Evaluating and Tuning n-fold Integer Programming

Kateřina Altmanová[1]
Department of Applied Mathematics, Charles University
Prague, Czech Republic
kacka@kam.mff.cuni.cz

Dušan Knop[2]
Department of Informatics, University of Bergen, Bergen, Norway and
Department of Applied Mathematics, Charles University, Prague, Czech Republic
dusan.knop@uib.no
https://orcid.org/0000-0003-2588-5709

Martin Koutecký[3]
Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology
Haifa, Israel
koutecky@technion.ac.il
https://orcid.org/0000-0002-7846-0053

Abstract

In recent years, algorithmic breakthroughs in stringology, computational social choice, scheduling, etc., were achieved by applying the theory of so-called n-fold integer programming. An n-fold integer program (IP) has a highly uniform block structured constraint matrix. Hemmecke, Onn, and Romanchuk [Math. Programming, 2013] showed an algorithm with runtime $a^{O(rst+r^2s)}n^3$, where a is the largest coefficient, r, s, and t are dimensions of blocks of the constraint matrix and n is the total dimension of the IP; thus, an algorithm efficient if the blocks are of small size and with small coefficients. The algorithm works by iteratively improving a feasible solution with augmenting steps, and n-fold IPs have the special property that augmenting steps are guaranteed to exist in a not-too-large neighborhood. However, this algorithm has never been implemented and evaluated.

We have implemented the algorithm and learned the following along the way. The original algorithm is practically unusable, but we discover a series of improvements which make its evaluation possible. Crucially, we observe that a certain constant in the algorithm can be treated as a tuning parameter, which yields an efficient heuristic (essentially searching in a smaller-than-guaranteed neighborhood). Furthermore, the algorithm uses an overly expensive strategy to find a "best" step, while finding only an "approximatelly best" step is much cheaper, yet sufficient for quick convergence. Using this insight, we improve the asymptotic dependence on n from n^3 to $n^2 \log n$ which yields the currently asymptotically fastest algorithm for n-fold IP.

Finally, we tested the behavior of the algorithm with various values of the tuning parameter and different strategies of finding improving steps. First, we show that decreasing the tuning parameter initially leads to an increased number of iterations needed for convergence and eventually to getting stuck in local optima, as expected. However, surprisingly small values of the parameter already exhibit good behavior. Second, our new strategy for finding "approximatelly best" steps wildly outperforms the original construction.

2012 ACM Subject Classification Software and its engineering → Software design engineering, Theory of computation → Parameterized complexity and exact algorithms

[1] Author was supported the project 17-09142S of GA ČR.
[2] Author supported supported by the project NFR MULTIVAL and the project P202/12/G061 of GA ČR.
[3] Author supported by Technion postdoctoral fellowship.

© Kateřina Altmanová, Dušan Knop, and Martin Koutecký;
licensed under Creative Commons License CC-BY

Keywords and phrases n-fold integer programming, integer programming, analysis of algorithms, primal heuristic, local search

Digital Object Identifier 10.4230/LIPIcs.SEA.2018.10

Supplement Material `https://github.com/katealtmanova/nfoldexperiment`

1 Introduction

In this article we consider the general integer linear programming (ILP) problem in standard form,

$$\min \{\mathbf{w}\mathbf{x} \mid A\mathbf{x} = \mathbf{b}\,,\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\,,\ \mathbf{x} \in \mathbb{Z}^n\}\,. \tag{ILP}$$

with A an integer $m \times n$ matrix, $\mathbf{b} \in \mathbb{Z}^m$, $\mathbf{w} \in \mathbb{Z}^n$, $\mathbf{l}, \mathbf{u} \in (\mathbb{Z} \cup \{\pm\infty\})^n$. It is well known to be strongly NP-hard, but models many important problems in combinatorial optimization such as planning [28], scheduling [13] and transportation [4] and thus powerful generic solvers have been developed for it [25]. Still, theory is motivated to search for tractable special cases One such special case is when the constraint matrix A has a so-called N-fold structure:

$$A = E^{(N)} = \begin{pmatrix} E_1 & E_1 & \cdots & E_1 \\ E_2 & 0 & \cdots & 0 \\ 0 & E_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & E_2 \end{pmatrix} .$$

Here, $r, s, t, N \in \mathbb{N}$, $\mathbf{u}, \mathbf{l}, \mathbf{w} \in \mathbb{Z}^{Nt}$, $\mathbf{b} \in \mathbb{Z}^{r+Ns}$, $E^{(N)}$ is an $(r + Ns) \times Nt$-matrix, $E_1 \in \mathbb{Z}^{r\times t}$ is an $r \times t$-matrix and $E_2 \in \mathbb{Z}^{s\times t}$ is an $s \times t$-matrix. We call $E^{(N)}$ the *N-fold product of* $E = \left(\begin{smallmatrix} E_1 \\ E_2 \end{smallmatrix}\right)$. Problem (ILP) with $A = E^{(N)}$ is known as *N-fold integer programming* (N-fold IP). Hemmecke, Onn, and Romanchuk [16] prove the following.

▶ **Proposition 1** ([16, Theorem 6.2]). *There is an algorithm that solves*[4] (ILP) *with* $A = E^{(N)}$ *encoded with* L *bits in time* $\Delta^{O(trs+t^2s)} \cdot n^3L$, *where* $\Delta = 1 + \max\{\|E_1\|_\infty, \|E_2\|_\infty\}$.

Recently, algorithmic breakthroughs in stringology [20], computational social choice [21], scheduling [5, 18, 22], etc., were achieved by applying this algorithm and its subsequent non-trivial improvements.

The algorithm belongs to the larger family of augmentation (primal) algorithms. It starts with an initial feasible solution $\mathbf{x}_0 \in \mathbb{Z}^{Nt}$ and produces a sequence of increasingly better solutions $\mathbf{x}_1, \ldots, \mathbf{x}_s$ (better means $\mathbf{w}\mathbf{x}_s < \mathbf{w}\mathbf{x}_{s-1} < \cdots < \mathbf{w}\mathbf{x}_0$). It is guaranteed that the algorithm terminates, that $\mathbf{x}_s$ is an optimal solution, and that the algorithm converges quickly, i.e., s is polynomial in the length of the input. A key property of N-fold IPs is that, if an augmenting step exists, then it can be decomposed into a bounded number of elements of the so-called *Graver basis* of A, which in turn makes it possible to compute it using dynamic programming [16, Lemma 3.1]. In a sense, this property makes the algorithm a local search algorithm which is always guaranteed to find an improvement in a not-too-large neighborhood. The bound on the number of elements or the size of the neighborhood which

[4] Given an IP, we say that to *solve* it is to either (i) declare it infeasible or unbounded or (ii) find a minimizer of it.

needs to be searched is called the *Graver complexity of E*. This, in turn, implies that, if an augmenting step exists, then there is always one with small ℓ_1-norm; for a matrix A, we denote this bound $g_1(A) = \max_{\mathbf{g}\in\mathcal{G}(A)} \|\mathbf{g}\|_1$ [24, Theorem 4]. However, the algorithm has never been implemented and evaluated.

1.1 Our Contributions

We have implemented the algorithm and tested it on two problems for which n-fold formulations were known: makespan minimization on uniformly related machines ($Q||C_{\max}$) and Closest String; we have used randomly generated instances.

In the course of implementing the algorithm we learn the following. The algorithm in its initial form is practically unusable due to an *a priori* construction of the Graver basis $\mathcal{G}(E_2)$ of size exponential in r, s, t and a related (even larger) set $Z(E)$. However, we discover a series of improvements (some building on recent insights [24]) which avoid the construction of these two sets. Moreover, we adjust the algorithm to treat $g_1(A)$ as a tuning parameter $\mathtt{g_1}$, which turns it into a heuristic.

We also study the *augmentation strategy*, which is the way the algorithm chooses an augmenting step among all the possible options. The original algorithm uses an overly expensive strategy to find a "best" step, which means that a large number of possible steps is evaluated in each iteration. We show that finding only an "approximatelly best" step is sufficient to obtain asymptotically equivalent convergence rate, and the work per iteration decreases exponentially. Using this insight, we improve the asymptotic dependence on N from N^3 to $N^2 \log N$. Together with recent improvements, this yields the currently asymptotically fastest algorithm for N-fold IP:

▶ **Theorem 2.** *Problem* (ILP) *with $A = E^{(N)}$ can be solved in time $\Delta^{r^2s+rs^2}(Nt)^2\log(Nt)M$, where $M = \log(\mathbf{wx}^* - \mathbf{wx}_0)$ for some minimizer $\mathbf{x}^*$ of $\mathbf{wx}$.*

Finally, we evaluate the behavior of the algorithm. We ask how is the performance of the algorithm (in terms of number of dynamic programming calls and quality of the returned solution) influenced by

1. the choice of the tuning parameter $1 < \mathtt{g_1} \leq g_1(A)$?
2. the choice of the augmentation strategy between "best step", "approximate best step" and "any step"?

As expected, as $\mathtt{g_1}$ moves from $g_1(A)$ to 1, we first see an increase in the number of iterations needed for convergence and eventually the algorithm gets stuck in local optima. However, surprisingly small values (e.g. $\mathtt{g_1} = 5$ when $g_1(A) > 10^5$) of the parameter already exhibit close to optimal behavior. Second, our new strategy for finding "approximatelly best" steps outperforms the original construction by orders of magnitude, while the naive "any step" strategy behaves erratically.

We note that at this stage we are *not* (yet) interested in showing supremacy over existing algorithms; we simply want to understand the practical behavior of an algorithm whose theoretical importance was recently highlighted. For this reasons our experimental focus is on the two aforementioned questions rather than simply measuring the time. Similarly, due to the rigid format of $E^{(N)}$ we are limited to few problems for which N-fold formulations are known. For Closest String we use the same instances as Chimani et al. [6]; for Makespan Minimization we generate our own data because standard benchmarks are not limited to short jobs or few types of jobs.

1.2 Related Work

Our work mainly relates to *primal heuristics* [3] for MIPs which are used to help reach optimality faster and provide good feasible solutions early in the termination process. Specifically, our algorithm is a *neighborhood* (or *local*) *search algorithm.* The standard paradigm is *Large Neighborhood Search* (LNS) [27] with specializations such as for example *Relaxation Induced Neighborhood Search* (RINS) [7] and *Feasibility Pump* [2]. Using this paradigm, our proposed algorithm searches in the neighborhood induced by the ℓ_1-distance from the current feasible solution and the search procedure is formulated as an ILP subproblem with the additional constraint $\|\mathbf{x}\|_1 \leq \mathbf{g}_1$. In this sense the closest technique to ours is *local branching* [11] which also searches in the ℓ_1-neighborhood; however, we treat a discovered step as a *direction* and apply it exhaustively, so, unlike in local branching, we make long steps. Moreover, local branching was mainly applied to binary programs without any additional structure of the constraint matrix.

On the theoretical side, very recently Koutecký et al. [24] have studied parameterized strongly polynomial algorithms for various block-structured ILPs, not just N-fold IP. Eisenbrand et al. [9] independently (and using slightly different techniques) arrive at the same complexity of N-fold IP as our Theorem 2.

2 Preliminaries

For positive integers m, n we set $[m, n] = \{m, \ldots, n\}$ and $[n] = [1, n]$. We write vectors in boldface (e.g., $\mathbf{x}, \mathbf{y}$) and their entries in normal font (e.g., the i-th entry of $\mathbf{x}$ is x_i). Given the problem (ILP), we say that $\mathbf{x}$ is *feasible* for (ILP) if $A\mathbf{x} = \mathbf{b}$ and $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$.

Graver bases and augmentation. Let us now introduce Graver bases and discuss how they can be used for optimization. We also recall N-fold IPs; for background, we refer to the books of Onn [26] and De Loera et al. [8].

Let $\mathbf{x}, \mathbf{y}$ be n-dimensional integer vectors. We call $\mathbf{x}, \mathbf{y}$ *sign-compatible* if they lie in the same orthant, that is, for each $i \in [n]$ the sign of x_i and y_i is the same. We call $\sum_i \mathbf{g}^i$ a *sign-compatible sum* if all $\mathbf{g}^i$ are pair-wise sign-compatible. Moreover, we write $\mathbf{y} \sqsubseteq \mathbf{x}$ if $\mathbf{x}$ and $\mathbf{y}$ are sign-compatible and $|y_i| \leq |x_i|$ for each $i \in [n]$. Clearly, $\sqsubseteq$ imposes a partial order called "conformal order" on n-dimensional vectors. For an integer matrix $A \in \mathbb{Z}^{m \times n}$, its *Graver basis* $\mathcal{G}(A)$ is the set of $\sqsubseteq$-minimal non-zero elements of the *lattice* of A, $\ker_{\mathbb{Z}}(A) = \{\mathbf{z} \in \mathbb{Z}^n \mid A\mathbf{z} = \mathbf{0}\}$. An important property of $\mathcal{G}(A)$ is the following.

▶ **Proposition 3** ([26, Lemma 3.4]). *Every integer vector $\mathbf{x} \neq \mathbf{0}$ with $A\mathbf{x} = \mathbf{0}$ is a sign-compatible sum $\mathbf{x} = \sum_{i=1}^{t} \alpha_i \mathbf{g}^i$, $\alpha_i \in \mathbb{N}$, $\mathbf{g}^i \in \mathcal{G}(A)$ and $t \leq 2n - 2$.*

Let $\mathbf{x}$ be a feasible solution to (ILP). We call $\mathbf{g}$ an $\mathbf{x}$-*feasible step* (or simply *feasible* if $\mathbf{x}$ is clear) if $\mathbf{x} + \mathbf{g}$ is feasible for (ILP). Further, we call a feasible step $\mathbf{g}$ *augmenting* if $\mathbf{w}(\mathbf{x} + \mathbf{g}) < \mathbf{w}\mathbf{x}$; note that $\mathbf{g}$ decreases the objective by $\mathbf{w}\mathbf{g}$. An augmenting step $\mathbf{g}$ and a *step length* $\gamma \in \mathbb{N}$ form an $\mathbf{x}$-*feasible step pair* with respect to a feasible solution $\mathbf{x}$ if $\mathbf{l} \leq \mathbf{x} + \gamma\mathbf{g} \leq \mathbf{u}$. A pair $(\gamma, \mathbf{g}) \in (\mathbb{N} \times \mathcal{G}(A))$ is a γ-*Graver-best step pair* and $\gamma\mathbf{g}$ is a γ-*Graver-best step* if it is feasible and for every feasible step pair $(\gamma, \mathbf{g}')$, $\mathbf{g}' \in \mathcal{G}(A)$, we have $\mathbf{w}\gamma\mathbf{g} \leq \mathbf{w}\gamma\mathbf{g}'$. An augmenting step $\mathbf{g}$ and a step length $\gamma \in \mathbb{N}$ form a *Graver-best step pair* if it is γ-Graver-best and it minimizes $\mathbf{w}\gamma'\mathbf{g}'$ over all $\gamma' \in \mathbb{N}$, where $(\gamma', \mathbf{g}')$ is a γ'-Graver-best step pair. We say that $\gamma\mathbf{g}$ is a *Graver-best step* if $(\gamma, \mathbf{g})$ is a Graver-best step pair.

The *Graver-best augmentation procedure* for (ILP) with a given feasible solution $\mathbf{x}_0$ and initial value $i = 0$ works as follows:

1. If there is no Graver-best step for $\mathbf{x}_i$, return it as optimal.
2. If a Graver-best step $\gamma\mathbf{g}$ for $\mathbf{x}_i$ exists, set $\mathbf{x}_{i+1} := \mathbf{x}_i + \gamma\mathbf{g}$, $i := i + 1$, and go to 1.

▶ **Proposition 4** (Convergence bound [26, Lemma 3.10]). *Given a feasible solution* $\mathbf{x}_0$ *for* (ILP)*, the Graver-best augmentation procedure finds an optimum in at most* $(2n-2)\log M$ *steps, where* $M = \mathbf{w}(\mathbf{x}_0 - \mathbf{x}^*)$ *and* $\mathbf{x}^*$ *is any minimizer of* $\mathbf{wx}$.

By standard techniques (detecting unboundedness etc.) we can ensure that $\log M \le L$.

N-fold IP. The structure of $E^{(N)}$ allows us to divide the Nt variables of $\mathbf{x}$ into N *bricks* of size t. We use subscripts to index within a brick and superscripts to denote the index of the brick, i.e., x^i_j is the j-th variable of the i-th brick with $j \in [t]$ and $i \in [N]$.

3 Approximate Graver-best Steps

In this section we introduce the notion of a c-approximate Graver-best step (Definition 5), show that such steps exhibit good convergence (Lemma 6), can be easily obtained (Lemma 7), and result in a significant speed-up of the N-fold IP algorithm (Theorem 2).

▶ **Definition 5** (c-approximate Graver-best step). Let $c \in \mathbb{R}$, $c \ge 1$. Given an instance of (ILP) and a feasible solution $\mathbf{x}$, we say that $\mathbf{h}$ is a *c-approximate Graver-best step for* $\mathbf{x}$ if, for every $\mathbf{x}$-feasible step pair $(\gamma, \mathbf{g}) \in (\mathbb{N} \times \mathcal{G}(A))$, we have $\mathbf{wh} \le \frac{1}{c} \cdot \gamma\mathbf{wg}$.

Recall the Graver-best augmentation procedure. We call its analogue where we replace a Graver-best step with a c-approximate Graver-best step the *c-approximate Graver-best augmentation procedure.*

▶ **Lemma 6** (c-approximate convergence bound). *Given a feasible solution* $\mathbf{x}_0$ *for* (ILP)*, the c-approximate Graver-best augmentation procedure finds an optimum of* (ILP) *in at most* $c \cdot (2n-2)\log M$ *steps, where* $M = \mathbf{w}(\mathbf{x}_0 - \mathbf{x}^*)$ *and* $\mathbf{x}^*$ *is any minimizer of* $\mathbf{wx}$.

Proof. The proof is a straightforward adaptation of the proof of Proposition 4. Let $\mathbf{x}^*$ be a minimizer and let $\mathbf{h} = \mathbf{x}^* - \mathbf{x}_0$. Since $A\mathbf{h} = \mathbf{0}$, by Proposition 3, $\mathbf{h} = \sum_{i=1}^{2n-2} \alpha_i \mathbf{g}^i$ for some $\alpha_i \in \mathbb{N}$, $\mathbf{g}^i \in \mathcal{G}(A)$, $i \in [2n-2]$. Thus, an $\mathbf{x}$-feasible step pair $(\gamma, \mathbf{g})$ such that $\gamma\mathbf{g}$ is a Graver-best step must satisfy $\mathbf{w}\gamma\mathbf{g} \le \frac{1}{2n-2}M$. In other words, any Graver-best step pair improves the objective function by at least a $\frac{1}{2n-2}$-fraction of the total optimality gap M, and thus $(2n-2)\log M$ steps suffice to reach an optimum. It is straightforward to see that a c-approximate Graver-best step satisfies $\mathbf{wx} - \mathbf{w}(\mathbf{x} + \gamma\mathbf{g}) \le \frac{c}{2n-2}M$, and thus $c(2n-2)\log M$ steps suffice. ◀

▶ Remark. Lemma 6 extends naturally to separable objectives; see the original proof [26, Lemma 3.10].

▶ **Lemma 7** (Powers of c step lengths). *Let* $c \in \mathbb{N}$, $\mathbf{x}$ *be a feasible solution of* (ILP)*, and let*

$$\Gamma_{c\text{-}apx} = \left\{c^i \mid \exists \mathbf{g} \in \mathcal{G}(A) : \mathbf{l} \le \mathbf{x} + c^i\mathbf{g} \le \mathbf{u}\right\} .$$

Let $(\gamma, \mathbf{g}) \in (\Gamma_{c\text{-}apx} \times \mathcal{G}(A))$ *be an* $\mathbf{x}$*-feasible step pair such that* $\gamma\mathbf{g} \le \gamma'\mathbf{g}'$ *for any* $\mathbf{x}$*-feasible step pair* $(\gamma', \mathbf{g}') \in (\Gamma_{c\text{-}apx} \times \mathcal{G}(A))$*. Then* $\gamma\mathbf{g}$ *is a c-approximate Graver-best step.*

Proof. Let $(\gamma, \mathbf{g})$ satisfy the assumptions, and let $(\tilde{\gamma}, \tilde{\mathbf{g}}) \in (\mathbb{N} \times \mathcal{G}(A))$ be a Graver-best step pair. Let γ' be a nearest smaller power of c from $\tilde{\gamma}$, and observe that $\gamma' \tilde{\mathbf{g}}$ is a c-approximate Graver-best step because $\gamma' \geq \frac{\tilde{\gamma}}{c}$. On the other hand, since $\gamma \mathbf{g}$ is a γ-Graver-best step, we have $\gamma \mathbf{g} \leq \gamma' \tilde{\mathbf{g}}$ and thus $\gamma \mathbf{g}$ is also a c-approximate Graver-best step. ◀

▶ **Theorem 2** (restated). *Problem* (ILP) *with* $A = E^{(N)}$ *can be solved in time* $\Delta^{O(r^2 s + rs^2)} (Nt)^2 \log(Nt) M$*, where* $M = \log(\mathbf{w}\mathbf{x}^* - \mathbf{w}\mathbf{x}_0)$ *for some minimizer* $\mathbf{x}^*$ *of* $\mathbf{w}\mathbf{x}$.

Proof. Recall that $\Delta = \|A\|_\infty + 1$. Koutecký et al. [24, Theorem 2] show that a γ-Graver-best step can be found in time $\Delta^{r^2 s + rs^2} Nt$. Moreover, Hemmecke et al. [15] prove a proximity theorem which allows the reduction of an instance of (ILP) to an equivalent instance with new bounds $\mathbf{l}', \mathbf{u}'$ satisfying $\|\mathbf{u}' - \mathbf{l}'\|_\infty \leq N t g_\infty$, with

$$g_\infty = \max_{\mathbf{g} \in \mathcal{G}(A)} \|\mathbf{g}\|_\infty \leq \max_{\mathbf{g} \in \mathcal{G}(A)} \|\mathbf{g}\|_1 \leq (\Delta rs)^{O(rs)} ,$$

where the last inequality can be found in the proof of [24, Theorem 4]. This bound implies that $\Gamma_{\text{2-apx}}$ from in Lemma 7 satisfies $|\Gamma_{\text{2-apx}}| \leq rs \log(\Delta N trs)$. By Lemma 7, finding a γ-Graver-best for each $\gamma \in \Gamma_{\text{2-apx}}$ and picking the minimum results in a 2-approximate Graver-best step, and can be done in time $\Delta^{r^2 s + rs^2} (Nt) \log(Nt)$. By Lemma 6, $(4n-4) \log M$ steps suffice to reach the optimum. ◀

4 Implementation

We first give an overview of the original algorithm, which is our starting point. Then we discuss our specific improvements and mention a few details of the software implementation.

4.1 Overview of the Original Algorithm

The key property of the N-fold product $E^{(N)}$ is that, for any $N \in \mathbb{N}$, the number of nonzero bricks of any $\mathbf{g} \in \mathcal{G}(E^{(N)})$ is bounded by some constant $g(E)$ called the *Graver complexity of* E, and, moreover, that the sum of all non-zero bricks of $\mathbf{g}$ can be decomposed into at most $g(E)$ elements of $\mathcal{G}(E_2)$ [16, Lemma 3.1]. This facilitates the following construction. Let

$$Z(E) = \left\{ \mathbf{z} \in \mathbb{Z}^t \mid \exists \mathbf{g}^1, \ldots, \mathbf{g}^k \in \mathcal{G}(E_2),\, k \leq g(E),\, \mathbf{z} = \sum_{i=1}^{k} \mathbf{g}^i \right\} .$$

Then, every prefix sum of the bricks of $\mathbf{g} \in \mathcal{G}(E^{(N)})$ is contained in $Z(E)$ and a γ-Graver-best step, $\gamma \in \mathbb{N}$, can be found using dynamic programming over the elements of $Z(E)$.

To ensure that a Graver-best step is found, a set of step-lengths Γ_{best} is constructed as follows. Observe that any Graver-best (and thus feasible) step pair $(\gamma, \mathbf{g}) \in \left(\mathbb{N} \times \mathcal{G}(E^{(N)})\right)$, must satisfy that in at least one brick $i \in [n]$ it is "tight", that is, $(\gamma, \mathbf{g})$ is $\mathbf{x}$-feasible while $(\gamma + 1, \mathbf{g})$ is not specifically because $\mathbf{l}^i \leq \mathbf{x} + \gamma \mathbf{g} \leq \mathbf{u}^i$ holds but $\mathbf{l}^i \leq \mathbf{x} + (\gamma + 1)\mathbf{g} \leq \mathbf{u}^i$ does not. Thus, for each $\mathbf{z} \in Z(E)$ and each $i \in [n]$, we find all the potentially "tight" step lengths γ and add them to Γ_{best}, which results in a bound of $|\Gamma_{\text{best}}| \leq |Z(E)| \cdot n$.

4.2 Replacing Dynamic Programming with ILP

We have started off by implementing the algorithm exactly as it is described by Hemmecke et al. [16]. The first obstacle is encountered almost immediately and is contained in the constant $g(E)$. This constant can be computed, but the computation is extremely difficult [10, 14].

Algorithm 1: Pseudocode of the algorithm of Hemmecke, Onn, and Romanchuk.

```
    input  : matrices $E_1, E_2$, positive integer $N$, and vectors $\mathbf{b}, \mathbf{l}, \mathbf{u}, \mathbf{w}$
    output : optimal solution to (ILP) with $A = E^{(N)}$
 1  $g = \texttt{GraverComplexity}(E_1, E_2)$;
 2  $\mathbf{x}_0 = \texttt{FindFeasibleSolution}(E, \mathbf{b}, \mathbf{l}, \mathbf{u}, \mathbf{w}),\ i = 0$;
 3  $\mathcal{G}(E_1) = \texttt{GraverBasis}(E_1, g)$;
 4  $Z(E) = \texttt{DynamicProgramStates}(\mathcal{G}(E_1), g)$;
 5  do
 6      $\Gamma = \texttt{BuildGamma}(\mathbf{x}_i)$;
 7      $i = i + 1$;
 8      foreach $\gamma \in \Gamma$ do
 9          $\mathbf{g}_\gamma = \texttt{gammaBestStep}(\gamma, \mathbf{g})$;
10      $\mathbf{x}_i = \mathbf{x}_{i-1} + \mathrm{argmin}_{\{\mathbf{g}_\gamma \mid \gamma \in \Gamma\}} \mathbf{w}\mathbf{g}_\gamma$;
11  while $\mathbf{x}_{i-1} \neq \mathbf{x}_i$;
12  return $\mathbf{x}_i$;
```

Another possibility is to estimate it, in which case it is almost always larger than N and thus is essentially meaningless. Finally, one can take the approach partially suggested in [16, Section 7], where we consider $g(E)$ in the construction of $Z(E)$ to be a tuning parameter and consider the approximate set $Z_{\texttt{gc}}(E)$, $\texttt{gc} \in \mathbb{N}$, obtained by taking sums of at most $\texttt{gc}$ elements of $\mathcal{G}(E_2)$. This makes the algorithm more practical, but turns it into a heuristic.

In spite of this sacrifice, already for small ($r = 3$, $s = 1$, $t = 7$, $n = 10$) instances and extremely small value of $\texttt{gc} = 3$, the dynamic programming based on the $Z_{\texttt{gc}}(E)$ construction was taking an unreasonably long time (over one minute). Admittedly this could be improved; however, already for $\texttt{gc} > 5$, it becomes infeasible to compute $Z_{\texttt{gc}}(E)$, and for larger instances ($r > 5$, $t > 12$) it becomes very difficult to compute even $\mathcal{G}(E_2)$. For these reasons we sought to completely replace the dynamic program involving $Z(E)$.

Koutecký et al. [24] show that all instances of (ILP) with the property that the so-called *dual treedepth* $\mathrm{td}_D(A)$ *of* A is bounded and the largest coefficient $\|A\|_\infty$ is bounded also have the property that $g_1(A) = \max_{\mathbf{g} \in \mathcal{G}(A)} \|\mathbf{g}\|_1$ is bounded, which implies that augmenting steps can be found efficiently. This class of ILPs contains N-fold IP.

The interpretation of the above fact is that, in order to solve (ILP), it is sufficient to repeatedly (for different $\mathbf{x}$ and γ) solve an auxiliary instance

$$\min \{\mathbf{w}\mathbf{h} \mid A\mathbf{h} = \mathbf{0}, \mathbf{l} \leq \mathbf{x} + \gamma\mathbf{h} \leq \mathbf{u}, \|\mathbf{h}\|_1 \leq g_1(A)\} \qquad \text{(AugILP)}$$

in order to find good augmenting steps; we note that the constraint $\|\mathbf{h}\|_1 \leq g_1(A)$ can be linearized [24, Lemma 25]. The heuristic approach outlined above transfers easily: we replace $g_1(A)$ in (AugILP) with some integer $\texttt{g}_1$, $1 < \texttt{g}_1 \leq g_1(A)$; this makes (AugILP) easier to solve at the cost of losing the guarantee that an augmenting step is found if one exists. In theory, solving (AugILP) should be easier than solving the original instance (ILP) due to the special structure of A [24, Lemma 25]. Our approach here is to simply invoke an industrial MILP solver on (AugILP) in order to find a γ-Graver-best step.

4.3 Augmentation Strategy: Step Lengths

Logarithmic Γ. The majority of algorithms based on Graver basis augmentation rely on the Graver-best augmentation procedure [5, 8, 16, 20, 22, 26] and thus require finding

Algorithm 2: Pseudocode of our new heuristic algorithm. The algorithm is exact if $g_1 \geq g_1(A) = \max_{\mathbf{g} \in \mathcal{G}(A)} \|\mathbf{g}\|_1$.

```
input  : matrices $E_1, E_2$, positive integers $N$, $c$, and $g_1$, and vectors $\mathbf{b}, \mathbf{l}, \mathbf{u}, \mathbf{w}$
output : a feasible solution to (ILP) with $A = E^{(N)}$
1  $\mathbf{x}_0 = \texttt{FindFeasibleSolution}(E, \mathbf{b}, \mathbf{l}, \mathbf{u}, \mathbf{w}),\ i = 0$;
2  do
3      $\Gamma = \emptyset;\ j = 0,\ i = i + 1$;
4      do
5          $\gamma = c^j$;
6          $\mathbf{g}_\gamma = \min\{\mathbf{w}\gamma\mathbf{g} \mid A\mathbf{g} = \mathbf{0},\ \mathbf{l} \leq \mathbf{x} + \gamma\mathbf{g} \leq \mathbf{u},\ \|\mathbf{g}\|_1 \leq g_1,\ \mathbf{g} \in \mathbb{Z}^{Nt}\}$;
7          $\gamma' = \texttt{ExhaustDirection}(\mathbf{g}_\gamma)$;
8          $\Gamma = \Gamma \cup \{\gamma'\},\ j = j + 1$;
9      while $\mathbf{g}_\gamma \neq \mathbf{0}$;
10     $\mathbf{x}_i = \mathbf{x}_{i-1} + \mathrm{argmin}_{\{\mathbf{g}_\gamma \mid \gamma \in \Gamma\}}\ \mathbf{w}\mathbf{g}_\gamma$;
11 while $\mathbf{x}_{i-1} \neq \mathbf{x}_i$;
12 return $\mathbf{x}_i$
```

(exact) Graver-best steps. In the aforementioned algorithms this is always done using the construction of a set Γ_{best} mentioned above, which is of size $f(k) \cdot n$ where k is the relevant parameter (e.g., $(ars)^{O(rst+st^2)}$ in the original algorithm for N-fold IP). We replace this construction with $\Gamma_{\text{2-apx}} = \{1, 2, 4, 8, \dots\}$ which, combined with the proximity technique, is only of size $O(\log n)$ (Theorem 2); in particular, independent of the function $f(k)$.

Exhausting γ. Moreover, we have noticed that sometimes the algorithm finds a step $\mathbf{g}$ for $\gamma = 2^k$ which is not tight in any brick, and then repeatedly applies it for shorter step-lengths $\gamma' < \gamma$. In other words, the discovered direction $\mathbf{g}$ is not *exhausted*. Thus, for each $\gamma \in \mathbb{N}$, upon finding the γ-Graver-best step $\mathbf{g}$, we replace γ with the largest $\gamma' \geq \gamma$ for which $(\gamma', \mathbf{g})$ is still $\mathbf{x}$-feasible.

Early termination. Another observation is that in any given iteration of the algorithm, if $\gamma > 1$ then *some* augmenting step has been found and if the computation is taking too long, we might terminate it and simply apply the best step found so far.

4.4 Software and Hardware

We have implemented our solver in the SageMath computer algebra system [31]. This was a convenient choice for several reasons. The SageMath system offers an interactive notebook-style web-based interface, which allows rapid prototyping and debugging. Data types for vectors and matrices, Graver basis algorithms [1], and a unified interface for MILP solvers are also readily available. We have experimented with the open-source solvers GLPK [30], Coin-OR CBC [29], and the commercial solver Gurobi [12] and have settled for using the latter since it performs the best. The downside of SageMath is that an implementation of the original dynamic program is likely much slower than a similar implementation in C; however this DP is impractical anyway as explained in Section 4.2. For random instance generation and subsequent data evaluation and graphing, we have used the Jupyter notebook environment [19] and Matplotlib library [17]. The computations were performed on a computer with an Intel® Xeon® E5-2630 v3 (2.40GHz) CPU and 128 GB RAM.

5 Evaluation

We begin our evaluation with two main questions, specifically, how is the performance of the algorithm (both in terms of the number of iterations and the quality of the returned solution) influenced by:

1. the tuning parameter g_1 and
2. the augmentation strategy?

Regarding our first question, theoretically we should see either an increase in the number of iterations, a decrease in the quality of the returned solution, or both. However, the range of the tuning parameter g_1 is quite large: any number between 2 and $g_1(A)$ is a valid choice, and in all our scenarios the true value of $g_1(A)$ exceeds 300. Thus, we are interested in the threshold values of g_1 when the algorithm no longer finds the true optimum or when its convergence rate drops significantly.

Regarding our second question, there are two main candidates for the set of step-lengths Γ. We can either use the "best step" construction Γ_{best} of the original algorithm, which assures that we always make a Graver-best step before moving to the next iteration. Or, we can use the "approximate best step" construction $\Gamma_{\text{2-apx}}$ of Theorem 2, which provides a 2-approximate Graver-best step. To make this comparison more interesting, we also consider $\Gamma_{\text{5-apx}}$ and also the trivial "any step" strategy where we always make the 1-Graver-best step, which corresponds to taking $\Gamma_{\text{any}} = \{1\}$. Recall that due to the trick of always exhausting the discovered direction, this strategy actually has a chance at quick convergence, unlike if we only made the step with $\gamma = 1$.

5.1 Instances

We choose two problems for which N-fold IP formulations were shown in the literature, namely the $Q||C_{\max}$ scheduling problem [22] and the Closest String problem [20].

Uniformly related machines makespan minimization ($Q||C_{\max}$)

Input: Set of m machines M, each with a speed $s_i \in \mathbb{N}$. A set of n jobs J, each with a processing time $p_j \in \mathbb{N}$.

Find: Find an assignment of jobs to machines such that the time when last job finishes (the makespan) is minimized; a job j scheduled on a machine i takes time p_j/s_i to execute.

Closest String

Input: A set of k strings $s_1, \ldots, s_k$ of length L over an alphabet Σ.

Find: Find a string $y \in \Sigma^L$ minimizing $\max_{i=1}^{k} d_H(s_i, y)$, where d_H is the Hamming distance.

For both problems we generate random instances as follows.

Scheduling. We view the problem as a decision problem where, given a number B, we ask whether a schedule of makespan at most B exists. This is equivalent to the multi-sized bin packing problem, where we have m bins of various capacities instead of m machines of different speeds, and we adopt this view as it is more convenient. We also view it as a high-multiplicity problem where the items are not given explicitely as a list of item sizes, but succinctly by a vector of item multiplicities. Because the algorithm is primarily an optimization algorithm, we follow the standard approach [16, Lemma 3.8] and turn the feasibility problem into an auxiliary optimization instance where finding a starting feasible solution is easy. Specifically for $Q||C_{\max}$ this means introducing auxiliary slack variables for "not-yet-scheduled" jobs and minimizing the total "not-yet-scheduled" length.

The input parameters of the instance generation are number of bins m, the smallest and largest capacities S and L, respectively, item sizes $p_1, \dots, p_k$ and probability weights $w_1, \dots, w_k$, $W = \sum_{i=1}^{k} w_i$, and a slack ratio σ. The instance is then generated as follows. First, we choose m capacities from $[S, L]$ uniformly at random. This determines the total capacity of the bins C. Our goal is to generate items whose total size is roughly $\sigma \cdot C$. We do this by repeatedly picking an item length from $p_1, \dots, p_k$, where p_j is selected with probability w_j/W, until the total size of items picked so far exceeds $\sigma \cdot C$, when we terminate and return the generated instance.

Closest String. As before, we view the problem as a decision problem: given a number $d \in \mathbb{N}$, decide whether there is a string y with $\max_{i=1}^{k} d_H(s_i, y) \leq d$. The random instance is generated exactly as done by Chimani et al. [6]: first, we generate a random "target" string $y \in \Sigma^L$ and create k copies $s_1, \dots, s_k$ of it; then, we make α random changes in $s_1, \dots, s_k$. This way, we have an upper bound α on the optimum. The input parameters of the instace generation are thus k, L, Σ and the distance ratio r such $\alpha = {}^{n}\!/_{r}$. Again, we solve an auxiliary instance where we essentially start with a string of "all blanks" and try to fill in all the blanks while staying in the specified distance d; the objective is thus the remaining number of blanks.

5.2 Results

Here we demonstrate the overall behavior of the algorithm on three selected instances; we encourage the reader to see the full data (incl. plots) at `https://github.com/katealtmanova/nfoldexperiment`. We have determined that the sensible range of values of $\mathtt{g_1}$ is between 2 and roughly 100 and beyond that the behavior does not change significantly. For all augmentation strategies there are values of $\mathtt{g_1}$ which take much longer to converge than any other values; because of these outliers we clip our figures. To expose the behavior of the algorithm we use two types of figures.

Outer loop. In the *outer loop figure* we focus on the behavior of the algorithm with respect to the loop starting at line 2 of Algorithm 2, i.e., where each iteration corresponds to making an augmenting step. There is a line plot for each tested value of $\mathtt{g_1}$. The x axis shows the iteration, the y axis shows the objective value attained in this iteration, i.e.,$\mathbf{w}\mathbf{x}_i$ for iteration i. We indicate the expensiveness of computing one augmentation by the thickness of the line in a given iteration – the thicker the line, the more times the (AugILP) has been solved in this iteration. The legend indicates the exact number of times (AugILP) has been solved for this value of $\mathtt{g_1}$.

Inner loop. In the *inner loop figure* we focus on the loop starting at line 4 of Algorithm 2, i.e., where iterations correspond to solutions of (AugILP). As before, each color corresponds to a tested value of $\mathtt{g_1}$. There is a line plot displaying the *minimum* over augmenting steps found in each outer iteration; however now there is also a semiopaque region above this line, indicating the values of all the augmenting steps (including the non-minimal ones) found in this iteration.

We chose two instances among the tested once as representative of the overall behavior:

- Our first instance is $Q||C_{\max}$ with parameters $m = 15$, $S = 2000$, $L = 10000$, item sizes $(2, 3, 13, 35)$ (so that we have an instance with nontrivial $\|A\|_\infty$), and weights $(6, 13, 2, 1)$ and $\sigma = 0.45$. The theoretical upper bound on $g_1(A)$ is $(rs\|A\|_\infty + 1)^{O(rs)}$, and here we

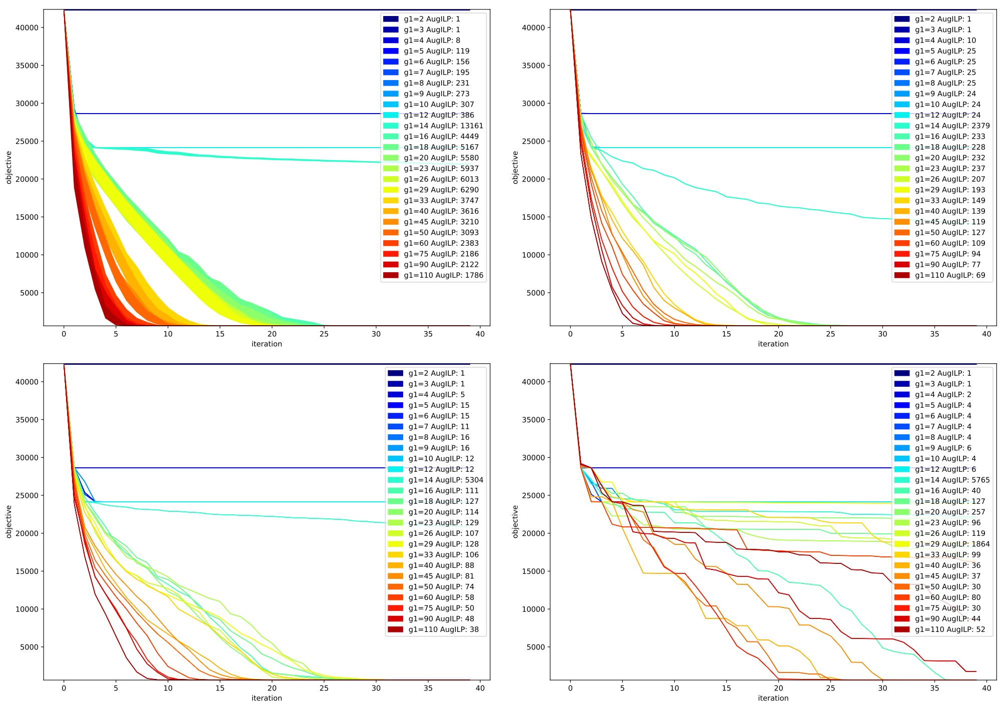

Figure 1 Outer loop results for Makespan Minimization when using (left to right) Γ_{best}, $\Gamma_{\text{2-apx}}$, $\Gamma_{\text{5-apx}}$, and Γ_{any}; clipped to 40 outer iterations.

have $r = 4$, $s = 1$ and $\|A\|_\infty = 35$; thus, without computing $g_1(A)$ exactly, we should consider it to be at least $(4 \cdot 36)^4$. See Figures 1 and 2.

- The third instance is Closest String with parameters $k = 5$, $|\Sigma| = 2$, $L = 10000$ and $r = 1$. The N-fold model has $r = 5$, $s = 1$ and $\|A\|_\infty = 1$, thus, without computing $g_1(A)$ exactly, we should consider it to be at least $(2 \cdot 5)^5$. See Figure 3.

5.3 Conclusions

Our main takeaway regarding Question #1 is that, while the theoretical upper bounds for $g_1(A)$ are huge, already small values of $\mathbf{g}_1$ ($\mathbf{g}_1 > 5$ for Closest String and $\mathbf{g}_1 > 20$ for Makespan Minimization) are sufficient for convergence. We remark that, in the case of Closest String, this hints at the possibility that the maximum value of any *feasible* augmenting step $\mathbf{g} \in \mathcal{G}(A)$ is bounded by $k^{O(1)}$ rather than $k^{O(k)}$, which would imply an algorithm with runtime $k^{O(k)} \log L$ while the currently best algorithm runs in time $k^{O(k^2)} \log L$ [20].

Regarding Question #2, we see that $\Gamma_{\text{2-apx}}$ provides essentially the same convergence rate as Γ_{best} but is orders of magnitude cheaper to compute. The "any step" augmentation strategy Γ_{any} usually converges surprisingly quickly, but our results make it clear that its behavior is erratic and unpredictable. The inner loop Figure 2 reveals that a good step (close to a Graver-best step) is usually found for larger step-length γ; this motivates adding the $\Gamma_{\text{5-apx}}$ augmentation strategy to the comparison, as it spends less time on short step-lengths than $\Gamma_{\text{2-apx}}$. Figure 1 shows that using 5-approximate Graver-best steps instead of 2-approximate does not affect the outer loop convergence much, and Figure 2 shows that in terms of the total number of (AugILP) calls it performs better.

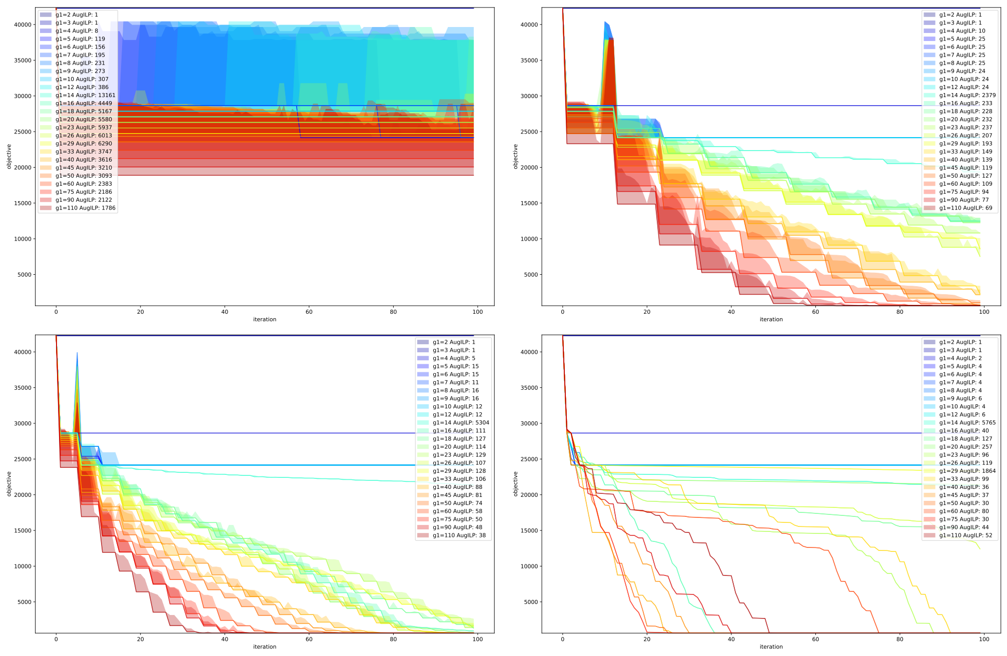

Figure 2 Inner loop results for MAKESPAN MINIMIZATION when using (left to right) Γ_{best}, $\Gamma_{\text{2-apx}}$, $\Gamma_{\text{5-apx}}$, and Γ_{any}; clipped to 100 inner iterations.

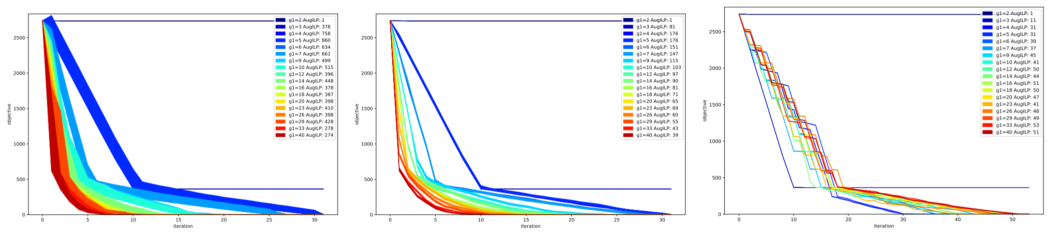

Figure 3 Outer loop results for the CLOSEST STRING instance when using (left to right) Γ_{best}, $\Gamma_{\text{2-apx}}$, and Γ_{any}.

Furthermore, we observe that solving (AugILP) using a MILP solver such as Gurobi typically takes essentially as much time as solving (ILP) itself; in other words, current MILP solvers are (without tuning) unable to make any use neither of the extra structure of $E^{(N)}$, nor the fact that we are seeking a solution with small ℓ_1 norm and the right hand side is $\mathbf{0}$. Moreover, with a growing number of bricks N, the time to solve (AugILP) using a MILP solver grows superlinearly, suggesting that, for large enough N, a specialized dynamic programming algorithm might be competitive with generic MILP solvers.

6 Outlook

We have initiated an experimental investigation of certain subclasses of ILP with block structured constraint matrices. Our results show that, as theory suggests, for such ILPs a primal algorithm always augmenting with steps of small ℓ_1 norm converges quickly. We close with a few interesting research directions. First, is there a way to tune generic MILP solvers to solve (AugILP) significantly faster than (ILP)? Second, what is the behavior of our algorithm on instances *other* than N-fold IP? For example, how large must be $\mathtt{g}_1$ in order to attain the optimum quickly for standard benchmark instances, e.g. MIPLIB [23]? Third, the approach of Koutecký et al. [24] suggests that a key property for the efficient solvability of (AugILP) is a certain "sparsity" and "shallowness" (formally captured by the graph parameter *tree-depth*) of graphs related to A; what are "natural" instances with small tree-depth?

References

1 4ti2 team. 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at www.4ti2.de.

2 Livio Bertacco, Matteo Fischetti, and Andrea Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007. Mixed Integer Programming. `doi:10.1016/j.disopt.2006.10.001`.

3 Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013.

4 Ralf Borndörfer, Martin Grötschel, and Ulrich Jäger. Planning problems in public transit. In *Production Factor Mathematics*, pages 95–121. Springer, 2010.

5 Lin Chen and Daniel Marx. Covering a tree with rooted subtrees–parameterized and approximation algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2801–2820. SIAM, 2018.

6 Markus Chimani, Matthias Woste, and Sebastian Böcker. A closer look at the closest string and closest substring problem. In *2011 Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 13–24. SIAM, 2011.

7 Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

8 Jesus A. De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and Geometric Ideas in the Theory of Discrete Optimization*, volume 14 of *MOS-SIAM Series on Optimization*. SIAM, 2013.

9 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. *arXiv preprint arXiv:1802.06289*, 2018.

10 Elisabeth Finhold and Raymond Hemmecke. Lower bounds on the graver complexity of m-fold matrices. *Annals of Combinatorics*, 20(1):73–85, 2016.

11 Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical programming*, 98(1-3):23–47, 2003.

12 Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2016. URL: `http://www.gurobi.com`.

13 Stefan Heinz, Wen-Yang Ku, and Christopher J. Beck. Recent improvements using constraint integer programming for resource allocation and scheduling. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 12–27. Springer, 2013.

14 Raymond Hemmecke. Exploiting symmetries in the computation of graver bases. *arXiv preprint math/0410334*, 2004.

15 Raymond Hemmecke, Matthias Köppe, and Robert Weismantel. Graver basis and proximity techniques for block-structured separable convex integer minimization problems. *Math. Program.*, 145(1-2, Ser. A):1–18, 2014.

16 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. n-fold integer programming in cubic time. *Math. Program.*, 137(1-2, Ser. A):325–341, 2013.

17 John D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. `doi:10.1109/MCSE.2007.55`.

18 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-ip-new ptas results for scheduling with setups times. *arXiv preprint arXiv:1801.06460*, 2018.

19 Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

20 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold integer programming and applications. In *Proc. ESA 2017*, volume 87 of *Leibniz Int. Proc. Informatics*, pages 54:1–54:14, 2017.

21 Dušan Knop, Martin Koutecký, and Matthias Mnich. Voting and bribing in single-exponential time. In *Proc. STACS 2017*, volume 66 of *Leibniz Int. Proc. Informatics*, pages 46:1–46:14, 2017.

22 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, Nov 2017. `doi:10.1007/s10951-017-0550-0`.

23 Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E Bixby, Emilie Danna, Gerald Gamrath, Ambros M Gleixner, Stefan Heinz, et al. Miplib 2010. *Mathematical Programming Computation*, 3(2):103, 2011.

24 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. *arXiv preprint arXiv:1802.05859*, 2018.

25 Andrea Lodi. Mixed integer programming computation. In *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer, 2010.

26 Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010.

27 David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

28 Yves Pochet and Laurence A Wolsey. *Production planning by mixed integer programming.* Springer Science & Business Media, 2006.

29 Matthew J. Saltzman. Coin-or: an open-source library for optimization. In *Programming languages and systems in computational economics and finance*, pages 3–32. Springer, 2002.

30 Tommi Sottinen. Operations research with gnu linear programming kit. *ORMS*, 1020:200, 2009.

31 The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.6)*, 2017. `http://www.sagemath.org`.