

Mind the Gap. Doubling Constant Parametrization of Weighted Problems: TSP, Max-Cut, and More

Mihail Stoian   

University of Technology Nuremberg, Germany

Abstract

Despite much research, hard weighted problems still resist super-polynomial improvements over their textbook solution. On the other hand, the unweighted versions of these problems have recently witnessed the sought-after speedups. Currently, the only way to repurpose the algorithm of the unweighted version for the weighted version is to employ a polynomial embedding of the input weights. This, however, introduces a pseudo-polynomial factor into the running time, which becomes impractical for arbitrarily weighted instances.

In this paper, we introduce a new way to repurpose the algorithm of the unweighted problem. Specifically, we show that the time complexity of several well-known NP-hard problems operating over the $(\min, +)$ and $(\max, +)$ semirings, such as TSP, Weighted Max-Cut, and Edge-Weighted k -Clique, is proportional to that of their unweighted versions when the set of input weights has small doubling. We achieve this by a meta-algorithm that converts the input weights into polynomially bounded integers using the recent constructive Freiman’s theorem by Randolph and Węgrzycki [ESA 2024] before applying the polynomial embedding.

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

Keywords and phrases doubling constant parametrization, weighted problems, traveling salesman, weighted max-cut, edge-weighted k -clique

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.79

Related Version *Full Version*: <https://arxiv.org/abs/2601.00768>

Acknowledgements The author thanks the anonymous reviewers of STACS’25 and STACS’26, whose detailed comments visibly improved the quality of the presentation.

1 Introduction

The renaissance witnessed in exact algorithm design over the last two decades has one of its roots in the quest to answer the following fundamental question:

Are textbook solutions of NP-hard problems the best we can hope for?

Indeed, there are many problems that resist improvements over the standard algorithm. Yet, we have arguments to be optimistic: The amount of work in this direction has led to impressive speedups for well-studied problems such as MAX-CUT [48], HAMILTONICITY [9, 10], SCHED [20], INDEPENDENT SET [23], DOMINATING SET [23, 44], and MIN k -CUT [32]. To some extent, these speedups have been achieved mainly for what we will refer to as *unweighted* problems. The *weighted* versions of many of these problems are still stuck with the textbook solutions that accept only modest, mostly polynomial-time, improvements.

A concrete example is that of the Traveling Salesman Problem (TSP), which asks to find the shortest tour through n cities. The textbook algorithm in $O(2^{2n})$ -time remains, up to some modest polynomial improvements [13], the best up to date for the general setting [7, 26, 35]. In contrast, its unweighted counterpart, the HAMILTONICITY problem, has been (non-trivially) sped up to $O^*(1.66^n)$ -time by Björklund [9].¹

¹ The O^* -notation hides polynomial factors in the input size.

Interestingly, virtually every new result on the unweighted version of a weighted problem has its own theorem stating that the original problem can be solved in the time of its unweighted counterpart, *yet* with an extra multiplicative pseudo-polynomial dependence on the largest input weight; this is known as the “(polynomial) embedding technique”. Notably, this situation also concerns TSP: Björklund proved that HAMILTONICITY can be leveraged to solve TSP by paying the price of a near-linear time-factor depending on the largest edge weight W [9, Thm. 3]. This becomes impractical for arbitrarily large W . The same holds true for the weighted MAX-CUT problem: its unweighted counterpart can be solved in $O^*(2^{\omega n/3})$ -time [48, 28], where $\omega < 2.371339$ is the matrix multiplication exponent [5], while its weighted version has to “accept” the additional pseudo-polynomial factor. On the other hand, if the weights are polynomially bounded, then the problematic factor becomes polynomial, and the running time of the weighted version matches, in the O^* -sense, that of the unweighted version.

Research Question. Given the ubiquity of the embedding technique, it is natural to ask whether the gap between the running times of the weighted and unweighted versions of a problem is overcome *only* in the regime of polynomially bounded weights. A negative answer would be consistent with the widespread adoption of the embedding approach, with its inherent pseudo-polynomial dependence [48, 28, 11, 12, 9, 31, 18]. In contrast, a positive answer would re-open the door to a more systematic re-examination of the weighted vs. unweighted gap, in the spirit of the “Losing Weights by Gaining Edges” technique for node-weighted problems [3] or in terms of approximation guarantees [17].

1.1 Our Results

Somewhat intriguingly, we show in this work that the polynomially bounded weights setting is *not* the only one that closes the gap between the weighted and unweighted versions of a problem. The meta-algorithm we introduce applies to a wide spectrum of NP-hard combinatorial problems that operate over the $(\min, +)$ and $(\max, +)$ semirings. Namely, it takes as input a weighted problem instance which guarantees that its weight set has *small doubling*, and solves it in time proportional to that of its unweighted counterpart. The set of input weights A is said to have small doubling if $|A + A| \leq C|A|$ for some constant $C > 1$, referred to as the *doubling constant*.

To maintain notation consistency with prior work on *unweighted* problems [37], we prefix problem names with “ \mathcal{C} -” if the set of input weights has doubling constant \mathcal{C} . We also write $O_{\mathcal{C}}$ to note that we suppressed factors that depend only on \mathcal{C} . Under this notation, all of our results conform to the following template:

► **Corollary 1.** *If HAMILTONICITY can be solved in time $T(n)$, then \mathcal{C} -TSP can be solved in time $O_{\mathcal{C}}^*(T(n))$.*

The applicability of our work goes beyond TSP. Indeed, we can consider any weighted NP-hard problem that operates over the $(\min, +)$ and $(\max, +)$ semirings, such as the WEIGHTED MAX-CUT, EDGE-WEIGHTED k -CLIQUE, or MINIMUM STEINER TREE problems.² The only requirement is that the problem at hand satisfies a property ϕ : its objective value is an additive combination of input weights, and it admits an (algebraic) algorithm as seen earlier; see Prop. 1 for a formal treatment.

² We kindly refer the reader to the relevant section for the problem definitions.

To avoid redundancy across different problems, we present our results under the umbrella of a meta-algorithm. Namely, given a weighted problem \mathbf{P}_w that satisfies the above property ϕ , its small-doubling counterpart can be solved in time matching that of the unweighted version:

► **Theorem 2 (Meta-algorithm).** *If problem \mathbf{P}_w satisfies property ϕ and the unweighted version can be solved in time $O(T(n))$ by an algebraic algorithm \mathcal{A} , then $\mathcal{C}\text{-}\mathbf{P}_w$ can be solved in time $O_{\mathcal{C}}^*(T(n))$.*

The following corollaries follow:

► **Corollary 3.** *If unweighted MAX-CUT can be solved in time $T(n, m)$, then $\mathcal{C}\text{-WEIGHTED MAX-CUT}$ can be solved in time $O_{\mathcal{C}}^*(T(n, m))$.*

► **Corollary 4.** *If $k\text{-CLIQUE}$ can be solved in time $T(n, m, k)$, then $\mathcal{C}\text{-EDGE-WEIGHTED } k\text{-CLIQUE}$ can be solved in time $O_{\mathcal{C}}^*(T(n, m, k))$.*

► **Corollary 5.** *If the unweighted version of MINIMUM STEINER TREE can be solved in time $T(n, m, k)$, then $\mathcal{C}\text{-MINIMUM STEINER TREE}$ can be solved in time $O_{\mathcal{C}}^*(T(n, m, k))$.*

We outline in the following how to obtain the results. Throughout the paper, we refer to the algebraic algorithm that exhibits the pseudo-polynomial dependence on W as the *bounded-input* algorithm.

1.2 Overview of Approach

Let us provide some intuition of our approach before we continue with the general framework.

Intuition. Consider TSP as a running example. Recall that the bounded-input algorithm works fine as long as W is small and the weights are in $[0, W]$. Intuitively, we can “hack” the algorithm to also work with weights that are a subset of a *simple* arithmetic progression $\{V, V + r, \dots, V + W \cdot r\}$, particularly for large values of V and r , namely: Map the weights to $[0, W]$ by subtracting V and dividing by r , compute the optimal tour as before on $[0, W]$, and, finally, recover the original objective by scaling by r and adding nV . Thus, as long as the arithmetic progression is small, the algorithm still works. Our meta-algorithm lifts this idea to weights lying in a small *generalized* arithmetic progression – a structure that captures small-doubling sets – as follows.

Meta-Algorithm. Let A denote the set of input weights of the problem at hand, such as the edge weights in TSP. The first observation is that the objective value of the problem is contained in a q -fold sumset of A , i.e., $A + \dots + A$ with A occurring q times, where q is problem-specific; for TSP we have $q = n$. The second observation is that if A has small doubling, then the cardinality of this very q -fold sumset is polynomial w.r.t. $O_{\mathcal{C}}$.³ This suggests that, *in principle*, we could run the bounded-input algorithm. However, the values may still be numerically as large as $\Theta(q \cdot \max A)$. The missing piece is a decomposition of the weights into constant-size coefficient-tuples coming from the generalized arithmetic progression A is a subset of (cf. Freiman’s theorem [24]).

To be able to apply this to any bounded-input algorithm out of the box, we must ensure that the coefficient representation preserves the total order of the original weights. We ensure this by designing an order-preserving monomorphism from the coefficient-tuple space into $\{0, \dots, |A'| - 1\}$, where $qA \subseteq A'$ and $|A'|$ remains polynomial w.r.t. $O_{\mathcal{C}}$.

³ Noteworthy, Plünnecke’s inequality is too loose in this setting, leading to an exponential bound.

1.3 Related Work

We survey applications of additive combinatorics to algorithm design and the literature on the unweighted-weighted dichotomy, focusing on the polynomial embedding technique.

Additive Combinatorics in Algorithm Design. Our work continues the effort to consider important results from additive combinatorics [42] as a new toolbox for algorithm design, a connection highlighted in works such as Ref. [43, 46, 8, 33]. One of the first seminal results is that of Chan and Lewenstein [14], who used the (constructive) Balog-Szemerédi-Gowers (BSG) theorem to speed up the monotone variant of the min-plus convolution on small integers. Recently, this has been further used and extended in #APSP [15], 3SUM [27], listing 4-cycles [1], and approximate distance oracles [1]. We observe that the constructive version of a fundamental theorem in additive combinatorics bears fruit in algorithm design. To this end, Randolph and Węgrzycki [37] made Freiman’s theorem [24], another fundamental result in additive combinatorics, constructive. Their main applications were to SUBSETSUM and ILP-FEASIBILITY, which have an inherent additive structure. Indeed, we extend their results to the area of weighted problems.

Embedding Technique. Repurposing the unweighted algorithm for the weighted version of a problem is a common theme in work that improves over the textbook solution, as in seminal results such as Ref. [48, 11, 12, 9]; notably, in all these examples, the weighted problem operates over the $(\min, +)$ and $(\max, +)$ semirings. The key idea is to represent the input weights of the problem instance as monomials so that the bounded-input algorithm can work in the $(+, \times)$ ring. The drawback is that this introduces a pseudo-polynomial dependence in the running time, since operations now incur an additional $\tilde{O}(W)$ factor,⁴ where W is the largest input weight. For instance, undirected HAMILTONICITY runs in $O^*(1.66^n)$ -time, hence TSP can be solved in $O^*(1.66^n W)$ -time. This, however, becomes impractical for arbitrarily large W . Still, if W is polynomially bounded, then up to polynomial factors, the weighted problem becomes equivalent to the unweighted one. For example, if $W = \text{poly}(n)$, then TSP can be solved in time $O^*(1.66^n)$, matching its unweighted counterpart.

Dodging Pseudo-Polynomiality. The pseudo-polynomial factor is often unwanted. One way to shave it is to enter the realm of approximation. In fact, there is a simple exercise to turn any bounded-input algorithm for a problem into an exponential-time $(1 + \varepsilon)$ -approximation algorithm. For the MAX-CUT problem, Williams mentions this idea in passing [48], inspired by the literature on the all-pairs shortest paths problem [49]; see, e.g., Ref. [41] for more such examples. It is interesting to ask whether the doubling constant could play a role in approximation algorithms.

1.4 Roadmap

We structure the paper as follows. We start with preliminaries on the embedding technique (Sec. 2.1) and the necessary notation in additive combinatorics (Sec. 2.2). Then, we introduce the meta-algorithm (Sec. 3), including the order-preserving monomorphism (Sec. 3.1). Subsequently, we instantiate the meta-algorithm on several weighted problems (Sec. 4). We conclude with an outlook on polynomial-time weighted problems (Sec. 5).

⁴ The \tilde{O} -notation hides polylogarithmic factors in the input size.

2 Preliminaries

Notation. For $a < b \in \mathbb{Z}$, define $[a, b] \triangleq \{a, \dots, b\}$. As shorthand, we define $[n] \triangleq [1, n]$. Moreover, we define $f(X)$ as $\{f(x) \mid x \in X\}$ for a function f and set X .

Time Complexities. As we will only deal with exponential-time algorithms, we use the O^* -notation to hide polynomial factors in the input size. For example, $O^*(2^n) = O(2^n \text{poly}(n))$. In particular, this notation will never hide pseudo-polynomial factors depending on the input weights; specifically, we mean the largest input weight, which will be denoted by W . Furthermore, we use the $O_{\mathcal{C}}$ -notation, already established in Ref. [37], to denote that we suppressed factors depending on \mathcal{C} , the doubling constant on which we parameterize the time-complexities. For instance, $O_{\mathcal{C}}(2^n) = f(\mathcal{C}) \cdot O(2^n)$ for a computable function f . To put into context, $O_{\mathcal{C}}^*(2^n)$ will thus mean $f(\mathcal{C}) \cdot O^*(2^n)$.

Graphs. Unless otherwise specified, all problems we consider in this work operate on an undirected graph $G = (V, E)$, most of the time endowed with a weight function w defined on the edge set $E(G)$ that attributes each edge e a positive value. The length of a path $\pi = (v_1, \dots, v_k)$ is $k - 1$, i.e., we count the number of edges, while the weight of the path is the sum of all edge weights, i.e., $w(\pi) := \sum_{i=1}^{k-1} w(v_i, v_{i+1})$.

2.1 Embedding Technique

We present the embedding technique, which is used by several works to leverage the running time of the unweighted problem for the corresponding weighted counterpart, at the cost of a pseudo-polynomial dependence on the largest input weight, W . Specifically, we consider NP-hard problems that admit formulations over the $(\min, +)$ and $(\max, +)$ semirings: combining substructures corresponds to addition, while selecting among candidate values corresponds to taking a minimum (for minimization) / maximum (for maximization).

Overview. The key idea is to encode weights as exponents in a polynomial ring, since (i) the “+” operator of our semirings translates to multiplication between monomials ($x^{a+b} = x^a \cdot x^b$), and (ii) aggregating candidate values corresponds to polynomial addition, which retains information about all previously attainable values.⁵ Hence, the output of the bounded-input algorithm is a *solution polynomial*:

► **Definition 6** (Solution polynomial). *Let I_w be an instance of a weighted combinatorial problem with a finite set $\mathcal{F}(I_w)$ of feasible solutions. For each $\sigma \in \mathcal{F}(I_w)$, let $v(\sigma) \in \mathbb{Z}_{\geq 0}$ denote its objective value. The solution polynomial of I_w is the ordinary generating function*

$$F_{I_w}(x) = \sum_{\sigma \in \mathcal{F}(I_w)} x^{v(\sigma)} = \sum_{s=0}^{V_{\max}} m_{I_w}(s) x^s,$$

where $m_{I_w}(s) := |\{\sigma \in \mathcal{F}(I_w) \mid v(\sigma) = s\}|$ and $V_{\max} := \max_{\sigma \in \mathcal{F}(I_w)} v(\sigma)$.

The solution polynomial abstracts away whether the problem is a minimization or maximization problem: for minimization we select the monomial with the smallest exponent, whereas for maximization we select the monomial with the largest exponent. To this end, we refer to an algorithm as *algebraic*, if it returns a solution polynomial for an instance of a problem.⁶

⁵ If the reader will, this operation “simulates” an additive inverse.

⁶ An alternative attribute could be *non-combinatorial*. For a treatment on the definition of a combinatorial algorithm, see Ref. [2].

Due to the additive structure of the input weights we will be exploiting, starting in Sec. 3, we will have to work with coordinate tuples instead of the standard setting of integer weights. To overcome this, we will propose a pairing function that transforms tuples into equivalent integer weights; see Sec. 3.1 for the technical details.

2.2 Additive Combinatorics

As the trend of importing results from additive combinatorics into algorithm design is itself rather young, we provide the necessary notation so that our algorithms can be easily followed.

Basic Notation. Let A and B be two sets. Define their sumset (also known as the Minkowski sum) $A + B$ in a commutative group as $\{a + b \mid a \in A, b \in B\}$. When referring to $A + A$, we directly note $2A$. Hence, the iterated sumset h -fold hA is defined recursively by $hA = (h - 1)A + A$.

Doubling Constant. An important definition is that of the doubling constant of a set A , which quantifies how large the sumset becomes w.r.t. to the original set:

$$\mathcal{C}(A) = \frac{|A + A|}{|A|}. \quad (1)$$

Whenever $\mathcal{C}(A)$ does not depend on the size of A , we directly write \mathcal{C} . We say that these sets have *small doubling*. Our work shows that whenever the set of input weights of NP-hard problems has small doubling, that instance can be solved faster.

This comes as a surprise given the simple definition of the doubling constant in Eq. (1). To gain an intuitive understanding, consider the case when A is a (simple) arithmetic progression, say $\{2, 4, 6, 8\}$. Its 2-fold sumset, $A + A$, is $\{4, 6, 8, 10, 12, 14, 16\}$. Taking a more general example will lead us to conclude that the sumset of a (simple) arithmetic progression does not explode that much. Indeed, $|A + A| = 2|A| - 1$, whenever A is a (simple) arithmetic progression. On the contrary, the set $\{3, 5, 9, 17\}$ results in a sumset that attains the maximum size for a 4-size set, namely 10 elements. In other words, there are no distinct pairs in $A \times A$ that have the same sum.

GAPs. Apart from the well-known *simple* arithmetic progression, additive combinatorics results target *generalized* arithmetic progressions (GAP). Formally, a GAP G is defined as

$$G = \{x_1 \ell_1 + \dots + x_d \ell_d \mid \forall i \in [d], \ell_i \in [0, L_i]\},$$

where d is the dimension of the GAP, $\{x_1, \dots, x_d\}$ are the generators, and $\{L_1, \dots, L_d\}$ are the dimension bounds; in particular, a 1-dimensional GAP is a simple arithmetic progression.

In our previous examples, we saw that if A is an arithmetic progression, then it has a small doubling constant; the interesting problems in additive combinatorics are indeed the *inverse* problems: *What does a set of small doubling look like?* This is where Freiman's theorem comes into play.

Freiman's Theorem. Given a set A of small doubling, Freiman's theorem [24] tells that, indeed, A is included in a GAP G , whose dimension is independent on $|A|$. In this work, we will need the *constructive* Freiman's theorem, recently designed by Randolph and Węgrzycki [37]:

► **Theorem 7** (Constructive Freiman’s Theorem [37]). *Let A be a set of n integers with $|A + A| \leq C|A|$. Then, there exists an $\tilde{O}_{\mathcal{C}}(n)$ -time algorithm that with probability $1 - n^{-\gamma}$, for an arbitrarily large constant $\gamma > 0$, returns a generalized arithmetic progression*

$$G = \{x_1 \ell_1 + x_2 \ell_2 + \dots + x_{d(\mathcal{C})} \ell_{d(\mathcal{C})} \mid \forall i, \ell_i \in [L_i]\} \supseteq A$$

with dimension $d(\mathcal{C})$ and volume $v(\mathcal{C})|A|$, where d and v are computable functions that depend only on the doubling constant C . Specifically, the theorem provides the values $x_1, x_2, \dots, x_{d(\mathcal{C})}$ and $L_1, L_2, \dots, L_{d(\mathcal{C})}$.

The above theorem gives us a GAP of dimension $d(\mathcal{C}) = 2^{C^{O(1)}}$ and volume $v(\mathcal{C}) = 2^{2^{C^{O(1)}}} n$, as in the original Freiman’s theorem [24]. Noteworthy, later works do optimize d and v [16, 40, 38, 39], yet these improvements are not captured by the constructive version.

This completes the necessary preliminaries on additive combinatorics. We are now ready to show that, in the $O_{\mathcal{C}}$ -sense, NP-hard weighted problems reduce to their unweighted versions whenever their input weights form a set of small doubling.

3 Meta-Algorithm

Before outlining the meta-algorithm, we first define the property that the considered weighted problems have to satisfy to be amenable to the recent speedups achieved for their unweighted counterparts. As motivated in the introduction, the key requirement is that the objective value has to be an additive combination of input weights. This additive structure ensures that objective values lie within a folded sumset of the input weights.

► **Property 1** (Property ϕ). *Let \mathbf{P}_w be a weighted problem and \mathbf{P} its unweighted counterpart. Let I be an instance of \mathbf{P}_w of size n , $w(I)$ its set of weights, and $W = \max w(I)$. Then, we say that \mathbf{P}_w has property ϕ if the following hold:*

1. *Any feasible solution to I is the total weight of a set $S \subseteq w(I)$,*
2. *There is an algebraic algorithm \mathcal{A} that solves \mathbf{P} in $O(T(n))$ -time and \mathbf{P}_w in $O^*(T(n) \cdot W)$ -time, and any intermediate solution to I produced by \mathcal{A} is the total weight of a polynomial-size multi-set with support in $w(I)$.*

Notably, the additional requirement in Condition 2 is inherent to any bounded-input algorithm whose running time has the near-linear dependence on the largest input weight W . Next, we introduce the pairing function that maps GAP coordinates to integers, allowing the bounded-input algorithm to operate directly on integers rather than raw coordinate tuples.

3.1 Pairing Function

To be able to actually run the bounded-input algorithm \mathcal{A} , we need to transform GAP coordinates into integers. In particular, the meta-algorithm will first enlarge the dimension bounds before encoding them as integers, so that the coordinates set is *well-defined*, namely: Given any two GAP coordinate tuples $\alpha = \langle \alpha_1, \dots, \alpha_d \rangle$ and $\beta = \langle \beta_1, \dots, \beta_d \rangle$ of dimension d , it holds that $\alpha_i + \beta_i \leq L_i, \forall i \in [d]$.

In addition, as we operate over the $(\min, +)$ and $(\max, +)$ semirings, the pairing function κ has to preserve the “additivity” of the coordinates, namely:⁷

$$\kappa(\alpha \oplus \beta) = \kappa(\alpha) + \kappa(\beta), \tag{2}$$

⁷ Note that we skip redundant information from κ , such as the dimension and the dimension bounds.

79:8 Mind the Gap. Doubling Constant Parametrization of Weighted Problems

where “ \oplus ” is the entrywise addition operator, i.e., $\langle \alpha \oplus \beta \rangle_i := \alpha_i + \beta_i, \forall i \in [d]$. Since κ must be injective, so that we can uniquely get back the GAP coordinates, κ has to be a monomorphism between a well-defined set of GAP coordinates and the encoded integers.

Our Pairing Function. We propose the following pairing function:

$$\kappa(d, \langle L_i \rangle_{i \in [d]}, \langle l_i \rangle_{i \in [d]}) = \begin{cases} l_1, & \text{if } d = 1, \\ l_d + (L_d + 1) \cdot \kappa(d - 1, \langle L_i \rangle_{i \in [d-1]}, \langle l_i \rangle_{i \in [d-1]}), & \text{if } d > 1. \end{cases}$$

It receives the dimension of the GAP, d , its dimension bounds $\langle L_i \rangle_{i \in [d]}$, and the GAP coordinates $\langle l_i \rangle_{i \in [d]}$ of an element, and returns the encoded value. If there is only one dimension, then the corresponding GAP coordinate is immediately returned. Otherwise, it builds the encoded value recursively, by multiplying the result for the remaining $d - 1$ dimensions by $L_d + 1$ and finally adding l_d . We show that in our setting it holds that κ is a monomorphism:

► **Lemma 8.** *Let $\langle L_i \rangle_{i \in [d]} \in \mathbb{N}^d$ be the bounds of a GAP of dimension d . Given a well-defined set $D \subseteq \prod_{i=1}^d [0, L_i]$ of d -size coordinate tuples, the pairing function κ is a monomorphism between (D, \oplus) and $([0, \prod_{i=1}^d (L_i + 1) - 1], +)$.*

Proof. We first prove that κ is a homomorphism and then prove that it is also injective.

Homomorphism. Let $\alpha = \langle \alpha_1, \dots, \alpha_d \rangle$ and $\beta = \langle \beta_1, \dots, \beta_d \rangle$ be two GAP-coordinates. Their entrywise addition $\alpha \oplus \beta$ is defined as

$$\alpha \oplus \beta = \langle \alpha_1 + \beta_1, \dots, \alpha_d + \beta_d \rangle.$$

At this point, we use the fact that D is well-defined, i.e., the dimension bounds are not exceeded: $\alpha_i + \beta_i \leq L_i, \forall i \in [d]$. (This is guaranteed by the fact that we enlarge the dimension bounds before running the bounded-input algorithm; compare line 3 in Alg. 1). Formally, we have to prove that

$$\kappa(d, \langle L_i \rangle_{i \in [d]}, \alpha \oplus \beta) = \kappa(d, \langle L_i \rangle_{i \in [d]}, \alpha) + \kappa(d, \langle L_i \rangle_{i \in [d]}, \beta). \quad (3)$$

In the sequel, we use the notation $\kappa_{d-1}(x)$ to denote that κ is applied only on the first $d - 1$ dimensions of a tuple of GAP coordinates x , i.e., $\kappa(d - 1, \langle L_i \rangle_{i \in [d-1]}, \langle x_i \rangle_{i \in [d-1]})$; analogously, κ_d will refer to the first d dimensions. To prove Eq. (3), we use induction on the number of dimensions. The base case, $d = 1$, follows by construction. Otherwise, assume Eq. (3) holds for $d - 1$ dimensions (IH1), i.e.,

$$\kappa_{d-1}(\alpha \oplus \beta) = \kappa_{d-1}(\alpha) + \kappa_{d-1}(\beta).$$

Then,

$$\begin{aligned} & \kappa_d(\alpha \oplus \beta) = \kappa_d(\alpha) + \kappa_d(\beta) \\ \stackrel{\kappa}{\iff} & (\alpha \oplus \beta)_d + (L_d + 1)\kappa_{d-1}(\alpha \oplus \beta) = \alpha_d + \beta_d + (L_d + 1)(\kappa_{d-1}(\alpha) + \kappa_{d-1}(\beta)) \\ \stackrel{\text{IH1}}{\iff} & (\alpha \oplus \beta)_d = \alpha_d + \beta_d, \end{aligned}$$

which is exactly the definition of $\alpha \oplus \beta$.

Injectivity. We next prove the injectivity of κ , by showing that κ actually satisfies a certain monotonicity property, outlined in the following:

$$\alpha \prec \beta \implies \kappa_d(\alpha) < \kappa_d(\beta), \quad (4)$$

where “ $\alpha \prec \beta$ ” means that α is strictly lexicographically smaller than β . We prove this property by induction on the number of dimensions. The base case, $d = 1$ follows by construction. Now, assume that property Eq. (4) holds for the first $d - 1$ dimensions (IH2), i.e.,

$$\langle \alpha_i \rangle_{i \in [d-1]} \prec \langle \beta_i \rangle_{i \in [d-1]} \implies \kappa_{d-1}(\alpha) < \kappa_{d-1}(\beta).$$

To prove the original Eq. (4), we differentiate between two cases when $\alpha \prec \beta$ holds true:

Case $\langle \alpha_i \rangle_{i \in [d-1]} = \langle \beta_i \rangle_{i \in [d-1]}$. This means that $\kappa_{d-1}(\alpha) = \kappa_{d-1}(\beta)$. Moreover, $\alpha_d < \beta_d$ (*), otherwise $\alpha \prec \beta$ cannot be true. Hence,

$$\begin{array}{l} \xleftrightarrow{\kappa} \\ \text{Case} \\ \xleftrightarrow{\quad} \end{array} \quad \begin{array}{l} \kappa_d(\alpha) < \kappa_d(\beta) \\ \alpha_d + (L_d + 1)\kappa_{d-1}(\alpha) < \beta_d + (L_d + 1)\kappa_{d-1}(\beta) \\ \alpha_d < \beta_d, \end{array}$$

which is true due to (*).

Case $\langle \alpha_i \rangle_{i \in [d-1]} \prec \langle \beta_i \rangle_{i \in [d-1]}$. In this case, there is no relation between α_d and β_d .⁸ Hence, we can apply IH2 and use the fact that the difference $\kappa_{d-1}(\beta) - \kappa_{d-1}(\alpha)$ is strictly greater than 0:

$$\begin{array}{l} \xleftrightarrow{\kappa} \\ \iff \\ \iff \\ \xleftrightarrow{\text{IH2}} \end{array} \quad \begin{array}{l} \kappa_d(\alpha) < \kappa_d(\beta) \\ \alpha_d + (L_d + 1)\kappa_{d-1}(\alpha) < \beta_d + (L_d + 1)\kappa_{d-1}(\beta) \\ \alpha_d - \beta_d < (L_d + 1)(\kappa_{d-1}(\beta) - \kappa_{d-1}(\alpha)) \\ \alpha_d - \beta_d < L_d + 1, \end{array}$$

which is true since the difference $\alpha_d - \beta_d$ can only lie in $[-L_d, L_d]$.

This completes the injectivity argument, hence κ is a monomorphism. \blacktriangleleft

As a by-product of the injectivity proof, we had to show that κ satisfies a certain monotonicity property:

$$\alpha \prec \beta \implies \kappa(\alpha) < \kappa(\beta), \quad (5)$$

where $\alpha \prec \beta$ stands for the (syntactic) lexicographical order of the coordinate tuples, e.g., $\langle 1, 2 \rangle \prec \langle 1, 3 \rangle$, $\langle 1, 2 \rangle \prec \langle 2, 1 \rangle$. However, even then, there is still an issue: *monomorphism alone does not suffice*. We discuss why this is the case in the following.

Why Monomorphism Alone Is Not Sufficient. The present form of κ does not suffice to show the correctness of our meta-algorithm. The salient issue is that κ does *not* preserve the total order of the tuples regarding their original, to-be-represented value. Consider for instance the coordinate tuples $\langle 1, 2 \rangle$ and $\langle 2, 1 \rangle$. While we know that $\langle 1, 2 \rangle \prec \langle 2, 1 \rangle$ in the lexicographical sense and implicitly $\kappa(\langle 1, 2 \rangle) < \kappa(\langle 2, 1 \rangle)$, according to Eq. (5), it could be that the generators of the GAP are $\langle 3, 10 \rangle$. In that case, the tuples should be ordered as $\langle 2, 1 \rangle \sqsubset \langle 1, 2 \rangle$, since $3 \cdot 2 + 10 \cdot 1 < 3 \cdot 1 + 10 \cdot 2$, where we refer to “ \sqsubset ” as the total order on the GAP coordinates.

⁸ To see why this is the case, note that in an English dictionary “root” \prec “rope”, even though the last letter of “rope”, “e”, comes before the “t” of “root”.

Restoring the Order. To address this, we augment the call to the bounded-input algorithm with a (fixed) *permutation* π that restores the order of the original values. Whenever the algorithm has to find the minimum / maximum monomial in the solution polynomial (Def. 6) – the final last step in the bounded-input algorithms we consider –, it uses π to recover the true rank of the current encoding e .

Building π . The permutation can be built in polynomial-time w.r.t. O_C , as follows: Once the dimension bounds of the GAP have been enlarged (see line 3 in the upcoming Alg. 1), we can generate the values $\sum_{i=1}^d x_i \cdot l_i$ for $l_i \in [0, \lambda L_i], \forall i \in [d]$, where λ is problem-specific (see the upcoming Sec. 3.2). By construction, these are all possible values that can be taken during a bounded-input algorithm’s run. To actually “freeze” the permutation π , we have to sort the generated values, which takes time $\tilde{O}(\lambda^d \prod_{i=1}^d (L_i + 1))$. We call this step BUILDPERMUTATION in the meta-algorithm.

Decoding. The decoding process follows a similar line and is outlined in the following:

$$\kappa^{-1}(d, \langle L_i \rangle_{i \in [d]}, e) = \begin{cases} \langle e \rangle, & \text{if } d = 1, \\ \kappa^{-1}\left(d-1, \langle L_i \rangle_{i \in [d-1]}, \left\lfloor \frac{e}{L_d+1} \right\rfloor\right) \circ \langle e \bmod (L_d + 1) \rangle, & \text{if } d > 1. \end{cases}$$

Namely, if there is only one dimension, the current encoding is returned, otherwise we append the result of $e \bmod (L_d + 1)$ to the tuple resulting from the remaining $d - 1$ dimensions. The recursive call is done with an e from which the contribution of dimension d has been removed.

3.2 Exploiting the Bounded-Input Algorithm

Let \mathbf{P}_w be a weighted problem satisfying property ϕ , and let I be an instance of \mathbf{P}_w of size n whose set of weights has small doubling. Let \mathcal{A} be the bounded-input algorithm from Condition 2. Furthermore, let $\lambda = \text{poly}(|w(I)|)$ be an upper-bound on the cardinality of the multi-sets from the same condition; as we will see, for all bounded-algorithms we consider in this paper it holds that $\lambda = O(|w(I)|)$. Then, we can state our meta-algorithm as follows:

■ **Algorithm 1** $\mathcal{M}(I, \mathcal{A})$.

-
- 1: $d(\mathcal{C}), \langle x_1, \dots, x_{d(\mathcal{C})} \rangle, \langle L_1, \dots, L_{d(\mathcal{C})} \rangle \leftarrow \text{CONSTRUCTIVEFREIMAN}(w(I))$ [Thm. 7]
 - 2: $w_{\text{coord}} \leftarrow \text{GETGAPCOORDINATES}(w(I), \langle x_1, \dots, x_{d(\mathcal{C})} \rangle, \langle L_1, \dots, L_{d(\mathcal{C})} \rangle)$
 - 3: $w' \leftarrow \kappa(d(\mathcal{C}), \langle \lambda L_1, \dots, \lambda L_{d(\mathcal{C})} \rangle, w_{\text{coord}})$
 - 4: $\pi \leftarrow \text{BUILDPERMUTATION}(d(\mathcal{C}), \langle x_1, \dots, x_{d(\mathcal{C})} \rangle, \langle \lambda L_1, \dots, \lambda L_{d(\mathcal{C})} \rangle, w_{\text{coord}})$
 - 5: $c' \leftarrow \mathcal{A}(w', \pi)$
 - 6: $\langle l'_1, \dots, l'_{d(\mathcal{C})} \rangle \leftarrow \kappa^{-1}(d(\mathcal{C}), \langle \lambda L_1, \dots, \lambda L_{d(\mathcal{C})} \rangle, c')$
 - 7: **return** $\sum_{i=1}^{d(\mathcal{C})} x_i l'_i$ ($=: c^*$)
-

► **Theorem 2 (Meta-algorithm).** *If problem \mathbf{P}_w satisfies property ϕ and the unweighted version can be solved in time $O(T(n))$ by an algebraic algorithm \mathcal{A} , then $\mathcal{C}\text{-}\mathbf{P}_w$ can be solved in time $O_C^*(T(n))$.*

Proof. We prove that the bounded-input algorithm actually receives as input polynomially bounded weights w.r.t. O_C via the new weight function w' (line 5). This will guarantee that the line runs in time $O_C^*(T(n))$. Note that this does not hide pseudo-polynomial factors in the largest input weight.

To this end, let G' be the GAP which $\lambda w(I)$, a superset of any intermediate solution – recall Condition 2, is a subset of. Moreover, the initial set of input weights $w(I)$ is also a subset of a GAP G . Let us see the connection between G' and the original G . First, the parameters of G are $\{x_1, \dots, x_{d(\mathcal{C})}\}$ and $\{L_1, \dots, L_{d(\mathcal{C})}\}$, i.e.,

$$G = \{x_1 l_1 + \dots + x_{d(\mathcal{C})} l_{d(\mathcal{C})} \mid \forall i, l_i \in [L_i]\}.$$

Then, since G' has to capture the weights in $\lambda w(I)$, we need to modify the bounds of the original GAP. We thus obtain the following new GAP:

$$G' = \{x_1 l_1 + \dots + x_{d(\mathcal{C})} l_{d(\mathcal{C})} \mid \forall i, l_i \in [\lambda L_i]\}.$$

Recall now that the coordinates are mapped to the integer range $[0, \prod_{i=1}^{d(\mathcal{C})} (\lambda L_i + 1) - 1]$, which is exactly the volume of G' . Bounding this, we obtain:

$$\begin{aligned} |G'| &= \prod_{i=1}^{d(\mathcal{C})} (\lambda L_i + 1) \leq \lambda^{d(\mathcal{C})} \prod_{i=1}^{d(\mathcal{C})} (L_i + 1) \\ &= \lambda^{d(\mathcal{C})} |G| \\ &= \lambda^{d(\mathcal{C})} v(\mathcal{C}) |w(I)| \\ &= |w(I)|^{O_{\mathcal{C}}(1)}. \end{aligned} \tag{6}$$

Therefore, the bounded-input algorithm (line 5) is run on an instance with polynomially large weights w.r.t. $O_{\mathcal{C}}$, represented by the weight function w' .

Let us analyze the total running time. We argue that all steps apart from line 5 take polynomial time w.r.t. $O_{\mathcal{C}}$ each. Namely, once the parameters of the GAP G have been computed (line 1), we can compute the coordinates corresponding to the values of the weight function w by a naive iteration of G (line 2); this takes time $O(|w(I)| \cdot |G|) = O(|w(I)| \cdot v(\mathcal{C}) |w(I)|) = O_{\mathcal{C}}(|w(I)|^2)$. Then, building the permutation π (line 4) takes time $\tilde{O}(|G'|) = \tilde{O}(|w(I)|^{O_{\mathcal{C}}(1)})$; see Sec. 3.1 and the above Eq. (6). Finally, applying κ on the GAP coordinates w_{coord} takes $O_{\mathcal{C}}(|w(I)|)$ -time (line 3), while lines 6 and 7 take $O_{\mathcal{C}}(1)$ -time and $\tilde{O}_{\mathcal{C}}(1)$ -time, respectively.

Noteworthy, when not parameterizing on the doubling constant, the running time of the meta-algorithm reads $O^*(\lambda^{d(\mathcal{C})} v(\mathcal{C}) T(n))$, which is $O_{\mathcal{C}}^*(T(n))$. ◀

4 Applications

In the following, we review several weighted NP-hard problems and show how to solve their small-doubling instances faster than the traditional algorithm by leveraging the previous meta-algorithm. This enables running times in the regime of the unweighted version of the problem for a large suite of problems, such as TSP, WEIGHTED MAX-CUT, EDGE-WEIGHTED k -CLIQUE, and MINIMUM STEINER TREE. We stress that, compared to prior work [37], we consider *weighted* problems that operate over tropical semirings.

4.1 TSP Meets Small Doubling

We define the doubling constant parametrization of TSP, \mathcal{C} -TSP, as follows:

C-TSP

Input: Undirected graph $G = (V, E)$, a cost $w(e) \in \mathbb{Z}_{\geq 0}$ for each $e \in E = E(G)$, such that $|w(E) + w(E)| \leq C|w(E)|$

Output: Tour $T \subseteq E$ that minimizes $\sum_{e \in T} w(e)$

Why the Embedding Technique Fails. First, note that bounded-input algorithm does not help much for C-TSP: The largest edge weight W can still be exponentially large. Therefore, the resulting running time would be even worse than that of the traditional algorithm. We show an alternative solution via our meta-algorithm (Alg. 1).

Intuition. Let $A := w(E)$ denote the set of edge weights. Recall that the weight of any tour in TSP is computed by the total sum of the edge weights on that tour. To understand how this actually relates to the doubling constant, consider the case when we only consider *paths* of length 2. The possible weights of these paths are all included in $A + A$. Note that not all path weights are also valid: some weights might come from paths using the same edge twice. This, however, is not an issue. The main benefit we get is that the cardinality of $A + A$ is at most $C|A|$. If we continue with longer paths, we will observe the same situation: Namely, the weights of the paths of length k will be included in kA , the k -fold sumset of A . In particular, the shortest tour length will be an element of nA .

► **Lemma 9.** *TSP satisfies property ϕ .*

Proof. Condition 1 holds since a tour's weight is the sum of the weights of its constituent edges. Next, we can take \mathcal{A} as Björklund's algorithm [9] for undirected HAMILTONICITY, which runs in time $O^*(1.66^n)$, and solves the weighted case in time $O^*(1.66^n W)$ [9, Thm. 3]. Due to the algorithm's design, namely that it counts cycle covers of n arcs, we have $\lambda \leq n$. ◀

► **Corollary 1.** *If HAMILTONICITY can be solved in time $T(n)$, then C-TSP can be solved in time $O_C^*(T(n))$.*

4.2 Weighted Max-Cut Under Small Doubling

Another well-studied problem whose unweighted version has been sped up and fits the small doubling setting well is the WEIGHTED MAX-CUT problem. In a breakthrough result, Williams [48] improved the time complexity of the unweighted MAX-CUT to $O^*(2^{\omega n/3})$. We show that we can leverage this result to the weighted setting when the edge weights form an integer set that has small doubling. We formally define the doubling constant parametrization of the problem in the following:

C-Weighted Max Cut

Input: Undirected graph $G = (V, E)$, a weight $w(e) \in \mathbb{Z}_{\geq 0}$ for each $e \in E = E(G)$ such that $|w(E) + w(E)| \leq C|w(E)|$

Output: Subset of vertices S so that $\sum_{e \in \delta(S)} w(e)$ is maximized

► **Lemma 10.** *WEIGHTED MAX-CUT satisfies property ϕ .*

Proof. Condition 1 holds since a cut’s weight is the sum of the weights of its constituent edges. Next, we can take \mathcal{A} as Williams’s algebraic algorithm [48] (and later improved by Koivisto [28]) for the unweighted MAX-CUT problem, which runs in time $O^*(2^{\omega n/3})$, and solves the weighted case in time $O^*(2^{\omega n/3} W)$ [28, Thm. 3]. ◀

Using the above lemma together with Thm. 2, we obtain:

► **Corollary 3.** *If unweighted MAX-CUT can be solved in time $T(n, m)$, then \mathcal{C} -WEIGHTED MAX-CUT can be solved in time $O_{\mathcal{C}}^*(T(n, m))$.*

4.3 Small Doubling and Edge-Weighted k -Cliques

We now turn to a problem whose complexity is a rather popular hypothesis in hardness proofs [6, 4, 30]: the edge-weighted k -clique problem. Compared to the problems considered so far, the algebraic algorithm for the unweighted k -Clique has already been known from the 80’s [36]. We show that it can be leveraged in the doubling constant parametrization of the edge-weighted version. Note that, apart from a mention in passing in Lincoln et al. [30] that Nešetřil-Poljak’s algorithm can also be used for polynomially bounded weights, we could not find any reference to a proof. For completeness, we show this in the following.

We show that Nešetřil-Poljak’s algorithm [36] can be extended to support small weights. We consider the $3k$ -case here; both the $3k + 1$ - and $3k + 2$ -cases are handled analogously.

► **Lemma 11.** *The EDGE-WEIGHTED $3k$ -CLIQUE problem on undirected graphs $G = (V, E)$ with integer edge weights in $[0, W]$ can be solved in time $O^*(n^{\omega k} W)$.*

Proof. We extend Nešetřil-Poljak’s construction to the edge-weighted setting. Let H be the auxiliary graph whose vertices are all k -cliques $Y \subseteq V$ of G , and where distinct Y_1, Y_2 are adjacent iff $Y_1 \cup Y_2$ induces a $2k$ -clique in G . Hence, H has $N = \Theta(n^k)$ vertices.

Denote by $w(e) \in [0, W]$ the weight of an edge $e \in E(G)$. For a k -clique Y , set $w'(Y) = w(E(G[Y]))$, the “internal” weight of node Y , and for adjacent Y_1, Y_2 in H , set

$$w'(Y_1, Y_2) = \sum_{\substack{u \in Y_1 \\ v \in Y_2}} w(uv),$$

representing the “cross” weight between Y_1 and Y_2 . If A, B, C form a triangle in H , the total weight of the corresponding $3k$ -clique $A \cup B \cup C$ is

$$W_{\text{clique}} = w'(A) + w'(B) + w'(C) + w'(A, B) + w'(B, C) + w'(C, A).$$

Now, since we would like to have edge weights only (in order to not have a specialized algorithm for coping with both cases), redefine the weight of the edge $\{Y_1, Y_2\}$ as:

$$w'(Y_1, Y_2) = 2 \sum_{\substack{u \in Y_1 \\ v \in Y_2}} w(uv) + w'(Y_1) + w'(Y_2),$$

that is, we double the previously edge weight, and add the internal weights of the two nodes. Hence, for every triangle $\{A, B, C\}$ in H , we have

$$w'(A, B) + w'(B, C) + w'(C, A) = 2W_{\text{clique}},$$

so minimizing triangle weight in H is equivalent to minimizing $3k$ -clique weight in G .

79:14 Mind the Gap. Doubling Constant Parametrization of Weighted Problems

Next, note that the minimizing edge weight on H satisfies

$$W_H^{\max} \leq (2k^2 + 2\binom{k}{2})W = (3k^2 - k)W = \Theta(k^2W).$$

A minimum-weight triangle in an N -vertex edge-weighted graph with integer weights in $[0, W_H^{\max}]$ can be found via a single distance $(\min, +)$ -product in time $\tilde{O}(W_H^{\max} N^\omega)$. Instantiating $N = \Theta(n^k)$ and $W_H^{\max} = \Theta(k^2W)$ yields time $O^*(n^{\omega k} W)$. Finally, explicitly constructing H and all $w'(\cdot, \cdot)$ values takes $O(n^{2k} \text{poly}(k) \log W)$ -time, which is dominated by $n^{\omega k}$ since $\omega > 2$. ◀

The doubling constant parametrization of edge-weighted k -Clique is the following:

C-Edge-Weighted k -Clique

Input: Undirected graph $G = (V, E)$, integer k , and a weight $w(e) \in \mathbb{Z}_{\geq 0}$ for each $e \in E = E(G)$ such that $|w(E) + w(E)| \leq \mathcal{C}|w(E)|$

Output: A vertex set $S \subseteq V$ with $|S| = k$ that induces a clique and maximizes $\sum_{e \in E(G[S])} w(e)$

► **Lemma 12.** *EDGE-WEIGHTED k -CLIQUE satisfies property ϕ .*

Proof. Condition 1 holds since a clique's weight is the sum of the weights of its constituent edges. Next, we can take \mathcal{A} as Nešetřil-Poljak's algebraic algorithm [36] for the unweighted version of the problem (or the improvement by Eisenbrand and Grandoni [22] for some values of k using rectangular matrix multiplication), which runs in time $O^*(n^{k\omega/3})$, and solves the weighted case in time $O^*(n^{k\omega/3} W)$ (Lemma 11). Notably, due to its design, namely that each original input weight is *doubled* in the auxiliary graph, the bounded-input algorithm (Lemma 11) has a $\lambda \leq 2|w(I)|$. ◀

In combination with Thm. 2, we obtain:

► **Corollary 4.** *If k -CLIQUE can be solved in time $T(n, m, k)$, then C-EDGE-WEIGHTED k -CLIQUE can be solved in time $O_{\mathcal{C}}^*(T(n, m, k))$.*

4.4 Steiner Meets Freiman

For reference, we also include the MINIMUM STEINER TREE, even though the general setting can be directly solved in $O((2 + \epsilon)^k p(n))$ [34], where $p(n)$ is a polynomial function of n , the degree of which grows rapidly as ϵ approaches zero, Its degree has been later refined to $12\sqrt{\epsilon^{-1} \ln \epsilon^{-1}}$, having as a result running times such as $O(2.1^k n^{57.6})$ or $O(2.5^k n^{14.2})$ [25]. Notably, Dreyfus and Wagner [21] were the first to design a dynamic programming recursion running in time $O(3^k n + 2^k n^2 + nm)$, where $n = |V|$, $m = |E|$, and $k = |T|$.

The doubling constant parametrization of the problem is the following:

C-Minimum Steiner Tree

Input: Undirected graph $G = (V, E)$, a weight $w(e) \in \mathbb{Z}_{\geq 0}$ for each $e \in E = E(G)$, and a set of vertices $K \subseteq V = V(G)$

Output: Subgraph H of G that connects the vertices in K and has the minimum total weight $\sum_{e \in E(H)} w(e)$ among all such subgraphs of G

Next, we show that the minimum Steiner tree is amenable for speedups in this regime:

► **Lemma 13.** *MINIMUM STEINER TREE satisfies property ϕ .*

Proof. Condition 1 holds since, as the edge weights are all non-negative, the optimal subgraph H is a tree with its leaves in K , and a tree’s weight is the sum of the weights of its constituent edges. Next, we can take \mathcal{A} as the fast subset convolution based algorithm due to Björklund, Husfeldt, Kaski, and Koivisto [11, Sec. 4.1.2], which runs in $O^*(2^k n^2 W)$ -time, or Lokshtanov–Nederlof’s polynomial-space algorithm [31] in the same running time. ◀

With this, using Thm. 2, we obtain:

► **Corollary 5.** *If the unweighted version of MINIMUM STEINER TREE can be solved in time $T(n, m, k)$, then \mathcal{C} -MINIMUM STEINER TREE can be solved in time $O_{\mathcal{C}}^*(T(n, m, k))$.*

This concludes the suite of applications of our meta-algorithm. We conclude with an outlook on the applicability of our framework beyond NP-hard weighted problems.

5 Outlook: Beyond NP-hard Problems

As shown in this paper, the constructive Freiman’s theorem can be converted in a rather effective application for NP-hard weighted problems as well. A natural question arises: Is it possible to extend its applicability to weighted problems in P?

Min-Plus Convolution. The answer seems to be a positive one, in particular for those polynomial problems for which the solution is an additive combination of the input weights. A simple application is that of the $(\min, +)$ -convolution: Given sequences $(a[i])_{i=0}^{n-1}$ and $(b[i])_{i=0}^{n-1}$, the goal is to compute $(c[i])_{i=0}^{n-1}$, where $c[k] = \min_{i=0, \dots, k} \{a[i] + b[k-i]\}$. The naive algorithm runs in $O(n^2)$ -time, and no $O(n^{2-\varepsilon})$ -time algorithm is known for $\varepsilon > 0$ [19, 29]. However, the convolution in the $(+, \times)$ -ring can be solved in $O(n \log n)$ -time via FFT. This resembles the setting we considered throughout the paper. Indeed, Węgrzycki outlines in his PhD thesis a bounded-input algorithm running in time $\tilde{O}(nW)$, where W is the largest input value [47, Lemma 5.7.2]. Applying the same key idea as for the NP-hard problems we considered, we can obtain an $\tilde{O}_{\mathcal{C}}(n)$ -time algorithm for the min-plus *self*-convolution problem, i.e., $a = b$, and the input sequence, when regarded as a set, has small doubling. The same setting holds true for the min-sum subset convolution [11].

It is thus interesting to ask how the framework behaves in more weighted applications, such as (specialized) $(\min, +)$ matrix products [45] or even APSP, where – similar to the NP-hard case studied here – the objective value is an additive combination of input weights.

References

- 1 Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-SUM Lower Bounds for Approximate Distance Oracles via Additive Combinatorics. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 391–404. ACM, 2023. doi:10.1145/3564246.3585240.
- 2 Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka. New Graph Decompositions and Combinatorial Boolean Matrix Multiplication Algorithms. In Bojan Mohar, Igor Shinkar, and Ryan O’Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 935–943. ACM, 2024. doi:10.1145/3618260.3649696.

- 3 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2014. doi:10.1007/978-3-662-44777-2_1.
- 4 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of Faster Alignment of Sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014. doi:10.1007/978-3-662-43948-7_4.
- 5 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More Asymmetry Yields Faster Matrix Multiplication. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.
- 6 Arturs Backurs and Christos Tzamos. Improving Viterbi is Hard: Better Runtimes Imply Faster Clique Algorithms. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 311–321. PMLR, 2017. URL: <http://proceedings.mlr.press/v70/backurs17a.html>.
- 7 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962. doi:10.1145/321105.321111.
- 8 Khodakhast Bibak. Additive Combinatorics: With a View Towards Computer Science and Cryptography - An Exposition. In Jonathan M. Borwein, Igor E. Shparlinski, and Wadim Zudilin, editors, *Number Theory and Related Fields, In Memory of Alf van der Poorten*, pages 99–128. Springer, 2013. doi:10.1007/978-1-4614-6642-0_4.
- 9 Andreas Björklund. Determinant Sums for Undirected Hamiltonicity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 173–182. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.24.
- 10 Andreas Björklund. Below All Subsets for Some Permutational Counting Problems. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, volume 53 of *LIPICs*, pages 17:1–17:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.SWAT.2016.17.
- 11 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier Meets Möbius: Fast Subset Convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.
- 12 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. doi:10.1137/070683933.
- 13 Timothy M. Chan. The Art of Shaving Logs. In Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, volume 8037 of *Lecture Notes in Computer Science*, page 231. Springer, 2013. doi:10.1007/978-3-642-40104-6_20.
- 14 Timothy M. Chan and Moshe Lewenstein. Clustered Integer 3SUM via Additive Combinatorics. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.
- 15 Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Fredman’s Trick Meets Dominance Product: Fine-Grained Complexity of Unweighted APSP, 3SUM Counting, and More. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 419–432. ACM, 2023. doi:10.1145/3564246.3585237.

- 16 Mei-Chu Chang. A polynomial bound in Freiman’s theorem. *Duke Mathematical Journal*, 113(3):399–419, 2002. doi:10.1215/S0012-7094-02-11331-3.
- 17 Pierluigi Crescenzi, Riccardo Silvestri, and Luca Trevisan. On Weighted vs Unweighted Versions of Combinatorial Optimization Problems. *Inf. Comput.*, 167(1):10–26, 2001. doi:10.1006/inco.2000.3011.
- 18 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 1st edition, 2015. doi:10.1007/978-3-319-21275-3.
- 19 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Włodarczyk. On Problems Equivalent to $(\min, +)$ -Convolution. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 22:1–22:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.22.
- 20 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Scheduling partially ordered jobs faster than $2n$. *Algorithmica*, 68(3):692–714, 2014. doi:10.1007/s00453-012-9694-7.
- 21 Stuart E. Dreyfus and Robert A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/net.3230010302.
- 22 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.
- 23 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009. doi:10.1145/1552285.1552286.
- 24 Gregory A Freiman. Foundations of a structural theory of set addition. *Translation of Math. Monographs*, 37, 1973.
- 25 Bernhard Fuchs, Walter Kern, Daniel Mölle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum steiner trees. *Theory Comput. Syst.*, 41(3):493–500, 2007. doi:10.1007/s00224-007-1324-4.
- 26 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In Thomas C. Rowan, editor, *Proceedings of the 16th ACM national meeting, ACM 1961, USA*, pages 71.201–71.204. ACM, 1961. doi:10.1145/800029.808532.
- 27 Ce Jin and Yinzhan Xu. Removing additive structure in 3sum-based reductions. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 405–418. ACM, 2023. doi:10.1145/3564246.3585157.
- 28 Mikko Koivisto. Optimal 2-constraint satisfaction via sum-product algorithms. *Inf. Process. Lett.*, 98(1):24–28, 2006. doi:10.1016/j.ipl.2005.11.013.
- 29 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-Grained Complexity of One-Dimensional Dynamic Programming. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.21.
- 30 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight Hardness for Shortest Cycles and Paths in Sparse Graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252. SIAM, 2018. doi:10.1137/1.9781611975031.80.
- 31 Daniel Lokshtanov and Jesper Nederlof. Saving Space by Algebraization. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, pages 321–330, 2010. doi:10.1145/1806689.1806735.

- 32 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. Breaking the All Subsets Barrier for Min k -Cut. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 90:1–90:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ICALP.2023.90.
- 33 Shachar Lovett. Additive Combinatorics and its Applications in Theoretical Computer Science. *Theory Comput.*, 8:1–55, 2017. doi:10.4086/toc.gs.2017.008.
- 34 Daniel Mölle, Stefan Richter, and Peter Rossmanith. A Faster Algorithm for the Steiner Tree Problem. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 561–570. Springer, 2006. doi:10.1007/11672142_46.
- 35 Jesper Nederlof. Bipartite TSP in $o(1.9999^n)$ time, assuming quadratic time matrix multiplication. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 40–53. ACM, 2020. doi:10.1145/3357713.3384264.
- 36 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2):415–419, 1985. URL: <http://eudml.org/doc/17394>.
- 37 Tim Randolph and Karol Wegrzycki. Parameterized Algorithms on Integer Sets with Small Doubling: Integer Programming, Subset Sum and k -SUM. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPICs*, pages 96:1–96:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.ESA.2024.96.
- 38 Tom Sanders. On the Bogolyubov-Ruzsa lemma. *Anal. PDE*, 5(3):627–655, 2012. doi:10.2140/apde.2012.5.627.
- 39 Tom Sanders. The structure theory of set addition revisited. *Bulletin of the American Mathematical Society*, 50(1):93–127, October 2012. doi:10.1090/s0273-0979-2012-01392-7.
- 40 Tomasz Schoen. Near optimal bounds in Freiman’s theorem. *Duke Mathematical Journal*, 158(1):1–12, 2011.
- 41 Mihail Stoian. Approximate min-sum subset convolution. In Marcin Bienkowski and Matthias Englert, editors, *Approximation and Online Algorithms - 22nd International Workshop, WAOA 2024, Egham, UK, September 5-6, 2024, Proceedings*, volume 15269 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2024. doi:10.1007/978-3-031-81396-2_14.
- 42 Terence Tao and Van H Vu. *Additive combinatorics*, volume 105. Cambridge University Press, 2006.
- 43 Luca Trevisan. Guest column: additive combinatorics and theoretical computer science. *SIGACT News*, 40(2):50–66, 2009. doi:10.1145/1556154.1556170.
- 44 Johan M. M. van Rooij, Jesper Nederlof, and Thomas C. van Dijk. Inclusion/exclusion meets measure and conquer. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 554–565. Springer, Springer, 2009. doi:10.1007/978-3-642-04128-0_50.
- 45 Virginia Vassilevska Williams and Yinzhao Xu. Truly Subcubic Min-Plus Product for Less Structured Matrices, with Applications. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 12–29. SIAM, 2020. doi:10.1137/1.9781611975994.2.
- 46 Emanuele Viola. Selected results in additive combinatorics: An exposition. *Theory Comput.*, 3:1–15, 2011. doi:10.4086/toc.gs.2011.003.

- 47 K. Węgrzycki. *Provably Optimal Dynamic Programming*. 2019. URL: <https://books.google.de/books?id=UmYfzwEACAAJ>.
- 48 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.
- 49 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, May 2002. doi:10.1145/567112.567114.