Mathematical Programming (2020) 184:1–34
https://doi.org/10.1007/s10107-019-01402-2

FULL LENGTH PAPER

Series A

Combinatorial *n*-fold integer programming and applications

Dušan Knop[1] · Martin Koutecký[2,3] · Matthias Mnich[4,5]

Received: 27 October 2017 / Accepted: 8 May 2019 / Published online: 20 November 2019

Abstract

Many fundamental NP-hard problems can be formulated as integer linear programs (ILPs). A famous algorithm by Lenstra solves ILPs in time that is exponential only in the dimension of the program, and polynomial in the size of the ILP. That algorithm became a ubiquitous tool in the design of fixed-parameter algorithms for NP-hard problems, where one wishes to isolate the hardness of a problem by some parameter. However, in many cases using Lenstra's algorithm has two drawbacks: First, the run time of the resulting algorithms is often double-exponential in the parameter, and second, an ILP formulation in small dimension cannot easily express problems involving many different costs. Inspired by the work of Hemmecke et al. (Math Program 137(1–2, Ser. A):325–341, 2013), we develop a single-exponential algorithm for so-called *combinatorial n-fold integer programs*, which are remarkably similar to prior ILP formulations for various problems, but unlike them, also allow variable dimension. We then apply our algorithm to many relevant problems problems like Closest String, Swap Bribery, Weighted Set Multicover, and several others, and obtain exponential speedups in the dependence on the respective parameters, the input size, or both. Unlike Lenstra's algorithm, which is essentially a bounded search tree algorithm, our result uses the technique of augmenting steps. At its heart is a deep result stating that in combinatorial *n*-fold IPs, existence of an augmenting step implies existence of a "local" augmenting step, which can be found using dynamic programming. Our results provide an important insight into many problems by showing that they exhibit this phenomenon, and highlights the importance of augmentation techniques.

Keywords Integer programming · Augmentation algorithm · Closest string · Fixed-parameter algorithms

Research supported by CE-ITI Grant Project P202/12/G061 of GA ČR, GA UK Grant Project 1784214, ERC Starting Grant 306465 (BeyondWorstCase), DFG Grant MN 59/4-1, Israel Science Foundation Grant 308/18, and Charles University project UNCE/SCI/004. An extended abstract of these results appeared in the Proceedings of the 25th European Symposium on Algorithms [45].

Extended author information available on the last page of the article

Mathematics Subject Classification 90C10 · 90C27 · 90C39

1 Introduction

The Integer Linear Programming (ILP) problem is fundamental as it models many combinatorial optimization problems. Since it is NP-complete, we naturally ask about the complexity of special cases. A fundamental algorithm by Lenstra from 1983 shows that ILPs can be solved in polynomial time when their number of variables (the dimension) d is fixed [52]; that algorithm is thus a natural tool to prove that the complexity of some special cases of other NP-hard problems is also polynomial.

A systematic way to study the complexity of "special cases" of NP-hard problems was developed in the past 25 years in the field of parameterized complexity. There, the problem input is augmented by some integer parameter k, and one then measures the problem complexity in terms of both the instance size n as well as k. Of central importance are algorithms with run times of the form $f(k)n^{\mathcal{O}(1)}$ for some computable function f, which are called *fixed-parameter algorithms*. The key idea is that the degree of the polynomial does not grow with k. For background on parameterized complexity, we refer to the monograph [13].

Kannan's improvement [42] of Lenstra's algorithm runs in time $d^{\mathcal{O}(d)}\langle I\rangle^{\mathcal{O}(1)}$, where $\langle I\rangle$ is the binary encoding length of the instance, which is thus a fixed-parameter algorithm for parameter d. Gramm et al. [33] pioneered the application of Lenstra's and Kannan's algorithm in parameterized complexity: they modeled Closest String with k input strings as an ILP of dimension $k^{\mathcal{O}(k)}$, and thereby concluded with the first fixed-parameter algorithm for Closest String. This success led Niedermeier [59] to propose in his book:

> [...] It remains to investigate further examples besides Closest String where the described ILP approach turns out to be applicable. More generally, it would be interesting to discover more connections between fixed-parameter algorithms and (integer) linear programming.

Since then, many more applications of Lenstra's and Kannan's algorithm for parameterized problems have been proposed. However, essentially all of them, e.g. [8,16,23,37,49,58], share a common trait with the algorithm for Closest String: they have a double-exponential run time dependence on the parameter. This is because solving an ILP takes time exponential in the number of variables, and these algorithms amount to solving ILP formulations with exponentially many variables. Moreover, it is difficult to find ILP formulations with small dimension for problems whose input contains many objects with varying cost functions, such as in Swap Bribery [6, Challenge #2].

1.1 Our contributions

We show that a certain form of ILP, which is closely related to the previously used formulations for Closest String and Swap Bribery and other problems, can be solved in single-exponential time, even if the dimension is allowed to be variable. For example, Gramm et al.'s [33] algorithm for Closest String runs in time $2^{2^{\mathcal{O}(k \log k)}} \mathcal{O}(\log L)$ for k strings of length L and has not been improved since 2003, while our algorithm runs in time $k^{\mathcal{O}(k^2)} \mathcal{O}(\log L)$. Moreover, our algorithm has a strong combinatorial flavor and is based on different notions than are typically encountered in parameterized complexity, most importantly so-called augmenting steps (cf. Sect. 1.3). We note that all formal definitions are postponed until or repeated in Sect. 2.

As an example of our form of ILP, its motivation and connections to earlier formulations, consider the Closest String problem. We are given k strings $s_1, \ldots, s_k$ of length L that come (after some preprocessing) from the alphabet $[k]:=\{1, \ldots, k\}$, and an integer d. The goal is to find a string $y \in [k]^L$ such that, for each s_i, the Hamming distance $d_H(y, s_i)$ between y and s_i, i.e., the number of positions on which y and s_i differ, is at most d, if such y exists.

Arranging the input strings in an $k \times L$ matrix whose ith row is s_i, we have that, for $j \in [L]$, $(s_1[j], \ldots, s_k[j])$ is the jth column of this input representation. Clearly, there are at most k^k different column types in the input, and we can represent the input succinctly with multiplicities $b^{\mathbf{f}}$ of each column type $\mathbf{f} \in [k]^k$. Moreover, there are k choices for the output string y in each column. Thus, we can encode the solution by describing, for each column type $\mathbf{f} \in [k]^k$ and each output character $e \in [k]$, how many solution columns are of type $(\mathbf{f}, e)$ with a variable $x_{\mathbf{f},e}$. This is the basic idea behind the formulation of Gramm et al. [33], as depicted on the left:

$$
\begin{array}{r|rll}
\displaystyle\sum_{e\in[k]}\sum_{\mathbf{f}\in[k]^k} d_H(e, f_j) x_{\mathbf{f},e} \le d & \displaystyle\sum_{\mathbf{f}\in[k]^k}\ \sum_{(\mathbf{f}',e)\in[k]^{k+1}} d_H(e, f_j) x^{\mathbf{f}}_{\mathbf{f}',e} \le d & & \forall j \in [k] \\
\displaystyle\sum_{e\in[k]} x_{\mathbf{f},e} = b^{\mathbf{f}} & \displaystyle\sum_{(\mathbf{f}',e)\in[k]^{k+1}} x^{\mathbf{f}}_{\mathbf{f}',e} = b^{\mathbf{f}} & & \forall \mathbf{f} \in [k]^k \\
x_{\mathbf{f},e} \ge 0 & & & \forall (\mathbf{f}, e) \in [k]^{k+1} \\
 & x^{\mathbf{f}'}_{\mathbf{f},e} = 0 & & \forall \mathbf{f}' \neq \mathbf{f}, \forall e \in [k] \\
 & 0 \le x^{\mathbf{f}}_{\mathbf{f},e} \le b^{\mathbf{f}} & & \forall \mathbf{f} \in [k]^k
\end{array}
$$

The formulation on the right is obtained from the one on the left by copying each variable once for each $\mathbf{f} \in [k]^k$, and "turning off" all variables $x^{\mathbf{f}'}_{\mathbf{f},e}$ with $\mathbf{f}' \neq \mathbf{f}$. In other words, the columns of the formulation on the left are a subset of the columns on the right, and variables corresponding to columns which only exist in the formulation on the right are set to zero.

Let $(1\ \ldots\ 1) = \mathbf{1}^\intercal$ be a row vector of all ones. Alternatively, we can view the above as

$$
\begin{pmatrix}
D_1 & D_2 & \cdots & D_{k^k} \\
\mathbf{1}^\intercal & 0 & \cdots & 0 \\
0 & \mathbf{1}^\intercal & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \mathbf{1}^\intercal
\end{pmatrix}
\mathbf{x}
\begin{array}{l}
\le \mathbf{d} \\
= b^1 \\
= b^2 \\
\ \vdots \quad \vdots \\
= b^{k^k}
\end{array}
\quad\Bigg|\quad
\begin{pmatrix}
D & D & \cdots & D \\
\mathbf{1}^\intercal & 0 & \cdots & 0 \\
0 & \mathbf{1}^\intercal & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \mathbf{1}^\intercal
\end{pmatrix}
\mathbf{x}
\begin{array}{l}
\le \mathbf{d} \\
= b^1 \\
= b^2 \\
\ \vdots \quad \vdots \\
= b^{k^k},
\end{array}
$$

where $D = (D_1\ D_2\ \dots\ D_{k^k})$ and $\mathbf{d} = (d \dots d)^{\mathsf{T}} \in \mathbb{Z}^k$. Our motivation for the formulation on the right is that it has the nicely uniform structure

$$E^{(n)} = \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}. \tag{1}$$

with $D \in \mathbb{Z}^{r \times t}$, $A \in \mathbb{Z}^{s \times t}$, and containing $n = k^k$ blocks D and A. Integer programming (IP) with constraint matrix of the form (1) is known as *n-fold integer programming*. An algorithm of Hemmecke et al. [36] solves n-fold IP in time $\Delta^{(rst+t^2s)} n^3 \langle I \rangle$, where $\Delta = 1 + \left\| E^{(n)} \right\|_\infty$. However, since the ILP on the right for Closest String has $t = k^k$, applying this algorithm gives no advantage over applying Lenstra's algorithm to solve the Closest String problem.

We overcome this impediment by harnessing the special structure of the ILP for Closest String. Observe that its constraint matrix A has the form

$$A = (1\ \dots\ 1) = \mathbf{1}^{\mathsf{T}} \in \mathbb{Z}^{1 \times t}. \tag{2}$$

We call any n-fold IP with this matrix A a *combinatorial n-fold IP*. Our main result is a fast algorithm for combinatorial n-fold IPs, which is exponentially faster in t than previous works for general n-fold IPs.

Theorem 1 (simplified) *Any combinatorial n-fold IP of size L can be solved in time* $t^{\mathcal{O}(r)} (\Delta r)^{\mathcal{O}(r^2)} \mathcal{O}(n^3 \langle I \rangle) + \mathrm{poly}(n, \langle I \rangle)$, *where* $\Delta = 1 + \|D\|_\infty$.

Observe that, when applicable, our algorithm is not only asymptotically faster than Lenstra's, but provides a fixed-parameter algorithm even if the dimension is variable and not a parameter. Moreover, note that n-fold IP is allowed to have not only a linear objective function, but also a separable convex objective function. Therefore, while Theorem 1 does indeed show that certain ILPs can be solved in single-exponential time, it also shows that certain IPs with more general objective functions can also be solved efficiently.

1.2 Applications

We apply Theorem 1 to several fundamental combinatorial optimization problems, for which we obtain exponential improvements in the parameter dependence, the input length, or both. For a summary of results, see Table 1; this list is not meant to be exhaustive. In fact, we believe that for any Lenstra-based result in the literature which only achieves double-exponential run times, there is a good chance that it can be sped up using our algorithm. The only significant obstacles seem to be either large coefficients in the constraint matrix or an exponential number of rows of the matrix D.

Table 1 Run time improvements resulting from this work for a few representative problems

Problem	Previous best run time	Our result
Closest String	$2^{2^{\mathcal{O}(k \log k)}} \mathcal{O}(\log L)$ [33]	$k^{\mathcal{O}(k^2)} \mathcal{O}(\log L)$
Optimal Consensus	FPT for $k \leq 3$, open for $k \geq 4$ [2]	$k^{\mathcal{O}(k^2)} \mathcal{O}(\log L)$
Score-Swap Bribery	$2^{2^{\mathcal{O}(\lvert C\rvert \log \lvert C\rvert)}} \mathcal{O}(\log \lvert V\rvert)$ [16]	$\lvert C\rvert^{\mathcal{O}(\lvert C\rvert^2)} \mathcal{O}(T^3 \log \lvert V\rvert)$,
	$\lvert C\rvert^{\mathcal{O}(\lvert C\rvert^6)} \mathcal{O}(\lvert V\rvert^3)$ [46]	with $T \leq \lvert V\rvert$
C1-Swap Bribery	$2^{2^{\mathcal{O}(\lvert C\rvert \log \lvert C\rvert)}} \mathcal{O}(\log \lvert V\rvert)$ [16]	$\lvert C\rvert^{\mathcal{O}(\lvert C\rvert^4)} \mathcal{O}(T^3 \log \lvert V\rvert)$,
	$\lvert C\rvert^{\mathcal{O}(\lvert C\rvert^6)} \mathcal{O}(\lvert V\rvert^3)$ [46]	with $T \leq \lvert V\rvert$
Weighted Set Multicover	$2^{2^{\mathcal{O}(k \log k)}} \mathcal{O}(n)$ [8]	$k^{\mathcal{O}(k^2)} \mathcal{O}(\log n)$
Huge *n*-fold IP	FPT with $D = I$ and	FPT with parameter-sized
	A totally unimodular	domains

We introduce and discuss the parameters (e.g. C) later in Sect. 4

We discuss applications in detail in Sect. 4, which is structured as follows. We first show (Sect. 4.1) it is possible to use inequalities (and not only equations) and discuss the representation of the input (Sect. 4.2). In Sect. 4.3 we use the Weighted Set Multicover problem (which has applications for example in graph algorithms and computational social choice) as a detailed example of modeling a problem as a combinatorial *n*-fold IP. Then, in Sects. 4.4 and 4.5 we discuss many variations on the Closest String and Bribery problems, respectively, and give combinatorial *n*-fold IP formulations for them. In Sect. 4.6 we discuss the Huge *n*-fold IP problem. Finally, in Sect. 4.7, we show that many other ILP formulations in the literature have a format very close to combinatorial *n*-fold IP and can in fact be modeled as one. Together, we obtain exponential speed-ups for all of the discussed problems.

1.3 Comparison with Lenstra's algorithm

The basic idea behind Lenstra's algorithm is the following. Given a system $A\mathbf{x} \leq \mathbf{b}$ it is possible to compute the volume of the polyhedron it defines and determine that it is either too large not to contain an integer point, or too small not to be flat in some direction. In the first case we are done. In the second case we can take d slices of dimension $d-1$ and recurse into them, achieving a $d^{\mathcal{O}(d)} \langle I \rangle^{\mathcal{O}(1)}$ runtime. Note that we only decide feasibility and optimization can be then done by binary search. On the other hand, the basic idea behind our algorithm is the following. We only focus on optimizing and later show that testing feasibility reduces to it. Starting from some feasible solution, the crucial observation is that if there is a step improving the objective, there is one which does not modify many variables, and can be found quickly by dynamic programming. Moreover, if the current solution is far from the optimum, then it is possible to make a long step, and polynomially many long steps will reach the optimum.

More concretely, consider the run of these two algorithms on an instance of Closest String consisting of k strings, each of length L. Lenstra's algorithm essentially either determines that the bounds are loose enough that there must exist a solution, or (oversimplifying) determines that there is a column type $\mathbf{f} \in [k]^k$ and a character $e \in [k]$ such that there are at most k^k consecutive choices for how many times the solution contains character e at a column of type $\mathbf{f}$. Then, we recurse, obtaining a $2^{2^{\mathcal{O}(k \log k)}} \mathcal{O}(\log L)$-time algorithm. On the other hand, our algorithm views the problem as an optimization problem, so we think of starting with a string of all blanks which is trivially at distance 0 from any string, and the goal is to fill in all blanks such that the result is still at distance at most d from the input strings. An augmenting step is a set of character swaps that decreases the number of blanks. The crucial observation is that if an augmenting step exists, then there is also one only changing few characters, and it can be found in time $k^{\mathcal{O}(k^2)} \mathcal{O}(\log L)$. Thus (omitting details), we can iteratively find augmenting steps until we reach the optimum.

Related work. Our main inspiration are augmentation methods based on Graver bases, especially a fixed-parameter algorithm for n-fold IP of Hemmecke et al. [36]. Our result improves the runtime of their algorithm for a special case. More generally, our work follows in the vein of Papadimitriou [64], whose algorithm was recently improved by Eisenbrand and Weismantel [19] and Jansen and Rohwedder [41]. Subsequent to the publication of the extended abstract version of this work, the main technical result (Theorem 1) was improved by Koutecký et al. [47] and Eisenbrand et al. [17]. All the following related work is orthogonal to ours in either the achieved result, or the parameters used for it.

In fixed dimension, Lenstra's algorithm [52] was generalized for arbitrary convex sets and quasiconvex objectives by Grötschel et al. [34, Theorem 6.7.10]. The currently fastest algorithm of this kind is due to Dadush et al. [14]. The first notable fixed-parameter algorithm for a non-convex objective is due to Lokshtanov [54], who showed that optimizing a quadratic function over the integers of a polytope is fixed-parameter tractable if all coefficients are small. Ganian and Ordyniak [30] and Ganian et al. [31] studied the complexity of ILP with respect to structural parameters such as treewidth and treedepth, and introduced a new parameter called *torso-width*.

Besides fixed-parameter tractability, there is interest in the (non-)existence of polynomial kernels of ILPs, which formalize the (im)possibility of various preprocessing procedures. Jansen and Kratsch [39] showed that ILPs containing parts with simultaneously bounded treewidth and bounded domains are amenable to polynomial kernelization, unlike ILPs containing totally unimodular parts. Kratsch [48] studied the kernelizability of sparse ILPs with small coefficients.

Outline. The paper is structured as follows. In Sect. 2 we provide the definitions and notions necessary for the proof of Theorem 1. Section 3 is dedicated to the proof of Theorem 1. Finally, in Sect. 4 we turn to applications of Theorem 1.

2 Preliminaries

For positive integers m, n we set $[m : n] = \{m, \ldots, n\}$ and $[n] = [1 : n]$. For a graph G we denote by $V(G)$ its set of vertices. We write vectors in boldface (e.g., $\mathbf{x}$, $\mathbf{y}$) and their entries in normal font (e.g., the ith entry of $\mathbf{x}$ is x_i).

For a matrix $A \in \mathbb{Z}^{m \times n}$, vectors $\mathbf{b} \in \mathbb{Z}^m$, $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^n$, and a function $f\colon \mathbb{Z}^n \to \mathbb{Z}$, let $(IP)_{A,\mathbf{b},\mathbf{l},\mathbf{u},f}$ be the problem

$$\min \left\{ f(\mathbf{x}) \mid A\mathbf{x} = \mathbf{b}\,,\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\,,\ \mathbf{x} \in \mathbb{Z}^n \right\}. \tag{IP}$$

We say that a vector $\mathbf{x}$ is a *feasible solution* to $(IP)_{A,\mathbf{b},\mathbf{l},\mathbf{u},f}$ if $A\mathbf{x} = \mathbf{b}$ and $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$.

n-fold IP. Let $r, s, t, n \in \mathbb{N}$ be integers, $\mathbf{u}, \mathbf{l} \in \mathbb{Z}^{nt}$ and $\mathbf{b} \in \mathbb{Z}^{r+ns}$ be vectors, and $f\colon \mathbb{Z}^{nt} \to \mathbb{Z}$ be a separable convex function (i.e., $f(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{t} f_j^i(x_j^i)$ with every $f_j^i\colon \mathbb{Z} \to \mathbb{Z}$ univariate convex). Let $D \in \mathbb{Z}^{r \times t}$ be an $r \times t$-matrix and $A \in \mathbb{Z}^{s \times t}$ be an $s \times t$-matrix. We define a matrix $E^{(n)}$ as

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}, \tag{3}$$

and we call $E^{(n)}$ the *n-fold product of* $E = \binom{D}{A}$. The problem (IP) with the constraint matrix $E^{(n)}$ is called *n-fold integer programming* $(IP)_{E^{(n)},\mathbf{b},\mathbf{l},\mathbf{u},f}$. By $\langle I \rangle = \langle \mathbf{b}, \mathbf{l}, \mathbf{u}, f \rangle$ we denote the length of the binary encoding of the (IP) instance, which is given by vectors $\mathbf{b}, \mathbf{l}, \mathbf{u}$ and the objective function f. Here, $\langle f \rangle = \langle \max_{\mathbf{x}:\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}} |f(\mathbf{x})| \rangle$ is the encoding length of the maximum absolute value attained by f over the feasible region.

Building on a dynamic program of Hemmecke, Onn, and Romanchuk [36] and a so-called proximity technique of Hemmecke, Köppe and Weismantel [35], Knop and Koutecký [43] proved that:

Proposition 1 ([43, Thm. 5]) *There is an algorithm that solves*[1] $(IP)_{E^{(n)},\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$ *encoded with* $\langle I \rangle = \langle \boldsymbol{b}, \boldsymbol{l}, \boldsymbol{u}, f \rangle$ *bits in time* $\Delta^{O(trs+t^2 s)} \cdot n^3 \langle I \rangle$*, where* $\Delta = 1 + \max\{\|D\|_\infty, \|A\|_\infty\}$.

The structure of $E^{(n)}$ (in equation (3)) allows us to divide any nt-dimensional object, such as the vector of variables $\mathbf{x}$, the bounds $\mathbf{l}$ and $\mathbf{u}$, or the objective function f, 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., we write $\mathbf{x} = \left(\mathbf{x}^1, \ldots, \mathbf{x}^n\right)$ with $\mathbf{x}^i$ being the ith brick of $\mathbf{x}$, and x_j^i denotes the jth variable of the ith brick with $j \in [t]$ and $i \in [n]$. We refer to the constraints $(D \cdots D) \cdot \mathbf{x} \leq \mathbf{d}$ as *globally uniform constraints* and we call the other constraints *locally uniform*.

Our focus here is on the following special form of n-fold IP which we call *combinatorial n-fold IP*.

Definition 1 (*Combinatorial n-fold IP*) Let $A = (1 \ \cdots \ 1) \in \mathbb{Z}^{1 \times t}$, let $D \in \mathbb{Z}^{r \times t}$ be a matrix, and let $E = \binom{D}{A}$. Let $f\colon \mathbb{Z}^{nt} \to \mathbb{Z}$ be a separable convex function represented by an evaluation oracle. A *combinatorial n-fold IP* is

$$\min \left\{ f(\mathbf{x}) \mid E^{(n)}\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^{nt} \right\}.$$

[1] 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.

An *approximate continuous relaxation oracle* is an algorithm which, queried on an instance of (IP), returns a solution $\mathbf{x}_\varepsilon$ of the continuous relaxation of (IP)[2] such that there exists an optimum $\tilde{\mathbf{x}}$ of the relaxation satisfying $\|\mathbf{x}_\varepsilon - \tilde{\mathbf{x}}\|_\infty \leq \varepsilon$. For any separable convex function such an oracle can be implemented in polynomial time by Chubanov's algorithm [11, Corollary 14]. We denote by $\mathcal{R}$ the time needed to realize one call to this oracle when the instance is clear from the context. For more details cf. De Loera et al. [15, Problem 4.3.1].

Graver bases and augmentation. Let us now introduce Graver bases and discuss how they can be used for optimization. For background, we refer to the books of Onn [61] and De Loera et al. [15].

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 pairwise 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 $i \in [n]$, and write $\mathbf{y} \sqsubset \mathbf{x}$ if at least one of the inequalities is strict. 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 2 ([61, Lemma 3.2]) *Every integer vector $\boldsymbol{x} \neq \mathbf{0}$ with $A\boldsymbol{x} = \mathbf{0}$ is a sign-compatible sum $\boldsymbol{x} = \sum_i \boldsymbol{g}^i$ of Graver basis elements $\boldsymbol{g}^i \in \mathcal{G}(A)$, with some elements possibly appearing with repetitions.*

Let $\mathbf{x}$ be a feasible solution to $(IP)_{A,\mathbf{b},\mathbf{l},\mathbf{u},f}$. We call a vector $\mathbf{g}$ a *feasible step* if $\mathbf{x} + \mathbf{g}$ is feasible for $(IP)_{A,\mathbf{b},\mathbf{l},\mathbf{u},f}$. Further, call a feasible step $\mathbf{g}$ *augmenting* if $f(\mathbf{x} + \mathbf{g}) < f(\mathbf{x})$. An augmenting step $\mathbf{g}$ and a *step length* $\alpha \in \mathbb{Z}$ form an $\mathbf{x}$-*feasible step pair* with respect to a feasible solution $\mathbf{x}$ if $\mathbf{l} \leq \mathbf{x} + \alpha\mathbf{g} \leq \mathbf{u}$. An augmenting step $\mathbf{g}$ and a step length $\alpha \in \mathbb{Z}$ form a *Graver-best step* if $f(\mathbf{x} + \alpha\mathbf{g}) \leq f(\mathbf{x} + \alpha'\tilde{\mathbf{g}})$ for all $\mathbf{x}$-feasible step pairs $(\tilde{\mathbf{g}}, \alpha') \in \mathcal{G}(A) \times \mathbb{Z}$.

The *Graver-best augmentation procedure* for $(IP)_{A,\mathbf{b},\mathbf{l},\mathbf{u},f}$ with given feasible solution $\mathbf{x}_0$ works as follows:

1. If there is no Graver-best step for $\mathbf{x}_0$, return it as optimal.
2. If a Graver-best step $(\alpha, \mathbf{g})$ for $\mathbf{x}_0$ exists, set $\mathbf{x}_0 := \mathbf{x}_0 + \alpha\mathbf{g}$ and go to 1.

Proposition 3 ([15, implicit in Theorem 3.4.1]) *Given a feasible solution $\boldsymbol{x}_0$ for $(IP)_{A,\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$ where f is separable convex, the Graver-best augmentation procedure finds an optimum of $(IP)_{A,\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$ in at most $(2n-2)\log M$ steps, where $M = f(\boldsymbol{x}_0) - f(\boldsymbol{x}^*)$ and $\boldsymbol{x}^* \in \mathbb{Z}^n$ is an optimum of $(IP)_{A,\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$.*

Graver complexity. 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 $\mathbf{g} \in \mathcal{G}\big(E^{(n)}\big)$ is bounded by some constant $g(E)$ called the *Graver complexity of E*. A proof is given for example by Onn [61, Lemma 4.3]. To emphasize the differences with our approach, we give a rough outline of the proof, omitting several details.

[2] The continuous relaxation of (IP) is the problem $\min\{f(\mathbf{x}) \mid A\mathbf{x} = \mathbf{b},\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{R}^n\}$.

Consider any $\mathbf{g} \in \mathcal{G}\big(E^{(n)}\big)$ and take its restriction to its nonzero bricks $\bar{\mathbf{g}}$. Since each brick $\bar{\mathbf{g}}^i$ of $\bar{\mathbf{g}}$ satisfies $A\bar{\mathbf{g}}^i = \mathbf{0}$, by Proposition 2 it can be decomposed into elements from $\mathcal{G}(A)$. Let $\mathbf{h}$ be a concatenation of all members of $\mathcal{G}(A)$ from the decompositions obtained above. Since $\mathbf{g} \in \mathcal{G}\big(E^{(n)}\big)$ and $\mathbf{h}$ is obtained from decompositions of its nonzero bricks, we have that $\sum_j D\mathbf{h}^j = \mathbf{0}$. Then, consider a compact representation $\mathbf{v}$ of $\mathbf{h}$ by counting how many times each element from $\mathcal{G}(A)$ appears. Let G be a matrix with the elements of $\mathcal{G}(A)$ as columns. It is then not difficult to show that $\mathbf{v} \in \mathcal{G}(DG)$. Since $\|\mathbf{v}\|_1$ is an upper bound on the number of bricks of $\mathbf{h}$ and thus of nonzero bricks of $\mathbf{g}$ and clearly does not depend on n, $g(E) = \max_{\mathbf{v}\in\mathcal{G}(DG)} \|\mathbf{v}\|_1$ is finite.

Let us make precise two observations from this proof, which we will use later.

Lemma 1 ([36, Lemma 3.1], [61, implicit in the proof of Lemma 4.3]) *Let $D \in \mathbb{Z}^{r\times t}$, $A \in \mathbb{Z}^{s\times t}$ and $E = \binom{D}{A}$.*

Let $(\boldsymbol{g}^1, \ldots, \boldsymbol{g}^n) \in \mathcal{G}\big(E^{(n)}\big)$. Then for $i = 1, \ldots, n$ there exist vectors $\boldsymbol{h}^{i,1}, \ldots, \boldsymbol{h}^{i,m_i} \in \mathcal{G}(A)$ such that $\boldsymbol{g}^i = \sum_{k=1}^{m_i} \boldsymbol{h}^{i,k}$, and $\sum_{i=1}^{n} m_i \leq g(E)$.

Lemma 2 ([36, Lemma 6.1], [61, implicit in the proof of Lemma 4.3]) *Let $D \in \mathbb{Z}^{r\times t}$, $A \in \mathbb{Z}^{s\times t}$ and let $G \in \mathbb{Z}^{t\times p}$ be the matrix whose columns are the elements of $\mathcal{G}(A)$. Then $|\mathcal{G}(A)| \leq \|A\|_\infty^{st}$, and for $E = \binom{D}{A}$ it holds*

$$g(E) \leq \max_{\boldsymbol{v}\in\mathcal{G}(DG)} \|\boldsymbol{v}\|_1 \leq |\mathcal{G}(A)| \cdot (r\|DG\|_\infty)^r \ .$$

3 Combinatorial n-fold IPs

This section is dedicated to proving Theorem 1:

Theorem 1 (repeated) *Let $D \in \mathbb{Z}^{r\times t}$, $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{Z}^{nt}$, $\boldsymbol{b} \in \mathbb{Z}^{r+n}$, and a separable convex function $f : \mathbb{Z}^{nt} \to \mathbb{R}$ be given. There is an algorithm that solves the combinatorial n-fold IP $(IP)_{E^{(n)},\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$ of size $\langle I\rangle = \langle \boldsymbol{b}, \boldsymbol{l}, \boldsymbol{u}, f\rangle$ in time $t^{\mathcal{O}(r)}(\Delta r)^{\mathcal{O}(r^2)}\mathcal{O}(n^3\langle I\rangle) + \mathcal{R}$, where $\Delta = 1 + \|D\|_\infty$, $E = \binom{D}{\mathbf{1}^\intercal}$, and $\mathcal{R}$ is the time required for one call to an optimization oracle for the continuous relaxation of $(IP)_{E^{(n)},\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$.*

We fix an instance of combinatorial n-fold IP, that is, a tuple $(n, D, \mathbf{b}, \mathbf{l}, \mathbf{u}, f)$ that we use throughout this whole section.

3.1 Graver complexity of combinatorial n-fold IP

Recall that $g(E)$ denotes the maximum number of non-zero bricks of any augmenting step $\mathbf{g} \in \mathcal{G}\big(E^{(n)}\big)$. The bound on $g(E)$ we get from Lemma 2 is exponential in t. Our goal now is to improve the bound on $g(E)$ in terms of t, exploiting the simplicity of the matrix A in combinatorial n-fold IPs.

To see this, we will need to understand the structure of $\mathcal{G}(\mathbf{1}^\intercal)$:

Lemma 3 *For any $i \neq j \in [t]$, let $\boldsymbol{\zeta}(i, j) \in \{-1, 0, 1\}^t$ satisfy $\zeta_i = 1$, $\zeta_j = -1$, and $\zeta_\ell = 0$ for all $\ell \in [t]\setminus\{i, j\}$. It holds that*

- $\mathcal{G}(\mathbf{1}^\intercal) = \{\boldsymbol{\zeta}(i,j) \mid i,j \in [t], i \neq j\}$,
- $p = |\mathcal{G}(\mathbf{1}^\intercal)| = t(t-1)$, *and*
- $\|\mathbf{g}\|_1 = 2$ *for all* $\mathbf{g} \in \mathcal{G}(\mathbf{1}^\intercal)$.

Proof Observe that the claimed set of vectors is clearly $\sqsubseteq$-minimal in $\ker_{\mathbb{Z}}(\mathbf{1}^\intercal)$. We are left with proving there is no other non-zero $\sqsubseteq$-minimal vector in $\ker_{\mathbb{Z}}(\mathbf{1}^\intercal)$. For contradiction assume there is such a vector $\mathbf{h}$. Since it is non-zero, it must have a positive entry h_i. On the other hand, since $\mathbf{1}^\intercal\mathbf{h} = \mathbf{0}$, it must also have a negative entry h_j. But then for a vector $\mathbf{g}$ with $g_i = 1$, $g_j = -1$ and $g_k = 0$ for all $k \notin \{i,j\}$ it holds that $\mathbf{g} \sqsubset \mathbf{h}$, a contradiction. The rest follows. □

With this lemma at hand, we can prove the following.

Lemma 4 *Let* $D \in \mathbb{Z}^{r\times t}$ *and* $\Delta = 1 + \|D\|_\infty$. *Then,* $g(\binom{D}{\mathbf{1}^\intercal}) \leq t^2(2r\Delta)^r$.

Proof We simply plug the correct values into the bound of Lemma 2. By Lemma 3, $p = t(t-1) \leq t^2$. Also, $\|DG\|_\infty \leq \max_{\mathbf{g}\in\mathcal{G}(\mathbf{1}^\intercal)} \{\|D\|_\infty \cdot \|\mathbf{g}\|_1\} \leq 2\Delta$, where the last inequality follows from $\|\mathbf{g}\|_1 = 2$ for all $\mathbf{g} \in \mathcal{G}(\mathbf{1}^\intercal)$, again by Lemma 3. □

3.2 Dynamic programming

Hemmecke et al. [36] devised a clever dynamic programming algorithm to find augmenting steps for a feasible solution of an n-fold IP. Lemma 1 is crucial in their approach, as they continue by building a set $Z(E)$ of all sums of at most $g(E)$ elements of $\mathcal{G}(A)$ and then use it to construct the dynamic program. However, such a set $Z(E)$ would clearly be of size exponential in t—too large to achieve our single-exponential run times. In their dynamic program, a directed acyclic graph is constructed whose layers correspond to partial sums of elements of $\mathcal{G}(A)$.

Our insight is to build a different dynamic program. In our dynamic program, we will exploit the simplicity of $\mathcal{G}(A) = \mathcal{G}(\mathbf{1}^\intercal)$ so that the constructed directed acyclic graph will have layers which correspond to individual coordinates h^i_j of an augmenting vector $\mathbf{h}$. Additionally, we also differ in how we enforce feasibility with respect to the globally uniform constraints $(D\ D\ \cdots\ D)\mathbf{x} = \mathbf{b}^0$.

We now give the details of our approach. The *signature set of* E is $\Sigma(E) = \prod_{j=1}^r [-2\Delta \cdot g(E) : 2\Delta \cdot g(E)]$; its elements are called *signatures*. Essentially, we will use the signature set to keep track of partial sums of prefixes of the augmenting vector $\mathbf{h}$ to ensure that it satisfies $D\mathbf{h} = \mathbf{0}$. Crucially, we notice that to ensure $D\mathbf{h} = \mathbf{0}$, it suffices to remember the partial sum of the prefixes of $\mathbf{h}$ multiplied by D, thus shrinking them from dimension t to dimension r. This is another insight which allows us to avoid the exponential dependence on t. Note that $|\Sigma(E)| = (1 + \Delta 4g(E))^r$.

Definition 2 (*Augmentation graph*) Let $\mathbf{x}$ be a feasible solution for $(IP)_{E^{(n)},\mathbf{b},\mathbf{l},\mathbf{u},f}$ and let $\alpha \in \mathbb{N}$. The *augmentation graph* $DP(\mathbf{x},\alpha)$ is a vertex-weighted directed layered graph with two distinguished vertices S and T called the *source* and the *sink*, and nt layers $\mathcal{L}(1,1),\ldots,\mathcal{L}(n,t)$ structured according to the bricks, such that for all $i \in [n]$, $j \in [t]$,

$$\mathcal{L}(i,j) = (i,j) \times [-g(E) : g(E)] \times [-g(E) : g(E)] \times \Sigma(E).$$

Thus, each vertex is a tuple $\left(i, j, h^i_j, \beta^i_j, \boldsymbol{\sigma}^i_j\right)$, with the following meaning:

- $i \in [n]$ is the index of the brick,
- $j \in [t]$ is the position within the brick,
- $h^i_j \in [-g(E) : g(E)]$ is the value of the corresponding coordinate of a proposed augmenting vector $\mathbf{h}$,
- $\beta^i_j \in [-g(E) : g(E)]$ is a brick prefix sum $\sum_{\ell=1}^{j} h^i_\ell$ of the proposed augmenting vector $\mathbf{h}$, and,
- $\boldsymbol{\sigma}^i_j \in \Sigma(E)$ is the signature that represents the prefix sum $\sum_{k=1}^{i} D\mathbf{h}^k + \sum_{\ell=1}^{j} D_\ell h^i_\ell$, where D_ℓ is the ℓth column of the matrix D.

A vertex $\left(i, j, h^i_j, \beta^i_j, \boldsymbol{\sigma}^i_j\right)$ has weight $f^i_j(\alpha h^i_j + x^i_j) - f^i_j(x^i_j)$.

Let $S = \left(0, t, 0, 0, \mathbf{0}\right)$ and $T = \left(n+1, 1, 0, 0, \mathbf{0}\right)$, where the last coordinate is an r-dimensional all-zero vector.

Edges to the first layer of a brick. Every vertex $\left(i, t, h^i_t, 0, \boldsymbol{\sigma}^i_t\right)$ has edges to each vertex $\left(i+1, 1, h^{i+1}_1, h^{i+1}_1, \boldsymbol{\sigma}^{i+1}_1\right)$ for which $l^{i+1}_1 \le x^{i+1}_1 + \alpha h^{i+1}_1 \le u^{i+1}_1$ and $\boldsymbol{\sigma}^{i+1}_1 = \boldsymbol{\sigma}^i_t + D_1 h^{i+1}_1$. We emphasize that there are no other outgoing edges from layer $\mathcal{L}(i,t)$ to layer $\mathcal{L}(i+1,1)$.

Edges within a brick. Every vertex $\left(i, j, h^i_j, \beta^i_j, \boldsymbol{\sigma}^i_j\right)$ with $j < t$ has edges to each vertex $\left(i, j+1, h^i_{j+1}, \beta^i_{j+1}, \boldsymbol{\sigma}^i_{j+1}\right)$ for which

- $l^i_{j+1} \le x^i_{j+1} + \alpha h^i_{j+1} \le u^i_{j+1}$,
- $\beta^i_{j+1} = \beta^i_j + h^i_{j+1}$, with $\beta^i_{j+1} \in [-g(E) \colon g(E)]$, and,
- $\boldsymbol{\sigma}^i_{j+1} = \boldsymbol{\sigma}^i_j + D_{j+1} h^i_{j+1}$.

See Fig. 1 for a scheme of the augmentation graph.

Note that by the bounds on $g(E)$ by Lemma 4 the number of vertices in each layer (recall that there is a layer for each $i \in [n]$ and $j \in [t]$) of $DP(\mathbf{x}, \alpha)$ is bounded by

$$\begin{aligned} L_{\max} &\le g(E)^2 \cdot |\Sigma(E)| \le \left(t^2(2r\Delta)^r\right)^2 \cdot \left(1 + 4\Delta \cdot \left(t^2(2r\Delta)^r\right)\right)^r \\ &= \left(t^2(2r\Delta)^r\right)^{\mathcal{O}(r)} = t^{O(r)} \cdot (r\Delta)^{O(r^2)} \ . \end{aligned} \tag{4}$$

Let P be an S–T path in $DP(\mathbf{x}, \alpha)$. We define the *P-augmentation vector* $\mathbf{h} \in \mathbb{Z}^{nt}$ by the h^i_j-coordinates of the vertices of P.

Let $\mathbf{x}$ be a feasible solution of the combinatorial n-fold IP instance fixed at the beginning of Sect. 3. We say that $\mathbf{h}$ is a *solution of* $DP(\mathbf{x}, \alpha)$ if there exists an S–T path P such that $\mathbf{h}$ is the P-augmentation vector. The weight $w(\mathbf{h})$ is then defined as the weight of the path P (i.e., the sum of weights of vertices of the path P). Note that $w(\mathbf{h}) = f(\mathbf{x} + \alpha\mathbf{h}) - f(\mathbf{x})$.

The following lemma relates solutions of $DP(\mathbf{x}, \alpha)$ to potential feasible steps in $\mathcal{G}\left(E^{(n)}\right)$.

Lemma 5 *Let $\boldsymbol{x} \in \mathbb{Z}^{nt}$ be a feasible solution, $\alpha \in \mathbb{N}$, and let $\boldsymbol{h}$ be a solution of $DP(\boldsymbol{x}, \alpha)$. Then $\boldsymbol{l} \le \boldsymbol{x} + \alpha\boldsymbol{h} \le \boldsymbol{u}$ and $E^{(n)}\boldsymbol{h} = \mathbf{0}$.*

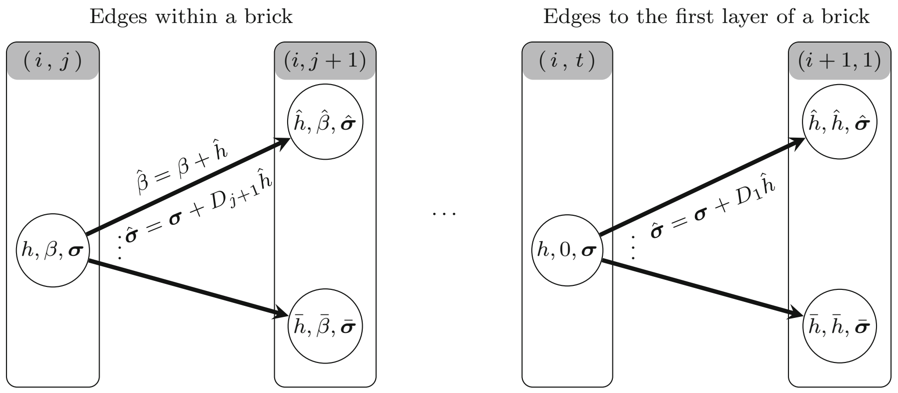
Edges within a brick
(i, j)
$(i, j+1)$
$\hat{h}, \hat{\beta}, \hat{\boldsymbol{\sigma}}$
$\hat{\beta} = \beta + \hat{h}$
$\hat{\boldsymbol{\sigma}} = \boldsymbol{\sigma} + D_{j+1}\hat{h}$
$h, \beta, \boldsymbol{\sigma}$
$\bar{h}, \bar{\beta}, \bar{\boldsymbol{\sigma}}$
Edges to the first layer of a brick
(i, t)
$(i+1, 1)$
$\hat{h}, \hat{h}, \hat{\boldsymbol{\sigma}}$
$\hat{\boldsymbol{\sigma}} = \boldsymbol{\sigma} + D_1\hat{h}$
$h, 0, \boldsymbol{\sigma}$
$\bar{h}, \bar{h}, \bar{\boldsymbol{\sigma}}$

Fig. 1 Transitions in the augmentation graph $DP(\mathbf{x}, \alpha)$

Proof To see that $\mathbf{l} \leq \mathbf{x} + \alpha\mathbf{h} \leq \mathbf{u}$, recall that there is no incoming edge to a vertex $(i, j, h^i_j, \beta^i_j, \boldsymbol{\sigma}^i_j)$ which would violate the bound $l^i_j \leq x^i_j + \alpha h^i_j \leq u^i_j$.

To see that $E^{(n)}\mathbf{h} = \mathbf{0}$, first observe that by the definition of β^i_j, and the condition that only if $\beta^i_t = 0$ for all $i = 1, \ldots, n$ there is an outgoing edge, we have that every brick $\mathbf{h}^i$ satisfies $\mathbf{1}^\intercal\mathbf{h}^i = 0$. Second, by the definition of $\boldsymbol{\sigma}^i_j$ and the edges incoming to T, we have that $D\mathbf{h} = \mathbf{0}$. Together, this implies $E^{(n)}\mathbf{h} = \mathbf{0}$. □

Lemma 6 *Let $\boldsymbol{x} \in \mathbb{Z}^{nt}$ be a feasible solution. Then every $\boldsymbol{g} \in \mathcal{G}\big(E^{(n)}\big)$ with $\boldsymbol{l} \leq \boldsymbol{x} + \alpha\boldsymbol{g} \leq \boldsymbol{u}$ is a solution of $DP(\boldsymbol{x}, \alpha)$.*

Proof Let $\mathbf{g} \in \mathcal{G}\big(E^{(n)}\big)$ satisfy $\mathbf{l} \leq \mathbf{x} + \alpha\mathbf{g} \leq \mathbf{u}$. We shall construct an S–T path P in $DP(\mathbf{x}, \alpha)$ such that $\mathbf{g}$ is the P-augmentation vector. We will describe which vertex is selected from each layer, and argue that this is well defined. Then, by the definition of $DP(\mathbf{x}, \alpha)$, it will be clear that the selected vertices are indeed connected by edges.

In layer $\mathcal{L}(1, 1)$, we select vertex $\big(1, 1, g^1_1, g^1_1, D_1 g^1_1\big)$. In layer $\mathcal{L}(i, j)$, we select vertex $\big(i, j, g^i_j, \beta^i_j, \boldsymbol{\sigma}^i_j\big)$ with $\beta^i_j = \beta^i_{j-1} + g^i_j$ and $\boldsymbol{\sigma}^i_j = \boldsymbol{\sigma}^i_{j-1} + D_j g^i_j$ if $j > 1$, and $\beta^i_j = \beta^{i-1}_t + g^i_j$ and $\boldsymbol{\sigma}^i_j = \boldsymbol{\sigma}^{i-1}_t + D_1 g^{i-1}_t$ otherwise.

We shall argue that this is well defined, i.e., that all of the specified vertices actually exist. From Lemma 1 it follows that $\mathbf{g}$ can be decomposed into $M \leq g(E)$ vectors $\tilde{\mathbf{g}}^1, \ldots \tilde{\mathbf{g}}^M \in \mathcal{G}(\mathbf{1}^\intercal)$. Moreover, since $M \leq g(E)$ and each $\tilde{g}^\ell$, $\ell \in [M]$, has one 1 and one -1 (again by Lemma 1), we have that for every $i \in [n]$ and every $j \in [t]$, $\left|\sum_{\ell=1}^{j} g^i_\ell\right| \leq g(E)$ (i.e., the brick prefix sum is also bounded by $g(E)$ in absolute value). Thus, vertices with the appropriate g^i_j- and β^i_j- coordinates exist. Regarding the $\boldsymbol{\sigma}^i_j$ coordinate, we make a similar observation: for every $i \in [n]$ and $j \in [t]$, $\left(\sum_{\hat{\ell}=1}^{i-1} D\mathbf{g}^{\hat{\ell}}\right) + \left(\sum_{\ell=1}^{j} D_\ell g^i_\ell\right) \in \Sigma(E)$.

From the definition of the edges in $DP(\mathbf{x}, \alpha)$, and the fact that $\mathbf{l} \leq \mathbf{x} + \alpha\mathbf{g} \leq \mathbf{u}$, the selected vertices create a path. □

Lemma 7 (optimality certification) *There is an algorithm that, given a feasible solution $\boldsymbol{x} \in \mathbb{Z}^{nt}$ for $(IP)_{E^{(n)},\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$ and $\alpha \in \mathbb{N}$, in time $t^{\mathcal{O}(r)}(\Delta r)^{\mathcal{O}(r^2)}n$ either finds a vector $\boldsymbol{h}$*

such that (i) $E^{(n)}\boldsymbol{h} = \mathbf{0}$*, (ii)* $\boldsymbol{l} \le \boldsymbol{x} + \alpha\boldsymbol{h} \le \boldsymbol{u}$*, and (iii)* $f(\boldsymbol{x} + \alpha\boldsymbol{h}) < f(\boldsymbol{x})$*, or decides that no such* $\boldsymbol{h}$ *exists.*

Proof It follows from Lemma 5 that all solutions of $DP(\mathbf{x})$ fulfil (i) and (ii). Observe that if we take $\mathbf{h}$ to be a solution of $DP(\mathbf{x}, \alpha)$ with minimum weight, then either $f(\mathbf{x}) = f(\mathbf{x} + \alpha\mathbf{h})$ or $f(\mathbf{x}) > f(\mathbf{x} + \alpha\mathbf{h})$. Due to Lemma 6 the set of solutions of $DP(\mathbf{x}, \alpha)$ contains all $\mathbf{h} \in \mathcal{G}\big(E^{(n)}\big)$ with $\mathbf{l} \le \mathbf{x} + \alpha\mathbf{h} \le \mathbf{u}$. Thus, by Proposition 3, if $f(\mathbf{x}) = f(\mathbf{x} + \alpha\mathbf{h})$, no $\mathbf{h}$ satisfying all three conditions (i), (ii), (iii) exists.

Our goal is then to find the lightest S–T path in the graph $DP(\mathbf{x}, \alpha)$. However, since edges of negative weight will be present, we cannot use, e.g., Dijkstra's algorithm. Still, it can be observed that $DP(\mathbf{x}, \alpha)$ is a directed acyclic graph, and moreover, finding the lightest path can be done in a layer-by-layer manner (by the standard algorithm of relaxing edges in a directed acyclic graph [12, Theorem 24.5]) in time $\mathcal{O}(|V(DP(\mathbf{x}, \alpha))| \cdot L_{\max}) = \mathcal{O}(ntL_{\max}^2)$, also cf. [36, Lemma 3.4]. The claimed run time follows from the bound on the maximum size of a layer (4). □

3.3 Step lengths

We have shown how to find a feasible step $\mathbf{h}$ for any given step length $\alpha \in \mathbb{N}$ such that $f(\mathbf{x} + \alpha\mathbf{h}) \le f(\mathbf{x} + \alpha\mathbf{g})$ for any feasible step $\mathbf{g} \in \mathcal{G}\big(E^{(n)}\big)$. Now, we will show that there are not too many step lengths that need to be considered in order to find a Graver-best step, which, by Proposition 3, leads to a good bound on the total required number of steps. Observe that since all feasible solutions are contained in a box $\mathbf{l} \le \mathbf{x} \le \mathbf{u}$, no steps of length $\alpha > \|\mathbf{u} - \mathbf{l}\|_\infty$ are feasible. Thus, if the quantity $\|\mathbf{u} - \mathbf{l}\|_\infty$ can be controlled, we may control the number of step-lengths which need to be checked.

In the following, we use the proximity technique pioneered by Hochbaum and Shantikumar [38] in the case of totally unimodular matrices and extended to the setting of Graver bases by Hemmecke et al. [35]. This technique allows to show that, provided some structure of the constraints (e.g., total unimodularity or bounded ℓ_∞-norm of its Graver elements), any continuous optimum is not too far from an integer optimum of the instance.

Proposition 4 ([35, Theorem 3.14]) *Consider a combinatorial n-fold IP* $(IP)_{E^{(n)},\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$*. Then for any optimal solution* $\hat{\boldsymbol{x}}$ *of its continuous relaxation there is an optimal solution* $\boldsymbol{x}^*$ *of* $(IP)_{E^{(n)},\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$ *with*

$$\left\|\hat{\boldsymbol{x}} - \boldsymbol{x}^*\right\|_\infty \le nt \cdot \max\left\{\|\boldsymbol{g}\|_\infty \mid \boldsymbol{g} \in \mathcal{G}\big(E^{(n)}\big)\right\} .$$

This allows us then to reduce the original instance to an "equivalent" instance contained in a small box, as hinted at above.

Lemma 8 (equivalent bounded instance) *Let* $(IP)_{E^{(n)},\boldsymbol{b},\boldsymbol{l},\boldsymbol{u},f}$ *be a combinatorial n-fold IP. With one call to an optimization oracle of its continuous relaxation, one can construct* $\hat{\boldsymbol{l}}, \hat{\boldsymbol{u}} \in \mathbb{Z}^{nt}$ *such that*

$$\min\left\{f(\boldsymbol{x}) \mid E^{(n)}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{l}\le\boldsymbol{x}\le\boldsymbol{u}\right\} = \min\left\{f(\boldsymbol{x}) \mid E^{(n)}\boldsymbol{x}=\boldsymbol{b}, \hat{\boldsymbol{l}}\le\boldsymbol{x}\le\hat{\boldsymbol{u}}\right\},$$

and $\left\|\hat{\boldsymbol{u}}-\hat{\boldsymbol{l}}\right\|_\infty \le nt\cdot g(E)$.

Proof Returning to Proposition 4, observe that the quantity $\max\left\{\|\mathbf{g}\|_\infty \mid \mathbf{g}\in\mathcal{G}\left(E^{(n)}\right)\right\}$ is bounded by $g(E)$ (Lemma 3 and Lemma 1). Hence, we can set new lower and upper bounds $\hat{\mathbf{l}}$ and $\hat{\mathbf{u}}$ defined by $\hat{l}^i_j:=\max\left\{\lfloor\hat{x}^i_j\rfloor - ntg(E), l^i_j\right\}$ and $\hat{u}^i_j:=\min\left\{\lceil\hat{x}^i_j\rceil + ntg(E), u^i_j\right\}$, and Proposition 4 assures that the integer optimum also lies within the new bounds. □

3.4 Finishing the proof

We are now going to prove Theorem 1 by combining the results we have established in the previous sections.

Proof of Theorem 1 We proceed in three steps.
Step 1: Bounding the feasible region. First, we use Lemma 8 to construct new lower and upper bounds $\hat{\mathbf{l}}$, $\hat{\mathbf{u}}$ satisfying $\|\hat{\mathbf{u}}-\hat{\mathbf{l}}\|_\infty \le nt\cdot g(E)$ and preserving the optimal value of $(IP)_{E^{(n)},\mathbf{b},\mathbf{l},\mathbf{u},f}$. Thus, we shall replace $\mathbf{l}$ and $\mathbf{u}$ by $\hat{\mathbf{l}}$ and $\hat{\mathbf{u}}$ from now on and assume that $\|\mathbf{u}-\mathbf{l}\|_\infty \le nt\cdot g(E)$.

Step 2: Optimization. Let us assume that we have an initial feasible solution $\mathbf{x}_0$ of $(IP)_{E^{(n)},\mathbf{b},\mathbf{l},\mathbf{u},f}$. Given a step length $\alpha\in\mathbb{N}$, it takes time $t^{\mathcal{O}(r)}(\Delta r)^{\mathcal{O}(r^2)}n$ by Lemma 7 to find a feasible step $\mathbf{h}$ satisfying $f(\mathbf{x}+\alpha\mathbf{h})\le f(\mathbf{x}+\alpha\mathbf{g})$ for all $\mathbf{g}\in\mathcal{G}\left(E^{(n)}\right)$. Recall that no $\mathbf{g}\in\mathcal{G}\left(E^{(n)}\right)$ can be feasible for $\alpha > ntg(E)$ by our bound on $\|\mathbf{u}-\mathbf{l}\|_\infty$. Thus, applying Lemma 7 for all $\alpha\in[nt\cdot g(E)]$ and choosing a pair $(\alpha,\mathbf{h})$ which minimizes $f(\mathbf{x}+\alpha\mathbf{h})$ surely finds a Graver-best step in time $t^{\mathcal{O}(r)}(\Delta r)^{\mathcal{O}(r^2)}n^2$ (or reports that no such $\mathbf{h}$ exists). In order to reach the optimum, by Proposition 3 we need to make at most $(2nt-2)\cdot\mathcal{O}(\langle I\rangle)$ Graver-best steps, where $\langle I\rangle = \langle\mathbf{b},\mathbf{0},\mathbf{u},f\rangle$. This follows from $\mathcal{O}(\langle I\rangle)$ being an upper bound on $f(\mathbf{x}^0)-f(\mathbf{x}^*)$ for some optimal solution $\mathbf{x}^*$ of $(IP)_{E^{(n)},\mathbf{b},\mathbf{l},\mathbf{u},f}$. In total, we need time $t^{\mathcal{O}(r)}(\Delta r)^{\mathcal{O}(r^2)}n^3\langle I\rangle$.

Step 3: Feasibility. Now we are left with the task of finding a starting feasible solution $\mathbf{x}_0$ in the case when we do not have it. We follow the lines of Hemmecke et al. [36, Lemma 3.8] and solve an auxiliary combinatorial n-fold IP given by the matrix $\bar{E}=\begin{pmatrix}\bar{D}\\ \bar{A}\end{pmatrix}$ with $\bar{D}:=(D\ I_r\ -I_r\ \mathbf{0})$ and $\bar{A}:=(A\ \mathbf{1}^\intercal_{2r+1})=\mathbf{1}^\intercal\in\mathbb{Z}^{t+2r+1}$, where I_r is the identity matrix of dimension r, $\mathbf{0}$ is the zero vector of length r and $\mathbf{1}_{2r+1}$ is the vector of all 1s of length $2r+1$. The variables $\bar{\mathbf{x}}$ of this auxiliary problem have a natural partition into nt variables $\mathbf{x}$ corresponding to the original problem fixed at the beginning of Sect. 3, and $n(2r+1)$ new auxiliary variables $\tilde{\mathbf{x}}$. Keep the original lower and upper bounds on $\mathbf{x}$ and introduce a lower bound 0 and upper bound $ntg(E)\Delta$ on each auxiliary variable. Finally, let the new objective be the linear function $f(\mathbf{x})=\bar{\mathbf{w}}^\intercal\bar{\mathbf{x}}$ which expresses the sum of the auxiliary variables, i.e., $\bar{\mathbf{w}}=\left(\bar{\mathbf{w}}^1,\ldots,\bar{\mathbf{w}}^n\right)$ and $\bar{\mathbf{w}}^i=(0,\ldots,0,1,\ldots,1)$ with t zeroes and $2r+1$ ones for all $i=1,\ldots,n$. Observe that it is easy to construct an initial feasible solution by setting $\mathbf{x} = \mathbf{l}$ and computing $\tilde{\mathbf{x}}$ accordingly: $\tilde{\mathbf{x}}$ serve the role of slack variables, and the slack in any constraint is at most $nt^2 g(E)\Delta$ by the fact that $\|\mathbf{u} - \mathbf{l}\|_\infty \leq nt \cdot g(E)$ and $\Delta = 1 + \|D\|_\infty$.

Then, applying Step 1 and Step 2 either finds a solution to the auxiliary problem with objective value 0, implying $\tilde{\mathbf{x}} = \mathbf{0}$, and thus $\mathbf{x}$ is feasible for the original problem, or no such solution exists, meaning that the original problem is infeasible. $\square$

4 Applications

Our aim in this section is to spell out the implications of Theorem 1 for several problems from the literature. In preparation for this, first we show how to deal with inequalities (Sect. 4.1), and second, we discuss some common encoding aspects (Sect. 4.2). Then, we give combinatorial n-fold IP formulations for several specific problems (Sects. 4.3–4.6), which constitute the bulk of this section. Finally, we show that many ILP formulations in the literature can be turned into a combinatorial n-fold IP formulations, implying speed-ups for the corresponding problems (Sect. 4.7).

All the formulations in this section use integer variables. We implicitly assume it throughout and specifically highlight it in the first formulation for the sake of completeness.

4.1 Inequalities in constraints

In applications, it is common for problems to exhibit a structure similar to that of combinatorial n-fold IP with the only difference being that some of the equations $E^{(n)}\mathbf{x} = \mathbf{b}$ are actually inequalities. Given an n-fold IP (in particular a combinatorial n-fold IP), we call the upper rows $(D\ D\ \cdots\ D)\mathbf{x} = \mathbf{b}^0$ *globally uniform constraints*, and the lower rows $A\mathbf{x}^i = \mathbf{b}^i$, for all $i \in [n]$, *locally uniform constraints*. So we first show that introducing inequalities into a combinatorial n-fold IP is possible. However, in the case of globally uniform constraints, we need a slightly different approach than in a standard n-fold IP to keep the rigid format of a combinatorial n-fold IP.

Lemma 9 *For an integer programming problem with equations and inequalities*

$$\min\{f(\boldsymbol{x}) \mid E^{(n)} \gtreqless \boldsymbol{b},\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u},\ \boldsymbol{x} \in \mathbb{Z}^{nt}\},$$

where $E^{(n)}$ is a combinatorial n-fold IP matrix, there exists an equivalent[3] *combinatorial n-fold IP instance $(IP)_{\bar{E}^{(n)},\boldsymbol{b},\bar{\boldsymbol{l}},\bar{\boldsymbol{u}},\bar{f}}$ of dimension $n\bar{t}$ with $\bar{t} \leq t + r + 1$ and $\langle \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \bar{f}\rangle \leq \mathcal{O}(\langle \boldsymbol{l}, \boldsymbol{u}, f\rangle)$.*

Remark For both types of constraints, a strict inequality "$<$" can be enforced by enforcing a "$\leq$" inequality and increasing the corresponding right hand side of the inequality by one, and similarly for "$>$".

[3] An instance I' is *equivalent* to I if there is an linear mapping φ from the feasible solutions of I' to the feasible solutions of I preserving objective values, such that φ is an injection. Specifically here I' uses auxiliary variables and φ is a mapping dropping these variables.

***Proof of Lemma* 9** *Inequalities in locally uniform constraints.* We add n variables x^i_{t+1} for all $i \in [n]$ and we replace D with $(D\ \mathbf{0})$. For each row i where we wish to enforce $\mathbf{1}^\intercal \mathbf{x}^i \leq b^i$, we set the upper bound on $u^i_{t+1} = \|\mathbf{u}^i - \mathbf{l}^i\|_1$ and lower bound $l^i_{t+1} = 0$. Similarly, for each row i where we wish to enforce $\mathbf{1}^\intercal \mathbf{x}^i \geq b^i$, we set a lower bound $l^i_{t+1} = -\|\mathbf{u}^i - \mathbf{l}^i\|_1$ and an upper bound $u^i_{t+1} = 0$. For all remaining rows we set $l^i_{t+1} = u^i_{t+1} = 0$, enforcing $\mathbf{1}^\intercal \mathbf{x}^i = b^i$.

Inequalities in globally uniform constraints. We replace the (already augmented) matrix D with $(D\ I_r)$, where I_r is the $r \times r$ identity matrix. Thus, we have introduced r new variables x^i_{t+j} with $i \in [n]$ and $j \in [r]$; however, we enforce them all to be 0 by setting $l^i_{t+j} = u^i_{t+j} = 0$ for all $i \in [n]$ and $j \in [r]$. Next, we introduce an $(n+1)$-st brick, set $l^{n+1}_j = u^{n+1}_j = 0$ for all $j \in [t]$ and set $b^{n+1} = \|D\|_\infty \cdot \|(b^1, \ldots, b^n)\|_1$. Then, for each row $i \in [r]$ where we wish to enforce a "$\leq$" inequality, we set $l^{n+1}_{t+i} = 0$ and $u^{n+1}_{t+i} = b^{n+1}$, and for each row $i \in [r]$ with a "$\geq$" inequality, we set $l^{n+1}_{t+i} = -b^{n+1}$ and $u^{n+1}_{t+i} = 0$. We let $l^{n+1}_{t+i} = u^{n+1}_{t+i} = 0$ for equality.

It remains to argue how for $i \in [n]$, $j \in [t]$ obtain the value of the original variable x^i_j. Note that we have only added new (slack) variables to the original IP and thus for this we only have to project out the newly added variables. An injection φ certifying the equivalence of I and I' is the mapping which projects out the new variables (x^i_{t+1} for all i, x^i_{t+1+j} for all $j \in [r]$, and x^{n+1}_j for all $j \in [t+r+1]$). $\square$

4.2 Succinctness

Before approaching specific applications we first discuss our particular choice of encoding.

A common aspect shared by all of our applications is that bounding some parameter of the instance makes it preferable to view the instance in a succinct way (following the terminology of Faliszewski et al. [21], Onn [62,63] calls these problems *huge* whereas Goemans and Rothvoß [32] call them *high multiplicity*). The *standard* way of viewing an instance is that the input is a collection of individual objects (bricks, matrix columns, voters, covering sets etc.). The *succinct* way of viewing an instance is by saying that identical objects are of the same *type*, giving a bound T on the number of distinct types, and then presenting the input as numbers $n_1, \ldots, n_T$ such that n_i is the number of objects of type i. Clearly, any standard instance can be converted to a succinct instance of roughly the same size (the number of objects is an upper bound on T), but the converse is not true as the numbers n_i might be large. Also, it is sometimes non-trivial to see (see Sect. 4.6) that the output can be represented succinctly, but we show that in all relevant cases it can.

In our applications we always state what are the types and what is the upper bound T on the number of types. We assume some arbitrary enumeration of the types. We also assume that the input is presented succinctly and thus we do not include the time needed to read and convert a standard instance into a succinct instance in the runtime of our algorithms.

4.3 Weighted set multicover

Bredereck et al. [8] defined the Weighted Set Multicover (WSM) problem, which is a significant generalization of the classical Set Cover problem:

Weighted Set Multicover

Input: A universe U of size k, a set system represented by a multiset $\mathcal{F} = \{F_1, \ldots, F_n\} \subseteq 2^U$, weights $w_1, \ldots, w_n \in \mathbb{N}$, and demands $d_1, \ldots, d_k \in \mathbb{N}$.

Find: A multisubset $\mathcal{F}' \subseteq \mathcal{F}$ minimizing $\sum_{F_i \in \mathcal{F}'} w_i$ and satisfying $\left|\{i \mid F_i \in \mathcal{F}', j \in F_i\}\right| \geq d_j$ for all $j \in [k]$.

The motivation of Bredereck et al. to study WSM was that it captures several problems from computational social choice and optimization problems on graphs implicit to previous works [24,28,49]. Bredereck et al. [8] designed an algorithm for WSM that runs in time $2^{2^{\mathcal{O}(k \log k)}} \mathcal{O}(n)$, using Lenstra's algorithm.

Our result yields an *exponential* improvement over the algorithm by Bredereck et al. [6], both in the dependence on the parameter and instance size:

Theorem 2 *There is an algorithm that solves* Weighted Set Multicover *in time* $k^{\mathcal{O}(k^2)} \mathcal{O}(\log n + \log w_{\max})$ *for succinctly represented instances of n sets over a universe of size k, where* $w_{\max}$ *is the maximum weight of any set.*

Proof Observe that there are at most 2^k different sets $F \in 2^U$; let T be the number of these sets. We classify each pair (F, w) in the input into one of $T \leq 2^k$ different types, where (F, w) and (F', w') are of the same type if $F = F'$. Let n_i, $i \in [T]$, be the number of sets of type i. Further, for any two pairs (F, w) and (F, w') with $w \leq w'$, for any solution containing (F', w') and not containing (F, w) there is another solution which is at least as good and contains (F, w). We thus order the pairs $(F, w), (F, w'), \cdots$ by non-decreasing weight, so that lighter elements are used before heavier ones in any optimal solution.

This allows us to represent the input instance in a succinct way by T functions $g^i, \ldots, g^T : [n] \to \mathbb{N}$ such that, for any $i \in [T]$, $g^i(\ell)$ is defined as the sum of the ℓ lightest elements of type i or $+\infty$ in case that there are less than ℓ elements of type i. Observe that since each g^i is a partial sum of a non-decreasing sequence of weights, it is a convex function.

We construct a combinatorial n-fold IP to solve the problem. Let $x^\tau_{\mathbf{f}}$ for each $\mathbf{f} \in 2^U$ and each $\tau \in [T]$ be a variable. Let $l^\tau_{\mathbf{f}} = u^\tau_{\mathbf{f}} = 0$ for each $\mathbf{f} \in 2^U$ such that $\mathbf{f} \neq F^\tau$, and let $l^\tau_{\mathbf{f}} = 0$ and $u^\tau_{\mathbf{f}} = n_\tau$ for $\mathbf{f} = F^\tau$. The variable $x^\tau_{\mathbf{f}}$ with $\mathbf{f} = F^\tau$ represents the number of sets of type τ in the solution. The IP formulation then reads

$$\begin{aligned} \min \sum_{\tau=1}^{T} g^\tau(x^\tau_{\mathbf{f}}) \qquad \text{s.t.} \sum_{\tau=1}^{T} \sum_{\mathbf{f} \in 2^U} f_j x^\tau_{\mathbf{f}} &\geq d_j && \text{for all } j \in [k] \\ \sum_{\mathbf{f} \in 2^U} x^\tau_{\mathbf{f}} &\leq n_\tau && \text{for all } \tau \in [T] \\ x^\tau_{\mathbf{f}} &\in \mathbb{N} && \text{for all } \tau \in [T], \mathbf{f} \in 2^U; \end{aligned}$$

note that f_i is 1 if $i \in \mathbf{f}$ and 0 otherwise.

This formulation is a combinatorial n-fold IP for the following reason. First, its objective function is separable convex. Second, its constrains are divided into two sets as follows. The first k constraints are described by a sum whose indices run from 1 to T and whose coefficients are identical for each $\tau \in [T]$ – these are the globally uniform constraints. The second T constraints are described by a sum whose indices run over $\mathbf{f} \in 2^U$ and each constraint only applies to a subset of variables determined by $\tau \in [T]$ – these are the locally uniform constraints.

Let us determine the parameters $\hat{\Delta}, \hat{r}, \hat{t}, \hat{n}$ and $\langle \hat{I} \rangle$ of this combinatorial n-fold IP instance $\hat{I}$. Clearly, the largest coefficient $||\hat{D}||_\infty$ is 1, the number of globally uniform constraints $\hat{r}$ is k, the number of variables per brick $\hat{t}$ is 2^k, the number of bricks $\hat{n}$ is T, and the length of the input $\langle \hat{I} \rangle$ is at most $\log n + \log w_{\max}$. Thus, instance $\hat{I}$ can be solved using Theorem 1 in the claimed time $k^{\mathcal{O}(k^2)} \mathcal{O}(\log n + \log w_{\max})$. □

4.4 Stringology

A typical problem from stringology is to find a string y satisfying certain distance properties with respect to k strings $s_1, \ldots, s_k$. All previous fixed-parameter algorithms for such problems we are aware of for parameter k rely on Lenstra's algorithm, or their complexity status was open (e.g., the complexity of Optimal Consensus [2] was unknown for all $k \geq 4$). Interestingly, Boucher and Wilkie [5] showed the counter-intuitive fact that Closest String is easier to solve when k is large, which makes the parameterization by k even more significant. Finding an algorithm with run time only single-exponential in k was a repeatedly posed problem, e.g. by Bulteau et al. [9, Challenge #1] and Avila et al. [3, Problem 7.1]. By applying our result, we close this gap for a wide range of problems.

In order to do so, we show a single-exponential algorithm for an artificial "meta-problem" called δ-Multi Strings which generalizes many previously studied problems in stringology.

δ-Multi Strings

Input: A set of strings $S = \{s_1, \ldots, s_k\}$, each of length L over alphabet $\Sigma \cup \{\star\}$, distance lower and upper bounds $d_1, \ldots, d_k \in \mathbb{N}$ and $D_1, \ldots, D_k \in \mathbb{N}$, distance function[4] $\delta\colon (\Sigma \cup \{\star\})^* \times \Sigma^* \to \mathbb{N}$ and a binary parameter $b \in \{0, 1\}$.

Find: An output string $y \in \Sigma^L$ with $d_i \leq \delta(s_i, y) \leq D_i$ for each $s_i \in S$, which minimizes $b \cdot \left(\sum_{i=1}^k \delta(s_i, y)\right)$.

We call a distance function $\delta\colon \Sigma^* \times \Sigma^* \to \mathbb{N}$ *character-wise wildcard-compatible* if $\delta(x, y) = \sum_{i=1}^L \delta(x[i], y[i])$ for any two strings $x, y \in \Sigma^L$, and $\delta(e, \star) = 0$ for all $e \in \Sigma$. Note that a character-wise wildcard-compatible can be described by a $|\Sigma| \times |\Sigma|$ sized table of the character distances.

Theorem 3 *There is an algorithm that solves instances of* δ-Multi Strings *in time* $K^{\mathcal{O}(k^2)} \mathcal{O}(\log L)$, *where* $K = \max\left\{|\Sigma|, k, \max_{e,f \in \Sigma} \delta(e, f)\right\}$ *and* δ *is a character-wise wildcard-compatible function.*

[4] We stress that here δ is part of the input.

Proof Let us fix an instance of δ-Multi Strings. We create an instance of combinatorial n-fold IP and show how solving it corresponds to solving the original δ-Multi Strings problem.

As is standard, we represent the input as an $L \times k$ matrix C with entries from $\Sigma \cup \{\star\}$ whose rows are the input strings $s_1, \ldots, s_k$. There are at most $T = (|\Sigma|+1)^k$ different *input column types*. Let $n_{\mathbf{e}}$ be the number of columns of type $\mathbf{e} \in (\Sigma \cup \{\star\})^k$ and denote $\mathcal{T}_c \subseteq (\Sigma \cup \{\star\})^k$ the set of input column types. A solution can be represented as an $L \times (k+1)$ matrix with entries from $\Sigma \cup \{\star\}$ whose last row does not contain any $\star$ symbol. Thus, there are at most $(|\Sigma|+1)^k \cdot |\Sigma|$ *solution column types* $\boldsymbol{\alpha} = (\mathbf{e}, f) \in \mathcal{T}_s$, where $\mathcal{T}_s = \mathcal{T}_c \times \Sigma$ is the set of all solution column types. We say that an input column type $\mathbf{e} \in \mathcal{T}_c$ is *compatible* with a solution column type $\boldsymbol{\alpha} \in \mathcal{T}_s$ if $\boldsymbol{\alpha} = (\mathbf{e}, f)$ for some $f \in \Sigma$.

Let us describe the combinatorial n-fold IP formulation. It consists of variables $x_{\boldsymbol{\alpha}}^{\mathbf{e}}$ for each $\boldsymbol{\alpha} \in \mathcal{T}_s$ and each $\mathbf{e} \in \mathcal{T}_c$. Intuitively, the variable $x_{\boldsymbol{\alpha}}^{\mathbf{e}}$ encodes the number of columns $\boldsymbol{\alpha}$ in the solution. However, to obey the format of combinatorial n-fold IP, we need a copy of this variable for each brick, hence the upper index $\mathbf{e}$. We set an upper bound $u_{\boldsymbol{\alpha}}^{\mathbf{e}} = n_{\mathbf{e}}$ for each two compatible $\mathbf{e}$ and $\boldsymbol{\alpha}$, and we set $u_{\boldsymbol{\alpha}}^{\mathbf{e}} = 0$ for each pair which is not compatible; all lower bounds are set to 0. The locally uniform constraints are simply $\sum_{\boldsymbol{\alpha} \in \mathcal{T}_s} x_{\boldsymbol{\alpha}}^{\mathbf{e}} = n_{\mathbf{e}}$ for all $\mathbf{e} \in \mathcal{T}_c$. The globally uniform constraints are

$$\sum_{\mathbf{e} \in \mathcal{T}_c} \sum_{\boldsymbol{\alpha} = (\mathbf{e}', f) \in \mathcal{T}_s} \delta(f, e_i') x_{\boldsymbol{\alpha}}^{\mathbf{e}} \geq d_i \qquad \text{for all } s_i \in S$$

$$\sum_{\mathbf{e} \in \mathcal{T}_c} \sum_{\boldsymbol{\alpha} = (\mathbf{e}', f) \in \mathcal{T}_s} \delta(f, e_i') x_{\boldsymbol{\alpha}}^{\mathbf{e}} \leq D_i \qquad \text{for all } s_i \in S$$

and the objective is

$$\min b \cdot \left(\sum_{i=1}^{k} \sum_{\mathbf{e} \in \mathcal{T}_c} \sum_{\boldsymbol{\alpha} = (\mathbf{e}', f) \in \mathcal{T}_s} \delta(f, e_i') x_{\boldsymbol{\alpha}}^{\mathbf{e}} \right) .$$

We then apply Theorem 1 with the following set of parameters:

- $\hat{\Delta}$ is one plus the largest coefficient in D, which is $1 + \max_{e, f \in \Sigma} \delta(e, f) \leq 1 + K$,
- $\hat{r}$ is the number of globally uniform constraints, which is $2k$,
- $\hat{t}$ is the number of variables per brick, which is $|\mathcal{T}_s| \leq (|\Sigma| + 1)^k |\Sigma| \leq K^{2k}$,
- $\hat{n}$ is the number of bricks, which is $|\mathcal{T}_c| \leq (|\Sigma| + 1)^k \leq (K + 1)^k$, and,
- $\langle \hat{I} \rangle$ is the size of the input $\langle \mathbf{b}, \mathbf{0}, \mathbf{u}, \mathbf{w} \rangle \leq \log L$.

Now, applying Theorem 1 yields the running time

$$\begin{aligned} \hat{t}^{\hat{r}} \cdot (\hat{\Delta}\hat{r})^{\mathcal{O}(\hat{r}^2)} \cdot (\hat{n}^3 \langle \hat{I} \rangle) + \mathcal{R} &\leq \left(K^{2k}\right)^{2k} \cdot ((1+K)2k)^{\mathcal{O}(2k)} \cdot (K+1)^k \log L + \mathcal{R} \\ &\leq K^{4k^2} \cdot K^{\mathcal{O}(k)} \cdot K^{\mathcal{O}(k)} \cdot \log L + \mathcal{R} \leq K^{\mathcal{O}(k^2)} \log L + \mathcal{R}, \end{aligned}$$

where we have used $k \leq K$ and $\mathcal{R}$ is the time required to solve the continuous relaxation of the above IP. Note that $\mathcal{R}$ is polynomial in $\hat{r}, \hat{t}, \hat{n}, \langle \hat{I} \rangle$. $\square$

When δ is the Hamming distance d_H, it is standard to first “normalize” the input to an equivalent instance over the alphabet $[k]$ [37, Lemma 1]. With a slight abuse of notation we also define the Hamming distance for the wildcard character $\star$ to be $d_H(e, \star) = 0$ for all $e \in \Sigma$. Note that the “normalization” procedure still works in that case, cf. [37, Lemma 1]. Thus, for $\delta = d_H$ we get rid of the dependence on $|\Sigma|$:

Corollary 1 *d_H-Multi Strings can be solved in time $k^{\mathcal{O}(k^2)}\mathcal{O}(\log L)$.*

That is, d_H-Multi Strings is fixed-parameter tractable for parameter k. Now we will show that many other string problems are single-exponential fixed-parameter tractable parameterized by k as well because they either can be seen as special cases of d_H-Multi Strings or (Turing) reduce to it. Foremost, this is the case for Closest String.

Closest String [53]
Input: Strings $s_1, \ldots, s_k \in \Sigma^L$, $d \in \mathbb{N}$.
Find: A string $y \in \Sigma^L$ such that $d_H(y, s_i) \leq d$ for all $s_i \in S$.

It is easy to see that Closest String is a special case of d_H- Multi Strings with $b = 0$, and $D_i = d$ and $d_i = 0$ for all $i \in [k]$.

Similar reasoning applies to many other problems. We first give their definitions and either explain how they reduce to Closest String, or in Table 2 explain how they reduce to d_H-Multi Strings.

Farthest String [51]
Input: Strings $s_1, \ldots, s_k \in \Sigma^L$, $d \in \mathbb{N}$.
Find: A string $y \in \Sigma^L$ such that $d_H(y, s_i) \geq d$ for all $s_i \in S$.

d-Mismatch [33,65]
Input: Strings $s_1, \ldots, s_k \in \Sigma^L$, $d \in \mathbb{N}$.
Find: A string $y \in \Sigma^{L'}$ with $L' \leq L$ and a position $p \in [L - L']$ such that $d_H(y, s_{i,p,L'}) \geq d$ for all $s_i \in S$, where $s_{i,p,L'}$ is the substring of s_i of length L' starting at position p.

Note. Gramm et al. [33] observed that d-Mismatch has a Turing reduction to Closest String which outputs L^2 instances.

Distinguishing String Selection (DSS) [50]
Input: Bad strings $S = \{s_1, \ldots, s_{k_1}\}$, good strings $S' = \{s'_1, \ldots, s'_{k_2}\}$, $k = k_1 + k_2$, $d_1, d_2 \in \mathbb{N}$, all strings of length L over alphabet Σ.
Find: A string $y \in \Sigma^L$ such that $d_H(y, s_i) \leq d_1$ for each bad string s_i and $d_H(y, s'_i) \geq L - d_2$ for each good string s'_i.

Neighbor String [55]
Input: Strings $s_1, \ldots, s_k \in \Sigma^L$, $D_1, \ldots, D_k \in \mathbb{N}$.
Find: A string $y \in \Sigma^L$ such that $d_H(y, s_i) \leq D_i$ for all $i \in [k]$.

Note. Neighbor String was introduced by Ma and Sun [55] and further studied by Nishimura and Simjour [60]. Neither explicitly state a fixed-parameter algorithm for parameter k.

Closest String with Wildcards [37]
Input: Strings $s_1, \dots, s_k \in (\Sigma \cup \{\star\})^L$, $d \in \mathbb{N}$.
Find: A string $y \in \Sigma^L$ such that $d_H(y, s_i) \leq d$ for all $i \in [k]$, where $d_H(e, \star) = 0$ for any $e \in \Sigma$.

Closest to Most Strings [4]
Input: Strings $s_1, \dots, s_k \in \Sigma^L$, $o \in [k]$, $d \in \mathbb{N}$.
Find: A string $y \in \Sigma^L$ and a set of outliers $O \subseteq \{s_1, \dots, s_k\}$ such that $d_H(y, s_i) \leq d$ for all $s_i \notin O$ and $|O| \leq o$.

Note. Closest to Most Strings (also known as Closest String with Outliers) has a Turing reduction (with 2^k instances) to Closest String [4].

c-Hamming Radius Clustering[5] (c-HRC) [1]
Input: Strings $s_1, \dots, s_k \in \Sigma^L$, $d \in \mathbb{N}$.
Find: A partition of $\{s_1, \dots, s_k\}$ into $S_1, \dots, S_c$ and output strings $y_1, \dots, y_c \in \Sigma^L$ such that $d(y_i, s_j) \leq d$ for all $i \in [c]$ and $s_j \in S_i$.

Note. c-HRC has a Turing reduction (with k^k instances) to Closest String [1].

Optimal Consensus [27]
Input: Strings $s_1, \dots, s_k \in \Sigma^L$, $d \in \mathbb{N}$.
Find: A string $y \in \Sigma^L$ such that $d_H(y, s_i) \leq d$ for all $i \in [k]$ and $\sum_{i \in [k]} d_H(y, s_i)$ is minimal.

See Table 2 for a summary of our improvements for the above-mentioned problems. Now, the following theorem is a simple corollary of Corollary 1 and the fact that d_H-Multi Strings generalizes all of the problems defined above.

Theorem 4 *The problems*

- Closest String, Farthest String, Distinguishing String Selection, Neighbor String, Closest String with Wildcards, Closest to Most Strings, c-HRC *and* Optimal Consensus *are solvable in time* $k^{\mathcal{O}(k^2)}\mathcal{O}(\log L)$, *and,*
- d-Mismatch *is solvable in time* $k^{\mathcal{O}(k^2)}\mathcal{O}(L^2 \log L)$,

for inputs consisting of k strings of length L succinctly encoded by multiplicities of identical columns.

4.5 Computational social choice

A typical problem in computational social choice takes as input an election consisting of a set V of voters and a set C of candidates which are ranked by the voters, and the objective is to manipulate the election in certain ways to let a desired candidate win the election under some voting rule $\mathcal{R}$. This setup leads to a class of bribery problems, a prominent example of which is $\mathcal{R}$-Swap Bribery where manipulation is by swaps

[5] We stress here c is a fixed constant.

Table 2 If the “specialization” row does not contain a value, it means its “default” value is assumed

Problem	Specialization of d_H-Multi Strings	Previous best run time
Closest String	$D_i = d$ for all $i \in [k]$	$2^{2^{\mathcal{O}(k \log k)}} \log^{\mathcal{O}(1)} n$ [33]
Farthest String	$d_i = d$ for all $i \in [k]$	$2^{2^{\mathcal{O}(k \log k)}} \log^{\mathcal{O}(1)} n$ [33, “implicit”]
d-Mismatch	Turing-reduces to Closest String [33]	$2^{2^{\mathcal{O}(k \log k)}} \log^{\mathcal{O}(1)} n$ [33]
Distinguishing String Selection	Special case of Neighbor String	$2^{2^{\mathcal{O}(k \log k)}} \log^{\mathcal{O}(1)} n$ [33, “implicit”]
Neighbor String	$D_i = D_i$ for all $i \in [k]$	$2^{2^{\mathcal{O}(k \log k)}} \log^{\mathcal{O}(1)} n$ [33, “implicit”]
Closest String with Wildcards	$D_i = d$ for all $i \in [k]$	$2^{2^{\mathcal{O}(k \log k)}} \log^{\mathcal{O}(1)} n$ [37]
Closest to Most Strings	Turing-reduces to Closest String [4]	$2^{2^{\mathcal{O}(k \log k)}} \log^{\mathcal{O}(1)} n$ [33, implicit]
c-HRC	Turing-reduces to Closest String [1]	$2^{2^{\mathcal{O}(k \log k)}} \log^{\mathcal{O}(1)} n$ [33, implicit]
Optimal Consensus	$D_i = d$ for all $i \in [k]$, $b = 1$	FPT for $k = 3$, open for $k > 3$ [2]

The default values are $b = 0$, and for all $i \in [k]$, $d_i = 0$, $D_i = L$. In each row corresponding to a problem, the last column gives the run time of the algorithm with the slowest-growing dependency on k. Most problems either reduce to Closest String and thus derive their time complexity from the result of Gramm et al. even though the original paper does not mention these problems – in that case, we write [33, implicit]. For some problems, no fixed-parameter algorithm was known before, but it is not difficult to see that the ILP formulation of Gramm et al. could be modified to model these problems as well – in that case, we write [33, “implicit”] of candidates which are consecutive in voters' preference orders. For a long time, the only known algorithms minimizing the number of swaps required run times which were double-exponential in $|C|$. Improving those run times was posed as a challenge [6, Challenge #1]. Recently, Knop et al. [46] solved the challenge using Proposition 1. However, Knop et al.'s result has a cubic dependence $\mathcal{O}(|V|^3)$ on the number of voters, and the dependence on the number of candidates is still quite large, namely $|C|^{\mathcal{O}(|C|^6)}$.

We improve their result to *logarithmic* dependence on $|V|$, and smaller dependence on $|C|$ – as we shall see. However, before we do so, we first introduce the necessary definitions and terminology.

Elections. An election (C, V) consists of a set C of candidates and a set V of voters, who indicate their preferences over the candidates in C, represented via a *preference order* $\succ_v$ for every $v \in V$ which is a total order over C. For a candidate c we denote by $\mathrm{rank}(c, v)$ their rank in $\succ_v$ (that is, $\mathrm{rank}(c, v) = \left|\left\{c' \in C \mid c \succ_v c'\right\}\right|$). Then, v's most preferred candidate has rank 1 and their least preferred candidate has rank $|C|$. For distinct candidates $c, c' \in C$, we write $c \succ_v c'$ if voter v prefers c over c'. To simplify notation, we sometimes identify the candidate set C with $\{1, \cdots, |C|\}$, in particular when expressing permutations over C. We sometimes identify a voter v with their preference order $\succ_v$, as long as no confusion arises.

Swaps. Let (C, V) be an election and let $\succ_v \in V$ be a voter. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ means to exchange the positions of c and c' in $\succ_v$. We denote the perturbed order by $\succ_v^s$. A swap $(c, c')_v$ is *admissible in* $\succ_v$ if $\mathrm{rank}(c, v) = \mathrm{rank}(c', v) - 1$. A set S of swaps is *admissible in* $\succ_v$ if they can be applied sequentially in $\succ_v$, one after the other, in some order, such that each one of them is admissible. Note that the perturbed vote, denoted by $\succ_v^S$, is independent from the order in which the swaps of S are applied (if each swap is admissible when applied). We also extend this notation for applying swaps in several votes and denote it V^S. We specify v's cost of swaps by a function $\sigma^v : C \times C \to \mathbb{Z}$, where $\sigma^v(c, c')$ is the cost of swapping c and c'.

Voting rules. A voting rule $\mathcal{R}$ is a function that maps an election (C, V) to a subset $W \subseteq C$ of *winners*. Let us define two significant classes of voting rules:

Scoring protocols. A scoring protocol is defined through a vector $\mathbf{s} = (s_1, \ldots, s_{|C|})$ of integers with $s_1 \geq \cdots \geq s_{|C|} \geq 0$. For each position $p \in \{1, \ldots, |C|\}$, value s_p specifies the number of points that each candidate c receives from each voter that ranks c as p^{th} best. Any candidate with the maximum number of points is a winner. Examples of scoring protocols include the Plurality rule with $\mathbf{s} = (1, 0, \ldots, 0)$, the d-Approval rule with $\mathbf{s} = (1, \ldots, 1, 0, \ldots, 0)$ with d ones, and the Borda rule with $\mathbf{s} = (|C| - 1, |C| - 2, \ldots, 1, 0)$. Throughout, we consider only *natural* scoring protocols for which $s_1 \leq |C|$, as is the case for the aforementioned popular rules.

C1 rules. A candidate $c \in C$ is a *Condorcet winner* if any other $c' \in C \setminus \{c\}$ satisfies $|\{\succ_v \in V \mid c \succ_v c'\}| > |\{v \in V \mid c' \succ_v c\}|$. Fishburn [26] classified voting rules as C1, C2 or C3, depending on the kind of information needed to determine the winner. For candidates $c, c' \in C$ let $v(c, c')$ be the number of voters who prefer c over c', that is, $v(c, c') = |\{\succ_v \in V \mid c \succ_v c'\}|$; we write $c <_M c'$ if c beats c' in a head-to-head contest, that is, if $v(c, c') > v(c', c)$.

A rule $\mathcal{R}$ *is C1* if knowing $<_M$ suffices to determine the winner, that is, for each pair of candidates c, c' we know whether $v(c, c') > v(c', c)$, $v(c, c') < v(c', c)$ or $v(c, c') = v(c', c)$. An example is the Copeland$^\alpha$ rule for $\alpha \in [0, 1]$, which specifies that for each head-to-head contest between two distinct candidates, if some candidate is preferred by a majority of voters then they obtain one point and the other candidate obtains zero points, and if a tie occurs then both candidates obtain α points; the candidate with the largest sum of points wins.

$\mathcal{R}$-Swap Bribery
Input: An election (C, V), a designated candidate $c^\star \in C$ and swap costs σ^v for $v \in V$.
Find: A set S of admissible swaps of minimum cost so that $c^\star$ wins the election (C, V^S) under the rule $\mathcal{R}$.

We say that two voters v, v' are of the same *type* if $\succ_v = \succ_{v'}$ and $\sigma^v = \sigma^{v'}$; clearly $T \le |V|$.

Theorem 5 $\mathcal{R}$-Swap Bribery *can be solved in time*

- $|C|^{\mathcal{O}(|C|^2)}\mathcal{O}(T^3(\log|V| + \log\sigma_{\max}))$ *for $\mathcal{R}$ any natural scoring protocol, and,*
- $|C|^{\mathcal{O}(|C|^4)}\mathcal{O}(T^3(\log|V| + \log\sigma_{\max}))$ *for $\mathcal{R}$ any C1 rule,*

where T is the number of voter types and $\sigma_{\max}$ is the maximum cost of a swap.

Remark For simplicity, we only show how Theorem 1 can be applied to speed up the $\mathcal{R}$-Swap Bribery for two representative voting rules $\mathcal{R}$.

Proof of Theorem 5 Let $n_1, \ldots, n_T$ be the numbers of voters of given types. We enumerate all total orders on C by numbers in $[|C|!]$; that is, a $j \in [|C|!]$ uniquely determines a total order on C. Let x^i_j for $j \in [|C|!]$ and $i \in [T]$ be a variable encoding the number of voters of type i that are bribed to be of order j in the solution. With slight abuse of notation, we denote by $\sigma^i(i, j)$ the cost of bribery for a voter of type i to change order to j (as by [20, Proposition 3.2] this cost is fixed). Regardless of the voting rule $\mathcal{R}$, the objective and the locally uniform constraints are identical:

$$\min \sum_{i=1}^{T} \sum_{j=1}^{|C|!} \sigma^i(i, j) x^i_j \text{ subject to } \sum_{j=1}^{|C|!} x^i_j = n_i \text{ for all } i \in [T] \ .$$

The number of variables per brick $\hat{t}$ is $|C|!$, the number of bricks $\hat{n}$ is T, and the size of the instance $\langle \hat{I} \rangle$ is $\log n + \log(|C|^2\sigma_{\max})$, because at most $|C|^2$ swaps suffice to permute any order $i \in [|C|!]$ to any other order $j \in [|C|!]$ [20, Proposition 3.2]. Let us now describe the globally uniform constraints separately for the two classes of voting rules which we study here.

Natural scoring protocol. Let $\mathbf{s} = (s_1, \ldots, s_{|C|})$ be a natural scoring protocol, i.e., $s_1 \ge \cdots \ge s_{|C|}$ and $\|\mathbf{s}\|_\infty \le |C|$. With slight abuse of notation, we denote $s_j(c)$, for

$j \in [|C|!]$ and $c \in C$, the number of points obtained by candidate c from a voter of order j. The globally uniform constraints then enforce that $c^\star$ gets at least as many points as any other candidate c:

$$\sum_{i=1}^{T}\sum_{j=1}^{|C|!} s_j(c)x_j^i \le \sum_{i=1}^{T}\sum_{j=1}^{|C|!} s_j(c^\star)x_j^i \qquad \text{for all } c \in C, c \neq c^\star \ .$$

The number $\hat{r}$ of these constraints is $|C|-1$, and the largest coefficient in them is $\|\mathbf{s}\|_\infty \le |C|$ (since $\mathbf{s}$ is a natural scoring protocol). Therefore, $\hat{\Delta} = 1 + \|\mathbf{s}\| \le 1 + |C|$.
Any C1 rule. Let $\alpha_j(c, c')$ be 1 if a voter with order $j \in [|C|!]$ prefers c to c' and 0 otherwise. Recall that a voting rule is C1 if, to determine the winner, it is sufficient to know, for each pair of candidates c, c', whether $v(c, c') > v(c', c)$, $v(c, c') < v(c', c)$ or $v(c, c') = v(c', c)$, where $v(c, c') = |\{v \mid c \succ_v c'\}|$. We call a tuple $<_M \in \{<, =, >\}^{|C|^2}$ a *scenario*. Thus, a C1 rule can be viewed as partitioning the set of all scenarios into those that select $c^\star$ as a winner and those that do not. Then, it suffices to enumerate all the at most $3^{|C|^2}$ scenarios $<_M$ where $c^\star$ wins, and for each of them to solve a combinatorial n-fold IP with the following globally uniform constraints enforcing the scenario $<_M$.

$$\sum_{i=1}^{T}\sum_{j=1}^{|C|!} \alpha_j(c, c')x_j^i > \sum_{i=1}^{T}\sum_{j=1}^{|C|!} \alpha_j(c', c)x_j^i \qquad \text{for all } c, c' \in C \text{ s.t. } c <_M c'$$

$$\sum_{i=1}^{T}\sum_{j=1}^{|C|!} \alpha_j(c, c')x_j^i = \sum_{i=1}^{T}\sum_{j=1}^{|C|!} \alpha_j(c', c)x_j^i \quad \text{for all } c, c' \in C \text{ which are incomparable.}$$

The number $\hat{r}$ of these constraints is $\binom{|C|}{2} \le |C|^2$, and the largest coefficient in them is 1, so $\hat{\Delta} = 2$. The proof is finished by plugging in the values $\hat{\Delta}, \hat{r}, \hat{t}, \hat{n}$ and $\langle \hat{I} \rangle$ into Theorem 1. □

Connections between stringology and computational social choice. Challenge #3 of Bulteau et al. [9] asks for connections between problems in stringology and computational social choice. We demonstrate that in both fields combinatorial n-fold IP is an important tool. An important feature of both Bribery-like problems and Closest String-like problems is that permuting voters or characters does not have any effect. This fits well the n-fold IP format, which does not allow any interaction between bricks. It seems that this feature is important, as when it is taken away, such as in Closest Substring, the problem becomes W[1]-hard [56], even for parameter $d + k$.

Another common feature is that both types of problems naturally admit ILP formulations for succinct variants of the problems, as mentioned above. Moreover, it was precisely this fact that made all previous algorithms double-exponential—the natural succinct formulation has exponentially many (in the parameter) variables and thus applying Lenstra's algorithm leads to a double-exponential runtime.

4.6 Huge n-fold integer programming with small domains

Onn [62] introduced a high-multiplicity version of the standard n-fold IP problem, where the number of bricks is now given in binary. It thus closely relates to the CUTTING STOCK problem, the high-multiplicity version of BIN PACKING, where the number of items of each size is given in binary. The complexity of CUTTING STOCK for constantly many item sizes was a long-standing open problem that was recently shown to be polynomial-time solvable by Goemans and Rothvoß [32] and by Jansen and Klein [40]. Previously, HUGE n-FOLD IP was shown to be fixed-parameter tractable when $D = I$ and A is totally unimodular. Using our result, we show that it is also fixed-parameter tractable when D and A are arbitrary, but the size of variable domains is bounded by a parameter.

HUGE n-FOLD IP concerns problems that can be formulated as an n-fold IP with the number of bricks n given in binary. Bricks are thus represented not explicitly, but succinctly by their multiplicity. It is at first unclear if this problem admits an optimal solution which can be encoded in polynomial space, but this is possible by a theorem of Eisenbrand and Shmonin [18, Theorem 2], as pointed out by Onn [62, Theorem 1.3 (1)].

Let $E = \binom{D}{A} \in \mathbb{Z}^{(r+s)\times t}$. Let T be a positive integer representing the *number of types of bricks*. We are given T positive integers $n_1, \ldots, n_T$ with $n = \sum_{i=1}^{T} n_i$ and vectors $\mathbf{b}^0 \in \mathbb{Z}^r$ and $\mathbf{l}^i, \mathbf{u}^i \in \mathbb{Z}^t$ and $\mathbf{b}^i \in \mathbb{Z}^s$, and for every $i \in [T]$ a separable convex function $f^i : \mathbb{Z}^t \to \mathbb{Z}$. For $i \in [T]$ and $\ell \in [n_i]$ we define the *index function* ι as $\iota(i, \ell) := \left(\sum_{j=1}^{i-1} n_j\right) + \ell$.

We call an n-fold IP instance given by the constraint matrix $E^{(n)}$ with the right hand side $\hat{\mathbf{b}}$ defined by $\hat{\mathbf{b}}^{\iota(i,\ell)} := \mathbf{b}^i$ and $\hat{\mathbf{b}}^0 := \mathbf{b}^0$, lower and upper bounds defined by $\hat{\mathbf{l}}^{\iota(i,\ell)} := \mathbf{l}^i$ and $\hat{\mathbf{u}}^{\iota(i,\ell)} := \mathbf{u}^i$ with the objective function $f(\mathbf{x}) := \sum_{i=1}^{T} \sum_{\ell=1}^{n_i} f^i\left(\mathbf{x}^{\iota(i,\ell)}\right)$ the *huge instance*.

For $i \in [T]$ and $\ell \in [n_i]$, and a feasible solution $\mathbf{x}$ of the huge instance, we say that the brick $\mathbf{x}^{\iota(i,\ell)}$ is of *type* i, and we say that $\mathbf{x}^{\iota(i,\ell)}$ has *configuration* $\mathbf{c} \in \mathbb{Z}^t$ with $\hat{\mathbf{l}}^i \le \mathbf{c} \le \hat{\mathbf{u}}^i$ if $\mathbf{x}^{\iota(i,\ell)} = \mathbf{c}$. The *succinct representation of* $\mathbf{x}$ is the set of tuples $\left\{\left(\mathbf{c}^{i,j}, m^{i,j}\right) \mid \mathbf{x} \text{ has } m^{i,j} \text{ bricks of type } i \text{ with configuration } \mathbf{c}^{i,j}\right\}$.

HUGE n-FOLD INTEGER PROGRAMMING

Input: Matrix $E = \binom{D}{A} \in \mathbb{Z}^{(r+s)\times t}$, positive integers $n_1, \ldots, n_T$, $\mathbf{b}^0 \in \mathbb{Z}^r$, for every $i \in [T]$ vectors $\mathbf{l}^i, \mathbf{u}^i \in \mathbb{Z}^t$ and $\mathbf{b}^i \in \mathbb{Z}^s$ and a separable convex function $f^i : \mathbb{Z}^t \to \mathbb{Z}$.

Find: The succinct representation of an optimal solution, if such exists.

In the special case of small domains we obtain the following:

Theorem 6 *Let $d_1, \ldots, d_t \in \mathbb{N}$ be such that $d_j = \max_{i\in[n]} u^i_j - l^i_j$, $d_{\max} = \max_{j\in[t]} d_j$ and let $\delta = \prod_{j=1}^{t} d_j$. Then the huge n-fold IP problem can be solved in time $\delta^{\mathcal{O}(r)}(t d_{\max} \|D\|_\infty r)^{\mathcal{O}(r^2)}\mathcal{O}(T^3 \log n)$.*[6]

[6] In fact, our result holds even in the case when f is an arbitrary (i.e. non-convex) function, but this does not imply any more power because of bounded domains.

This result is useful for the following reason. Knop et al. [46] obtained n-fold IP formulations with small domains for the very general $\mathcal{R}$-MULTI BRIBERY problem. The purpose of $\mathcal{R}$-MULTI BRIBERY is similar to that of the δ-MULTI STRINGS problem: it is an artificial meta-problem designed to capture many problems from the literature. In particular, it captures many additional problems not dealt with by Theorem 5. Together with Theorem 6, this immediatelly implies an exponential speedup in the number of bricks, without having to reformulate these problems as combinatorial n-fold IPs. However, there are still benefits in using Theorem 1 directly (as shown in previous sections) as it leads to better dependence on the respective parameters.

Proof of Theorem 6 Let $E, n_1, \ldots, n_T, \mathbf{b}, \mathbf{l}, \mathbf{u}, f$ be an instance I of HUGE n- FOLD IP. First, we shall prove that we can restrict our attention to the case where $\mathbf{l} = \mathbf{0}$ and $\mathbf{u}^i \leq (d_1, \ldots, d_t)^\intercal = \mathbf{d}$ for all $i \in [T]$. Consider a variable x^i_j with $l^i_j \neq 0$ and any row $\mathbf{ex} = b$ of the system $E^{(n)}\mathbf{x} = \mathbf{b}$. Because the contribution of x^i_j to the right hand side is $e^i_j x^i_j$, we have that

$$\mathbf{ex} = b \Leftrightarrow \mathbf{ex} - e^i_j l^i_j = b - e^i_j l^i_j \ .$$

Let I' be an instance of HUGE n- FOLD IP obtained from I by, for every row $\mathbf{ex}^i = b$, changing the right hand side from b to $b - \sum_{j=1}^t e^i_j l^i_j$, and setting $u^i_j := u^i_j - l^i_j$ and $l^i_j := 0$. Clearly there is a bijection between the feasible solutions of I and I' such that if $\mathbf{x} - \mathbf{l}$ is a feasible solution of I', $\mathbf{x}$ is a feasible solution of I, and thus minimizing $f(\mathbf{x} + \mathbf{l})$ over I' is equivalent to minimizing $f(\mathbf{x})$ over I. Thus, from now on assume that I satisfies $\mathbf{l} = \mathbf{0}$ and $\mathbf{u}^i \leq \mathbf{d}$ for all $i \in [T]$.

Let $\mathcal{C}^i$ for $i \in [T]$ be the set of all possible configurations of a brick of type i, defined as $\mathcal{C}^i = \left\{\mathbf{c} \in \mathbb{Z}^t \mid A\mathbf{c} = \mathbf{b}^i, \mathbf{0} \leq \mathbf{c} \leq \mathbf{u}^i\right\}$ and let $\mathcal{C} = \prod_{j=1}^t [0 : d_j]$ be the set of all configurations. Clearly, $\mathcal{C}^i \subseteq \mathcal{C}$ for all $i \in [T]$, and $|\mathcal{C}| = \delta$. Let $C \in \mathbb{Z}^{t \times \delta}$ be a matrix whose columns are all configurations from $\mathcal{C}$.

We shall give a combinatorial n-fold IP formulation solving the huge n-fold IP instance I. The formulation contains variables $y^i_{\mathbf{c}}$ for each $\mathbf{c} \in \mathcal{C}$ and each $i \in [T]$ encoding how many bricks of type i have configuration $\mathbf{c}$ in the solution of I. The formulation then is

$$\begin{aligned}
\min \quad & \hat{f}(\mathbf{y}) = \sum_{i=1}^T \sum_{\mathbf{c} \in \mathcal{C}} f^i(\mathbf{c}) y^i_{\mathbf{c}} & & (5)\\
\text{s.t.} \quad & DC\mathbf{y} = \mathbf{b}^0 & & (6)\\
& \mathbf{1}^\intercal \mathbf{y}^i = n_i & \text{for all } i \in [T] \quad & (7)\\
& y^i_{\mathbf{c}} = 0 & \text{for all } \mathbf{c} \notin \mathcal{C}^i, i \in [T] \quad & (8)\\
& 0 \leq y^i_{\mathbf{c}} \leq n_i & \text{for all } \mathbf{c} \in \mathcal{C}^i, i \in [T]\ . \quad & (9)
\end{aligned}$$

It remains to verify that the formulation above corresponds to the huge n-fold IP instance I. The objective (5) clearly has the same value. Consider the globally uniform constraints (6). In the huge n-fold IP instance, a configuration $\mathbf{c} \in \mathcal{C}^i$ of a brick of type

$i \in [T]$ contributes $D\mathbf{c}$ to the right hand side in the first r rows. This corresponds in our program to the column $D\mathbf{c}$ of the matrix DC. The locally uniform constraints (7) simply state that the solution needs to contain exactly n_i bricks of type i. Finally, since a brick of type i can never have a configuration $\mathbf{c} \notin \mathcal{C}^i$ we set all variables $y^i_{\mathbf{c}}$ with $\mathbf{c} \notin \mathcal{C}^i$ to zero with the upper bound (8), and place no restrictions on $y^i_{\mathbf{c}}$ with $\mathbf{c} \in \mathcal{C}^i$ (9).

The parameters $\hat{\Delta}, \hat{r}, \hat{t}, \hat{n}, \langle \hat{I} \rangle$ of the resulting combinatorial n-fold IP are:

- $\hat{\Delta}$ is one plus the largest coefficient in the upper matrix $\hat{D} = DC$, which is $\|DC\|_\infty \le t d_{\max} \|D\|_\infty$,
- the number of globally uniform constraints $\hat{r} = r$,
- the number of variables in a brick $\hat{t} = \delta$,
- the number of bricks $\hat{n} = T$, and,
- the input length $\langle \hat{I} \rangle = \langle \hat{\mathbf{b}}, \mathbf{0}, \hat{\mathbf{u}}, \hat{f} \rangle \le \log n \cdot \left(\max_{i \in [T]} \max_{\mathbf{c} \in \mathcal{C}^i} f^i(\mathbf{c})\right)$.

Thus, the proof is completed by applying Theorem 1. □

4.7 Combinatorial pre-*n*-fold IPs

Recall from Sect. 1 our comparison of the Gramm et al. [33] ILP for Closest String to a similar combinatorial n-fold IP:

$$\begin{pmatrix} D_1 & D_2 & \cdots & D_{k^k} \\ \mathbf{1}^\intercal & 0 & \cdots & 0 \\ 0 & \mathbf{1}^\intercal & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{1}^\intercal \end{pmatrix} \mathbf{x} \begin{array}{c} \le \mathbf{d} \\ = b^1 \\ = b^2 \\ \vdots \quad \vdots \\ = b^{k^k} \end{array} \quad \Bigg| \quad \begin{pmatrix} D & D & \cdots & D \\ \mathbf{1}^\intercal & 0 & \cdots & 0 \\ 0 & \mathbf{1}^\intercal & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{1}^\intercal \end{pmatrix} \mathbf{x} \begin{array}{c} \le \mathbf{d} \\ = b^1 \\ = b^2 \\ \vdots \quad \vdots \\ = b^{k^k}, \end{array}$$

where $D = (D_1 \ D_2 \ \ldots \ D_{k^k})$.

This similarity strongly suggests a general way how to construct the formulation on the right given the formulation on the left. Since formulations like the one on the left are ubiquitous in the literature, this would immediately imply exponential speed-ups for all such problems.

Definition 3 (*Combinatorial pre-n-fold IP*) Let $T, r, t_1, \ldots, t_T \in \mathbb{N}$ and $D_i \in \mathbb{Z}^{r \times t_\tau}$ for each $\tau \in [T]$. Let $t = t_1 + \cdots + t_T$ and $D = (D_1 \ldots D_T)$. Let

$$F = \begin{bmatrix} D_1 & D_2 & \cdots & D_T \\ \mathbf{1}^\intercal & 0 & \cdots & 0 \\ 0 & \mathbf{1}^\intercal & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{1}^\intercal \end{bmatrix}.$$

Moreover, let $\mathbf{b} = (\mathbf{b}^0, b^1, \ldots, b^T)$ with $\mathbf{b}^0 \in \mathbb{Z}^r$ and $b^\tau \in \mathbb{Z}$ for each $\tau \in [T]$, $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^t$, and let $f : \mathbb{Z}^t \to \mathbb{Z}$ be a separable convex function. Then for $\diamondsuit \in \{<, \le, =, \ge, >\}^{r+T}$, a *combinatorial pre-n-fold IP* is the problem

$$\min \left\{ f(\mathbf{x}) \mid F\mathbf{x} \diamondsuit \mathbf{b}, \ \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \ \mathbf{x} \in \mathbb{Z}^t \right\}.$$

Corollary 2 *Any combinatorial pre-n-fold IP with $\langle I\rangle = \langle \boldsymbol{b}, \boldsymbol{l}, \boldsymbol{u}, f\rangle$ and $\Delta = 1 + \|D\|_\infty$ can be solved in time $t^{\mathcal{O}(r)}(\Delta r)^{\mathcal{O}(r^2)}\mathcal{O}(n^3\langle I\rangle) + \mathcal{R}$, where $\mathcal{R}$ is the time required by one call to an optimization oracle of the continuous relaxation of the given IP.*

Proof We shall create a combinatorial n-fold IP instance based on the input combinatorial pre-n-fold IP. Let

$$E^{(n)} = \begin{bmatrix} D & D & \cdots & D \\ \mathbf{1}^\intercal & 0 & \cdots & 0 \\ 0 & \mathbf{1}^\intercal & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{1}^\intercal \end{bmatrix}.$$

Recall $D = (D_1 \dots D_T)$ in an $r \times t$ matrix. Let $\bar{T}_\tau = \sum_{j=1}^{\tau} t_j$ for each $\tau \in [T]$. Let $\varphi\colon \bigcup_{\tau=1}^{T} \big(\{\tau\} \times [t_\tau]\big) \to [T] \times [t]$ be an injective mapping from the original variables to the new ones, defined as follows: for each $\tau \in [T]$ and $j \in [t_\tau]$, $\varphi(\tau, j) = \big(\tau, \bar{T}_{\tau-1} + j\big)$. Note that the first argument is fixed by φ, that is $\varphi(\tau, \cdot) = (\tau, \cdot)$. We call any pair $(\tau, j) \in [T] \times [t]$ without a preimage in φ a *dummy pair*.

Then, for any $\tau \in [T]$ and $j \in [t_\tau]$, let $(\tau, j') = \varphi(\tau, j)$, and set $\hat{l}^\tau_{j'} = l^\tau_j$, $\hat{u}^\tau_{j'} = u^\tau_j$, and $\hat{f}^\tau_{j'} = f^\tau_j$. For any $\tau \in [T]$ and $j \in [t]$ which form a dummy pair, set $\hat{u}^\tau_j = \hat{l}^\tau_j = 0$ and let $\hat{f}^\tau_j$ be the zero function.

Now we see that

$$\min\left\{\hat{f}(\hat{\mathbf{x}}) \mid E^{(n)}\hat{\mathbf{x}} \diamond \mathbf{b},\ \hat{\mathbf{l}} \le \hat{\mathbf{x}} \le \hat{\mathbf{u}},\ \mathbf{x} \in \mathbb{Z}^{T\cdot t}\right\}$$

is a combinatorial n-fold IP (with inequalities) and thus can be solved in the claimed time by Theorem 1. □

Remark We remark that the $\mathcal{O}(\cdot)$ constants of the construction above are not optimal, but do not harm the asymptotic complexity of the algorithm. Moreover, embedding a combinatorial pre-n-fold in the strict format of a combinatorial n-fold IP is essentially unnecessary since it is already in the much more permissive *generalized n-fold IP* format, and has bounded *dual treedepth*, and thus is solvable by subsequent faster algorithms [17,47].

Parameterizing by the number of numbers. Fellows et al. [22] argued that many problems' NP-hardness construction is based on having many distinct objects, when it would be quite natural that the number of distinct objects, and thus the numbers representing them, is bounded by a parameter. For example, Chrobak et al. [10] consider the Heat-sensitive Scheduling problem and prove its NP-hardness; however, the construction uses many jobs with distinct but increasingly close "heat levels". Fellows et al. [22, Theorem 5] gave a fixed-parameter algorithm for a certain Mealy automaton problem which models, for example, the Heat- sensitive Scheduling problem parameterized by the number of distinct heat levels. Their algorithm relies on Lenstra's algorithm and has a double-exponential dependence on the parameter. In the conclusions of their paper, the authors state that "[o]ur main FPT result, Theorem 5, has a poor worst-case running-time guarantee. Can this be improved–at least in important special cases?"

It is easy to observe that their ILP only has σ constraints, and that the coefficients are bounded by $|S|^{|S|^2}$. Thus, applying our algorithm with $t = \Delta = \mathcal{O}(|S|^{|S|^2})$ and $r = \sigma$ yields an algorithm with runtime $|S|^{\mathcal{O}(\sigma|S|^2)}$, which is single-exponential in their (combined) parameter.

Scheduling meets fixed-parameter tractability. Parameterized complexity of scheduling problems is topic of increasing interest; see the survey by Mnich and van Bevern [57]. Mnich and Wiese [58] studied the parameterized complexity of fundamental scheduling problems, starting with Makespan Minimization on identical machines. They gave an algorithm with double-exponential dependence for parameter maximum job length $p_{\max}$, and polylogarithmic dependence on the number m of machines. Knop and Koutecký [43] used standard n-fold IP to reduce the dependence on $p_{\max}$ to single-exponential, however their algorithm depends polynomially on m.

We observe that in Mnich and Wiese's proof [58, Theorem 2] there is an ILP related to a set of configurations which is of size $p_{\max}^{p_{\max}}$. The coefficients of this ILP are unbounded, but they give a reduction [58, Lemma 1] which lets us assume that all coefficients differ by at most $p_{\max}^{p_{\max}}$. Moreover, because the number of machines m is fixed before the construction of the ILP, we can appropriately subtract from the right-hand sides and decrease all coefficients such that $\Delta \leq p_{\max}^{p_{\max}}$. Then, take the constraints (2) as globally uniform and notice that there are $r = p_{\max}$ of them. In conclusion, we have $t = \Delta \leq p_{\max}^{p_{\max}}$ and $r = p_{\max}$, which yields an algorithm with run time $p_{\max}^{p_{\max}^2} \cdot (\log n + \log m)$. This improves both the algorithm by Mnich and Wiese [58] as well as the algorithm by Knop and Koutecký [43].

Lobbying in multiple referenda. Bredereck et al. [7] studied the computational social choice problem of lobbying in multiple referenda, and showed a Lenstra-based fixed-parameter algorithm for the parameter m = "number of choices". The number of choices induces the parameter ℓ = "number of ballots", which is clearly bounded by $\ell \leq 2^m$. This leads to an ILP formulation with $2^{\mathcal{O}(m)}$ variables, and thus to a run time double-exponential in m.

We point out that Bredereck et al.'s proof [7, Theorem 9] contains a combinatorial pre-n-fold IP, with one set of constraints indexed over i, and another set of constraints indexed by j, which are simply sums of variables over all i. Since $i \in [m]$ and $j \in [\ell]$, we have the parameters $\Delta = 1, t = 2^m, r = m$. Therefore, we improve their algorithm to only single-exponential in m, namely $m^{\mathcal{O}(m^2)} \log n$.

Weighted set multicover (WSM) in graph algorithms. In Sect. 4.3, we gave a combinatorial n-fold IP formulation for WSM. While WSM was studied in the context of computational social choice by Bredereck et al. [8], it appeared implicitly several times in algorithms for restricted classes of graphs, namely graphs of bounded vertex cover number and neighborhood diversity. We briefly mention some of these results which we improve here.

Fiala et al. [25] showed that Equitable Coloring and $L(p, 1)$-Coloring parameterized by vertex cover are fixed-parameter tractable. Fiala et al. give two proofs [25, Theorem 4, Theorem 9] where they construct a combinatorial pre-n-fold IP. Lampis [49] introduced the neighborhood diversity parameter and used combinatorial pre-n-folds to show that GRAPH COLORING and HAMILTONIAN CYCLE are fixed-parameter tractable with this parameterization. Ganian [29] later showed in a similar fashion that VERTEX- DISJOINT PATHS and PRECOLORING EXTENSION are also fixed-parameter tractable parameterized by neighborhood diversity. Similarly, Fiala et al. [24] showed that the UNIFORM CHANNEL ASSIGNMENT problem is (triple-exponential) fixed-parameter tractable parameterized by neighborhood diversity and the largest edge weight. Ganian and Obdržálek [30] and later Knop et al. [44] studied extensions of the MSO logic, and provided fixed-parameter algorithms for their model checking on graphs of bounded neighborhood diversity. These algorithms again use combinatorial pre-n-folds under the hood.

We can speed up all aforementioned algorithms by applying Corollary 2.

5 Discussion

We established new and fast fixed-parameter algorithms for a class of n-fold IPs, which led to the first single-exponential time algorithms for several well-studied problems. Many intriguing questions arise, for example, is HUGE n-FOLD IP fixed-parameter tractable for parameter $(r, s, t, \varDelta)$? One sees that optimality certification is fixed-parameter tractable using ideas similar to those of Onn's [62]; yet, one possibly needs exponentially (in the input size) many augmenting steps.

For most of our applications, complexity lower bounds are not known to us. Our algorithms yield complexity upper bounds of $k^{\mathcal{O}(k^2)}$ on the dependence on parameter k for various problems, such as CLOSEST STRING, WEIGHTED SET MULTICOVER, Score-SWAP BRIBERY or even MAKESPAN MINIMIZATION [43]. Is this just a common feature of our algorithm, or are there hidden connections between some of these problems? And what are their actual complexities? All we know so far is a trivial ETH-based $2^{o(k)}$ lower bound for CLOSEST STRING based on its reduction from SATISFIABILITY [27].

References

1. Amir, A., Ficler, J., Roditty, L., Shalom, O.S.: On the efficiency of the Hamming C-centerstring problems. In: Proceedings of CPM 2014, Lecture Notes in Computer Science, vol. 8486, pp. 1–10 (2014)
2. Amir, A., Landau, G.M., Na, J.C., Park, H., Park, K., Sim, J.S.: Efficient algorithms for consensus string problems minimizing both distance sum and radius. Theor. Comput. Sci. **412**(39), 5239–5246 (2011)
3. Avila, L.F., Garcıa, A., Serna, M.J., Thilikos, D.M.: A list of parameterized problems in bioinformatics. Tech. Rep. LSI-06-24-R, Technical University of Catalonia (2006)
4. Boucher, C., Ma, B.: Closest string with outliers. BMC Bioinform. **12**((S–1)), S55 (2011)
5. Boucher, C., Wilkie, K.: Why large closest string instances are easy to solve in practice. In: Proceedings of SPIRE 2010, Lecture Notes in Computer Science, vol. 6393, pp. 106–117 (2010)

6. Bredereck, R., Chen, J., Faliszewski, P., Guo, J., Niedermeier, R., Woeginger, G.J.: Parameterized algorithmics for computational social choice: nine research challenges. Tsinghua Sci. Tech. **19**(4), 358–373 (2014)
7. Bredereck, R., Chen, J., Hartung, S., Kratsch, S., Niedermeier, R., Suchý, O., Woeginger, G.J.: A multivariate complexity analysis of lobbying in multiple referenda. J. Artif. Intell. Res. **50**, 409–446 (2014)
8. Bredereck, R., Faliszewski, P., Niedermeier, R., Skowron, P., Talmon, N.: Elections with few candidates: Prices, weights, and covering problems. In: Proceedings of ADT 2015, Lecture Notes in Computer Science, vol. 9346, pp. 414–431 (2015)
9. Bulteau, L., Hüffner, F., Komusiewicz, C., Niedermeier, R.: Multivariate algorithmics for NP-hard string problems. Bull. EATCS **114**, (2014)
10. Chrobak, M., Dürr, C., Hurand, M., Robert, J.: Algorithms for temperature-aware task scheduling in microprocessor systems. In: Proceedings of AAIM 2008, Lecture Notes in Computer Science, vol. 5034, pp. 120–130 (2008)
11. Chubanov, S.: A polynomial-time descent method for separable convex optimization problems with linear constraints. SIAM J. Opt. **26**(1), 856–889 (2016)
12. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1990)
13. Cygan, M., Fomin, F.V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, New York (2015)
14. Dadush, D., Peikert, C., Vempala, S.: Enumerative lattice algorithms in any norm via M-ellipsoid coverings. Proc. FOCS **2011**, 580–589 (2011)
15. De Loera, J.A., Hemmecke, R., Köppe, M.: Algebraic and geometric ideas in the theory of discrete optimization. MOS-SIAM series on optimization. SIAM, **14** (2013)
16. Dorn, B., Schlotter, I.: Multivariate complexity analysis of swap bribery. Algorithmica **64**(1), 126–151 (2012)
17. Eisenbrand, F., Hunkenschröder, C., Klein, K.M.: Faster algorithms for integer programs with block structure. In: Proceedings of ICALP 2018, Leibniz International Proceedings in Informatics, vol. 107, pp. 49:1–49:13 (2018)
18. Eisenbrand, F., Shmonin, G.: Carathéodory bounds for integer cones. Oper. Res. Lett. **34**(5), 564–568 (2006)
19. Eisenbrand, F., Weismantel, R.: Proximity results and faster algorithms for integer programming using the Steinitz lemma. In: Proceedings of SODA 2018, pp. 808–816 (2018)
20. Elkind, E., Faliszewski, P., Slinko, A.: Swap bribery. In: Proceedings of SAGT 2009, Lecture Notes in Computer Science, vol. 5814, pp. 299–310 (2009)
21. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L.A.: The complexity of bribery in elections. In: Proceedings of AAAI 2006, pp. 641–646 (2006)
22. Fellows, M.R., Gaspers, S., Rosamond, F.A.: Parameterizing by the number of numbers. Theory Comput. Syst. **50**(4), 675–693 (2012)
23. Fellows, M.R., Lokshtanov, D., Misra, N., Rosamond, F.A., Saurabh, S.: Graph layout problems parameterized by vertex cover. In: Proceedings of ISAAC 2008, Lecture Notes in Computer Science, vol. 5369, pp. 294–305. Springer, Berlin (2008)
24. Fiala, J., Gavenčiak, T., Knop, D., Koutecký, M., Kratochvíl, J.: Parameterized complexity of distance labeling and uniform channel assignment problems. Discrete Appl. Math. **248**, 46–55 (2018)
25. Fiala, J., Golovach, P.A., Kratochvíl, J.: Parameterized complexity of coloring problems: treewidth versus vertex cover. Theor. Comput. Sci. **412**(23), 2513–2523 (2011)
26. Fishburn, P.C.: Condorcet social choice functions. SIAM J. Appl. Math. **33**(3), 469–489 (1977)
27. Frances, M., Litman, A.: On covering problems of codes. Theory Comput. Syst. **30**(2), 113–119 (1997)
28. Gajarský, J., Lampis, M., Ordyniak, S.: Parameterized algorithms for modular-width. In: Proceedings of IPEC 2013, Lecture Notes in Computer Science, vol. 8246, pp. 163–176 (2013)
29. Ganian, R.: Using neighborhood diversity to solve hard problems. Tech. rep. (2012). https://arxiv.org/abs/1201.3091
30. Ganian, R., Ordyniak, S.: The complexity landscape of decompositional parameters for ILP. Artif. Intell. **257**, 61–71 (2018)
31. Ganian, R., Ordyniak, S., Ramanujan, M.S.: Going beyond primal treewidth for (M)ILP. Proc. AAAI **2017**, 815–821 (2017)
32. Goemans, M.X., Rothvoß, T.: Polynomiality for bin packing with a constant number of item types. Proc. SODA **2014**, 830–839 (2014)

33. Gramm, J., Niedermeier, R., Rossmanith, P.: Fixed-parameter algorithms for closest string and related problems. Algorithmica **37**(1), 25–42 (2003)
34. Grötschel, M., Lovász, L., Schrijver, A.: Geometric algorithms and combinatorial optimization, Algorithms and Combinatorics, vol. 2, 2nd edn. Springer-Verlag, Berlin (1993)
35. Hemmecke, R., Köppe, M., Weismantel, R.: Graver basis and proximity techniques for block-structured separable convex integer minimization problems. Math. Program. **145**(1–2, Ser. A), 1–18 (2014)
36. Hemmecke, R., Onn, S., Romanchuk, L.: n-fold integer programming in cubic time. Math. Program. **137**(1–2, Ser. A), 325–341 (2013)
37. Hermelin, D., Rozenberg, L.: Parameterized complexity analysis for the closest string with wildcards problem. Theor. Comput. Sci. **600**, 11–18 (2015)
38. Hochbaum, D.S., Shanthikumar, J.G.: Convex separable optimization is not much harder than linear optimization. J. ACM **37**(4), 843–862 (1990)
39. Jansen, B.M.P., Kratsch, S.: A structural approach to kernels for ILPs: treewidth and total unimodularity. In: Proceedings of ESA 2015, Lecture Notes in Computer Science, vol. 9294, pp. 779–791 (2015)
40. Jansen, K., Klein, K.: About the structure of the integer cone and its application to bin packing. In: Proceedings of SODA 2017, pp. 1571–1581 (2017)
41. Jansen, K., Rohwedder, L.: On integer programming and convolution. In: Proceedings of ITCS 2019, The Leibniz International Proceedings in Informatics, vol. 124, pp. 43:1–43:17 (2018)
42. Kannan, R.: Minkowski's convex body theorem and integer programming. Math. Oper. Res. **12**(3), 415–440 (1987)
43. Knop, D., Koutecký, M.: Scheduling meets n-fold integer programming. J. Sched. **21**(5), 493–503 (2018)
44. Knop, D., Koutecký, M., Masařík, T., Toufar, T.: Simplified algorithmic metatheorems beyond MSO: Treewidth and neighborhood diversity. In: Proceedings of WG 2017, Lecture Notes in Computer Sciencce, vol. 10520, pp. 344–357 (2017)
45. Knop, D., Koutecký, M., Mnich, M.: Combinatorial n-fold integer programming and applications. In: Proceedings of ESA 2017, The Leibniz International Proceedings in Informatics, vol. 87, pp. 54:1–54:14 (2017)
46. Knop, D., Koutecký, M., Mnich, M.: Voting and bribing in single-exponential time. In: Proceedings of STACS 2017, The Leibniz International Proceedings in Informatics, vol. 66, pp. 46:1–46:14 (2017)
47. Koutecký, M., Levin, A., Onn, S.: A parameterized strongly polynomial algorithm for block structured integer programs. In: Proceedings of ICALP 2018, Leibniz International Proceedings in Informatics, vol. 107, pp. 85:1–85:14 (2018)
48. Kratsch, S.: On polynomial kernels for sparse integer linear programs. J. Comput. Syst. Sci. **82**(5), 758–766 (2016)
49. Lampis, M.: Algorithmic meta-theorems for restrictions of treewidth. Algorithmica **64**(1), 19–37 (2012)
50. Lanctôt, J.K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. Inf. Comput. **185**(1), 41–55 (2003)
51. Lanctôt, J. Kevin: Some string problems in computational biology. Ph.D. thesis, University of Waterloo (2000)
52. Lenstra Jr., H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. **8**(4), 538–548 (1983)
53. Li, M., Ma, B., Wang, L.: On the closest string and substring problems. J. ACM **49**(2), 157–171 (2002)
54. Lokshtanov, D.: Parameterized integer quadratic programming: variables and coefficients. Tech. rep. (2015). http://arxiv.org/abs/1511.00310
55. Ma, B., Sun, X.: More efficient algorithms for closest string and substring problems. SIAM J. Comput. **39**(4), 1432–1443 (2009)
56. Marx, D.: Closest substring problems with small distances. SIAM J. Comput. **38**(4), 1382–1410 (2008)
57. Mnich, M., van Bevern, R.: Parameterized complexity of machine scheduling: 15 open problems. Comput. Oper. Res. **100**, 254–261 (2018)
58. Mnich, M., Wiese, A.: Scheduling and fixed-parameter tractability. Math. Program. **154**((1–2, Ser. B)), 533–562 (2015)
59. Niedermeier, R.: Ubiquitous parameterization—invitation to fixed-parameter algorithms. In: Proceedings of MFCS 2004, Lecture Notes in Computer Science, vol. 3153, pp. 84–103 (2004)
60. Nishimura, N., Simjour, N.: Enumerating neighbour and closest strings. In: Proceedings of IPEC 2012, Lecture Notes in Computer Science, vol. 7535, pp. 252–263 (2012)

61. Onn, S.: Nonlinear discrete optimization. Zurich Lectures in Advanced Mathematics, European Mathematical Society (2010)
62. Onn, S.: Huge multiway table problems. Discrete Optim. **14**, 72–77 (2014)
63. Onn, S., Sarrabezolles, P.: Huge unimodular n-fold programs. SIAM J. Discrete Math. **29**(4), 2277–2283 (2015)
64. Papadimitriou, C.H.: On the complexity of integer programming. J. ACM **28**(4) (1981)
65. Stojanovic, N., Berman, P., Gumucio, D., Hardison, R.C., Miller, W.: A linear-time algorithm for the 1-mismatch problem. In: Proceedings of WADS 1997, Lecture Notes in Computer Science, vol. 1272, pp. 126–135 (1997)

Affiliations

Dušan Knop[1] · **Martin Koutecký**[2,3] · **Matthias Mnich**[4,5]

✉ Matthias Mnich
matthias.mnich@tuhh.de

Dušan Knop
dusan.knop@fit.cvut.cz

Martin Koutecký
koutecky@iuuk.mff.cuni.cz

1 Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic

2 Technion - Israel Institute of Technology, Haifa, Israel

3 Charles University, Prague, Czech Republic

4 TU Hamburg, Institute for Algorithms and Complexity, Hamburg, Germany

5 Universität Bonn, Bonn, Germany