Optimization Letters
https://doi.org/10.1007/s11590-023-02017-5

ORIGINAL PAPER

Structured $(\min, +)$-convolution and its applications for the shortest/closest vector and nonlinear knapsack problems

D. V. Gribanov[1] · **I. A. Shumilov**[2] · **D. S. Malyshev**[1]

Received: 29 September 2022 / Accepted: 8 May 2023

Abstract
In this work we consider the problem of computing the $(\min, +)$-convolution of two sequences a and b of lengths n and m, respectively, where $n \geq m$. We assume that a is arbitrary, but $b_i = f(i)$, where $f(x) : [0, m) \to \mathbb{R}$ is a function with one of the following properties: f is linear, f is monotone, f is convex, f is concave, f is piece-wise linear, f is a polynomial function of a fixed degree. To the best of our knowledge, the concave, piece-wise linear and polynomial cases were not considered in literature before. We develop true sub-quadratic algorithms for them. We apply our results to the knapsack problem with a separable nonlinear objective function, shortest lattice vector, and closest lattice vector problems.

Keywords Convolution · Nonlinear knapsack · Separable objective · Shortest vector problem · Closest vector problem · Dynamic programming · Integer programming · Piece-wise linear

The article was prepared within the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE). The authors would like to thank A. Vanin and the anonymous reviewers for useful discussions and remarks during preparation of this article.

✉ D. V. Gribanov
dimitry.gribanov@gmail.com

I. A. Shumilov
ivan.a.shumilov@gmail.com

D. S. Malyshev
dsmalyshev@rambler.ru

[1] National Research University Higher School of Economics, 25/12 Bolshaja Pecherskaja Ulitsa, Nizhny Novgorod, Russian Federation 603155

[2] Lobachevsky State University of Nizhny Novgorod, 23 Gagarina Avenue, Nizhny Novgorod, Russian Federation 603950

Published online: 27 May 2023

1 Introduction

1.1 Structured (min, +)-convolution

The standard (min, +)-*convolution problem* is formulated in the following way. For given $a = \{a_0, a_1, \dots, a_{n-1}\}$ and $b = \{b_0, b_1, \dots, b_{m-1}\}$, where $n \geq m$, it is to compute $a \star b := c$, defined by the formula:

$$c_k = \min_{i+j=k}\{a_i + b_j\}, \quad \text{for} \quad k \in \{0, \dots, n+m-2\}. \tag{MinConv}$$

In the current paper, it is more natural for us to work with another problem that is clearly linear-time equivalent to the original one. The problem is to compute $a \bullet b := c$, defined by the formula:

$$c_k = \min_{0 \leq i \leq m-1}\{a_{k+i} + b_i\}, \quad \text{for} \quad k \in \{0, \dots, n-m\}. \tag{ReducedMinConv}$$

This formulation of MinConv can be found, for example, in [51]. We will call it ReducedMinConv. Unlike the standard $(+, \times)$-convolution, it is not known whether the (min, +)-convolution admits the existence of truly sub-quadratic algorithms. Moreover, the lack of truly sub-quadratic algorithms, despite considerable efforts, has led researchers to postulate the *MinConv-hypothesis* that MinConv cannot be solved in $O(n^{2-\delta})$ time, for any constant $\delta > 0$ [17, 35]. Many problems are known to have conditional lower bounds from the MinConv hypothesis, see, for example, [8, 13, 17, 29, 35, 36].

The trivial $O(n^2)$ running time can be improved to $n^2/2^{\Omega(\sqrt{\log n})}$ using a reduction to the (min, +)-matrix product, due to Bremner et al. [12], and using the Williams' algorithm for all the pairs shortest path (APSP) problem [51], which was derandomized later by Chan and Williams [15]. However, the $O(n^2)$-time barrier can be beaten for different special cases. Let $b_i = f(i)$, for some function $f : [0, m) \to \mathbb{R}$. For some f, the computational complexity of MinConv can be significantly reduced. In this paper, we consider the following cases:

(i) the linear case, when $f(i) = \alpha \cdot i + \beta$;
(ii) the convex case, when $f(i+1) - f(i) \geq f(i) - f(i-1)$;
(iii) the concave case, when $f(i+1) - f(i) \leq f(i) - f(i-1)$;
(iv) the polynomial case,[1] when $f(x) \in \mathbb{Z}^d[x]$, for some fixed d;
(v) the piece-wise linear case, when $f(x)$ is represented by a piece-wise linear function with p pieces;
(vi) the monotone case, when a and b are both monotone sequences and $|a_i| = O(n), |b_i| = O(n)$.

[1] Actually, a more general class of rational functions $f(x)/g(x)$, where $f, g \in \mathbb{Z}^d[x]$, could be considered by the cost of using $2d$ instead of d in the complexity bound. But for the sake of simplicity, this case is not considered. More generally, our approach is applicable to any functions $f(x)$ with a fixed number of inflection points.

Table 1 Complexity bounds for (MinConv)

Case	Time	Reference
General	$n^2/2^{\Omega(\sqrt{\log n})}$	Bremner et al. [12] & Williams [51]
		See also [15]
Monotone	$\tilde{O}(n^{1.5})$	Shucheng et al. [16]
		See also [14]
f is linear	$O(n)$	Folklore
		Another variant due to Pferschy [43]
f is convex	$O(n)$	Axiotis & Tzamos [7] (2019)
	$O(n \cdot \log(n))$	Kellerer and Pferschy [31] (2004)
f is concave	$O(n^{4/3} \cdot \log^2(n))$	This work
f is piece-wise linear	$O(p \cdot n \cdot \log(n))$	This work
		p is a number of pieces
$f \in \mathbb{Z}^d[x]$	$O\left(d^3 \cdot n^{1+\frac{\sigma-1}{\sigma}} \cdot \log^2(n)\right)$	This work
		$\sigma = \log_2(d) + \frac{1}{1+\log_2(d)}$
Examples:		
$f \in \mathbb{Z}^2[x]$	$\tilde{O}(n^{4/3})$	
$f \in \mathbb{Z}^3[x]$	$\tilde{O}(n^{1.493})$	
$f \in \mathbb{Z}^4[x]$	$\tilde{O}(n^{11/7})$	

Definition 1 For the second and third cases, we assume that f is defined by *the evaluation oracle*, which is denoted by *EV* in our paper. Given f and $x \in \mathrm{dom}\,(f)$, this oracle returns (if it is possible) the value of $f(x)$. For the sake of simplicity, we assume that *EV* can also compare the values of given f in pairs of different points $x, y \in \mathrm{dom}\,(f)$.

To the best of our knowledge, the third, fourth, and fifth cases are new, and we develop the first sub-quadratic algorithms for them in the current paper. For the third and fifth cases, we simultaneously estimate the oracle and arithmetic complexities. In the following Table 1, we emphasize the best known complexity bounds for the cases, mentioned above. Details and the space complexity bounds could be found in Sect. 3.2 and in Theorems 2, 3 of Sect. 3.3.

Despite that the linear case is a special variant of the convex case, we make a separate category for it, because of the beautiful folklore linear-time solution. This solution is based on the folklore queue data structure that additionally supports queries to a minimal element and contains a really low constant term, which is hiding inside the O-notation. Since we did not find a description of this solution in literature, we will give it in Sect. 3.1. Probably, an $O(n)$-time solution for the MinConv linear case was firstly implicitly presented by Pferschy in [43], where an $O(n \cdot W)$ -time dynamic programming algorithm (DP-algorithm) for the bounded knapsack problem was presented. But, since the description of a knapsack DP-algorithm from [43] is quite complex, it is hard to extract an algorithm for MinConv from it, and the folklore solution looks more natural and effective.

1.2 The nonlinear knapsack problem

Let $w \in \mathbb{Z}^n_{>0}$, $W \in \mathbb{Z}_{>0}$, $u \in \mathbb{Z}^n_{>0}$, and $f : \mathbb{Z}^n \to \mathbb{R}$ be *a separable function*. That is $f(x) = f_1(x_1) + \cdots + f_n(x_n)$, where $f_i(x) : \mathbb{Z} \to \mathbb{R}$. Let us consider *the bounded knapsack problem with a general separable objective function*, defined as follows:

$$f(x) \to \max$$

$$\begin{cases} w^\top x = W \\ 0 \le x \le u \\ x \in \mathbb{Z}^n. \end{cases} \tag{KNAP-GEN}$$

We are interested in the following special cases for $f(x)$ that define the corresponding problems:

$$f(x) = c^\top x, \quad \text{for some} \quad c \in \mathbb{Z}^n_{>0}; \tag{KNAP-LIN}$$

$$f_i(x) \text{ are all convex functions;} \tag{KNAP-CONV}$$

$$f_i(x) \text{ are all concave functions;} \tag{KNAP-CONC}$$

$$f_i(x) \text{ are all piece-wise linear, with } p \text{ pieces;} \tag{KNAP-PLIN}$$

$$f_i(x) \in \mathbb{Z}^d[x], \quad \text{for some fixed } d. \tag{KNAP-POLY}$$

Note that any algorithm that solves a variant of the bounded knapsack problem also can be used to solve the unbounded one. Additionally, note that KNAP-POLY is a natural generalization of KNAP-LIN, since KNAP-POLY with $d = 1$ is equivalent to KNAP-LIN.

1.2.1 Some results about KNAP-LIN

Denote $w_{\max} = \|w\|_\infty$, $c_{\max} = \|c\|_\infty$, and $u_{\max} = \|u\|_\infty$. The paper [37] gives a reduction of KNAP-LIN to $\{0,1\}$-knapsack of $O\big(n \cdot \log(u_{\max})\big)$ weights, bounded by $O(u_{\max} \cdot w_{\max})$. Together with the basic dynamic programming technique, due to Bellman [10], it gives $\tilde{O}(n \cdot W)$-time algorithm for KNAP-LIN. The paper [43] removes the logarithmic term and gives an $O(n \cdot W)$-time algorithm. The linear-time algorithm for the monotone convex $(\min, +)$-convolution, due to [7], together with the principle to put equivalent-weight items into buckets, due to [31] (see also [7, 45]), reduces the running time to $O(n + m \cdot W)$, where m is the number of unique weights. Since $m \le \min\{w_{\max}, n\}$, it gives an $O(n + w_{\max} \cdot W)$-time algorithm.

The paper [21] introduces an elegant proximity argument and uses it to give an $\tilde{O}(n \cdot w^2_{\max})$-time algorithm. The work [23] combines the above proximity

Table 2 Complexity bounds for KNAP-LIN with respect to W and $w_{\max}$

Time	Reference
$\tilde{O}(n \cdot W)$	Bellman [10] & Lawler [37]
$O(n \cdot W)$	Pferschy [43]
$O(n + m \cdot W) = O(n + w_{\max} \cdot W)$	Axiotis & Tzamos [7]
	See also [31] and [45]
$\tilde{O}(n \cdot w_{\max}^2)$	Eisenbrand & Weismantel [21]
$O(n \cdot w_{\max}^2)$	Gribanov [23]
	See also [25]
$O(n + m \cdot w_{\max}^2) = O(n + w_{\max}^3)$	Polak, Rohwedder & Węgrzycki [45]
	$m \leq \min\{n, w_{\max}\}$ is a number of unique weights

argument with the folklore linear (min, +)-convolution algorithm and reduces logarithmic factors in the last complexity bound to give an $O(n \cdot w_{\max}^2)$-time algorithm. The same algorithm is presented in [25] for more general class of problems, which contains Δ-modular simplicies and closed polyhedra. Finally, the paper [45] carefully combines ideas of a part of the previous papers to give the state of the art $O(n + m \cdot w_{\max}^2)$-time algorithm for KNAP-LIN. The state of the art algorithms with different parametrizations by $c_{\max}$ are given in [9, 45]. The following Table 2 gives some comparison of the above results.

1.2.2 Nonlinear separable objective function

Many tricks, developed for KNAP-LIN, do not work in this case. To the best of our knowledge, the best known algorithm, parameterized by W or $w_{\max}$ that we can apply for the problem KNAP-GEN, is a straightforward application of the Bellman's DP-principle [10] that gives an $O(n \cdot W^2)$-time algorithm. A straightforward application of the linear-time monotone convex (min, +)-convolution algorithm, due to [7], gives an $O(n \cdot W)$-time algorithm for KNAP-CONV. We can use different variants of the (min, +)-convolution, considered in our paper, to construct pseudopolynomial algorithms for the problems KNAP-CONC, KNAP-PLIN, and KNAP-POLY. Related results are given in the following Table 3 (details could be found in Theorem 4):

1.3 The shortest and closest vector problems

Let $A \in \mathbb{Z}^{n \times n}$, $\Delta = |\det(A)| > 0$, $q \in \mathbb{Q}^n$, and $\Lambda(A) = \{At : t \in \mathbb{Z}^n\}$. Clearly, $\Lambda(A)$ is a full rank integer lattice with determinant Δ. *The shortest lattice vector problem* and the *the closest lattice vector problem* with respect to the l_p-norm can be formulated as follows:

$$\min\left\{\|x\|_p : x \in \Lambda(A) \setminus \{\mathbf{0}\}\right\}, \tag{SVP}$$

Table 3 Complexity bounds for KNAP-LIN, the nonlinear cases

Case	Time	Reference
KNAP-GEN	$O(n \cdot W^2)$	Bellman [10]
KNAP-CONV	$O(n \cdot W)$	Axiotis & Tzamos [7]
KNAP-CONC	$\tilde{O}(n \cdot W^{4/3})$	This work
KNAP-PLIN	$\tilde{O}(p \cdot n \cdot W)$	This work
KNAP-POLY	$\tilde{O}(d^3 \cdot n \cdot W^{1+\frac{\sigma-1}{\sigma}})$	This work
	$\sigma = \log_2(d) + \frac{1}{1+\log_2(d)}$	
Examples:		
$d = 2$	$\tilde{O}(n \cdot W^{4/3})$	
$d = 3$	$\tilde{O}(n \cdot W^{1.493})$	
$d = 4$	$\tilde{O}(n \cdot W^{11/7})$	

$$\min\left\{\|x - q\|_p \,:\, x \in \Lambda(A)\right\}. \tag{CVP}$$

In our paper, we consider only exact algorithms for SVP and CVP with theoretically provable complexity bounds. So, we avoid many works about approximate solutions or efficient practical algorithms. For a recent survey, see [52], see also [26]. Exact algorithms for SVP and CVP have a rich history. The first direction of enumeration-based algorithms dates back to the papers of Pohst [44], Kannan [30], and Fincke & Pohst [22]. The Kannan's paper [30] gives an $n^{O(n)}$-time algorithm for SVP and CVP, and many others improved upon his technique to achieve better running times [22, 27, 28, 41]. An important feature of these algorithms is that they are of polynomial space. To the best of our knowledge, the state of the art complexity bound $n^{\frac{n}{2e}+o(n)}$ for SVP via enumeration-based approach is given in [41], due to Micciancio & Walter.

Another direction is sieving-based algorithms for SVP. It is dated to the seminal paper of Ajtai et al. [3]. The algorithms from this class have better theoretical running time $2^{O(n)}$ with respect to enumeration-base algorithms, but use exponential $2^{O(n)}$ space. Many extensions and improvements of sieving technique have been proposed in [1, 4, 6, 11, 26, 38, 39, 42, 46]. The paper [1], due to Aggarwal, Dadush & Regev, gives the state of the art $2^{n+o(n)}$-complexity bound. In fact, this paper solves the more general *Discrete Gaussian Sampling* (DGS) problem. Note that above papers do not give single-exponential $2^{O(n)}$-time algorithms for CVP. The first paper that generalizes the sieving approach to solve CVP in single-exponential time is due to Aggarwal, Dadush & Stephens-Davidowitz [2]. This paper extends the DGS sampling technique from [1] and solves CVP by an $2^{n+o(n)}$-time algorithm.

The last direction that we want to refer concerns algorithms, which use *the Voronoi cell of a lattice* – the centrally symmetric polytope, corresponding to the points closer to the origin than to any other lattice point. This direction is started from the paper [48], due to Sommer, Feder & Shalvi. The seminal work of Micciancio & Voulgaris [40], which is built upon the approach of [48], gives first known

deterministic single exponential time algorithms for SVP and CVP. More precisely, it gives $2^{2n+o(n)}$-time algorithms. The space usage is $2^{n+o(n)}$.

The existence of $2^{O(n)}$-time polynomial-space exact algorithms for SVP or CVP is the major open problem in the lattice algorithms field. The works, mentioned above, are mainly concerned with the Euclidean norm $\|\cdot\|_2$. Some results about SVP-solvers for other norms are presented, for example, in [11, 18–20]. The paper [20] (see also the monograph [18]) presents a general technique to extend any Euclidean norm solvers to arbitrary norms with an additional $2^{O(n)}$-time multiplicative factor.

Now, let us discuss our motivation. All the algorithms, mentioned above, are *fixed polynomial tractable* (FPT) with respect to the dimension n parameter. In other words, a complexity bound of any of the algorithms above looks like $f(n) \cdot \mathrm{poly}(\mathrm{size}(A, q))$, where $f(n)$ is a computable function, depending only on n, and $\mathrm{size}(A, q)$ is the input encoding size. Is it possible to choose another parameterization? For example, could we build an algorithm, parameterized by the lattice determinant Δ? The paper [24] answers positively and gives an $O\big(n^\omega \cdot \log(\Delta) + n \cdot \Delta^2 \cdot \log(\Delta)\big)$-time dynamic programming algorithm for both SVP and CVP with respect to any $\|\cdot\|_p$, for $p \geq 1$, where w is the matrix multiplication exponent. In our work, we improve this running time to $O(n^\omega \cdot \log(\Delta) + m \cdot \Delta \cdot \log(\Delta))$, where $m = \min\{n, \Delta\}$. The improvement consists just in careful using of the linear-time monotone convex (min, +)-convolution algorithm, due to [7]. Our algorithm uses $O(\Delta)$ space.

Strictly speaking, we solve the following slightly more general problems than SVP and CVP. Let $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ be a monotone and convex function. We define *the generalized shortest lattice vector problem* in the following way:

$$\min\left\{\sum_{i=1}^{n} f\big(|x_i|\big) \,:\, x \in \Lambda(A) \setminus \{\mathbf{0}\}\right\}. \tag{GENERALIZED-SVP}$$

Clearly, the original SVP problem is equivalent to GENERALIZED-SVP with $f(x) = x^p$. We also define *the generalized closest vector problem* in the following way:

$$\min\left\{\sum_{i=1}^{n} f\big(|x_i - q_i|\big) \,:\, x \in \Lambda(A)\right\}. \tag{GENERALIZED-CVP}$$

Again, the original CVP problem is equivalent to GENERALIZED-CVP with $f(x) = x^p$.

In the following Table 4, we group the state of the art results for different cases, mentioned above, together with our new result. All the algorithms from the table below are deterministic, except the sieving-based algorithm. Details could be found in Theorems 5 and 6.

Remark 1 Strictly speaking, the algorithms, presented in Theorems 5 and 6, use of $O(n \cdot \Delta)$ space. But, they can be transformed to $O(\Delta)$-space algorithms without

Table 4 Complexity bounds for SVP and CVP

Technique	Time	Space	Reference
Enumeration	$n^{\frac{n}{2e}+o(n)}$	$n^{O(1)}$	Micciancio and Walter [41]
Sieving	$2^{n+o(n)}$	$2^{n+o(n)}$	Aggarwal et al. [1, 2]
Voronoi cell	$2^{2n+o(n)}$	$2^{n+o(n)}$	Micciancio and Voulgaris [40]
DP	$\tilde{O}(n^{\omega}+n\cdot\Delta^2)$	$O(n+\Delta)$	Gribanov et al. [24]
DP	$\tilde{O}(n^{\omega}+m\cdot\Delta)$	$O(n+\Delta)$	This work $m=\min\{n,\Delta\}$

significant effort. Definitely, our algorithms use dynamic tables with $O(n\cdot\Delta)$ entries. It can be easily seen that if we want only to compute the optimal value of the objective function, then it is sufficient to store only one row of these tables at each computational step, which reduces the space requirement to $O(\Delta)$. However, if we want to compute an optimal solution vector, then this simple trick is not applicable and more sophisticated technique need to be used. Such a technique is described, for example, in [32, Paragraph 3.3] (see also [43]), and it can be applied for our dynamic programming algorithms without any restrictions to produce space complexity bounds of the type $O(n+\Delta)$. The same trick works for knapsack space complexity bounds from Table 3, i.e. bounds of the type $O\left(n\cdot W^{O(1)}\right)$ can be replaced by $O\left(n+W^{O(1)}\right)$ bounds.

2 Data structures

In this Section, we describe data structures that will be used for our $(\min,+)$-convolution algorithms. The first two of them, the *queue with minimum operation* and the *compressed segment tree*, are classical. The third data structure is our modification of the segment tree, we call it the *augmented segment tree*.

2.1 Queue with minimum support

The queue with minimum support is a classical data structure that looks to be folklore. We did not able to find any correct historical references on it. This data structure just represents a generic queue that stores elements of some linearly ordered set, but with an additional operation *min*() that returns the current minimum. Let Q be an instance of the queue, we list all the operations with their complexities:

(i) the operation $Q.min()$ returns a current minimum in Q. Its complexity is $O(1)$ in the worst case;
(ii) the operation $Q.push(x)$ inserts an element x to the tail of Q. Its complexity is $O(1)$ in the worst case;
(iii) the operation $Q.pop()$ removes an element from the head of Q. Its complexity is $O(1)$ amortised.

Since we are not able to give a correct reference to this data structure, we give a brief explanation of *how it works*. First of all, note that it is easy to implement a stack with minimum support and with the worst-case complexity $O(1)$, for all the operations. To do this, we just need to create a second stack, which will store the current minimum.

A queue Q with minimum support can be implemented just by using two stacks S_h and S_t that are glued by the bottom side. The stack S_t represents a tail of Q, and the stack S_h represents a head of Q. Now, the operation $Q.min()$ can be implemented just by taking the minimum value between $S_t.min()$ and $S_h.min()$. When we need to insert a new element x to Q, we just need to call $S_t.push(x)$. Finally, the operation $Q.pop()$ can be implemented in the following way: if S_h is not empty, we just call $S_h.pop()$. If S_h is empty, we move all the elements from S_t to S_h and call $S_h.pop()$ after that. Clearly, the worst case complexity of $Q.pop()$ is $O(n)$. But, since any element of Q can be moved from S_t to S_h only ones, the amortised complexity of $Q.pop()$ is $O(1)$.

2.2 Segment tree

The segment tree is a classical data structure to perform the range minimum, the sum or update queries in sub-intervals of a given array, using only logarithmic worst-case time. We did not find any correct historical references on it, but a detailed description could be found in the internet [5]. The brief description of the weaker version without range update operations could be found in [34, Section A.3]. Additionally, the work [34] gives a good survey and interesting new results about queries on arbitrary semigroups.

Let T be an instance of a segment tree. We list the required operations with their complexities:

(i) the operation $T.build(A)$ builds the data structure on an array A of length n. Its worst-case complexity is $O(n)$;
(ii) the operation $T.min(i, j)$ returns a minimal element in the sub-array $A[i, j)$. Its worst-case complexity is $O(\log(n))$;
(iii) the operation $T.add(i, j, x)$ adds the value of x to all the elements of the sub-array $A[i, j)$. Its worst-case complexity is $O(\log(n))$.

Let us assume that $n = 2^d$. We call an interval $[i, j)$ *basic*, if $[i,j) = [i \cdot 2^{d-k}, (i+1) \cdot 2^{d-k})$, for some $i \in \{0, \dots, 2^k - 1\}$ and $k \in \{0, \dots, d\}$. The segment tree is represented as a full binary tree, where each node v corresponds to some basic interval $[i_v,j_v)$ and additionally stores a minimum element in the sub-array $A[i_v,j_v)$. If v is not leaf, then it has two children a and b, corresponding to the intervals $[i_a,j_a) = [i_v, i_v + h)$ and $[i_b,j_b) = [i_v + h,j_v)$, where $h = (j_v - i_v)/2$. The leafs correspond to intervals of length 1, associated with all the elements of A. If v is a root node, we just have $[i_v,j_v) = [0, 2^d)$ and the minimum value in v corresponds to the minimum in A.

The key idea that helps to compute the minimum value in a general interval $[i, j)$ is a special algorithm that splits a given interval $[i, j)$ into at most $2d$ basic intervals. We emphasise this in the following lemma, which will be used later:

Lemma 1 *For any given interval $[i, j)$, there is an $O(d)$-complexity algorithm that splits $[i, j)$ into at most $2d$ basic intervals, corresponding to the nodes of T.*

2.3 Augmented segment tree

Assume again that n is a power of 2. Let A be an array of length n, and let $f(x) : [0, n) \to \mathbb{R}$ be a function. Given $x \in \{0, \dots, n-1\}$ and $i, j \in \{0, \dots, n\}$, we are interested in the problem to efficiently compute a function

$$\begin{aligned}&\mathcal{F}([i,j);x)\\ &\quad= \min\big\{A[i]+f(x),\ A[i+1]+f(x+1),\ \dots,\ A\big[i+(j-i-1)\big]\\ &\qquad\qquad +f(x+(j-i-1))\big\} = \min_{0\le k<j-i}\{A[i+k]+f(x+k)\},\end{aligned}$$

assuming that some preprocessing is done for A. To compute the values of f for $x \in [0, n)$, the *EV*-oracle can be used (see Definition 1). The following main property of the function $\mathcal{F}([i, j); x)$ can be checked straightforwardly:

Proposition 1 *Let an interval $[i, j)$ be partitioned into the intervals $[i_1, j_1)$ and $[i_2, j_2)$, i.e. $[i, j) = [i_1, j_1) \cup [i_2, j_2)$, where $i_2 \ge i_1$. Then,*

$$\mathcal{F}([i,j);x) = \min\big\{\mathcal{F}\big([i_1,j_1);x\big),\ \mathcal{F}\big([i_2,j_2);x+j_1-i_1\big)\big\}. \tag{1}$$

Definition 2 The interval $[a, b) \subseteq \mathbb{R}$ is called *integer* if its end-points a and b are integers.

Definition 3 Let $f : [0, n) \to \mathbb{R}$ and $\mathcal{I} \subseteq [0, n)$ be an integer interval. Let $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_m$ be a set of integer intervals, such that

(i) the intervals $\mathcal{I}_1, \dots, \mathcal{I}_m$ partition $\mathcal{I}$;
(ii) for any j, we have either $f(x) \ge 0$ or $f(x) \le 0$ for all points inside $\mathcal{I}_j$.

A minimal set of such intervals $\mathcal{I}_1, \dots, \mathcal{I}_m$ is called a *minimal sign partition of f*. The set of all such minimal partitions is denoted by $\mathscr{P}(f, \mathcal{I})$.

Definition 4 By $\mathcal{SP}$ we denote the *minimal sign partition oracle*. For given f and $\mathcal{I} \subseteq \operatorname{dom}(f)$, it returns some minimal sign partition from $\mathscr{P}(f, \mathcal{I})$.

Next, we need to define a special characteristic p_f of a function $f : \mathcal{I} \to \mathbb{R}$, defined on an integer interval $\mathcal{I}$, which will be extensively used further.

Definition 5 Let $f : \mathcal{I} \to \mathbb{R}$ be a function, defined on an integer interval $\mathcal{I}$. Let us define a value p_f in the following way. For $a \in \mathbb{Z}_{\geq 0}$ and $b \in \mathbb{Z}$, let us consider a function $g_{ab}(x) = f(x + a) - f(x) + b$. Let

$$p_f(a, b, \mathcal{J}) \text{ be the size of some minimal sign partition of } g_{ab} \text{ on } \mathcal{J}, \text{ and}$$
$$p_f = \max\left\{p_f(a, b, \mathcal{J}) : a \in \mathbb{R}_{\geq 0},\ b \in \mathbb{R},\ \mathcal{J} \subseteq \operatorname{dom}(g_{ab})\right\}$$

In other words, p_f is the maximal size of a minimal sign partition that $f(x + a) - f(x) + b$ can have on $\mathcal{J}$, for the all possible values of a, b and correct integer sub-intervals $\mathcal{J} \subseteq \mathcal{I}$.

The following theorem defines the augmented segment tree data structure:

Theorem 1 *Assume that EV and SP oracles are available. Let $f : [0, n) \to \mathbb{R}$ be a function, A be an array of length n, which is a power of 2, and $p := p_f$. There exists a data structure T, called the augmented segment tree, that supports the following list of operations*:

(i) *the operation $T.build(A)$ builds the data structure for the array A of length n. The worst-case SP-oracle and arithmetic complexities are $O(n^{\log_2(p)+\frac{1}{1+\log_2(p)}})$;*
(ii) *the operation $T.query(i, j, x)$ returns the value of $\mathcal{F}([i, j); x)$. The worst-case EV-oracle and arithmetic complexities are $O(\log^2(n))$.*

The data structure uses $O(n^{\log_2(p)+\frac{1}{1+\log_2(p)}})$ space. Calls to *SP* oracle are performed for functions of the type $g(x) = f(x + a) - f(x + b) + c$, where $a, b \in \mathbb{Z}_{\geq 0}$, and $c \in \mathbb{Z}$. Calls to *EV* are performed for f.

The theorem proof is moved into Appendix Sect. 6.1.

3 Structured (min, +)-convolution algorithms

In this Section, we describe how to solve the problem ReducedMinConv for the linear ($f(x) = \alpha \cdot x + \beta$), piece-wise linear ($f(x)$ is represented by a piece-wise linear function with p pieces), polynomial ($f \in \mathbb{Z}^d[x]$) and concave cases.

3.1 The linear case

W.l.o.g. we can assume that $b_i = \alpha \cdot i$, for some $\alpha \in \mathbb{Q}$. We will use a queue Q, which was described in Sect. 2.1. The algorithm consists of $m - n$ steps: at the first step, we just initialise Q with the elements

$$a_0 + b_0,\ a_1 + b_1,\ \ldots,\ a_{m-1} + b_{m-1},$$

which can be done in $O(m)$-time. After that, we assign $c_0 := Q.min()$, which can be done in $O(1)$-time. Note that the difference between elements in Q is exactly α. We will support the following invariant:

after the k-th step the queue Q contains the following elements: $a_k + \alpha \cdot k,\ a_{k+1} + \alpha \cdot (k+1),\ \dots,\ a_{k+m-1} + \alpha \cdot (k+m-1)$.

Assuming that the k-th step has been done and c_k has been computed, let us show how to perform the $(k+1)$-th step. We call the $Q.pop()$ and after that call $Q.push(x)$, for $x = a_{k+m} + \alpha \cdot (k+m)$. The last operations will satisfy the invariant at the $(k+1)$ -th step. Now, we can put $c_{k+1} := Q.min() - \alpha \cdot (k+1)$, due to the invariant, it is the correct value of c_{k+1}. Since the amortised complexity of each step is $O(1)$, the total arithmetical complexity bound is $O(m + (n-m)) = O(n)$.

3.2 The piece-wise linear case

W.l.o.g. we can assume that $f(x)$ is defined on $[0, m)$ by the following three vectors: $\alpha, \beta \in \mathbb{Q}^p$, and $u \in \mathbb{Z}^{p+1}_{\geq 0}$. We assume, that $u_0 = 0$, $u_p = n$, and $u_{j-1} < u_j$, for $j \in \{1, \dots, p\}$. The formula for f is:

$$f(x) = \beta_k + \alpha_k \cdot x, \quad \text{for } x \in [u_{k-1}, u_k).$$

Assuming $b_i = f(i)$, let us show how to compute the elements of c. We will use the compressed segment tree data structure T, described in Sect. 2.2. The algorithm consists of $n - m$ steps: at the first step, we construct the array $A := a$ and assign $A[i] := A[i] + b_i$, for all $i \in \{0, \dots, m-1\}$. It takes $O(n)$ arithmetic operations. Next, we initialise T, by calling $T.build(A)$. It also takes $O(n)$-time. We will support the following invariant:

after the k-th step has been done, the sub-array $A[k, k+m)$ consists of the elements: $a_k + b_0,\ a_{k+1} + b_1,\ \dots,\ a_{k+m-1} + b_{m-1}$.

Consequently, the equality $T.query(k, k+m) = c_k$ holds after the k-th step has been finished.

Now, let us show how to perform the $(k+1)$-th step with the complexity $O(p \cdot \log(n))$. Fix a number $j \in \{1, \dots, p\}$ and consider the sub-array $A[k+1+u_{j-1}, k+1+u_j)$. Let $d_j = u_j - u_{j-1}$. By the invariant, the first $d_j - 1$ elements of this array are equal to

$$a_{k+1+u_{j-1}} + \beta_j + \alpha_j \cdot u_{j-1},\ \dots,\ a_{k+u_j-1} + \beta_j + (u_j - 1) \cdot \alpha_j.$$

The last element $A[k+u_j]$ is equal to $a_{k+u_j} + f(u_j) = a_{k+u_j} + \beta_{j+1} + \alpha_{j+1} \cdot u_j$. Consequently, to make the first $d_j - 1$ elements of the sub-array to satisfy the invariant, we need to make the $update(k+1+u_{j-1}, k+u_j, -\alpha_j)$ operation, which can be done in $O(\log(n))$-time. Since $k + u_j = (k+1) + (u_j - 1)$, the last element $A[k+u_j]$ must be assigned to $a_{k+u_j} + f(u_j - 1)$, which can also be done in $O(\log(n))$-time. After apply-

ing this procedure for all $k \in \{1, \dots, p\}$, the invariant for the $k+1$-th step will be satisfied, and the value of c_{k+1} can be computed just by using the $query(k+1, k+1+m)$ operation. The complexity of the considered step is $O(p \cdot \log(n))$. The total arithmetic complexity of the algorithm is $O(n + (n-m) \cdot p \cdot \log(n)) = O(p \cdot n \cdot \log(n))$. Note that the segment tree data structure needs additional $O(n)$ space.

3.3 The polynomial and concave cases

Consider a function $f : [0, m) \to \mathbb{R}$ and assume that *EV* and *SP* oracles are supported. Let us estimate the oracle and arithmetic complexities to solve Reduced-MinConv with $b_i = f(i)$. We will use the augmented segment tree data structure, described in Sect. 2.3. Let us assume that n be a power of 2 and choose a block size B, which is also be a power of 2.

We assign $A := a$ and split A into n/B consecutive blocks of size B. Let $\mathcal{B}_j$ be the interval representing the j-th block, i.e. $\mathcal{B}_j = [(j-1) \cdot B, j \cdot B)$. Now, for each $j \in \{1, \dots, n/B\}$, we construct the augmented segment tree data structure T_j by calling the operation $T_j.build(A[\mathcal{B}_j])$. Due to Theorem 1, the oracle, arithmetic, and space complexities of this step can be expressed by the formula

$$O\left(\frac{n}{B} \cdot B^{\log_2(p) + \frac{1}{1+\log_2(p)}}\right) = O\left(n \cdot B^{\sigma-1}\right), \tag{2}$$

where $\sigma = \log_2(p) + \frac{1}{1+\log_2(p)}$.

The algorithm consist of $n-m$ steps. At the k-th step we try to compute the value of c_k, using the hint that $c_k = \mathcal{F}([k, k+m); 0)$. Let $\mathcal{B}_j, \dots, \mathcal{B}_{j+t}$ be the consecutive blocks, affected by the window $[k, k+m)$, where $t = O(m/B)$. If $t = 1$, then we can just put $c_k := T_j.query(\nu, \tau, 0)$, where $\nu = k - B \cdot (j-1)$ and $\tau = k - B \cdot (j-1) + m$ are the relative coordinates of the $[k, k+m)$ window in $\mathcal{B}_j$. Now, let us consider the case $t \geq 2$. Let $s = j \cdot B - k$ be the size of the intersection of $\mathcal{B}_j$ with $[k, k+m)$. Using Proposition (19), we have

$$\begin{aligned} c_k &:= \mathcal{F}([k, k+m); 0) \\ &= \min\Big\{\mathcal{F}\big(\mathcal{B}_j \cap [k, k+m); 0\big),\ \mathcal{F}\big(\mathcal{B}_{j+1}; s\big),\ \mathcal{F}\big(\mathcal{B}_{j+2}; s+B\big),\ \dots \\ &\qquad \mathcal{F}\big(\mathcal{B}_{j+t-1}; s+B\cdot(t-2)\big),\ \mathcal{F}\big(\mathcal{B}_{j+t} \cap [k, k+m); s+B\cdot(t-1)\big)\Big\} \end{aligned}$$

The interval $\mathcal{B}_j \cap [k, k+m)$ is a suffix of $\mathcal{B}_j$. So, the first value

$$\mathcal{F}\big(\mathcal{B}_j \cap [k, k+m); 0\big)$$

can be computed by a call to $T_j.query(B-s,\ B,\ 0)$. The interval $\mathcal{B}_{j+t} \cap [k, k+m)$ is a prefix of $\mathcal{B}_{j+t}$. So, the last value

$$\mathcal{F}\big(\mathcal{B}_{j+t} \cap [k, k+m); s+B\cdot(t-1)\big)$$

can be computed by a call to

$$T_{j+t}.query(0,\, k+m-B\cdot(t-1),\, s+B\cdot(t-1)).$$

Here $k+m-B\cdot(t-1)$ is the size of $\mathcal{B}_{j+t}\cap[k,k+m)$. The intermediate values $\mathcal{F}\big(\mathcal{B}_{j+i};s+B\cdot(i-1)\big)$, for $i\in\{1,\dots,t-1\}$, can be computed by calls to $T_{j+i}.query(0,\, B,\, s+B\cdot(i-1))$.

Therefore, c_k can be computed as the minimum between all this values. The respective arithmetic and oracle complexity is expressed by $O\Big(\frac{m}{B}\cdot\log^2(B)\Big)$. Consequently, the total complexity of $m-n$ steps without the initial preprocessing can be estimated as

$$O\Big((n-m)\cdot\frac{m}{B}\cdot\log^2(m)\Big)=O\Big(\frac{n^2}{B}\cdot\log^2(n)\Big).$$

Now, the total algorithm complexity (together with the preprocessing, see the formula (2)) can be expressed by the formula

$$O\Big(n\cdot B^{\sigma-1}+\frac{n^2}{B}\cdot\log^2(n)\Big).$$

Actually, a more detailed formula holds

$$O\Big(n\cdot B^{\sigma-1}\cdot T_{SP}+\frac{n^2}{B}\cdot\log^2(n)\cdot T_{EV}\Big). \tag{3}$$

We will try to balance this formula, solving the equation

$$n\cdot B^{\sigma-1}=\frac{n^2}{B}\quad\Longrightarrow\quad B^{\sigma}=n.$$

So, the B parameter could be chosen as $B=n^{\frac{1}{\sigma}}$. After this substitution, the total time and space complexities become

$$O\Big(\big(T_{SP}+\log^2(n)\cdot T_{EV}\big)\cdot n^{1+\frac{\sigma-1}{\sigma}}\Big),\quad\text{and}\quad O\Big(n^{1+\frac{\sigma-1}{\sigma}}\Big). \tag{4}$$

Now, let us consider the polynomial case.

Theorem 2 *Let $f\in\mathbb{Z}^d[x]$. Then, ReducedMinConv can be solved by an algorithm with the arithmetic complexity bound*

$$O\Big(d^3\cdot n^{1+\frac{\sigma-1}{\sigma}}\cdot\log^2(n)\Big),\quad\text{where }\sigma=\log_2(d)+\frac{1}{1+\log_2(d)}.$$

The space complexity is $O\Big(d^2+n^{1+\frac{\sigma-1}{\sigma}}\Big)$.

Proof Clearly, the complexity of EV for f is $O(d)$. Let us estimate the complexity of SP for polynomials of the type $g(x)=f(x+a)+b-f(x)$. Since $\deg(g)\le d-1$, the size p of any minimal sign partition g is bounded by d. To calculate this partition

on a given interval $\mathcal{I}$, we clearly need to localize all the roots of g inside $\mathcal{I}$. Since intervals in the resulting sign partition need to have integer end-points, we do not need to compute the roots exactly. Instead of that, we simply can calculate them with the additive accuracy 1/3 and round-off to the nearest integer. To localize the roots of g on $\mathcal{I}$, we will use the classical Budan–Fourier theorem. It states that for any interval (ν, τ) with $g(\nu) \neq 0$ and $g(\tau) \neq 0$ the number of roots of $g(x)$ in the interval (ν, τ) is equal or less than the value of $W(\nu) - W(\tau)$, where $W(x)$ is the number of sign changes in the sequence $f(x), f'(x), f''(x), \ldots$. Note that the Budan–Fourier theorem does not calculate the exact number of roots, the real number of roots in $[\nu, \tau)$ can be less by an even number. But, in our case, we only need to know how the sign changes, when x crosses an integer landmark point. So, this method can be used without restrictions.

Clearly, the sequence of $g(x)$-derivatives can be computed, using $O(d^2)$ arithmetic operations. After that, given a point x, the value of $W(x)$ could also be computed, using $O(d^2)$ arithmetic operations. Then, using the standard dichotomy principle, we could localize all the roots of g on $\mathcal{I}$ with the additive accuracy 1/3, using $O(d \cdot \log(n))$ calls to $W(x)$. Hence, the complexity of $\mathcal{SP}$ on g is $O(d^3 \cdot \log(n))$. Let us consider the formula (4). Since $\sigma = \sigma(p)$ is a monotone function, we have $\sigma(p) \leq \sigma(d)$. Then, together with the complexity bounds for $\mathcal{SP}$ and EV, the formula (4) gives the desired complexity bound for ReducedMinConv. □

Now, we are going to consider the concave case. First of all, we need some auxiliary lemmas:

Lemma 2 *Let (x_0,f_0), (x_1,f_1), …, (x_{n-1},f_{n-1}) be a sequence of pairs from $\mathbb{R}^2$. By d_i we denote $(f_i - f_{i-1})/(x_i - x_{i-1})$. Assume that*

$$x_0 \leq x_1 \leq \cdots \leq x_{n-1}, \quad \text{and} \quad d_1 \leq \cdots \leq d_{n-1}.$$

Then, there exists a C^1-smooth convex function $f : \mathbb{R} \to \mathbb{R}$ with $f(x_i) = f_i$.

The lemma proof is moved into Appendix Sect. 6.2.

Lemma 3 *Let $f : [\alpha, \beta) \to \mathbb{R}$ be a convex/concave function ($\alpha, \beta \in \mathbb{R}$). The following statements hold*:

(i) $p_f \leq 2$;
(ii) *Assume that EV oracle is supported for f. Let $g(x) = f(x + a) + b - f(x)$, for some $a, b \in \mathbb{R}$. Then, given a bounded integer interval $\mathcal{I}$ of length n, some minimal sign partition from $\mathcal{P}(g, \mathcal{I})$ can be computed, using $O(\log(n))$ calls to EV.*

The lemma proof is moved into Appendix Sect. 6.3.

Theorem 3 *Let $f : [0, m) \to \mathbb{R}$ be a concave function. Assume that EV oracle is available. Then, ReducedMinConv can be solved by an algorithm with the*

arithmetic and oracle complexity, bounded by $O\left(n^{4/3} \cdot \log^2(n)\right)$. *The space complexity is* $O\left(n^{4/3}\right)$.

Proof Let $g(x) = f(x+a) + b - f(x)$ and $\mathcal{I} \subseteq [0, n)$. Due to Lemma 3, the complexity of *SP*-oracle with the input pair $(g, \mathcal{I})$ is bounded by $O(\log(n))$ calls to *EV*. Additionally, $p := p_f \leq 2$. Let us consider the formula (4). Since $\sigma = \sigma(p)$ is a monotone function, we have $\sigma(p) \leq \sigma(2) = 3/2$. Together with the complexity bound for *SP*, the formula (4) gives the desired complexity bound for ReducedMinConv. □

4 Applications for the bounded knapsack

Let us consider KNAP-GEN. W.l.o.g. we can assume that $u_k \leq \lfloor W/w_k \rfloor$, for $k \in \{1, \dots, n\}$. Additionally, we consider the minimization problem instead of maximization, since we can work with $-f$ instead f, which preserves the separability property. Let us consider a very basic dynamic program, dated to Bellman [10]. For $k \in \{1, \dots, n\}$ and $w_0 \in \{0, \dots, W\}$, by $DP(k, w_0)$ we denote the following problem:

$$\sum_{i=1}^{k} f_i(x_i) \to \min \qquad \begin{cases} \sum\limits_{i=1}^{k} w_i x_i = w_0 \\ 0 \leq x_i \leq u_i \\ x \in \mathbb{Z}^k. \end{cases} \tag{5}$$

Clearly, the problem $DP(n, W)$ is equivalent to the original problem KNAP-GEN. For $2 \leq k \leq n$, the value $DP(k, w_0)$ can be computed, using the values $DP(k-1, \cdot)$ by the following formula:

$$DP(k, w_0) = \min_{j \in \{0, \dots, u_k\}} \left\{ DP(k-1, w_0 - w_k \cdot j) + f_k(j) \right\}, \tag{6}$$

assuming that $DP(k, w_0) = +\infty$ for $w_0 < 0$. For $k = 1$, the formula is

$$DP(1, w_0) = \begin{cases} f_1(w_0/w_1) & \text{if } w_1 \mid w_0 \text{ and } 0 \leq w_0/w_1 \leq u_1 \\ +\infty & \text{, in the opposite case.} \end{cases} \tag{7}$$

Let us estimate the complexity to compute all the values $DP(k, w_0)$, for $k \in \{1, \dots, n\}$ and $w_0 \in \{0, \dots, W\}$. Clearly, the values $DP(1, w_0)$ can be computed with $O(W)$ operations. Definitely, each of the values $DP(1, 0)$, $DP(1, w_1)$, $DP(1, 2w_1)$, etc. can be computed with $O(1)$ operations, using the formula (7). For other values of w_0, we just set $DP(1, w_0) = +\infty$. The computation of $DP(k, w_0)$, for $k \geq 2$, can be reduced to $(\min, +)$-convolution. Fix $k \geq 2$ and some residue r modulo w_k. We define the sequences $\{a_i\}_{i \in \{0, \dots, u_k\}}$, $\{b_i\}_{i \in \{0, \dots, u_k\}}$, and $\{c_i\}_{i \in \{0, \dots, u_k\}}$ as follows:

$$a_i = DP(k-1, r + i \cdot w_k), \quad b_i = f_k(i), \quad c_i = DP(k, r + i \cdot w_k).$$

Assuming that $a_i = b_i = c_i = 0$, for $i < 0$, and due to (6), we have

$$c_i = \min_{j\in\{0,\dots,i\}} \{a_{i-j} + b_j\}. \tag{8}$$

That gives $c = (a \star b)[0, u_k]$. Therefore, considering all the values of r, the complexity to compute the level $DP(k, \cdot)$, in the assumption that the level $DP(k-1, \cdot)$ has already been computed, can be expressed by $O\big(w_k \cdot T_{conv}(u_k)\big)$, where $T_{conv}(\cdot)$ denotes the complexity of the (min, +)-convolution. The total complexity of the whole dynamic programming scheme is

$$O\left(W + \sum_{k=2}^{n} w_k \cdot T_{conv}(u_k)\right). \tag{9}$$

Using the previous formula, the inequality $u_i \le \lfloor W/w_k \rfloor$, and different T_{conv}-complexity results, due to Sect. 3.2 and Theorems 2, 3 of Sect. 3.3, we obtain the following result:

Theorem 4 *The following statements hold*:

(i) *The problem KNAP-PLIN can be solved by $O(p \cdot n \cdot W \cdot \log(W))$ arithmetic complexity algorithm*;

(ii) *The problem KNAP-CONC can be solved by an algorithm with the arithmetic complexity bound*

$$O\Big(W^{4/3} \cdot \left(w_2^{-1/3} + w_3^{-1/3} + \cdots + w_n^{-1/3}\right) \cdot \log^2(W)\Big) = O\big(n \cdot W^{4/3} \cdot \log^2(W)\big);$$

(iii) *Denote $\sigma = \log_2(d) + \frac{1}{1+\log_2(d)} \ge 1$. The problem KNAP-POLY can be solved by an algorithm with the arithmetic complexity bound*

$$O\left(d^3 \cdot W^{1+\frac{\sigma-1}{\sigma}} \cdot \left(w_2^{\frac{1-\sigma}{\sigma}} + w_3^{\frac{1-\sigma}{\sigma}} + \cdots + w_n^{\frac{1-\sigma}{\sigma}}\right) \cdot \log^2(W)\right) = O\Big(d^3 \cdot n \cdot W^{1+\frac{\sigma-1}{\sigma}} \cdot \log^2(W)\Big).$$

All the computations are performed with integer numbers of the size $O(\log(W))$.

5 SVP and CVP dynamic programming algorithms

5.1 SVP problem

Let us consider the generalized problem GENERALIZED-SVP. It is a known fact (see, for example, [47, 49, 53]) that there exist unimodular matrices $P \in \mathbb{Z}^{n\times n}$ and $Q \in \mathbb{Z}^{n\times n}$, such that $PAQ = S$, where $S \in \mathbb{Z}^{n\times n}_{\ge 0}$ is a diagonal non-degenerate matrix. Moreover,

$\prod_{i=1}^{k} S_{ii} = \Delta_{\gcd}(A, k)$, for any $k \in \{1, \dots, n\}$, and, consequently, $S_{ii} \mid S_{(i+1)(i+1)}$, for $i \in \{1, \dots, n-1\}$. Here $\Delta_{\gcd}(A, k)$ denotes the greatest common divisor of $k \times k$ subdeterminants of A. The matrix S is called the *Smith Normal Form* (or, shortly, the SNF) of A. Using the SNF, we can reformulate the problem (GENERALIZED-SVP):

$$\sum_{i=1}^{n} f(x_i) \to \min \qquad \begin{cases} Px \equiv \mathbf{0} \pmod{S \cdot \mathbb{Z}^n} \\ x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}. \end{cases} \tag{10}$$

Let us consider the quotient group $\mathcal{G} = \mathbb{Z}^n / S \cdot \mathbb{Z}^n$ (with respect to addition in $\mathbb{Z}^n$), and define $g_i = P_i \bmod \operatorname{diag}(S)$, where P_i is the i-th column of P and $i \in \{1, \dots, n\}$. We identify the vectors g_i with the elements of the group $\mathcal{G}$. Clearly, $|\mathcal{G}| = \Delta$. Then, the problem (10) can be reformulated as a minimization problem on $\mathcal{G}$:

$$\sum_{i=1}^{n} f(x_i) \to \min \qquad \begin{cases} \sum\limits_{i=1}^{n} x_i \cdot g_i = 0 \\ x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}. \end{cases} \tag{11}$$

Remark 2 Note that since $|\det(S)| = \Delta$, the diagonal of S contains at most $\log_2(\Delta)$ of elements that are not equal 1. Therefore, the arithmetic complexity of one operation with elements of $\mathcal{G}$ is $O(\log(\Delta))$.

Remark 3 W.l.o.g. we can assume that $g_i \neq \pm g_j$, for different i, j. Consequently, $n \leq \Delta/2 + 1$. Definitely, if for example $g_1 = \pm g_2$, then the vector $(1, \mp 1, 0, \dots, 0)^\top \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ is a feasible solution of (11). Clearly, the only solutions, which can be better, are solutions of the type $\pm e_i$, which are be feasible only if $g_i = 0$. The duplicates and zeros inside $g_1, g_2, \dots, g_n$ can be detected by an algorithm, like the radix-sort using $O(n \cdot \Delta)$ group operations or by any comparison-based sorting using $O(n \cdot \log(n))$ group operations.

To solve the problem (11), we will use the following dynamic programming scheme. For $g_0 \in \mathcal{G}$ and $k \in \{1, \dots, n\}$, we define the problem $DP(k, g_0)$ in the following way:

$$\sum_{i=1}^{k} f_i(x_i) \to \min \qquad \begin{cases} \sum\limits_{i=1}^{k} x_i \cdot g_i = g_0 \\ x \in \mathbb{Z}^k \setminus \{\mathbf{0}\}. \end{cases} \tag{12}$$

Clearly, the problem $DP(n, 0)$ is equivalent to the problem 11. Denote

$$\begin{aligned} \psi_+(k, g_0) &= \min_{j \in \{0, \dots, \Delta\}} \left\{ DP(k-1, g_0 - j \cdot g_k) + f_k(j) \right\}, \\ \psi_-(k, g_0) &= \min_{j \in \{0, \dots, \Delta\}} \left\{ DP(k-1, g_0 + j \cdot g_k) + f_k(j) \right\}, \\ \eta(k, g_0) &= \min \left\{ f_k(j) : j \cdot g_k = g_0,\ j \in \{-\Delta, \dots, \Delta\} \setminus \{0\} \right\}. \end{aligned}$$

If the set, where we take min for $\eta(k, g_0)$, is empty, then we set $\eta(k, g_0) = +\infty$. Since f_k is monotone and even, we have $DP(1, g_0) = \eta(1, g_0)$. Similarly, for $k \geq 2$, it can be straightforwardly checked out that

$$DP(k, g_0) = \min\left\{\psi_+(k-1, g_0),\ \psi_-(k-1, g_0),\ \eta(k-1, g_0)\right\}. \tag{13}$$

Note that we can not only use the values of $\psi_+(k-1, g_0)$ and $\psi_-(k-1, g_0)$ in the previous formula, because in this case we are missing out the solutions of the type $(0, 0, \dots, 0, j)^\top \in \mathbb{Z}^k$. So, we need additionally to take into account the values of $\eta(k-1, g_0)$.

First of all, fix k and let us show how to compute the values $\eta(k, g_0)$, for all $g_0 \in \mathcal{G}$, using $O(\Delta)$ group operations. Note that $\eta(k, g_0) \neq +\infty$ if and only if $g_0 = g_k \cdot j$, for some $j \in \mathbb{Z} \setminus \{0\}$. Hence, we need to fill only the values $\eta(k, j \cdot g_k)$, for other values of g_0 we can just set $\eta(k, g_0) = +\infty$. To fill $\eta(k, j \cdot g_k)$, we can do the following:

1. Assign $\eta(k, g_0) := +\infty$, for all $g_0 \in \mathcal{G}$;
2. For $j \in \{1, \dots, \Delta - 1\}$, do the following:
3. If $\eta(k, j \cdot g_k) = +\infty$, then assign $\eta(k, j \cdot g_k) := f_k(j)$;
4. If $\eta(k, -j \cdot g_k) = +\infty$, then assign $\eta(k, -j \cdot g_k) := f_k(j)$;

To see that the algorithm is correct, assume that $j^* \in \mathbb{Z}\setminus\{0\}$ is the value such that $g_0 = j^* \cdot g_k$ and $|j^*|$ is minimal. Then, clearly $\eta(k, g_0) = f_k(j^*)$. If $j^* > 0$, we will find it during the 3-th step. If $j^* < 0$, the 4-th step will give the correct value. This value will not be rewritten, because only the values with $\eta(k, g_0) = +\infty$ could be assigned to something. Therefore, the values $\eta(k, g_0)$, for $k \in \{1, \dots, n\}$ and $g_0 \in \mathcal{G}$, can be computed using $O(n \cdot \Delta)$ group operations.

Fix $k \geq 2$. Let us estimate the complexity to compute $\psi_+(k, \cdot)$ in assumption that the layer $DP(k-1, \cdot)$ is already computed. Let us consider the quotient group $\mathcal{Q} = \mathcal{G}/\langle g_k\rangle$ and fix $\mathcal{Q} \in \mathcal{Q}$. Let $d_k = \left|\langle g_k\rangle\right|$. Clearly, $\mathcal{Q} = q + \langle g_k\rangle$, where $q \in \mathcal{G}$ is a representative of $\mathcal{Q}$, and $d_k = |\mathcal{Q}|$. Let us define the sequences $\{a_i\}_{i\in\{0,\dots,d_k-1\}}$, $\{b_i\}_{i\in\{0,\dots,d_k-1\}}$, and $\{c_i\}_{i\in\{0,\dots,d_k-1\}}$ as follows:

$$a_i = DP(k-1, q + i \cdot g_k), \quad b_i = f_k(i), \quad c_i = \psi_+(k, q + i \cdot g_k).$$

Assuming that $a_i = b_i = c_i = 0$ for $i < 0$, and due to the definition of ψ_+, we have

$$c_i = \min_{j\in\{0,\dots,i\}}\left\{a_{i-j} + b_i\right\}.$$

That gives $c = (a \star b)[0, d_k - 1]$. Therefore, considering all the cosets $\mathcal{Q} \in \mathcal{Q}$, the group operation complexity to compute $\psi_+(k, \cdot)$, in the assumption that the level $DP(k-1, \cdot)$ has already been computed, can be expressed by $O\big(d_k \cdot T_{conv}(\Delta/d_k)\big)$, where $T_{conv}(\cdot)$ denotes the complexity of the $(\min, +)$-convolution. The values of $\psi_-(k, \cdot)$ can be computed in a similar way with the same complexity bound.

Consequently, the layer $DP(k, \cdot)$, again in the assumption that the level $DP(k-1, \cdot)$ has already known, can be computed, using $O\big(d_k \cdot T_{conv}(\Delta/d_k)\big)$ group operations. The total group operations complexity is

$$O\left(n \cdot \Delta + \sum_{k=1}^{n} d_k \cdot T_{conv}(\Delta/d_k)\right).$$

Since f_k is convex, due to [7], $T_{conv}(k) = O(k)$. Consequently, the last bound becomes $O(n \cdot \Delta)$. Due to Remark 2, the arithmetic complexity of group operations is $O(\log(\Delta))$. Hence, the total arithmetic complexity to solve the problem (11) can be expressed by

$$O(n \cdot \Delta \cdot \log(\Delta)).$$

Finally, assuming that the original group problem (11) contains duplicates, we can remove them, using Remark 3 with $O(n \cdot \log(n) \cdot \log(\Delta))$ arithmetic operations. Denoting the dimension of the resulting problem by $m \leq \Delta/2 + 1$ and taking into account the SNF computational complexity, denoted by $T_{SNF}(n, \Delta)$, we get the complexity bound of the whole algorithm

$$O\big(T_{SNF}(n, \Delta) + n \cdot \log(n) \cdot \log(\Delta) + m \cdot \Delta \cdot \log(\Delta)\big).$$

Due to [49], $T_{SNF}(n, \Delta) = O(n^{\omega} \cdot \log(\Delta))$ of arithmetic operations with integers of the size $O(\log(\Delta))$. So, the following theorem has been proven.

Theorem 5 *The problem GENERALIZED-SVP can be solved by an algorithm with arithmetic complexity bound*

$$O(n^{\omega} \cdot \log(\Delta) + \min\{n, \Delta\} \cdot \Delta \cdot \log(\Delta)) = \tilde{O}(n^{\omega} + \min\{n, \Delta\} \cdot \Delta),$$

where all the computations are performed with integer numbers of the size $O(\log(\Delta))$ and ω is the matrix multiplication exponent.

5.2 CVP problem

Let us consider the generalized problem GENERALIZED-CVP. After the maps $x \to x - b$ and $q - b \to q$, where $b = -\lfloor q \rceil$, the problem GENERALIZED-CVP transforms to:

$$\min\left\{\sum_{i=1}^{n} f\big(|x_i - q_i|\big) \,:\, x \in b + \Lambda(A)\right\}, \tag{14}$$

with $\|q\|_{\infty} < 1/2$. Additionally, we can assume that $q \geq 0$, because we can map x_i to $-x_i$, for $q_i < 0$. Finally, we can sort q_i, so we have

$$1/2 > q_1 \geq q_2 \geq \cdots \geq q_n \geq 0. \tag{15}$$

Denote $f_i(x) = f\big(|x - q_i|\big)$. It is easy to check that the following properties hold for $f_i(x)$:

(i) For any $i \in \{1, \dots, n\}$, f_i is monotone on the sets $\mathbb{Z}_{\geq 0}$ and $\mathbb{Z}_{\leq 0}$ and convex on $\mathbb{Z}$;
(ii) For any $x \in \mathbb{Z}_{\geq 1}$, $f_1(x) \leq f_2(x) \leq \cdots \leq f_n(x)$;
(iii) For any $x \in \mathbb{Z}_{\leq 0}$, $f_1(x) \geq f_2(x) \geq \cdots \geq f_n(x)$;
(iv) For any $x \in \mathbb{Z}_{\geq 1}$,

$$f_1(x) - f_1(x-1) \leq f_2(x) - f_2(x-1) \leq \cdots \leq f_n(x) - f_n(x-1);$$

(v) For any $x \in \mathbb{Z}_{\leq 0}$,

$$f_1(x-1) - f_1(x) \geq f_2(x-1) - f_2(x) \geq \cdots \geq f_n(x-1) - f_n(x).$$

The property (i) could be checked directly. The properties (ii-iii) hold, due to (15) and the the monotonicity of f. The properties (iv-v) hold, due to (15) and the convexity of f.

As in Sect. 5.1, using the SNF decomposition $PAQ = S$, we transform (14) to:

$$\sum_{i=1}^{n} f_i(x_i) \to \min \qquad \begin{cases} Px \equiv Pb \pmod{S \cdot \mathbb{Z}^n} \\ x \in \mathbb{Z}^n. \end{cases} \tag{16}$$

Let us define $\mathcal{G}$ and g_i (for $i \in \{1, \dots, n\}$) as it was done in Sect. 5.1. Let us define $G = Pb \bmod \operatorname{diag}(S)$ and reformulate (16) in the group minimization style:

$$\sum_{i=1}^{n} f_i(x_i) \to \min \qquad \begin{cases} \sum\limits_{i=1}^{n} g_i \cdot x_i = G \\ x \in \mathbb{Z}^n. \end{cases} \tag{17}$$

Now, we going to remove duplicates from $g_1, g_2, \dots, g_n$, but it is a bit more tricky problem than its analogue discussed in Remark 3. Assume that $g_1 = g_2 = \cdots = g_k$. We want to replace the variables $x_1, \dots, x_k$ by only one variable $y = x_1 + \cdots + x_k$ attached to g_1. To this end, we need to replace the objective $\sum_{i=1}^{k} f_i(x_i)$ with a new equivalent objective $h(y)$. The following lemma explains how to choose h.

Lemma 4 *Let $g_1 = g_2 = \cdots = g_k$, for $k \in \{1, \dots, n\}$. There exists a function $h(x) : \mathbb{Z} \to \mathbb{R}$, such that the problem (17) and the following problem*

$$h(y) + \sum_{i=k+1}^{n} f_i(x_{i-k}) \to \min$$
$$\begin{cases} g_1 \cdot y + \sum\limits_{i=k+1}^{n} g_i \cdot x_{i-k} = G \\ x \in \mathbb{Z}_{\geq 0}^{n-k} \\ y \in \mathbb{Z}_{\geq 0} \end{cases} \tag{18}$$

are equivalent. The function h can be defined in the following way:

(i) *If* $y \in \mathbb{Z}_{\geq 0}$, *then compute* $r = y \bmod k$ *and* $a = \lfloor y/k \rfloor$. *Let us construct the vector* $z = (a+1, \dots, a+1, a, \dots, a)^\top$, *where* $a+1$ *is taken* r *times. Put* $h(y) := \sum_{i=1}^{k} f_i(z_i)$;

(ii) *If* $y \in \mathbb{Z}_{<0}$, *then compute* $r = (-y) \bmod k$ *and* $a = \lfloor (-y)/k \rfloor$. *Let us construct the vector* $z = -(a, \dots, a, a+1, \dots, a+1)^\top$, *where* $a+1$ *is taken* r *times. Put* $h(y) := \sum_{i=1}^{k} f_i(z_i)$.

Additionally, for any $m \geq 0$, the sequences

$$h(0),\ h(1),\ \dots,\ h(m), \quad \text{and} \quad h(0),\ h(-1),\ \dots,\ h(-m)$$

can be computed using $O(m)$ *operations*.

The lemma proof is moved into Appendix Sect. 6.4.

Using the previous lemma, we can remove all the duplicates and assume that all the elements $g_1, g_2, \dots, g_n$ are unique. The remaining part of our algorithm is very close to the algorithm from Sect. 5.1. For $k \in \{1, \dots, n\}$ and $g_0 \in \mathcal{G}$, we define $DP(k, g_0)$, $\psi_+(k, g_0)$ and $\psi_-(k, g_0)$. Clearly, the problem $DP(n, G)$ is equivalent to the problem 17. The values $\psi_+(k, g_0)$ and $\psi_-(k, g_0)$ can be computed with the same formulas and algorithms. The only minor difference is the recurrent formula for $DP(k, g_0)$:

$$DP(k, g_0) = \min\left\{\psi_+(k-1, g_0), \psi_-(k-1, g_0)\right\}.$$

Therefore, we have proven our conclusive result.

Theorem 6 *The problem GENERALIZED-CVP can be solved by an algorithm with arithmetic complexity bound*

$$O(n^\omega \cdot \log(\Delta) + \min\{n, \Delta\} \cdot \Delta \cdot \log(\Delta)) = \tilde{O}(n^\omega + \min\{n, \Delta\} \cdot \Delta),$$

where all computations are performed with integer numbers of the size $O(\log(\Delta))$ *and* ω *is the matrix multiplication exponent*.

Appendix

Proof of Theorem 1

Proof Description of data structure: Our new data structure is a common segment tree T with some additional augmentations. Here we will use the same notations, as in Sect. 2.2. We augment each vertex v of T with an additional data, represented by a finite set $\mathcal{G}_v$ of functions $g : \mathcal{D}_g \cap \mathbb{Z} \to \mathbb{Z}$, where each domain $\mathcal{D}_g$ is an integer sub-interval of $[0, n)$ and g acts on $\mathcal{D}_g$ as a function $g(x) = A[j] + f(x + t)$, for some $j, t \in \{0, \dots, n-1\}$. We will support the following four invariants, for any $v \in T$:

– Invariant 1: the intervals $\mathcal{D}_g$, for any $g \in \mathcal{G}_v$, must split $[0, n)$:

$$[0, n) = \bigsqcup_{g \in \mathcal{G}_v} \mathcal{D}_g;$$

– Invariant 2: for any $x \in [0, n)$, there exists a unique function $g \in \mathcal{G}_v$, such that $x \in \mathcal{D}_g$, and

$$\mathcal{F}\big([i_v, j_v); x\big) = g(x);$$

– Invariant 3: the functions $g \in \mathcal{G}_v$ are stored in the sorted order with respect to the end-points of their domains $\mathcal{D}_g$;
– Invariant 4: for any $g \in \mathcal{G}_v$, the function $g(x)$ acts on $\mathcal{D}_g$ like $g(x) = A[j] + f(x + t)$, for $j, t \in \{0, \dots, n-1\}$;

Description and analysis of the query operation for basic intervals: For the definition of *basic intervals*, see Sect. 2.2. Assume that a vertex $v \in T$ is given, and we want to perform the $query(i_v, j_v, x)$ operation with respect to the basic interval $[i_v, j_v)$. Due to Invariant 2, we just need to find an appropriate function g from the set $\mathcal{G}_v$. Due to Invariant 3, the function g can be found in $O(\log(n))$ time, because $\mathcal{G}_v$ contains at most n functions. Due to Invariant 4, $g(x)$ looks like $A[j] + f(x + t)$, so it can be computed, using a single call to EV. The total complexity is $O(\log_2(n))$.

Description and analysis of the query operation for general intervals: Assume that an interval $[i, j)$ is given, and we want to perform the $query(i, j, x)$ operation. Due to Lemma 1, there exist $m \le 2\log_2(n)$ vertices $v_1, v_2, \dots, v_m \in T$, such that $[i, j)$ is partitioned into the basic intervals $[i_{v_k}, j_{v_k})$, for $k \in \{1, \dots, m\}$. Let us assume that $i_{v_1} < i_{v_2} < \dots < i_{v_m}$, and let $h_k = i_{v_k} - i_{v_1}$. Due to the property (1), we have

$$\begin{aligned}
&\mathcal{F}([i, j); x) \\
&\quad = \min\left\{\mathcal{F}\big([i_{v_1}, j_{v_1}); x + h_1\big), \dots, \mathcal{F}\big([i_{v_m}, j_{v_m}); x + h_m\big)\right\} \\
&\quad = \min_{k \in \{1, \dots, m\}} \left\{\mathcal{F}\big([i_{v_k}, j_{v_k}); x + h_k\big)\right\}.
\end{aligned}$$

Consequently, due to the complexity bound on queries for basic intervals, the complexity of the $query(i, j, x)$ operation is $O(\log^2(n))$.

Description and analysis of the preprocessing:

First of all, let us construct the standard segment tree T, described in Sect. 2.2, for the array A. It will take $O(n)$ time and space. We need to show how to compute $\mathcal{G}_v$, for any $v \in T$, and satisfy all the invariants. The algorithm is recursive: it starts from the leafs, and moves upper, until it meats the root of T. Let v be a leaf. Since $j_v - i_v = 1$, $\mathcal{F}\big([i_v, j_v); x\big) = A[i_v] + f(x)$. Consequently, $\mathcal{G}_v$ consists of only one function $g(x) = A[i_v] + f(x)$, and $\mathcal{D}_g = [0, n)$.

Next, we assume that v is not a leaf, and let u and w be the children of v. We will show how the set $\mathcal{G}_v$ can be constructed from the sets $\mathcal{G}_u$ and $\mathcal{G}_w$, based on the formula

$$\mathcal{F}\big([i_v, j_v); x\big) = \min\left\{\mathcal{F}\big([i_u, j_u); x\big),\ \mathcal{F}\big([i_w, j_w); x + j_u - i_u\big)\right\}, \tag{19}$$

which is a direct application of (1). Let $\mathcal{P}_u$ and $\mathcal{P}_w$ be the sets of end-points of intervals, representing the domains of functions inside $\mathcal{G}_u$ and $\mathcal{G}_w$. We assume that $0, n \in \mathcal{P}_u$ and $0, n \in \mathcal{P}_w$. Clearly, $\left|\mathcal{P}_u\right| = \left|\mathcal{G}_u\right| + 1$ and $\left|\mathcal{P}_w\right| = \left|\mathcal{G}_w\right| + 1$. Due to Invariant 3, we can assume that $\mathcal{P}_u$ and $\mathcal{P}_w$ are sorted. Next, we merge $\mathcal{P}_u$ and $\mathcal{P}_w$ into $\mathcal{P}_v$, maintaining the same sorting order, and remove the duplicates. The last step can be done in $O(\left|\mathcal{G}_u\right| + \left|\mathcal{G}_w\right|)$-time, since $\mathcal{P}_u$ and $\mathcal{P}_w$ are sorted. Since the points 0, n are common for both $\mathcal{P}_u$ and $\mathcal{P}_v$, we have

$$\left|\mathcal{P}_v\right| \leq \left|\mathcal{G}_u\right| + \left|\mathcal{G}_w\right|. \tag{20}$$

Take a pair ν, τ of consecutive points in $\mathcal{P}_v$. Due to Invariant 2, there exist unique functions $g_u \in \mathcal{G}_u$ and $g_w \in \mathcal{G}_w$, such that $[\nu, \tau) \subseteq \mathcal{D}_{g_u} \cap \mathcal{D}_{g_w}$. Due to the formula (19), for $x \in [\nu, \tau)$, we have

$$\mathcal{F}\left([i_v, j_v); x\right) = \min\left\{g_u(x),\ g_w(x + j_u - i_u)\right\}.$$

Let $h(x) = g_u(x) - g_w(x + j_u - i_u)$, defined on $[\nu, \tau)$. Due to Invariant 4, the function $h(x)$ has the form $f(x + a) - f(x + b) + c$, for some $a, b \in \mathbb{Z}_{\geq 0}$ and $c \in \mathbb{Z}$.

To efficiently precompute $\mathcal{F}\left([i_v, j_v); x\right)$ for $x \in [\nu, \tau) \cap \mathbb{Z}$, we need to compute a minimal sign partition $\mathcal{S} \in \mathscr{P}(h, [\nu, \tau))$. It can be done by a single call to $\mathcal{SP}$. Now, for any interval $\mathcal{I} \in \mathcal{S}$, if $h(x) \geq 0$ on $\mathcal{I}$, then $\mathcal{F}\left([i_v, j_v); x\right) = g_u(x)$ and $\mathcal{F}\left([i_v, j_v); x\right) = g_w(x + j_u - i_u)$ in the opposite case $h(x) \leq 0$. Consequently, for any such interval $\mathcal{I}$, we create a new function $g_{\mathcal{I}}$ and put it inside $\mathcal{G}_v$ in the sorted order with respect to endpoints of $\mathcal{I}$. Hence, the interval $[\nu, \tau)$ will be decomposed into at most p new sub-intervals, and the same number of new functions will be added into $\mathcal{G}_v$.

Now, let us estimate the time and space requirements to build the set $\mathcal{G}_v$. As it was shown before, for any pair $[\nu, \tau)$ of consecutive points from $\mathcal{P}_v$, we add at most p functions to $\mathcal{G}_v$. Therefore, due to (20), we have

$$\left|\mathcal{G}_v\right| \leq \left(\left|\mathcal{G}_u\right| + \left|\mathcal{G}_w\right| - 1\right) \cdot p.$$

Denote $N(m) = \max\left\{\left|\mathcal{G}_v\right| : v \in T,\ j_v - i_v = m\right\}$, for $m \leq n$ being a power of 2. Since $N(1) = 1$, we have

$$N(m) \leq 2 \cdot N(m/2) \cdot p \leq (2p)^{\log_2(m)} = m^{1+\log_2(p)}.$$

And, since we always work in the interval $[0, n)$,

$$N(m) \leq \min\{m^{1+\log_2(p)},\ n\}. \tag{21}$$

By analogy with $N(m)$, let us denote the maximal time to construct $\mathcal{G}_v$ (in the assumption that $\mathcal{G}_u$ and $\mathcal{G}_w$ are already constructed) by $t_{node}(m)$, where $m = j_v - i_v$. By the word "time", we mean both arithmetical and oracle complexities. Clearly, the definition is correct, because the value of m is the same for all the vertices of the same level in T. Since the complexity to compute $\mathcal{G}_v$ is linear with respect to the resulting size of $\mathcal{G}_v$, due to (21),

$$t_{node}(m) = O(N(m)) = O\big(\min\{m^{1+\log_2(p)},\ n\}\big). \tag{22}$$

Note additionally that the space requirements to store $\mathscr{G}_v$ with the whole information, related to v, can be described by the same function $t_{node}(m)$. Now, let us compute the total time and space complexity to construct the final augmented tree T. It can be expressed by the function

$$t(n) = \sum_{k=0}^{\log_2(n)} 2^k \cdot t_{node}(n/2^k).$$

Let $s = \left\lceil \log_2 \left(n^{\frac{1}{1+\log_2(p)}} \right) \right\rceil$. To calculate the asymptotic of $t(n)$, we split the sum into two parts and estimate elements of each sum, using (22):

$$\begin{aligned} t(n) &\lesssim \sum_{k=0}^{s} 2^k \cdot n + \sum_{k=s+1}^{\log_2(n)} 2^k \cdot (n/2^k)^{1+\log_2(p)} \\ &\lesssim n^{1+\frac{1}{1+\log_2(p)}} + n^{1+\log_2(p)} \cdot \sum_{k=s+1}^{\log_2(n)} 2^{-k\cdot\log_2(p)}. \end{aligned}$$

Estimating the sum at the end of the last formula, we have:

$$\begin{aligned} \sum_{k=s+1}^{\log_2(n)} 2^{-k\cdot\log_2(p)} &= \sum_{k=s+1}^{\log_2(n)} p^{-k} = \frac{p}{p-1} \cdot \left(\frac{1}{p^{1+s}} - \frac{1}{p^{1+\log_2(n)}} \right) \\ &= \frac{1}{p-1} \cdot \left(\frac{1}{p^s} - \frac{1}{p^{\log_2(n)}} \right) \le \frac{1}{p-1} \cdot \left(\frac{1}{n^{\frac{\log_2(p)}{1+\log_2(p)}}} - \frac{1}{n^{\log_2(p)}} \right). \end{aligned}$$

Finally, the total time and space requirements can be estimated as follows

$$\begin{aligned} t(n) &\lesssim n^{1+\frac{1}{1+\log_2(p)}} + \frac{1}{p} \cdot n^{1+\log_2(p)-\frac{\log_2(p)}{1+\log_2(p)}} \\ &= n^{1+\frac{1}{1+\log_2(p)}} \cdot \left(1 + \frac{1}{p} \cdot n^{-1+\log_2(p)} \right) \\ &= O\left(n^{\log_2(p)+\frac{1}{1+\log_2(p)}} \right). \end{aligned}$$

□

Proof of Lemma 2

Proof Put $\mathcal{I} = \{0, \dots, n-1\}$. To prove the lemma, we will use the criteria, given in [50, Corollary 1]. For given $g_0, g_1, \dots, g_{n-1}$, it states that the $C^{1,L}$-smooth convex function f with $f'(x_i) = g_i$ exists if and only if the following conditions are satisfied:

$$f_i \geq f_j + g_j \cdot (x_i - x_j) + \frac{1}{2L} \cdot \left| g_i - g_j \right|, \quad \text{for } i, j \in \mathcal{I}. \tag{23}$$

We construct g_i in the following way. We choose $g_0 < d_1$ and $g_{n-1} > d_{n-1}$. For $i \in \{1, \dots, n-2\}$, if $d_i < d_{i+1}$, we choose g_i strictly between d_i and d_{i+1}: $d_i < g_i < d_{i+1}$. In the opposite case, when $d_i = d_{i+1}$, we just set $g_i := d_i$.

Fix $i, j \in \{1, \dots, n-2\}$. Assume firstly that $d_k = d_l$, for all k, l between (inclusively) i, j. Then, the following equality, for any $\varepsilon > 0$, holds, which follows from the definition of d_i:

$$f_i = f_j + g_j \cdot (x_i - x_j) = f_j + g_j \cdot (x_i - x_j) + 0 \cdot \left| g_i - g_j \right|.$$

Now, assume that $d_k \neq d_{k+1}$, for some k between (inclusively) i, j. Then, since $g_k > d_k$, we have

$$f_i > f_j + g_j \cdot (x_i - x_j) \quad \Longrightarrow \quad f_i \geq f_j + g_j \cdot (x_i - x_j) + \varepsilon \cdot \left| g_i - g_j \right|,$$

for any sufficiently small $\varepsilon > 0$. Finally, if $i = 0$ or $j = n-1$, the same inequality holds, because $g_0 < d_1$ and $g_{n-1} > d_{n-1}$.

Therefore, since $\mathcal{I}$ is finite, we can choose ε sufficiently small, such that the following inequality will hold, for any $i, j \in \mathcal{I}$:

$$f_i \geq f_j + g_j \cdot (x_i - x_j) + \varepsilon \cdot \left| g_i - g_j \right|, \quad \text{which satisfies (23) with } L = 1/(2\varepsilon).$$

□

Proof of Lemma 3

Proof Assume that f is convex. In the opposite case, we can consider a function $-g(x) = (-f(x+a)) - b - (-f(x))$. Clearly, all the sign partitions of $g(x)$ and $-g(x)$ are equivalent. Next, we can assume that f is C^1-smooth. Definitely, if f is not C^1-smooth, due to Lemma 2, there exists a convex C^1-smooth function h, such that $h(x) = f(x)$, for all $x \in [\alpha, \beta) \cap \mathbb{Z}$. Since any minimal sign partition of f consists of intervals with integer end-points, the sign partitions for h and f are equivalent, and we can use h instead of f.

Additionally, we assume that $a > 0$, because, in the opposite case, both statements are trivial. Now, we claim that the equality $g(x) = 0$ is possible only for points in some connected interval $[\nu, \tau] \subseteq [\alpha, \beta)$. Assume that $g(\nu) = 0$ and $g(\tau) = 0$, for some $\nu, \tau \in [\alpha, \beta)$ with $\tau > \nu$. By definition, we have

$$f(\nu + a) - f(\nu) = -b \quad \text{and} \quad f(\tau + a) - f(\tau) = -b.$$

Due to C^1-smoothness of f, we can use the fundamental theorem of calculus:

$$\int_{\nu}^{\nu+a} f'(x)dx = -b, \quad \text{and} \quad \int_{\tau}^{\tau+a} f'(x)dx = -b. \tag{24}$$

Put $\delta = \tau - \nu$. Make a change of variables $x \to x - \delta$ and $x \to x + \delta$ in (24):

$$\int_{\tau}^{\tau+a} f'(x-\delta)dx = -b, \quad \text{and} \quad \int_{\nu}^{\nu+a} f'(x+\delta)dx = -b. \tag{25}$$

Combining (24) and (25), we get

$$\int_{\nu}^{\nu+a} \big(f'(x+\delta) - f'(x)\big)dx = 0, \quad \text{and} \quad \int_{\tau}^{\tau+a} \big(f'(x) - f'(x-\delta)\big)dx = 0.$$

Since f is convex, $f'(x+\delta) - f'(x) \geq 0$ and $f'(x) - f'(x-\delta) \geq 0$, and, consequently,

$$\forall x \in [\nu, \nu+a] \quad f'(x) = \text{ const}, \quad \text{and} \quad \forall x \in [\tau, \tau+a] \quad f'(x) = \text{ const}.$$

Again, since $f'(x)$ is convex, $f'(x) = \text{const}$, for the whole interval $[\nu, \tau]$. Consequently, $g'(x) = 0$ and $g(x) = \text{const}$, for $x \in [\nu, \tau]$. Since $g(\nu) = 0$, it holds that $g(x) = 0$, for $x \in [\nu, \tau]$. So, the claim is proved.

Therefore, any given interval $\mathcal{I} \subseteq [\alpha, \beta)$, where g is well-defined, can be partitioned into at most three parts: strict inequalities $g(x) > 0$ or $g(x) < 0$ on the left and right sides, and equality $g(x) = 0$ in the middle. Consequently, any minimal sign partition of g on $\mathcal{I}$ consists of at most 2 pieces, which proves the inequality $p_f \leq 2$. To calculate such a partition, we need to find an integer point $z \in \mathcal{I}$ with $g(z) = 0$. Or, in the case, when $g(x) \neq 0$ for all $x \in \mathcal{I} \cap \mathbb{Z}$, we need to calculate a point z, such that $g(z) < 0$ and $g(z+1) > 0$, or vice-versa. Clearly, in both cases we can use the standard dichotomy principle that takes $O(\log(n))$ calls to EV. □

Proof of Lemma 4

Proof First we need to prove the following auxiliary lemma:

Lemma 5 *Let $g_1 = g_2 = \cdots = g_k$, for $k \in \{1, \ldots, n\}$, and x^* be an optimal solution of 17. Denote $S = x_1^* + \cdots + x_k^*$. Then, there exists an optimal solution z^* with the following structure*:

(i) *If $S \geq 0$, then*:

$$z^* = (a+1, a+1, \ldots, a+1, a, a, \ldots, a)^\top, \quad \text{where } a \in \mathbb{Z}_{\geq 0}. \tag{26}$$

(ii) *If $S < 0$, then*:

$$z^* = -(a, a, \dots, a, a+1, a+1, \dots, a+1)^\top, \quad \text{where } a \in \mathbb{Z}_{\geq 0}. \tag{27}$$

Proof Note that the expressions $g_1 \cdot x_1^* + \dots + g_k \cdot x_k^*$ and $g_1 \cdot S$ are equivalent in therms of constraints of (17). Assume that $S \geq 0$. First of all, we claim that there exists an optimal solution z^* with the property $z_i^* \geq 0$, for $i \in \{1, \dots, k\}$. Assume that there exist $i, j \in \{1, \dots, k\}$ with $x_i^* \geq 1$ and $x_j^* \leq -1$. Since $S \geq 0$, if x_j^* exists, then x_i^* exists also. Next, we construct a vector z^*, which coincides with x^* in all the coordinates, except i, j. Put $z_i^* = x_i^* - 1$ and $z_j^* = x_j^* + 1$. Due to Property 1, we have $\sum_{i=1}^k f_i(z_i^*) \leq \sum_{i=1}^k f_i(x_i^*)$. Such a procedure can be repeated until no negative coordinates remain. Consequently, it can be assumed that $x_i^* \geq 0$, for $i \in \{1, \dots, k\}$. Let us consider the following auxiliary optimization problem:

$$\begin{cases} \sum_{i=1}^k f_i(x_i) \to \min \\ \begin{cases} x_1 + \dots + x_k = S \\ x \in \mathbb{Z}_{\geq 0}. \end{cases} \end{cases}$$

Clearly, $x^*[1, k]$ gives an optimal solution of this problem, and vice versa, an optimal solution of 6.4 could be used to generate the first k coordinates of x^*. Let us consider the set $\mathcal{S} = \{x \in \mathbb{Z}_{\geq 0}^k : x_1 + \dots + x_k \leq S\}$. Elements of $\mathcal{S}$ could be treated as the characteristic vectors of multisets with the cardinality S. Identifying vectors with multisets, we can see that $\mathcal{S}$ is a matroid, see, for example, [33, Proposition 13.4, Part 13. Matroids]. The vectors $z \in \mathcal{S}$ with $z_1 + \dots + z_k = S$ are the bases of $\mathcal{S}$. Consequently, an optimal solution of 6.4 is exactly a base of $\mathcal{S}$ with the minimal possible value of the objective function. Since $\mathcal{S}$ is a matroid, an optimal solution of 6.4 can by found by the following greedy algorithm:

1. Assign $s := 0$, $x := \mathbf{0}^k$, and $F := f_1(0) + \dots + f_k(0)$;
2. While $s \leq S$ do the following:
3. Choose $i \in \{1, \dots, k\}$, such that the value $f_i(x_i + 1) - f_i(x_i)$ is minimal;
4. Assign $x_i := x_i + 1$, $s := s + 1$, and $F := F + f_i(x_i + 1) - f_i(x_i)$;
5. Move to the step 2;
6. Return x as a greedy solution and F as $f(x)$;

Due to the properties 4,5, there exists a greedy solution z^* that looks like (26). This proves the lemma for the case $S \geq 0$. The case $S < 0$ is absolutely similar. □

If x^* is an optimal solution of (17), then, due to Lemma 5, there exists an optimal solution z^* of (17), such that the first k components of z^* look like (26) or (27). By the definition of h, we have $\sum_{i=1}^k f_i(z_i^*) = h(y)$, where $y = z_1^* + \dots z_k^*$. Note that $(y, x^*[k+1, n])$ is a feasible solution of (18) with the same value of the objective function. In the opposite direction, let (y, x^*) be an optimal solution of (18). Let

us construct the vector z as it was described in the lemma definition. Clearly, the vector $\begin{pmatrix} z \\ x^* \end{pmatrix}$ is a feasible solution of (17) with the same value of the objective function.

Finally, let us explain how to compute $h(0), h(1), \ldots, h(m)$ with $O(m)$ operations. Assume that $h(i)$ has already been computed, and let $r = i \bmod k$ and $a = \lfloor i/k \rfloor$. Then, by the definition of h, we have $h(i+1) = h(i) + f_{r+1}(a+1) - f_{r+1}(a)$. Hence, we need $O(1)$ operations to compute $h(i+1)$ and $O(m)$ operations to compute the whole sequence. A similar algorithm works for $h(0), h(-1), \ldots, h(-m)$. The proof of Lemma 4 is finished. □

References

1. Aggarwal, D., Dadush, D., Regev, O., Stephens-Davidowitz, N.: Solving the shortest vector problem in 2n time using discrete gaussian sampling. In: Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing, pp. 733–742 (2015)
2. Aggarwal, D., Dadush, D., Stephens-Davidowitz, N.: Solving the closest vector problem in 2^n time–the discrete gaussian strikes again! In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, pp. 563–582. IEEE (2015)
3. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of computing, pp. 601–610 (2001)
4. Ajtai, M., Kumar, R., Sivakumar, D.: Sampling short lattice vectors and the closest lattice vector problem. In: Proceedings 17th IEEE Annual Conference on Computational Complexity, pp. 53–57. IEEE (2002)
5. cp algorithms.com: Segment tree (2022). https://cp-algorithms.com/data_structures/segment_tree.html
6. Arvind, V., Joglekar, P.S.: Some sieving algorithms for lattice problems. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2008)
7. Axiotis, K., Tzamos, C.: Capacitated Dynamic Programming: Faster Knapsack and Graph Algorithms. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S. (eds.) 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019), Leibniz International Proceedings in Informatics (LIPIcs), vol. 132, pp. 19:1–19:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.19
8. Backurs, A., Indyk, P., Schmidt, L.: Better approximations for tree sparsity in nearly-linear time. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 2215–2229. SIAM (2017)
9. Bateni, M., Hajiaghayi, M., Seddighin, S., Stein, C.: Fast algorithms for knapsack via convolution and prediction. In: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, pp. 1269–1282 (2018)
10. Bellman, R.: Dynamic programming. Science **153**(3731), 34–37 (1966)
11. Blömer, J., Naewe, S.: Sampling methods for shortest vectors, closest vectors and successive minima. Theoret. Comput. Sci. **410**(18), 1648–1665 (2009)
12. Bremner, D., Chan, T.M., Demaine, E.D., Erickson, J., Hurtado, F., Iacono, J., Langerman, S., Taslakian, P.: Necklaces, convolutions, and $x + y$. In: European Symposium on Algorithms, pp. 160–171. Springer (2006)
13. Chan, T.M., Har-Peled, S.: Smallest k-enclosing rectangle revisited. Discrete Comput. Geom. **66**(2), 769–791 (2021)
14. Chan, T.M., Lewenstein, M.: Clustered integer 3sum via additive combinatorics. In: Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing, STOC '15, pp. 31–40. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2746539.2746568

15. Chan, T.M., Williams, R.: Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In: Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms, pp. 1246–1255. SIAM (2016)
16. Chi, S., Duan, R., Xie, T., Zhang, T.: Faster min-plus product for monotone instances (2022)
17. Cygan, M., Mucha, M., Węgrzycki, K., Włodarczyk, M.: On problems equivalent to (min, +)-convolution. ACM Trans. Algorithm (TALG) **15**(1), 1–25 (2019)
18. Dadush, D.: Integer programming, lattice algorithms, and deterministic volume estimation. Georgia Institute of Technology, ProQuest Dissertations Publishing, Ann Arbor (2012)
19. Dadush, D., Peikert, C., Vempala, S.: Enumerative lattice algorithms in any norm via m-ellipsoid coverings. In: 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, pp. 580–589 (2011). https://doi.org/10.1109/FOCS.2011.31
20. Eisenbrand, F., Hähnle, N., Niemeier, M.: Covering cubes and the closest vector problem. In: Proceedings of the Twenty-Seventh Annual Symposium on Computational Geometry, pp. 417–423 (2011)
21. Eisenbrand, F., Weismantel, R.: Proximity results and faster algorithms for integer programming using the steinitz lemma. ACM Trans. Algorithms (2019). https://doi.org/10.1145/3340322
22. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Math. Comput. **44**(170), 463–471 (1985)
23. Gribanov D., V.: An FPTAS for the Δ-Modular Multidimensional Knapsack Problem (2021). https://doi.org/10.1007/978-3-030-77876-7_6
24. Gribanov, D.V., Malyshev, D.S., Pardalos, P.M., Veselov, S.I.: FPT-algorithms for some problems related to integer programming. J. Comb. Optim. **35**, 1128–1146 (2018). https://doi.org/10.1007/s10878-018-0264-z
25. Gribanov D., V., Shumilov I., A., Malyshev D., S., Pardalos P., M.: On δ-modular integer linear problems in the canonical form and equivalent problems. J. Glob. Optim. (2022). https://doi.org/10.1007/s10898-022-01165-9
26. Hanrot, G., Pujol, X., Stehlé, D.: Algorithms for the shortest and closest lattice vector problems. In: International Conference on Coding and Cryptology, pp. 159–190. Springer (2011)
27. Hanrot, G., Stehlé, D.: Improved analysis of kannan's shortest lattice vector algorithm. In: Annual International Cryptology Conference, pp. 170–186. Springer (2007)
28. Helfrich, B.: Algorithms to construct minkowski reduced and hermite reduced lattice bases. Theoret. Comput. Sci. **41**, 125–139 (1985)
29. Jansen, K., Rohwedder, L.: On integer programming and convolution. In: 10th Innovations in Theoretical Computer Science Conference (ITCS 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
30. Kannan, R.: Minkowski's convex body theorem and integer programming. Math. Oper. Res. **12**(3), 415–440 (1987)
31. Kellerer, H., Pferschy, U.: Improved dynamic programming in connection with an fptas for the knapsack problem. J. Comb. Optim. **8**(1), 5–11 (2004)
32. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer Science & Business Media, Berlin (2013)
33. Korte, B., Vygen, J.: Combinatorial Optimization. Springer, Berlin (2011)
34. Kulikov, A.S., Mikhailin, I., Mokhov, A., Podolskii, V.: Complexity of Linear Operators. In: P. Lu, G. Zhang (eds.) 30th International Symposium on Algorithms and Computation (ISAAC 2019), Leibniz International Proceedings in Informatics (LIPIcs), vol. 149, pp. 17:1–17:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2019). https://doi.org/10.4230/LIPIcs.ISAAC.2019.17
35. Künnemann, M., Paturi, R., Schneider, S.: On the Fine-Grained Complexity of One-Dimensional Dynamic Programming. In: I. Chatzigiannakis, P. Indyk, F. Kuhn, A. Muscholl (eds.) 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017), Leibniz International Proceedings in Informatics (LIPIcs), vol. 80, pp. 21:1–21:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017). https://doi.org/10.4230/LIPIcs.ICALP.2017.21. http://drops.dagstuhl.de/opus/volltexte/2017/7468
36. Laber, E.S., Bardales, W., Cicalese, F.: On lower bounds for the maximum consecutive subsums problem and the (min, +)-convolution. In: 2014 IEEE International Symposium on Information Theory, pp. 1807–1811. IEEE (2014)
37. Lawler, E.L.: Fast approximation algorithms for knapsack problems. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), pp. 206–213. IEEE (1977)

38. Liu, M., Wang, X., Xu, G., Zheng, X.: Shortest lattice vectors in the presence of gaps. Cryptology ePrint Archive (2011)
39. Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms, pp. 1468–1480. SIAM (2010)
40. Micciancio, D., Voulgaris, P.: A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. SIAM J. Comput. **42**(3), 1364–1391 (2013)
41. Micciancio, D., Walter, M.: Fast lattice point enumeration with minimal overhead. In: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 276–294. SIAM (2014)
42. Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. J. Math. Cryptol. **2**(2), 181–207 (2008)
43. Pferschy, U.: Dynamic programming revisited: Improving knapsack algorithms. Computing **63**(4), 419–430 (1999). https://doi.org/10.1007/s006070050042
44. Pohst, M.: On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. ACM Sigsam Bull. **15**(1), 37–44 (1981)
45. Polak, A., Rohwedder, L., Wegrzycki, K.: Knapsack and subset sum with small items (2021). arXiv:2105.04035v1 [cs.DS]
46. Pujol, X., Stehlé, D.: Solving the shortest lattice vector problem in time 2ˆ 2.465 n. Cryptology ePrint Archive (2009)
47. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Chichester (1998)
48. Sommer, N., Feder, M., Shalvi, O.: Finding the closest lattice point by iterative slicing. SIAM J. Discret. Math. **23**(2), 715–731 (2009)
49. Storjohann, A.: Near optimal algorithms for computing Smith normal forms of integer matrices. In: Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, ISSAC '96, pp. 267–274. Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/236869.237084
50. Taylor, A.B., Hendrickx, J.M., Glineur, F.: Smooth strongly convex interpolation and exact worst-case performance of first-order methods. Math. Program. **161**(1), 307–345 (2017)
51. Williams, R.R.: Faster all-pairs shortest paths via circuit complexity. SIAM J. Comput. **47**(5), 1965–1985 (2018)
52. Yasuda, M.: A survey of solving svp algorithms and recent strategies for solving the svp challenge. In: International Symposium on Mathematics, Quantum Theory, and Cryptography, pp. 189–207. Springer, Singapore (2021)
53. Zhendong, W.: Computing the Smith forms of Integer Matrices and Solving Related Problems. University of Delaware, Newark, DE (2005)