


New Algorithm for Combinatorial n -Folds and Applications

Klaus Jansen 


Department of Computer Science, Kiel University, Germany

Kai Kahler 

Department of Computer Science, Kiel University, Germany

Lis Piroton 

Department of Computer Science, Kiel University, Germany

Malte Tutas 

Department of Computer Science, Kiel University, Germany

Abstract

Block-structured integer linear programs (ILPs) play an important role in various application fields. We address n -fold ILPs where the matrix \mathcal{A} has a specific structure, i.e., where the blocks in the lower part of \mathcal{A} consist only of the row vectors $(1, \dots, 1)$. In this paper, we propose an approach tailored to exactly these combinatorial n -folds. We utilize a divide and conquer approach to separate the original problem such that the right-hand side iteratively decreases in size. We show that this decrease in size can be calculated such that we only need to consider a bounded amount of possible right-hand sides. This, in turn, lets us efficiently combine solutions of the smaller right-hand sides to solve the original problem. We can decide the feasibility of, and also optimally solve, such problems in time $(nr\Delta)^{O(r)} \log(\|b\|_\infty)$, where n is the number of blocks, r the number of rows in the upper blocks and $\Delta = \|\mathcal{A}\|_\infty$.

We complement the algorithm by discussing applications of the n -fold ILPs with the specific structure we require. We consider the problems of (i) scheduling on uniform machines, (ii) closest string and (iii) (graph) imbalance. Regarding (i), our algorithm results in running times of $p_{\max}^{O(d)} |I|^{O(1)}$, matching a lower bound derived via ETH. For (ii) we achieve running times matching the current state-of-the-art in the general case. In contrast to the state-of-the-art, our result can leverage a bounded number of column-types to yield an improved running time. For (iii), we improve the parameter dependency on the size of the vertex cover.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases integer linear programming, n -fold, parameterized complexity, scheduling, uniform machines

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.15

Related Version *Full Version*: <https://arxiv.org/abs/2409.04212> [17]

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project numbers 453769249 and 528381760.

Acknowledgements We thank Martin Koutecký for the helpful discussions regarding the topic of $P\|C_{\max}$.

1 Introduction

At an abstract level, computer science is about solving hard problems. In a seminal work, Karp formulated a list of twenty-one NP-hard problems [24]. Among these is the problem of integer linear programming. An integer linear program (ILP) in standard form is defined by

$$\max\{c^T x \mid \mathcal{A}x = b, x \in \mathbb{Z}_{\geq 0}\},$$



© Klaus Jansen, Kai Kahler, Lis Piroton, and Malte Tutas;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Parameterized and Exact Computation (IPEC 2025).

Editors: Akanksha Agrawal and Erik Jan van Leeuwen; Article No. 15; pp. 15:1–15:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with constraint matrix $\mathcal{A} \in \mathbb{Z}^{m \times n}$, right-hand side (RHS) $b \in \mathbb{Z}^m$ and objective function vector $c \in \mathbb{Z}^n$. The constraints $Ax = b$ and the objective function $c^T x$ are required to be linear functions. Such integer programs have been the subject of vast amounts of research, such as [35, 22, 36, 21], with the latter two results representing the current state of the art of running times regarding n and m respectively. However, with integer programming being an NP-hard problem, research has flourished into different structures of integer programs, which utilize a more specific structure to yield improved running times. One such structure is the n -fold ILP.

n -fold ILP

Let $r, s, t, n \in \mathbb{Z}_{\geq 0}$. An n -fold ILP is an integer linear program with RHS vector $b \in \mathbb{Z}^{r+ns}$ and with a constraint matrix of the form

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

where $A_1, \dots, A_n \in \mathbb{Z}^{r \times t}$ and $B_1, \dots, B_n \in \mathbb{Z}^{s \times t}$ are matrices, and $x = (x_1, \dots, x_n)$ is the desired solution. In other words, if we delete the first r constraints, which are called the *global* constraints, the problem decomposes into independent programs, which are called *local*. If Δ is the largest absolute value in A_i and B_i , n -fold integer programs can be solved in time $(nt)^{(1+o(1))} \cdot 2^{O(rs^2)} (rs\Delta)^{O(r^2s+s^2)}$ [8]. This algorithm comes as the latest development in a series of improvements spanning almost twenty years [1, 8, 9, 32, 10, 16, 20, 30]. n -fold integer programs have already found applications in countless problems, often yielding improved running times over other techniques. As the total number of applications is too large, we refer the reader to [7, 32, 14, 19, 25, 26, 28] for some examples.

In the following, we focus on combinatorial n -fold ILPs where we allow blocks of different width, i.e., $t \in \mathbb{Z}_{\geq 0}^n$ is a vector and therefore, the block $i \in [n]$ has t_i columns. Also, we have $B_i = (1, \dots, 1)^T = \mathbb{1}_{t_i}$ for all $i \in [n]$ which implies $s = 1$. Thus, we have the following ILP

$$Ax = \begin{pmatrix} A_1 & A_2 & \dots & A_n \\ \mathbb{1}_{t_1} & 0 & \dots & 0 \\ 0 & \mathbb{1}_{t_2} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \mathbb{1}_{t_n} \end{pmatrix} x = b, \quad x \in \mathbb{Z}_{\geq 0}^h \quad (\spadesuit)$$

with $A_i \in \mathbb{Z}^{r \times t_i}$ for all $i \in [n]$, $b \in \mathbb{Z}^{r+n}$ and where $h := \sum_{i \in [n]} t_i$ is the total number of columns in \mathcal{A} . The currently best known algorithms for such combinatorial n -fold ILPs have parameter dependency $(r\Delta)^{O(r^2)}$ [8, 28]. In our work, we often consider the upper part of the vector b differently from its lower part. Therefore, we introduce the following notation: $b = (b^\uparrow, b^\downarrow)^T$ with $b^\uparrow \in \mathbb{Z}^r$ and $b^\downarrow \in \mathbb{Z}^n$. Also, define $b_{\max}^\uparrow := \max_{j \in [r]} b_j^\uparrow$, $b_{\max}^\downarrow := \max_{k \in [n]} b_k^\downarrow$ and $b_{\text{def}} := \min(b_{\max}^\uparrow, b_{\max}^\downarrow)$.

Our contribution

In this paper, we provide a novel approach to solving such combinatorial n -fold ILPs. While the idea of the algorithm we provide is inspired by Jansen and Rohwedder [21], we note that we utilize the structure of n -fold ILPs to generate different techniques to split solutions apart into smaller sub-solutions. Using these techniques, we achieve the following result.

► **Theorem 1.** *Combinatorial n -fold ILPs (\spadesuit) with respect to the objective of maximizing $c^T x$, $c \in \mathbb{Z}_{\geq 0}^h$ and where Δ is the largest absolute value in \mathcal{A} can be solved in time $(nr\Delta)^{O(r)} \log(b_{\text{def}})$.*

Comparatively, the algorithm in [21] yields a running time of $O(\sqrt{r+n}\Delta)^{2(r+n)} + O(h \cdot (r+n))$ when applied to our setting, while the state-of-the-art algorithm [8] achieves a running time of $(n\|t\|_\infty)^{(1+o(1))} \cdot 2^{O(r)} (r\Delta)^{O(r^2)}$.

To complement the general techniques we provide to solve combinatorial n -fold ILPs, we show how our result can be applied to common problems. First, we consider scheduling on uniform machines with the objective of makespan minimization ($Q\|C_{\max}$) and Santa Claus ($Q\|C_{\min}$). After publication of a preliminary version of our results, Rohwedder [37] recently bounded Δ by $p_{\max}^{O(1)}$ for $Q\|C_{\max}$, where p_{\max} is the largest processing time. They also provide an (almost) tight algorithm for $Q\|C_{\max}$ with running time $(dp_{\max})^{O(d)}(h+M)^{O(1)}|I|^{O(1)}$, where d is the number of job-types, h is the total number of columns in the configuration ILP and M is the total number of machines. Replacing a subroutine in [37] by our algorithm improves their result to $(dp_{\max})^{O(d)} \log(m_{\text{def}})|I|^{O(1)}$, where $m_{\text{def}} := \min(n_{\max}, m_{\max})$ i.e., we achieve a better result with respect to parameter h and M . This answers the open question by Koutecký and Zink in [31]. They pose the question whether $Q\|C_{\max}$ can be solved in time $p_{\max}^{O(d)}|I|^{O(1)}$. In more detail, the calculation of the Multi-Choice IP in [37] has a running time of $(nr\Delta)^{O(r)}(h + \|b^\downarrow\|_1)$, while our n -fold algorithm runs in $(nr\Delta)^{O(r)} \log(b_{\text{def}})$, i.e., solves these Multi-Choice IPs more quickly.

We also manage to extend these techniques to $Q\|C_{\min}$. This yields running times that are (almost) tight by an ETH-lower bound.

► **Theorem 2.** *$Q\|C_{\max}$ and $Q\|C_{\min}$ can be solved in time $p_{\max}^{O(d)}|I|^{O(1)}$.*

We also apply our algorithm to the CLOSEST STRING problem and the IMBALANCE problem (see Section 4 for formal definitions). The current best running times [28] for both problems have a quadratic dependency of k in the exponent, where k is the number of considered strings (CLOSEST STRING), respectively the size of the vertex cover (IMBALANCE). When the number of column-types (CLOSEST STRING), respectively vertex-types (IMBALANCE) is bounded by T , our algorithm achieves the following results.

► **Corollary 3.** *The CLOSEST STRING problem can be solved in time $((T+1)k)^{O(k)} \log(L)$.*

► **Corollary 4.** *The IMBALANCE problem with n vertices can be solved in time $((T+2)k)^{O(k)} \log(kn)$, where k is the size of a vertex cover and T is the number of vertex-types.*

Generally, T is bounded by k^k for CLOSEST STRING. In this case, we meet the state-of-the-art parameter dependency. For IMBALANCE, T may take any integer value up to 2^k . The resulting parameter dependency of 2^{k^2} improves on the state-of-the-art k^{k^2} which is the application of [28] to the problem. Additionally, when T is bounded by $k^{O(1)}$, we are able to reduce the dependency in the exponent to linear for both problems. When applying Rohwedder's algorithm for the Multi-Choice IP [37, Theorem 2] to those problems, the resulting running times are $((T+1)k)^{O(k)}O(T+dk)$, where d is the upper bound on string distance (CLOSEST STRING) and $((T+2)k)^{O(k)}O(Tk+nk)$ (IMBALANCE).

We expect that our techniques can be extended to several other problems that can be formulated as a combinatorial n -fold ILP as well, such as the string problems given in [28].

Related Work

Integer linear programming is one of the twenty-one problems originally proven to be NP-hard by Karp in his seminal work [24]. Since then, ILPs have been used to solve countless NP-hard problems, for examples see e.g. [14]. The currently best known running times can

be separated by their parameterization. When considering a small number of constraints m , the best known running time is given by Jansen and Rohwedder [21], and runs in time $O(\sqrt{m}\Delta)^{(1+o(1))m} + O(nm)$. For a small number of variables n , the current best known algorithm is given by Reis and Rothvoss [36], with a running time of $(\log n)^{O(n)} \cdot |\mathcal{A}|^{O(1)}$.

With the general ILP being a hard problem, research into certain substructures that are more easily solvable began. For an overview of this process, see e.g., [11, 14]. Of particular interest to this paper is the n -fold ILP, whose current best known algorithm is given by Cslovjcek, Eisenbrand, Hunkenschröder, Rohwedder and Weismantel [8]. Their algorithm runs in time $(nt)^{(1+o(1))} \cdot 2^{O(rs^2)}(rs\Delta)^{O(r^2s+s^2)}$ and is the latest result in a series of improvements that spans close to twenty years, see [9, 32, 10, 16, 20, 30] for works complementing this line of research.

Scheduling is one of the most well-known operations research problems. We now specifically turn towards uniform machines. By formulating the problem as an n -fold ILP, Knop and Koutecký [29] showed that $Q\|C_{\max}$ can be solved in time $p_{\max}^{O(p_{\max}^2)}|I|^{O(1)}$ if the number of machines is encoded in unary. By solving a relaxation to schedule many of the jobs in advance and then applying a dynamic program, Koutecký and Zink [31] improved this to $p_{\max}^{O(d^2)}|I|^{O(1)}$, still with the number of machines M encoded in unary. Leveraging the huge- n -fold machinery, Knop et al. [27] get rid of the assumption on M and obtain a $p_{\max}^{O(d^2)}|I|^{O(1)}$ -time algorithm. Koutecký and Zink [31] pose the open question whether the dependency $p_{\max}^{O(d^2)}$ for the objective $Q\|C_{\max}$ can be improved, as it has been for identical machines. In a recent result, Rohwedder [37] gives an algorithm for $Q\|C_{\max}$ running in time $p_{\max}^{O(d)}|I|^{O(1)}$. We note that our work has been compiled independently and prior to Rohwedders result, but we can leverage his novel ideas to reduce the value of Δ to guarantee a similar result. Chen, Jansen and Zhang show that, for a given makespan T , there is no algorithm solving $P\|C_{\max}$ in time $2^{T^{1-\delta}}|I|^{O(1)}$ [6]. Jansen, Kahler and Zwanger extended these results to derive a lower bound of $p_{\max}^{O(d^{1-\delta})}|I|^{O(1)}$ [18]. As $Q\|C_{\max}$ is a generalization of $P\|C_{\max}$, these results extend to it.

Regarding the CLOSEST STRING problem with k input strings of length L , Gramm et al. [15] formulated the problem as an ILP of dimension $k^{O(k)}$. This led to an algorithm with running time $2^{2^{O(k \log(k))}}O(\log(L))$. Knop et al. [28] improved this double-exponential dependency to the currently best known algorithm with running time $k^{O(k^2)}O(\log(L))$. Rohwedder and Węgrzycki [38] showed that the CLOSEST STRING problem (with binary alphabet) cannot be solved in time $2^{O(k^{2-\varepsilon})}\text{poly}(L)$ with $\varepsilon > 0$ unless the ILP Hypothesis fails. The ILP Hypothesis states that for every $\varepsilon > 0$, there is no $2^{O(m^{2-\varepsilon})}\text{poly}(L)$ -time algorithm for ILPs with $\Delta = O(1)$.

Imbalance is known to be NP-complete for various special graph classes [4, 23]. Fellows, Lokshtanov, Misra, Rosamond and Saurabh [13] showed that the problem is fixed-parameter tractable (FPT) parameterized by the size k of the vertex cover. Their algorithm has a double exponential running time. There also exist FPT algorithms for this problem parameterized by other parameters, for instance imbalance [33], neighborhood diversity [2] and the combined parameter of treewidth and maximum degree [33]. Misra and Mittal [34] showed that IMBALANCE is in XP when parameterized by twin cover, and FPT when parameterized by the twin cover and the size of the largest clique outside the twin cover.

2 Overview

In this section, we give an overview of the paper and the techniques within. We begin by giving some concepts essential to the functionality of our algorithm. Then, we prove the feasibility version of Theorem 1 under the assumption that \mathcal{A} contains only non-negative

entries. At its core, we use a divide and conquer approach to separate the entire problem into smaller sub-problems, which we can then solve more quickly via dynamic programming. Then, we combine these small solutions such that they solve iteratively larger sub-problems, again requiring a novel structural result to show that such combinations are feasible. A core strength of this dynamic programming formulation and approach is that we only need to solve the dynamic program once for each recombination step. This procedure can be iterated until we have solved the entire n -fold ILP. After that, we show how we can reduce the general problem which also allows negative coefficients to the special case where \mathcal{A} is non-negative. After that, we prove Theorem 1.

The algorithm we provide is complemented by a discussion on its applications to relevant problems. We begin this by elaborating on the problem of scheduling on uniform machines in Section 4. Here, we can adapt some novel concepts to our techniques such that we achieve a running time that is (almost) tight according to a lower bound derived via the exponential time hypothesis. This yields Theorem 2. Finally, in the full version [17] we provide a brief outlook onto other relevant problems, the CLOSEST STRING problem which results in Corollary 3 and the IMBALANCE problem which results in Corollary 4.

The full proofs as well as omitted statements can be found in the full version [17].

Preliminaries

Before we define concepts essential to the algorithm and its functionality, we first specify some notation. We denote $[k] := \{i \in \mathbb{Z}_{\geq 0} \mid 1 \leq i \leq k\}$ and use base 2 logarithms, i.e., we assume $\log(2) = 1$. A vector that contains a certain value in all entries is stylized, e.g. the zero vector 0_d is written as $\mathbf{0}_d$. We omit the dimension d if it is apparent from the context. For vector v , we refer to its maximum value $\|v\|_\infty$ with v_{\max} . The support of a d -dimensional vector v is the set of its indices with a non-zero entry. It is denoted by $\text{supp}(v) = \{i \in [d] \mid v_i \neq 0\}$. We define a *brick* $x^{(i)} \in \mathbb{Z}^{t_i}$ as the i -th block of the solution vector of (\spadesuit) , i.e., we have $x = (x^{(1)}x^{(2)} \dots x^{(n)})^T$, with $x^{(i)} \in \mathbb{Z}^{t_i}$ for all $i \in [n]$.

Support of a Solution

By applying the result by Eisenbrand and Shmonin [12, Theorem 1 (ii)] to the n -fold ILP, one can show the following support bound:

► **Lemma 5.** *Consider the ILP (\spadesuit) and let Δ be the largest absolute value of any entry in the coefficient matrix \mathcal{A} . Then there exists a solution $x = (x_1, \dots, x_n)^T$ to (\spadesuit) with*

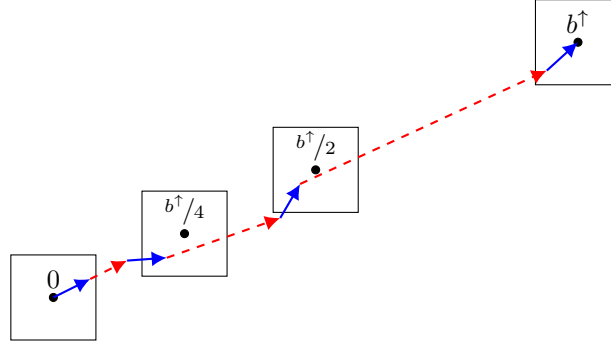
$$|\text{supp}(x_k)| \leq 2(r+1) \log(4(r+1)\Delta), \quad \forall k \in [n].$$

Proof. Let $y = (y_1, \dots, y_n)^T$ be a solution to (\spadesuit) . Then for each $k \in [n]$, the ILP

$$\begin{pmatrix} A_k \\ \mathbf{1}^T \end{pmatrix} x_k = \begin{pmatrix} A_k y_k \\ b_{r+k} \end{pmatrix} \quad x_k \in \mathbb{Z}_{\geq 0}^c$$

is feasible and we can apply the support bound by Eisenbrand and Shmonin [12, Theorem 1 (ii)]. Hence, there exists a solution x_k with $|\text{supp}(x_k)| \leq 2(r+1) \log(4(r+1)\Delta)$. Combining these solutions for all $k \in [n]$ yields a solution x with the desired property. ◀

Note that this bound only holds for the feasibility problem. Using the support bound by Eisenbrand and Shmonin [12, Corollary 5 (ii)], the same technique achieves the following result $|\text{supp}(x_k)| \leq 2(r+2)(\log(r+2) + \Delta + 2)$. Other bounds, for instance [3], can also be applied in our approach and may achieve better results in specific application settings.



■ **Figure 1** Example of the procedure with four iterations. The continuous (blue) arrows indicate a feasible upper RHS of the sub-problem in the current iteration. The dashed (red) arrows double the intermediate solution of the previous iteration.

3 Combinatorial n -fold Algorithm

On a high level, our algorithm implements an elegant idea. Instead of solving the – often times large – entire n -fold ILP in one fell swoop, we manage to reduce it to a set of smaller problems, which we then use to solve the large problem. More specifically, we show that there exists a feasible solution to (♠) that can be constructed by doubling a solution *near* $b^\uparrow/2$ and adding the solution of a small sub-problem. This concept can be repeated iteratively, i.e., a solution near $b^\uparrow/2$ can be computed by doubling a solution near $b^\uparrow/4$ and adding the solution of another small sub-problem, and so on. We repeat this until the remaining problem only contains a small upper RHS b^\uparrow . For an illustration of this process, see Figure 1. A core strength of this approach is that we only need to compute this dynamic program once for every iteration, independent of the amount of candidate solutions.

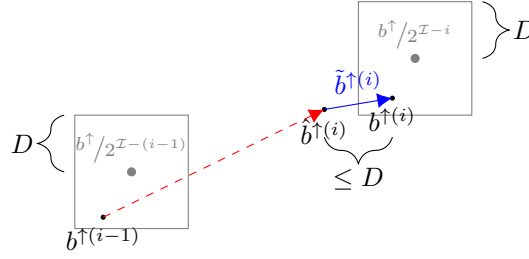
Taking a closer look, we use insights generated from the structure of our repeatedly doubled intermediary solutions to uniquely determine the lower RHS b^\downarrow at every step of the algorithm. As we double solutions in each step, we can calculate the total number of iterations our algorithm needs. We denote this number by $\mathcal{I} \in \mathbb{Z}_{\geq 1}$. Later in this section, we give the exact value of \mathcal{I} and provide a proof that our algorithm needs exactly \mathcal{I} iterations.

For any iteration $i \in [\mathcal{I}]$, we denote the lower RHS by $b^{\downarrow(i)}$. In the first iteration, and every time we solve a small sub-problem, we know by Lemma 5 that the support of the individual bricks is bounded by

$$K := \lfloor 2 \cdot (r+1) \cdot \log(4 \cdot (r+1) \cdot \Delta) \rfloor, \quad (1)$$

which also limits the upper RHS, which we refer to as D , of these sub-problems. Later, we show how this bound D is determined. The more detailed process is illustrated in Figure 2. Here, the continuous (blue) arrow contains the solution to the small sub-problem, while the dashed (red) arrow represents the doubling of an earlier solution.

We begin the description of our algorithm by showing that the lower RHS in each of these iterations is uniquely determinable. We further show that we can indeed find solutions close to each $b^\uparrow/2^{i-\mathcal{I}}$ that can be doubled to produce a feasible solution. Finally, we show how to utilize these insights to construct a feasible schedule while only needing to iteratively solve small sub-problems.



■ **Figure 2** Zoom in on a single iteration i including an illustration of the designation of the upper RHS $b^{\uparrow(i-1)}$, $\hat{b}^{\uparrow(i)}$, $\tilde{b}^{\uparrow(i)}$ and $b^{\uparrow(i)}$ and the upper bound D of the box sizes and $\|\tilde{b}^{\uparrow(i)}\|_{\infty}$.

3.1 Structure of Intermediary Solutions

In this section, we show that if (\spadesuit) is feasible, there exists a feasible upper RHS near $b^{\uparrow}/2^{T-i}$, for all $i \in [T]$, which then can be extended to solve main n -fold. We say that a RHS b or upper RHS b^{\uparrow} is feasible if and only if there exists a solution x to $\mathcal{A}x = (b^{\uparrow}, b^{\downarrow})^T$. We bound the size of the RHS in the small sub-problems needed to be solved in order to extend these intermediary solutions in each step.

Within this section, we view the problem top-down, i.e., in each iteration $i \in [T]$, we divide the problem into two sub-problems. The first sub-problem, which we call *parity constrained*, contains only even entries in its lower RHS \hat{b}^{\downarrow} . Because of this, that sub-problem can then be divided in two equal parts which we solve inductively. We call the second sub-problem, with lower RHS \tilde{b}^{\downarrow} , *small*. The *small* sub-problem fulfills $\|\tilde{b}^{\downarrow}\|_{\infty} \leq K$ and can therefore be solved efficiently using dynamic programming.

For a given iteration i , we denote the RHS of the considered problem by $b^{(i)}$. Note that we have $b^{(T)} = b$. The division into the two sub-problems implies $b^{(i)} = \hat{b}^{(i)} + \tilde{b}^{(i)}$. As we divide parity constrained problem into two equal problems and then repeat the procedure, we have $b^{(i-1)} = \hat{b}^{(i)}/2$. We ensure, that the upper RHS $b^{\uparrow(i)}$ also contains only even entries.

With this notation in mind, we begin by determining the lower RHS values for each iteration and their sub-problems. To determine the value of the k -th entry in the lower RHS of the small sub-problems $\tilde{b}^{\downarrow(i)}$, we consider the corresponding entry $b_k^{\downarrow(i)}$. If that entry is small, that is if $b_k^{\downarrow(i)} \leq K$, we set $\tilde{b}_k^{\downarrow(i)} = b_k^{\downarrow(i)}$. This implies $\hat{b}_k^{\downarrow(i)} = b_k^{\downarrow(i)} - \tilde{b}_k^{\downarrow(i)} = 0$ and $b_k^{\downarrow(i-1)} = \hat{b}_k^{\downarrow(i)}/2 = 0$. Therefore, for all following iterations this entry remains 0, i.e., $b_k^{\downarrow(i')} = \tilde{b}_k^{\downarrow(i')} = 0$ and $\hat{b}_k^{\downarrow(i')} = 0$ for all $i' < i$. However, if $b_k^{\downarrow(i)}$ is large, that is if $b_k^{\downarrow(i)} > K$, we set $\tilde{b}_k^{\downarrow(i)}$ to K or $K-1$ such that $\hat{b}_k^{\downarrow(i)} = b_k^{\downarrow(i)} - \tilde{b}_k^{\downarrow(i)}$ is even. Thus, the parity constrained entry is larger than 0 and can be divided by 2. Now, set $b_k^{\downarrow(i-1)} = \hat{b}_k^{\downarrow(i)}/2$ and repeat the procedure for the next smaller iteration.

More formally, for known value of $b_k^{\downarrow(i+1)} > K$ we can derive $b_k^{\downarrow(i)}$ as follows:

$$b_k^{\downarrow(i)} = \frac{b_k^{\downarrow(i+1)} - (K + z_k^{(i+1)})}{2} \quad \text{with} \quad (2)$$

$$z_k^{(i+1)} = \begin{cases} 0, & \text{if } (b_k^{\downarrow(i+1)} \bmod 2) = (K \bmod 2) \\ 1, & \text{otherwise.} \end{cases}$$

The following lemmas derive that, despite of the uncertainty due to the parity-constraint, the lower RHS in each iteration can be uniquely determined from the input (\mathcal{A}, b) to (\spadesuit) .

► **Lemma 6** (\spadesuit). *Let $b_k^{\downarrow} \in \mathbb{Z}_{\geq 0}^n$ and $k \in [n]$. Then $b_k^{\downarrow(i)}$ is integer for all $i \in [T]$, where values are determined by the procedure iteratively applying (2).*

Using this property and (2), we achieve the desired result by elegantly formulating the term as a continued fraction and using a property of the geometric sum.

► **Lemma 7** (\asymp). *Let $b_k^\downarrow \in \mathbb{Z}_{\geq 0}^n$ and $k \in [n]$. Then $b_k^{\downarrow(i)}$, $i \in [\mathcal{I}]$ is determinable as*

$$b_k^{\downarrow(i)} = \begin{cases} 0, & \text{if } i \neq \mathcal{I} \text{ and } b_k^{\downarrow(i+1)} \leq K \\ \left\lfloor b_k^\downarrow / 2^{x-i} - K \cdot \sum_{\ell=i}^{\mathcal{I}-1} (1/2^{x-\ell}) \right\rfloor, & \text{otherwise.} \end{cases} \quad (3)$$

Proof Idea. We determine the interval in which $b_k^{\downarrow(i)}$ lies. The interval size is less than one. With Lemma 6, this implies the lemma. ◀

We now focus on the case, when $b_k^{\downarrow(i+1)}$ is large. Since we reduce the lower RHS in each entry by at most K such that the remaining entries are even, the remaining lower RHS can be divided by two. However, this does not directly imply that we can split the complete ILP into two equal sub-ILPs, as the upper RHS might have odd entries. To provide this, we give a procedure on how to reduce the solution of the n -fold within the iterations. We show that after the reduction, there exists a feasible solution for the remaining problem, which contains only even entries, which then implies that the complete RHS is even. Again, we make use of the support theorem (Lemma 5) of Eisenbrand and Shmonin [12, Theorem 1 (ii)]. Define

$$\tilde{x}_k^{(i)} := \begin{cases} x_k^{(i)}, & \text{if } \|x_k^{(i)}\|_1 \leq K \\ \left(\tilde{x}_{k,1}^{(i)}, \dots, \tilde{x}_{k,t_k}^{(i)} \right)^T, & \text{otherwise.} \end{cases} \quad (4)$$

In the latter case, determine $\left(\tilde{x}_{k,1}^{(i)}, \dots, \tilde{x}_{k,t_k}^{(i)} \right)^T$ as follows (for details we refer to the full version [17], where we present a pseudocode description of this procedure).

1. Set $\tilde{x}_k^{(i)} = \mathbf{0}_{t_i}$.
2. For all odd entries $\ell \in [t_k]$ in $x_{k,\ell}^{(i)}$, add 1 to $\tilde{x}_{k,\ell}^{(i)}$. This ensures the correct parity.
3. Until $\|\tilde{x}_k^{(i)}\|_1$ equals the desired value K or $K-1$, add 2 to an arbitrary component $\tilde{x}_{k,\ell}^{(i)}$, with $x_{k,\ell}^{(i)} - \tilde{x}_{k,\ell}^{(i)} \geq 0$. Note that there always exists such entry as otherwise, we would have $b_k^{\downarrow(i)} \leq K$ and the RHS of the remaining problem would be $\mathbf{0}$.

This procedure ensures that $x^{(i)} - \tilde{x}^{(i)} =: \hat{x}^{(i)}$ only consists of even components. Therefore, the RHS $\hat{b}^{(i)}$ of such ILP $\mathcal{A}\hat{x}^{(i)} = \hat{b}^{(i)}$ does not have any odd entries. Therefore, the complete problem can be split into two equal sub-problems.

Using the previously described reduction strategies, we now deduce how the upper RHS $b^{\uparrow(i)}$ changes during an iteration. As we do not know the solution x of (\spadesuit) , the intermediate solutions $x^{(i)}$ are also unknown. Note that the described algorithm only gives a procedure to derive the current intermediate solution, assuming the prior one is known. Therefore, we cannot uniquely determine the upper RHS of the sub-problems. In the following, we often refer to a feasible RHS b as a feasible upper RHS or point b^\uparrow when b^\downarrow is apparent from the context. Recall that it is enough to determine the feasible points near $b^\uparrow / 2^{x-i}$ for all $i \in [\mathcal{I}]$. Now, we determine the size of the boxes in which a feasible point must lie if (\spadesuit) is feasible. For this purpose set $D := n \cdot K \cdot \Delta$. The next lemma states that for given small lower RHS $\tilde{b}^{\downarrow(i)}$ the size of the upper RHS $\tilde{b}^{\uparrow(i)}$ is bounded by D in each dimension (Figure 2).

► **Lemma 8** (\asymp). *Let $\tilde{b}^{(i)} = (\tilde{b}^{\uparrow(i)}, \tilde{b}^{\downarrow(i)})^T$ with $i \in [\mathcal{I}]$, $\|\tilde{b}^{\downarrow(i)}\|_\infty \leq K$ and assume $\mathcal{A}\tilde{x}^{(i)} = \tilde{b}^{(i)}$ is feasible. Then it holds that $\|\tilde{b}^{\uparrow(i)}\|_\infty \leq D$.*

With this property, we can derive the boxes with all relevant feasible upper RHS of the sub-problems. The center of the box in iteration i is $b^\uparrow/2^{x-i}$ and the side length in each dimension is $2D$ (Figure 2).

► **Lemma 9** (\asymp). *Let $b = (b^\uparrow, b^\downarrow)^T$ and assume $\mathcal{A}x = b$ is feasible. Then the following two properties hold:*

- (i) *For all $i \in \{2, \dots, \mathcal{I}\}$, there exists $b^{\uparrow(i)} \in \mathbb{Z}_{\geq 0}^r$ with $\|b^\uparrow/2^{x-i} - b^{\uparrow(i)}\|_\infty \leq D$ and $\mathcal{A}x^{(i)} = (b^{\uparrow(i)}, b^{\downarrow(i)})^T$ is feasible.*
- (ii) *There exists $b^{\uparrow(1)} \in \mathbb{Z}_{\geq 0}^r$ with $\|b^{\uparrow(1)}\|_\infty \leq D$ and $\mathcal{A}x^{(1)} = (b^{\uparrow(1)}, b^{\downarrow(1)})^T$.*

This means that if there exists a feasible point $b^{\uparrow(i)}$ near $b^\uparrow/2^{x-i}$, there also exists another feasible point $b^{\uparrow(i-1)}$ near $b^\uparrow/2^{x-(i-1)}$ for all iterations $i \in \{3, \dots, \mathcal{I}\}$. We also find a feasible point $b^{\uparrow(1)}$ near $\mathbf{0}$ if there is a feasible point $b^{\uparrow(2)}$ near $b^\uparrow/2^{x-2}$. Altogether, this gives us a path from $\mathbf{0}$ to n , with at least one feasible point near each $b^\uparrow/2^{x-i}$ for all $i \in \{2, \dots, \mathcal{I}\}$.

Now, we determine the total number of needed iterations by showing how to determine the number of iterations until the lower RHS equals $\mathbf{0}$. Let \mathcal{I}_k be the number of *non-zero* iterations of machine-type k . An iteration i of machine-type k is non-zero when $b_k^{\downarrow(i)} \neq 0$.

► **Lemma 10** (\asymp). *Let $k \in [n]$, $b_k^\downarrow \in \mathbb{Z}_{\geq 0}$. Then the number of non-zero iterations \mathcal{I}_k can be determined by*

$$\mathcal{I}_k = \begin{cases} \log((b_k^\downarrow + K)/(2K+1)) + 2, & \text{if } \log((b_k^\downarrow + K)/(2K+1)) \text{ is integer} \\ \lceil \log((b_k^\downarrow + K)/(2K+1)) \rceil + 1, & \text{otherwise.} \end{cases}$$

Proof Idea. We use (3) in Lemma 7 to determine the iteration i with $b_k^{\downarrow(i)} > 0$ and $b_k^{\downarrow(i-1)} = 0$. As calculating $b_k^{\downarrow(i)}$ is deterministic, this can be calculated uniquely. ◀

Having determined the number of non-zero iterations of each machine-type, we can directly derive the total number of iterations \mathcal{I} , as $\mathcal{I} = \max_{k \in [n]} \mathcal{I}_k$.

► **Lemma 11** (\asymp). *For the total number of iterations \mathcal{I} it holds that*

$$\mathcal{I} = \begin{cases} \log((b_{\max}^\downarrow + K)/(2K+1)) + 2, & \text{if } \log((b_{\max}^\downarrow + K)/(2K+1)) \text{ is integer} \\ \lceil \log((b_{\max}^\downarrow + K)/(2K+1)) \rceil + 1, & \text{otherwise.} \end{cases}$$

With this in mind, we are now able to derive an algorithm where we start with a small n -fold ILP in the first iteration and build up the overall solution of (\spadesuit) step by step.

3.2 Solving the n -fold ILP

In this section, we introduce an algorithm that determines the feasibility of the n -fold ILP (\spadesuit) with only non-negative entries in \mathcal{A} . Combined with the reduction in Section 3.3, this results in the following:

► **Lemma 12.** *The feasibility of combinatorial n -fold ILPs (\spadesuit) where \mathcal{A} contains only non-negative coefficients and Δ is the largest coefficient in \mathcal{A} can be determined in time $(nr\Delta)^{O(r)} \log(b_{\text{def}})$.*

Proof Idea. In the preceding section we derived the property that we only have to solve small sub-problems (Lemma 8) and that we can combine their solutions in an efficient way to solve (\spadesuit). We split the algorithm into two parts. In the preprocessing we generate solutions for all required small sub-problems. Then, we generate the final solution by combining the intermediate solutions. A detailed proof of both the dynamic program and the combining step can be found in the full version [17].

Preprocessing. First, determine the number of non-zero iterations \mathcal{I}_k for all $k \in [n]$ and the total number of iterations \mathcal{I} with Lemma 10 and Lemma 11. Then determine all required lower RHS $b^{\downarrow(i)}$, $\tilde{b}^{\downarrow(i)}$ and $\hat{b}^{\downarrow(i)}$ for all $i \in [\mathcal{I}]$ as described in Section 3.1. For each vector $\tilde{b}^{\downarrow(i)}$ define the set of required upper RHS in iteration i :

$$\tilde{N}^{(i)} = \left\{ \tilde{b}^{\uparrow(i)} \in \mathbb{Z}_{\geq 0}^r \mid \mathcal{A}\tilde{x}^{(i)} = (\tilde{b}^{\uparrow(i)}, \tilde{b}^{\downarrow(i)})^T, \tilde{x}^{(i)} \in \mathbb{Z}_{\geq 0}^h, \tilde{b}^{\uparrow(i)} \leq D \right\}.$$

The elements in $\tilde{N}^{(i)}$ can be calculated via dynamic programming (Algorithm 1). Instead of determining the feasibility of the complete n -fold $\mathcal{A}\tilde{x}^{(i)} = (\tilde{b}^{\uparrow(i)}, \tilde{b}^{\downarrow(i)})^T$, we split it into its bricks. For each brick $k \in [n]$ and upper RHS $\nu \in \{0, \dots, K\Delta\}^r$, we determine the feasibility of $A_k \tilde{x}_k = \nu$ with $\|\tilde{x}_k\|_1 = \tilde{b}_k^{\downarrow(i)}$. These feasible points are then combined into the set $\tilde{N}^{(i)}$.

■ **Algorithm 1** dynamicProgram. The base table (BT) contains the feasibility of all possible points for each individual block. The dynamic table DT combines the feasibility of the sub-problems. The set of required points is returned.

Input: Lower RHS $\tilde{b}^{\downarrow(i)}$

Output: Set $\tilde{N}^{(i)}$ of required points (upper RHS)

```

1: for all  $k \in [n]$  and  $\nu \in \{0, \dots, K\Delta\}^r$  do                                     ▷ Base Case
2:    $BT(\nu, k) \leftarrow \begin{cases} \text{true}, & \text{if } A_k \tilde{x}_k = \nu \text{ with } \|\tilde{x}_k\|_1 = \tilde{b}_k^{\downarrow(i)}, \tilde{x}_k \in \mathbb{Z}_{\geq 0}^{t_i} \text{ is feasible} \\ \text{false}, & \text{otherwise} \end{cases}$ 
3:  $DT(\nu, 1) \leftarrow BT(\nu, 1)$ 
4: for all  $k = 2, \dots, n$  and  $\nu \in \{0, \dots, D\}^r$  do                               ▷ Inductive Step
5:    $DT(\nu, k) \leftarrow \bigvee_{\substack{\nu' + \nu'' = \nu, \\ \nu', \nu'' \text{ integral vectors}}} (DT(\nu', k-1) \wedge BT(\nu'', k))$ 

return  $\{\nu \in \{0, \dots, D\}^r \mid DT(\nu, n) = \text{true}\}$ 

```

The feasibility of the small sub-problems (base case) can be determined using a standard ILP algorithm. Applying the Jansen-Rohwedder algorithm [21] yields a running time of

$$O(\sqrt{r+1}\Delta)^{(1+o(1))(r+1)} + O(t_k(r+1)). \quad (5)$$

Combining the Solutions. Having solved all small sub-problems, we are now able to iteratively derive the feasibility of (\spadesuit) . In the first iteration, set $N^{(1)} := \tilde{N}^{(1)}$ as the set of feasible points after this iteration. Note that for all $b^{\uparrow(1)} \in N^{(1)}$ it holds that $\mathcal{A}x^{(1)} = (b^{\uparrow(1)}, b^{\downarrow(1)})^T$ is feasible.

In every other iteration $i = 2, \dots, \mathcal{I}$, we first double the solutions of the previous iteration and obtain the feasible ILP $\mathcal{A}\hat{x}^{(i)} = (\hat{b}^{\uparrow(i)}, \hat{b}^{\downarrow(i)})^T$, with $\hat{x}^{(i)} = 2x^{(i-1)}$, $\hat{b}^{\downarrow(i)} = 2b^{\downarrow(i-1)}$ and $\hat{b}^{\uparrow(i)} = 2b^{\uparrow(i-1)}$. Let $\hat{N}^{(i)} := \{\hat{b}^{\uparrow(i)} \in \mathbb{Z}_{\geq 0}^r \mid \hat{b}^{\uparrow(i)} = 2b^{\uparrow(i-1)}, b^{\uparrow(i-1)} \in N^{(i-1)}\}$ be the set of feasible points after that step. We now construct the set $N^{(i)}$ of the feasible points after this iteration by combining all vectors in $\hat{N}^{(i)}$ with all vectors in $\tilde{N}^{(i)}$ and only keep the ones close enough to $b^{\uparrow}/2^{x-i}$ (Lemma 9), i.e.,

$$N^{(i)} := \{b^{\uparrow(i)} \in \mathbb{Z}_{\geq 0}^r \mid b^{\uparrow(i)} = \hat{b}^{\uparrow(i)} + \tilde{b}^{\uparrow(i)}, \hat{b}^{\uparrow(i)} \in \hat{N}^{(i)}, \tilde{b}^{\uparrow(i)} \in \tilde{N}^{(i)}, \left\| \frac{\hat{b}^{\uparrow}}{2^{x-i}} - b^{\uparrow(i)} \right\|_{\infty} \leq D\}.$$

Note again that for all $b^{\uparrow(i)} \in N^{(i)}$ it holds that $\mathcal{A}x^{(i)} = (b^{\uparrow(i)}, b^{\downarrow(i)})^T$ is feasible.

Repeating these steps results in a set $N^{(\mathcal{I})}$ of feasible points with $\mathcal{A}x^{(\mathcal{I})} = (b^{\uparrow(\mathcal{I})}, b^{\downarrow(\mathcal{I})})^T$ and $b^{\downarrow(\mathcal{I})} = b^{\downarrow}, b^{\uparrow(\mathcal{I})} \in N^{(\mathcal{I})}$. Therefore, if $b^{\uparrow} \in N^{(\mathcal{I})}$ the n -fold ILP (\spadesuit) is feasible.

Running Time. The running time of the preprocessing is dominated by the dynamic program. In the worst case, it is called $O(n)$ times. Creating the base table (BT) of the DP, we have to solve $n(K\Delta + 1)^r$ small sub-problems, each having a running time of (5). The dynamic program then constructs a dynamic table (DT) with $n(D+1)^r$ entries. Each can be calculated by doing a boolean convolution using FFT which takes time $O(n(D+1)^r \cdot \log(n(D+1)^r))$ (see e.g. [5]). With $t_k \leq h$ for all $k \in [n]$, the total running time for the preprocessing amounts to

$$\underbrace{n(K\Delta + 1)^r \cdot O(\sqrt{r+1}\Delta)^{(1+o(1))(r+1)} + O(h(r+1))}_{\text{Base Case}} + \underbrace{n^2(D+1)^{O(r)} \cdot \log(n(D+1)^r)}_{\text{Inductive Step}} \\ = (D+1)^{O(r)}$$

The number of iterations, we need to combine the solutions is generally bounded by $\log(b_{\max}^{\downarrow})$. However, if the upper RHS is relatively small in all entries, it becomes \mathcal{O} before b^{\downarrow} does. In order to be feasible in that case, we only need to check whether a \mathcal{O} in each A_i where the corresponding entry in the lower RHS is not 0, exists. Thus, these sub-problems are trivial to solve and therefore, we may say that we need $\log(b_{\text{def}})$ iterations, where $b_{\text{def}} := \min(b_{\max}^{\uparrow}, b_{\max}^{\downarrow})$. Each set $N^{(i)}, \tilde{N}^{(i)}$ and $\hat{N}^{(i)}$ contains at most $(D+1)^r$ vectors. Therefore, determining each set is possible in time $D^{O(r)}$. With $h \leq (\Delta+1)^r$ this yields a total running time of $(nr\Delta)^{O(r)} \log(b_{\text{def}})$. \blacktriangleleft

3.3 Allowing Negative Coefficients

In this section, we provide a linear-time reduction from a general combinatorial n -fold ILP (\spadesuit) to an ILP of the same form which only allows non-negative entries in \mathcal{A} . More concretely, we prove:

► **Lemma 13.** *Let $X \subseteq \mathbb{Z}_{\geq 0}^h$ be the set of feasible solutions of the n -fold ILP (\spadesuit) with sub-matrices $A_i \in \mathbb{Z}^{r \times t_i} \forall i \in [n]$ and RHS vector $b \in \mathbb{Z}^{r+n}$ and let Δ be the largest absolute value in A_i , i.e., $\Delta = \max_{i \in [n]} \|A_i\|_{\infty}$. Then there are matrices $A'_i \in \mathbb{Z}_{\geq 0}^{r \times t_i} \forall i \in [n]$ and a vector $b' \in \mathbb{Z}_{\geq 0}^{r+n}$ such that X is the set of feasible solutions to $\mathcal{A}'x = b'$ of the form (\spadesuit) with sub-matrices A_i and the entries in A'_i are bounded by $\Delta' = 2\Delta$.*

Proof. Let $i \in [n]$. We construct A'_i as by adding Δ to each entry in A_i , i.e., $A'_i[k, \ell] := A_i[k, \ell] + \Delta$, where $A_i[k, \ell]$ is the entry in A_i that is in the k -th row and ℓ -th column (analogous notation for $A'_i[k, \ell]$). Since the absolute entries in A_i are smaller than or equal to Δ , all entries in A'_i are non-negative and upper bounded by $2\Delta =: \Delta'$.

Now we show how the RHS is adapted. Note that the local constraints of (\spadesuit) define the 1-norm of the solution vectors, i.e., for all $x \in X$ it holds that $\|x\|_1 = \|b^{\downarrow}\|_1$. Viewing the matrix-multiplication as summing up vectors (columns of the matrix), we are summing up exactly $\|x\|_1$ vectors. Therefore, by changing the constraint matrix as described above, we know that exactly $\|b^{\downarrow}\|_1 \cdot \Delta$ are added to each entry of the upper RHS. Thus, we define $b^{\uparrow'}$ as follows: For each $j \in [r]$ set $b_j^{\uparrow'} := b_j^{\uparrow} + \|b^{\downarrow}\|_1 \cdot \Delta$. Since the local constraints do not change, the lower RHS stays the same that is $b^{\downarrow'} := b^{\downarrow}$. \blacktriangleleft

3.4 Solving the Optimization Problem

While we focused our discussions on the problem of deciding whether a feasible solution to a given combinatorial n -fold exists, we note that some modifications further extend our results to finding an optimal solution according to some objective function, for instance $\max c^T x, c \in \mathbb{Z}^h$. We need to adapt the upper bound on the support (Lemma 5) and the dynamic program (Algorithm 1) according to this new objective function. For $\max c^T x, c \in \mathbb{Z}^h$, the running time of the dynamic program changes as follows: Instead of determining the feasibility of the small sub-problems in the base case, we need the optimal solution of those. The currently best algorithm by Jansen and Rohwedder [21] achieves a running time of $O(\sqrt{r+1}\Delta)^{2(r+1)} + O(h(r+1))$. In the inductive step, we replace the boolean convolution with $(\max, +)$ -convolution, which has generally has quadratic running time. This and the change of the support bound to $2(r+2)(\log(r+2) + \Delta + 2)$ does not affect the asymptotic running time which yields:

► **Theorem 1.** *Combinatorial n -fold ILPs (\spadesuit) with respect to the objective of maximizing $c^T x, c \in \mathbb{Z}_{\geq 0}^h$ and where Δ is the largest absolute value in \mathcal{A} can be solved in time $(nr\Delta)^{O(r)} \log(b_{\text{def}})$.*

4 Applying the Algorithm

To complete the discussion of our algorithm, we present exemplary applications of the algorithm to different contexts. Here, we introduce the considered problems, while we refer to the full version [17] for the detailed applications and resulting running times.

Scheduling on Uniform Machines

A scheduling instance is defined by the number of jobs $n \in \mathbb{Z}_{\geq 0}^d$ with processing times $p \in \mathbb{Z}_{\geq 0}^d$ and the number of machines $m \in \mathbb{Z}_{\geq 0}^{\tau}$ with speed values $s \in \mathbb{Z}_{\geq 1}^{\tau}$. This means that $N := \|n\|_1$ jobs of d different sizes have to be scheduled on $M := \|m\|_1$ machines with τ_I different speeds. Since the processing times are integer, the total number of different job-types d is bounded by $d \leq p_{\max}$. Each job has to be executed on a single machine.

Let \mathcal{M} be the set of all machines. A *schedule* $\sigma: \mathbb{Z}_{\geq 0}^d \rightarrow \mathcal{M}$ is a mapping that assigns all jobs to the available machines. The *load* $L(m^{(k)})^{(\sigma)}$ of machine $m^{(k)} \in \mathcal{M}$ is the sum of the sizes of all jobs assigned to that machine by the schedule σ . Assume $m^{(k)}$ has speed s_k . Then we define the *completion time* of machine $m^{(i)}$ by $C(m^{(i)})^{(\sigma)} := L(m^{(i)})^{(\sigma)} / s_k$, i.e., the time when machine $m^{(i)}$ has finished processing all jobs that were assigned to it by σ .

The objective functions that we consider in this work are:

- **Makespan Minimization (C_{\max}):** $\min_{\sigma} \left(\max_{m^{(i)} \in \mathcal{M}} C(m^{(i)})^{(\sigma)} \right)$
- **Santa Claus (C_{\min}):** $\max_{\sigma} \left(\min_{m^{(i)} \in \mathcal{M}} C(m^{(i)})^{(\sigma)} \right)$.

Closest String Problem

When considering the CLOSEST STRING problem, we are given a set of k strings $S = \{s_1, \dots, s_k\}$ of length $L \in \mathbb{Z}_{\geq 1}$ from alphabet $[k]$ and an integer $d \in \mathbb{Z}_{\geq 0}$. The task is to find a string $c \in [k]^L$ of length L (called the “closest string”) such that the Hamming distance $d_H(c, s_i)$ between c and any string $s_i \in S$ is at most d . The Hamming distance for two equal-length strings is the number of positions at which they differ.

Imbalance Problem

The input to the IMBALANCE problem is an undirected graph $G = (V, E)$ with n vertices. An ordering of the vertices in G is a bijective function $\pi = V \rightarrow [n]$. For $v \in V$, we define neighborhood of v by $N(v) := \{u \in V \mid (u, v) \in E\}$, the set of left neighbors $L(v) := \{u \in N(v) \mid \pi(u) < \pi(v)\}$ and the set of right neighbors $R(v) := \{u \in N(v) \mid \pi(v) < \pi(u)\}$. Note that $R(v) = N(v) \setminus L(v)$. The imbalance of at $v \in V$ is defined as $\iota_\pi(v) = ||R(v)| - |L(v)||$ and the total imbalance of the ordering π is $\iota(\pi) = \sum_{v \in V} \iota_\pi(v)$. The aim is now to find an ordering π that minimizes the total imbalance.

5 Conclusion

We have shown that we can solve combinatorial n -fold ILPs in time $(nr\Delta)^{O(r)} \log(b_{\text{def}})$. Our algorithm builds on insights regarding the partition of the entire problem into smaller and smaller sub-problems, which we solve and use to reconstruct the entire problem in a novel manner. Complementing our algorithmic results, we present applications of our algorithm to the world of scheduling on uniform machines, the closest string problem and imbalance. In the case of scheduling on uniform machines, we can apply techniques to bound Δ and achieve an algorithm with a running time matching a lower bound provided by the ETH for both makespan minimization and Santa Claus. For closest string our algorithm matches the current state of the art, but can, in contrast to existing algorithms, leverage bounded number of column-types to achieve improved running times. Finally, when applied to imbalance, our algorithm results in an improved running time over the state-of-the-art. For both closest string and imbalance our algorithm matches the current state of the art, but can, in contrast to existing algorithms, leverage bounded number of column/vertex types to achieve improved running times.

References

- 1 Matthias Aschenbrenner and Raymond Hemmecke. Finiteness theorems in stochastic integer programming. *Foundations of Computational Mathematics*, 7(2):183–227, 2007. doi:10.1007/S10208-005-0174-1.
- 2 Olav Røthe Bakken. Arrangement problems parameterized by neighbourhood diversity. *Master's thesis, University of Bergen*, 2018.
- 3 Sebastian Berndt, Hauke Brinkop, Klaus Jansen, Matthias Mnich, and Tobias Stamm. New support size bounds for integer programming, applied to makespan minimization on uniformly related machines. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 283 of *LIPIcs*, pages 13:1–13:18, 2023. doi:10.4230/LIPICS.ISAAC.2023.13.
- 4 Therese Biedl, Timothy M. Chan, Yashar Ganjali, Mohammad Taghi Hajiaghayi, and David R. Wood. Balanced vertex-orderings of graphs. *Discrete Applied Mathematics*, 148(1):27–48, 2005. doi:10.1016/J.DAM.2004.12.001.
- 5 Karl Bringmann and Vasileios Nakos. Fast n -fold boolean convolution via additive combinatorics. In *International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 198 of *LIPIcs*, pages 41:1–41:17, 2021. doi:10.4230/LIPICS.ICALP.2021.41.
- 6 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *Journal of Computer and System Sciences*, 96:1–32, 2018. doi:10.1016/J.JCSS.2018.03.005.
- 7 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *Symposium on Theoretical Aspects of Computer Science, (STACS)*, volume 66 of *LIPIcs*, pages 22:1–22:14, 2017. doi:10.4230/LIPICS.STACS.2017.22.

- 8 Jana Cslovjceksek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1666–1681, 2021. doi:10.1137/1.9781611976465.101.
- 9 Jana Cslovjceksek, Friedrich Eisenbrand, Michal Pilipczuk, Moritz Venzin, and Robert Weismantel. Efficient sequential and parallel algorithms for multistage stochastic integer programming using proximity. In *European Symposium on Algorithms, ESA*, volume 204 of *LIPICs*, pages 33:1–33:14, 2021. doi:10.4230/LIPICs.ESA.2021.33.
- 10 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In *International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 107 of *LIPICs*, pages 49:1–49:13, 2018. doi:10.4230/LIPICs.ICALP.2018.49.
- 11 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming, 2022. arXiv:1904.01361.
- 12 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Operations Research Letters*, 34(5):564–568, 2006. doi:10.1016/J.ORL.2005.09.008.
- 13 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 5369 of *LNCS*, pages 294–305, 2008. doi:10.1007/978-3-540-92182-0_28.
- 14 Tomas Gavenciak, Martin Koutecký, and Dusan Knop. Integer programming in parameterized complexity: Five miniatures. *Discrete Optimization*, 44(Part):100596, 2022. doi:10.1016/J.DISOPT.2020.100596.
- 15 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, 37(1):25–42, 2003. doi:10.1007/S00453-003-1028-3.
- 16 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. n -fold integer programming in cubic time. *Mathematical Programming*, 137(1-2):325–341, 2013. doi:10.1007/S10107-011-0490-Y.
- 17 Klaus Jansen, Kai Kahler, Lis Piroton, and Malte Tutas. New algorithm for combinatorial n -folds and applications, 2025. arXiv:2409.04212.
- 18 Klaus Jansen, Kai Kahler, and Esther Zwanger. Exact and approximate high-multiplicity scheduling on identical machines. In *Conference on Algorithms and Complexity (CIAC)*, volume 15679 of *LNCS*, pages 1–17, 2025. doi:10.1007/978-3-031-92932-8_1.
- 19 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-IP: new PTAS results for scheduling with setup times. *Mathematical Programming*, 195(1):367–401, 2022. doi:10.1007/S10107-021-01694-3.
- 20 Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. Near-linear time algorithm for n -fold ILPs via color coding. *SIAM Journal on Discrete Mathematics*, 34(4):2282–2299, 2020. doi:10.1137/19M1303873.
- 21 Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution. *Mathematics of Operations Research*, 48(3):1481–1495, 2023. doi:10.1287/MOOR.2022.1308.
- 22 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.
- 23 Jan Kára, Jan Kratochvíl, and David R. Wood. On the complexity of the balanced vertex ordering problem. *Discrete Mathematics & Theoretical Computer Science*, 9(1), 2007. doi:10.46298/DMTCS.383.
- 24 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

- 25 Dusan Knop and Martin Koutecký. Scheduling meets n -fold integer programming. *Journal of Scheduling*, 21(5):493–503, 2018. doi:10.1007/S10951-017-0550-0.
- 26 Dusan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Parameterized complexity of configuration integer programs. *Operations Research Letters*, 49(6):908–913, 2021. doi:10.1016/J.ORL.2021.11.005.
- 27 Dusan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. High-multiplicity n -fold IP via configuration LP. *Mathematical Programming*, 200(1):199–227, 2023. doi:10.1007/S10107-022-01882-9.
- 28 Dusan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n -fold integer programming and applications. *Mathematical Programming*, 184(1):1–34, 2020. doi:10.1007/S10107-019-01402-2.
- 29 Dušan Knop and Martin Koutecký. Scheduling Meets N -Fold Integer Programming. *Journal of Scheduling*, 21(5):493–503, October 2018. doi:10.1007/S10951-017-0550-0.
- 30 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 107 of *LIPICs*, pages 85:1–85:14, 2018. doi:10.4230/LIPICs.ICALP.2018.85.
- 31 Martin Koutecký and Johannes Zink. Complexity of scheduling few types of jobs on related and unrelated machines. *Journal of Scheduling*, 28(1):139–156, 2025. doi:10.1007/S10951-024-00827-8.
- 32 Jesús A. De Loera, Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. N -fold integer programming. *Discrete Optimization*, 5(2):231–241, 2008. doi:10.1016/J.DISOPT.2006.06.006.
- 33 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Imbalance is fixed parameter tractable. *Information Processing Letters*, 113(19-21):714–718, 2013. doi:10.1016/J.IPL.2013.06.010.
- 34 Neeldhara Misra and Harshil Mittal. Imbalance parameterized by twin cover revisited. *Theoretical Computer Science*, 895:1–15, 2021. doi:10.1016/J.TCS.2021.09.017.
- 35 Christos H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.
- 36 Victor Reis and Thomas Rothvoss. The subspace flatness conjecture and faster integer programming. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 974–988, 2023. doi:10.1109/FOCS57990.2023.00060.
- 37 Lars Rohwedder. Eth-tight FPT algorithm for makespan minimization on uniform machines. In *International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 334 of *LIPICs*, pages 126:1–126:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICs.ICALP.2025.126.
- 38 Lars Rohwedder and Karol Wegrzycki. Fine-grained equivalence for problems related to integer linear programming. In *Innovations in Theoretical Computer Science (ITCS)*, volume 325 of *LIPICs*, pages 83:1–83:18, 2025. doi:10.4230/LIPICs.ITCS.2025.83.