Approximating Multidimensional Subset Sum and the Minkowski Decomposition of Polygons

Ioannis Emiris, Anna Karasoulou, Charilaos Tzovas

Approximating Multidimensional Subset Sum and the Minkowski Decomposition of Polygons[☆]

Ioannis Z. Emiris[1], Anna Karasoulou[1], Charilaos Tzovas[1,]

[a]*Department of Informatics & Telecommunications, University of Athens, Athens, Greece, and "Athena" Research Center, Maroussi, Greece*
[b]*Department of Mathematics, University of Athens,Athens, Greece*

Abstract

We consider the approximation of two NP-hard problems: Minkowski Decomposition (MinkDecomp) of lattice polygons in the plane and the closely related problem of Multidimensional Subset Sum (kD-SS) in arbitrary dimension. In kD-SS we are given an input set S of k-dimensional vectors, a target vector t and we ask, if there exists a subset of S that sums to t. We prove, through a gap-preserving reduction, that, for general dimension k, kD-SS is not in APX although the classic $1D$-SS is in PTAS. On the positive side, we present an $O(n^3/\epsilon^2)$ approximation grid based algorithm for $2D$-SS, where n is the cardinality of the set and ϵ bounds the difference of some measure of the input polygon and the sum of the output polygons. We also describe two more approximation algorithms with a better experimental ratio. Applying one of these algorithms, and a transformation from MinkDecomp to $2D$-SS, we can approximate MinkDecomp. For an input polygon Q and parameter ϵ, we return two summands A and B such that $A+B=Q'$ with Q' being bounded in relation to Q in terms of volume, perimeter, or number of internal lattice points, an additive error linear in ϵ and up to quadratic in the diameter of Q. A similar function bounds the Hausdorff distance between Q and Q'. We offer experimental results based on our implementation.

1. Introduction

Every polynomial is related with its Newton polytope and, using a theorem of Ostrowski, Gao devised an irreducibility test for a polynomial. Here, we consider the problem of decomposition of integral polygons. A polygon Q is called an (integral) *lattice polygon*, when all its vertices are points with integer coordinates.

A polygon Q is called an (integral) *lattice polygon*, when all its vertices are points with integer coordinates.

Definition 1. The Minkowski sum of two sets of vectors A and B in Euclidean space is defined by adding each vector in A to each vector in B, namely: $A+B=$

[☆]The authors were partially supported by H2020 M. Sklodowska-Curie Innovative Training Network "ARCADES: Algebraic Representations in Computer-Aided Design for complEx Shapes", 2016-2019.

*Corresponding author.

$\{a + b \mid a \in A, b \in B\}$.

Problem 2. Minkowski Decomposition (MinkDecomp).
Given a lattice convex polygon Q, decide if it is decomposable, that is, if there are nontrivial lattice polygons A and B such that $A + B = Q$, where $+$ denotes the Minkowski sum. The polygons A and B are called *summands*.

?? is proven NP-complete in [**?**] and can be reduced to $2D$-SS. For the reduction see **??**. The approximation version can be defined as follows.

Problem 3. MinkDecomp-μ-approx
Input: A lattice polygon Q, a parameter $0 < \epsilon < 1$ and a function μ.
Output: Lattice polygons A, B such that $0 \leq \mu(A + B) - \mu(Q) < \epsilon \cdot \phi(D)$, where D is the diameter of Q and ϕ a polynomial. We call such an output an $\epsilon \cdot \phi(D)$-solution.

For μ we may consider the functions $vol(Q)$: the volume, $per(Q)$: the perimeter, $i(Q)$: the internal lattice points of Q or $d_H(Q, A + B)$: the Hausdorff distance between Q and $A + B$. An interesting question is what other functions we can use to measure the similarity of two polygons.

Problem 4. kD-Subset Sum (kD-SS)
Input: A vector set $S = \{v_i \mid v_i \in \mathbb{Z}^k, 1 \leq i \leq n, k \geq 1\}$ and a target vector $t \in \mathbb{Z}^k$.
Output: Decide, whether there exists a vector subset $S' \subseteq S$ such that $\sum v_i = t$, $v_i \in S'$.

This is a generalization of the classic $1D$-SS problem, and as such, is also NP-complete.

Let P_i be the set of all possible vector sums that can be produced by adding at most i vectors among the first vectors in S. Then, $P_n \subseteq \mathbb{Z}^k$ is the set of all possible vector sums. Here is the approximation version:

Problem 5. kD-SS-opt
Input: A set $S = \{v_i \mid v_i \in \mathbb{Z}^k, 1 \leq i \leq n, k \geq 1\}$, a nonzero target $t \in P_n$ and $0 < \epsilon < 1$.
Output: Find a subset $S' \subseteq S$ whose vector sum $t' = \sum v_i, v_i \in S'$ that

$$\text{minimize } dist(t, t').$$

We consider the Euclidean distance l_2 here, but our method is easily generalized to any l_p, $1 \leq p < \infty$. For more details see Theorem 8.22 in [**?**].

Definition 6. A PTAS (Polynomial Time Approximation Scheme) is an algorithm, which takes an instance of an optimization problem, a parameter $\epsilon > 0$ and in polynomial time produces a solution, that is within a factor $1+\epsilon$ of being optimal for minimization problems or $1 - \epsilon$ for maximization problems.

We further recall the classes EPTAS (Efficient PTAS), where time complexity is polynomial in the input size (but can have any dependence on ϵ) and FPTAS (Fully PTAS), where the time complexity is polynomial in both input size and the parameter ϵ. The class APX contains every problem that can be approximated within a constant factor c.

Previous work $1D$-SS and kD-SS are not strongly NP-complete and can be solved exactly in pseudo-polynomial time: $1D$-SS is solved in $O(n|t|)$, see [?]. Generalizing this idea kD-SS is solved in $O(n|M|^k)$, where $M = \max P_n$ is the farthest reachable point. Moreover, $1D$-SS is in FPTAS, see [?]. A related problem is the Multidimensional Knapsack. Firstly, it was proved in [?], that this problem does not have an FPTAS. Later in [?], does not have an EPTAS, while Knapsack is in FPTAS. $1D$-SS is a special case of Knapsack as both are defined as maximization problems. In two or higher dimension, it makes more sense to define kD-SS as a minimization problem, since a vector sum with maximum length may be far from the target vector. So, in dimension two or higher, the problem is not related to Multidimensional Kanspack but rather to CVP.

A closely connected problem to kD-SS is the Closest Vector Problem (CVP): we are given a set of basis vectors $B = \{b_1, \ldots, b_n\}$, where $b_i \in \mathbb{Z}^k$, and a target vector $t \in \mathbb{Z}^k$, and we ask what is the closest vector to t in the lattice $\mathcal{L}(B)$ generated by B. This is $\mathcal{L}(B) = \{\sum_{i=1}^{m} a_i b_i \mid a_i \in \mathbb{Z}\}$ and thus kD-SS is a special case of CVP, where $a_i \in \{0, 1\}$. CVP is not in APX and cannot even be approximated within a factor of $2^{\log^{1-\epsilon} n}$ with $\epsilon = (\log(\log n))^c$ for $c < 1/2$ [? ?].

MinkDecomp has its fair share of attention. One application is in the factorization of bivariate polynomials through their Newton polygons. As noticed by Ostrowski in 1921, if a polynomial factors, then its Newton polygon has a Minkowski decomposition. An algorithm for polynomial irreducibility testing using MinkDecomp is presented in [?] motivated by previous similar work in [?]. They present a criterion for MinkDecomp that reduces the decision problem into a linear programming question. Continuing the work of [? , sections 4,5] we propose a polynomial time algorithm, that solves MinkDecomp approximately using a solver for $2D$-SS. Here, we are interested in finding approximate solutions. We will discuss these problems in **??**. MinkDecomp is NP-complete for lattice polygons, and a pseudopolynomial algorithm exists [?].

Our contribution We introduce the kD-SS problem. It is clearly NP-complete; we prove that it cannot be approximated efficiently. For $k \geq 2$ it cannot be approximated within a constant factor (although the classic 1D-SS has a FPTAS). We design an algorithm for $2D$-SS-approx: given a set S, $|S| = n$, target t and $0 < \epsilon < 1$, the algorithm returns, in $O(n^3\epsilon^{-2})$ time, a subset of S whose vectors sum to t' such that $dist(t, t') \leq \epsilon M$, where $M = \max P_n$. We also describe two more approximation algorithms with a better experimental ratio.

Applying one of these algorithms yields an approximation algorithm for MinkDecomp (**??**): If Q is the input polygon the algorithm returns polygons A and B whose Minkowski sum defines polygon Q' such that $vol(Q) \leq vol(Q') \leq vol(Q) + \epsilon D^2$, $per(Q) \leq per(Q') \leq per(Q) + 2\epsilon D$, $i(Q) \leq i(Q') \leq i(Q) + \epsilon D^2$, where D is the diameter of Q. The Hausdorff distance of Q and Q' is bounded by $d_H(Q, Q') \leq \epsilon/2D$.

2. *kD*-SS is not in APX

To prove that kD-SS-opt is not in APX we will apply the idea used to prove that CVP is not in APX, see [?] for more details. We apply their proof to our problem, in order to prove something similar for the kD-SS-opt.

Proposition 7. *[?] For every $c > 1$ there is a polynomial time reduction that, given an instance ϕ of SAT, produces an instance of Set Cover $\{\mathcal{U}, (S_1, \ldots, S_m)\}$ where $\mathcal{U}$ is the input set of integers and $S_1, \ldots, S_m$ are subsets of $\mathcal{U}$, and integer K with the following property: If ϕ is satisfiable, there is an exact cover of size K, otherwise all set covers have size more than cK.*

Given a CNF formula ϕ we invoke **??** and get an instance of the Set Cover problem. This is a gap introducing reduction, because if ϕ is statisfiable then the instance of Set Cover has a solution of size exactly K and if ϕ is not statisfiable every solution has size at least cK for a constant c. From this instance of Set Cover we create an instance for kD-SS that preserves the gap. Now, if ϕ is satisfiable, the closest vector to a given target t has distance exactly K. If ϕ is not statisfiable, the closest vector in target t has distance at least cK.

We reduce kD-SS to Set Cover for norm l_1, but this can easily be generalized to any l_p, where p is a positive integer. We say that a cover is *exact* if the sets in the cover are pairwise disjoint.

Theorem 8. *Given a CNF formula ϕ and $c > 1$ we create an instance $\{v_1, \ldots, v_m; t\}$ of kD-SS. If ϕ is satisfiable, then the minimum distance of a possible vector sum from t is smaller than K otherwise, it is larger than cK.*

Proof. Let $\{\mathcal{U}, (S_1, \ldots, S_m), K\}$ be the instance of Set-Cover obtained in Proposition **??** for the formula ϕ. We transform it to an instance of kD-SS with input set $\{v_1, \ldots, v_m\}$ and target t, such that the distance of t from the nearest point in the set of all possible points P_n is either K or $\geq cK$.

Let $v_i \in \mathbb{Z}^{n+m}$, where $|\mathcal{U}| = n$. We will create such a vector v_i for every set S_i; $1 \leq i \leq m$. Let $L = cK$. Then the first n coordinates of each vector v_i have their j'th-coordinate ($j \leq n$) equal to L if the corresponding j'th-element belongs to set S_i, or 0 otherwise. The remaining m coordinates have 1 in the $(n+i)$'th-coordinate and zeros everywhere else:

$$v_i = (L \cdot \chi_{S_i}, 0, \ldots, 1, \ldots 0) = (L \cdot \chi_{S_i}, e_i),$$

where χ_{S_i} is the characteristic function of the set S_i. The target vector t has in the first n coordinates L and the last m coordinates are zeros, $t = (L, \ldots, L, 0, \ldots, 0)$.

Now, let the instance of Set-Cover have an exact cover of size K. We will prove that the minimum distance of every $v \in P_n$ from target t is less than K. Without loss of generality, let the solution be $\{S_1, \ldots, S_K\}$. For each S_i, $1 \leq i \leq K$, sum the corresponding vectors v_i and let this sum be $\zeta \in \mathbb{Z}^{n+m}$:

$$\zeta = \sum_{i=1}^{K} v_i = (\underbrace{L, \ldots, L}_{n}, \underbrace{1, \ldots, 1}_{K} \underbrace{0, \ldots, 0}_{m-K}).$$

The first n coordinates must sum up to $(L, L \ldots, L)$, because if one of the coordinates was 0, the solution would not be a cover and if one of them was greater than L, then some element is covered more than once and the solution would not be exact. Note that each of the first n coordinates is either 0 or greater than L. The key point is that in the last m coordinates we will have exactly K units and everything else 0. The distance of this vector ζ from t is

$$\| - t + \zeta\|_1 = \|(\underbrace{0, \ldots, 0}_{n}, \underbrace{1, \ldots, 1}_{K} \underbrace{0, \ldots, 0}_{m-K})\|_1 = K$$

Thus, there is a point in P_n that its distance from t is at most K.

Let us consider the other direction, where the Set Cover instance has a solution set greater than $cK = L$. We will show that the closest vector to t has distance at least L from t. This solution must have at least $cK = L$ sets. As before, $\| - t + \zeta\|_1 \geq L$ (this time the cover need not be exact).

Towards a contradiction, suppose there exists a vector ξ such that $\| - t + \xi\|_1 < L$. If the corresponding sets do not form a cover of S then one of the first n coordinates of ξ is 0 and this alone is enough for $\| - t + \xi\|_1 > L$. If the sets form a cover that is not exact, then in at least one of the first n coordinates of ξ will be greater than L (for the element that is covered more than once) and will force $\| - t + \xi\|_1$ to be greater than L. Finally, if the sets form an exact cover, the first n coordinates of $\| - t + \xi\|_1$ will be 0. For the distance to be less than L, in the last m coordinates there must be less than L units implying that the sets in the cover are less than L contradicting our hypothesis.

In all cases, there cannot exist a vector whose distance from t is $< cK$. □

Theorem 9. *There is no APX for kD-SS-approx unless P=NP.*

Proof. Let ϕ be a given formula as an instance of SAT. Use **??** to get an instance of Set Cover and then the reduction from **??** to get an instance of kD-SS. Suppose there exists an algorithm $\mathcal{A}$ for kD-SS-opt that is in APX. $\mathcal{A}$ returns a vector t' such that $\|t - t'\|_1 \leq (1 + \epsilon)OPT$, where $OPT = \|t - t^*\|_1$ and t^* is the closest vector in P_n. From **??**, if ϕ is satisfiable then $OPT \leq K$ and if ϕ is not satisfiable $OPT > cK$.

We must run algorithm $\mathcal{A}$ with a suitable parameter ϵ so we can distinguish if the optimum solution t^* is within distance K or not. When ϕ is satisfiable we would want $(1+\epsilon)K < cK \implies \epsilon < c-1$. Set $c' < c-1$, call $\mathcal{A}$ with parameter $\epsilon = c'$ and let t' be the returned vector. In the case where ϕ is satisfiable and $OPT \leq K$ we have

$$\|t - t'\|_1 \leq (1 + \epsilon)OPT < cK$$

Of course if ϕ is not satisfiable for any t' we have that $\|t - t'\|_1 > cK$. Thus, $\|t - t'\|_1 < cK$ if and only if ϕ is satisfiable. Since ϵ is a constant and $\mathcal{A}$ is in APX we can decide SAT in polynomial time. □

Although there can be no algorithm that returns a constant factor approximation solution for general dimension k, we will present algorithms that provide different kind of approximation. Specifically, the returned vector t' of our algorithm is an $(OPT + \epsilon M)$ solution, where $M = \max P_n$ is the longest possible vector sum.

3. Three approximation algorithms for 2*D*-SS

3.1. annulus-slice algorithm

The idea here is to create all possible vectors step by step. At each step, if two vectors are close to each other, one is deleted. Whenever we refer to distance it is the Euclidean distance. We begin with some notation.

- Input: the set $S = \{v_1, v_2, \ldots, v_n\}$ with $v_i = (x_i, y_i) \in \mathbb{Z}^2$ and $|S| = n$, parameter $0 < \epsilon < 1$.

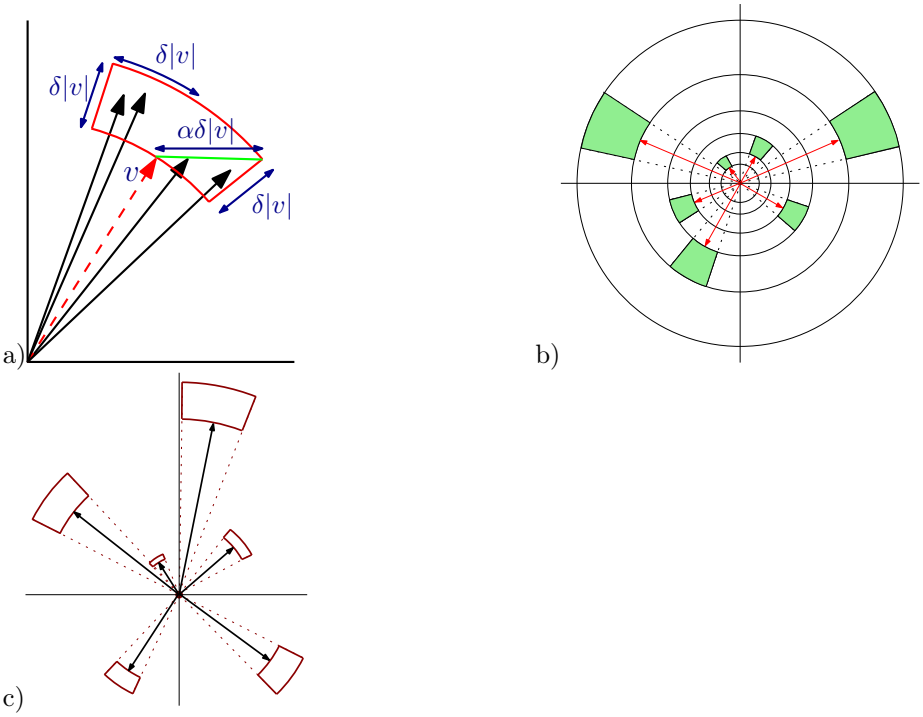

Figure 1: a) A single cell for the dashed vector v. All vectors in the cell will be deleted. The distances are shown and the furthest point is in distance $\alpha\delta|v|$. b) How space is divided. c) A few cells. The shorter the vector the smaller the cell.

- P_i is the set of all possible vectors that can be produced by adding at most i elements from the first i vectors in S. P_n is the set of all possible vector sums.
- $E_i = L_{i-1} \cup \{w + v_i \mid w \in L_{i-1}\}$ is the list created at the beginning of every step and that is about to get trimmed.
- $L_i = trim(E_i, \delta)$ is the trimmed list and $0 \leq \delta \leq 1$.

At the beginning of the i-th step we create the list $E_i = L_{i-1} \cup \{w + v_i \mid w \in L_{i-1}\}$. Notice that addition is over $\mathbb{Z}^2$. After a point is found we calculate its length, sort E_i based on the lengths and call $trim(E_i, \epsilon/2n)$. For each vector $u \in E_i$ with length $|u|$ and angle $\theta(u)$ from the x-axis, check all the vectors $u' \in E_i$ that have length $|u| \leq |u'| \leq (1+\delta)|u|$. If also $\theta(u) - \delta \leq \theta(u') \leq \theta(u) + \delta$, remove u' from E_i. The remaining trimmed list is the list L_i. The two conditions ensure that $dist(u', u) \leq \alpha\delta|u|$, where $1 \leq \alpha \leq 2$ is a constant. Every vector that is deleted from E_i is not very far away from a vector in L_i:

$$\forall u \in E_i, \exists w \in L_i : u = w + r_w,\ |r_w| \leq \alpha\delta|w| \tag{1}$$

hence, $|w| \leq |u| \leq (1 + \delta)|w|$. See **??**.

Since all vectors have integer coordinates, any vector $u \in E_i$ such that $|u| < 1/\alpha\delta < \sqrt{2}n/\epsilon$ implies that $\alpha\delta|u| < 1$. Thus, the area around u does not contain any other other lattice points except u.

Lemma 10. *Using the above notation, call function L_i =trim(E_i, δ), with parameter $\delta = \epsilon/2n$ and let $M_i = max\{|u| : u \in E_i\}$, the vector in E_i with the largest length. It holds that $|L_i| = O(n^2\epsilon^{-2} \log M_n)$ for $1 \leq i \leq n$.*

Proof. Every vector in E_i has length between $(1+\delta)^k$ and $(1+\delta)^{k+1}$. These are circles with center $(0,0)$ and radius $(1+\delta), (1+\delta)^2, \ldots, (1+\delta)^k$ for some k. Every two successive circles form an annulus that we call it a zone. We must cover all $u \in P_n$ and k is the minimum such that $(1+\delta)^k > M_n$. Solving $(1+\delta)^k \geq M_n$ for k, there are $O(n \log M_n/\epsilon) = O(n^2/\epsilon)$ many zones that can be created.

Every zone is divided into cells. Each cell is taken in such a way that it will cover $2\delta R$ of the inner circle of the zone, where R is the radius of this circle (**??**a). Thus, every zone between the circles with radious R and $(1+\delta)R$ has at most $2\pi R/\delta R = 4\pi n/\epsilon$ cells.

Since a list L_i has at most an entry for every cell created in every zone, its size can be at most $(n^2/\epsilon) \cdot (4\pi n/\epsilon) = O(n^3\epsilon^{-2})$. □

For function trim, the time required is $|E_i|$ to consider all vectors and, in the worst case, we have to check each vector in E_i with all the others leading to a running time of $O(|E_i|^2) = O(|L_i|^2)$. The running time for **??** is $n \cdot T(trim) = O(n|L_n|^2)$ and overall, from **??**, it requires time $O(n^5\epsilon^{-4} \log^2 M_n)$. The algorithm only stores at each step the list L_i so the space consumption is $O(n^2\epsilon^{-2} \log M_n)$.

Corollary 11. *For $\delta = \epsilon/2n$ the running time of* **??** *is $O(n^5\epsilon^{-4} \log^2 M_n)$ and space required is $O(n^2\epsilon^{-2} \log M_n)$.*

Algorithm 1: trim

input : $E \subset \mathbb{Z}^2$, $0 \leq \delta \leq 1$
output: a trimmed list L

```
sort(E)
for v_k ∈ E do
    i = 1
    while |v_{k+i}| ≤ (1+δ)|v_k| do
        if θ(v_{k+i}) − δ ≤ θ(v_k) ≤ θ(v_{k+i}) + δ then
            remove v_{k+i} from E
        i = i + 1
return E
```

Theorem 12. *For a set of vectors $S = \{v_i \mid v_i \in \mathbb{Z}^2, 0 \leq i \leq n\}$, every possible vector sum $v \in P_n$ can be approximated by a vector w such that*

$$\forall v \in P_n, \exists w \in L_n, \exists r_w \in \mathbb{Z}^2 : v = w + r_w,\ |r_w| \leq n\delta M_n,$$

Proof. The proof is by induction. The base step, it is easy to see that if we only have one element the theorem holds. The induction hypothesis

$$\forall v \in P_{n-1}, \exists w \in L_{n-1}, \exists r_w : v = w + r_w,\ |r_w| \leq (n-1)\delta M_{n-1}.$$

Algorithm 2: approx-2D-SS

input : $S \subset \mathbb{Z}^2,\ 0 \leq \epsilon \leq 1$
output: all approximation points L_n

$L_0 = \emptyset$
for $v_i \in S$ **do**
  $E_i = L_{i-1} \cup \{L_{i-1} + v_i\}$
  $L_i = \text{trim}(E_i, \epsilon/2n)$
return L_n

Now suppose $v \in P_n \setminus P_{n-1}$, because if $v \in P_{n-1}$ the theorem holds straight from the induction hypothesis. We write v as $v = z + v_n,\ z \in P_{n-1}$ and the induction hypothesis holds for z, thus

$$\exists p \in L_{n-1}, \exists r_p : z = p + r_p,\ |r_p| \leq (n-1)\delta M_{n-1}. \tag{2}$$

Since $p \in L_{n-1}$ this means that $p + v_n \in E_n$ and $L_n = trim(E_n)$. From the guarantee of function trim we know that

$$\exists q \in L_n : p + v_n = q + r_q,\ |r_q| \leq \delta |q|. \tag{3}$$

From **????** we get $v = z + v_n = p + v_n + r_p = q + r_q + r_p$. This proves that for $v \in P_n$ there exists a vector $q \in L_n$ that approximates it; but how close are they? We will bound the length $|r_q + r_p|$. From **????**,

$$|r_p| \leq (n-1)\delta \max\{L_{n-1}\} \leq (n-1)\delta M_n$$
$$|r_q| \leq \delta |q|,\ q \in L_n \implies |r_q| \leq \delta M_n$$

Thus

$$|r_q + r_p| \leq |r_q| + |r_p| \leq (n-1)\delta M_n + \delta M_n \leq n\delta M_n.$$

□

Setting $\delta = \epsilon/2n$ we can ensure that every possible vector sum will be approximated by a vector in L_n at most ϵM_n far. Implementing and testing the algorithm, much better bounds are obtained, see **??**.

3.2. A grid based algorithm

Our input is a list of vector $S = \{v_i \mid v_i \in \mathbb{Z}^2, i \leq n\}$. We define the list $E_i = L_{i-1} \cup \{w + v_i \mid w \in L_{i-1}\}$ and at each step i we trim it by a parameter δ to get the trimmed list $L_i = trim(E_i, \delta)$. Also, P_i is the set of all possible vector sums using a subset of the first i vectors of S, $P_i = \{\sum_j^i a_j v_j \mid a_j \in \{0,1\}, v_j \in S\}$.

It turns out that the same approximation ratio ϵM_n can be achieved by a faster algorithm that separates the plane into a grid, where M_n is the length of the largest vector in E_i. Instead of creating this different annulus-slice cells we have a regular orthogonal grid where each square cell has side $d = \epsilon M_n/2n$. Many thanks to Günter Rote for the fruitful conversation.

Let the cell side length be $d = \epsilon M_n/2n$, and for each $v(x,y) \in E_i$ store in the trimmed list L_i the vector with its coordinates rounded in the integer multiple of d:

$$\forall v(x,y) \in E_i\ \exists w(x',y') \in L_i : x' = \left\lfloor \frac{x}{d} \right\rfloor,\ y' = \left\lfloor \frac{y}{d} \right\rfloor$$

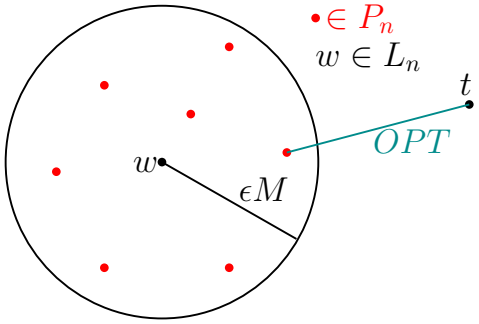

Figure 2: For every t the returned vector is at most ϵM from the optimum

and

$$dist(v, w) \leq \sqrt{2}d \leq \frac{\epsilon M_n}{n}$$

and the maximum value reaches when v and w are in the diagonal of the cell.

The whole grid has size $2M_n$ and since $d = \epsilon M_n/2n$ the grid has $O((M_n/d)^2) = O((n/\epsilon)^2)$ cells. In the worst case we will have a vector in every cell and this means that the time to traverse the lists E_i at each step is $O(n^2\epsilon^{-2})$. Since we have n lists the total running time of the new algorithm is $O(n^3\epsilon^{-2})$ and the space requirements are $O(n^2\epsilon^{-2})$.

Also, every $u \in P_n$ is the sum of at most n vectors from S. In the worst case, every time we call trim, we represent a vector $u \in E_i$ by another one that has distance form u at most d. In that case we lost at most $nd = \epsilon M_n$:

$$\forall v \in P_n \exists w \in L_n : dist(v, w) \leq \epsilon M_n.$$

Thus, for every given target vector t the algorithm will return an approximation solution that is $nd = \epsilon M_n$ far from being optimum **??**.

Corollary 13. *The grid-based algorithm runs in time $O(n^3\epsilon^{-2})$, requires space $O(n^2\epsilon^{-2})$ and returns a solution t' such that $dist(t, t') \leq OPT + \epsilon M_n$*

In 2D case there is a factor $\sqrt{2}$ that we also omit it from our approximation. This happens because the maximum distance inside a cell is not d but $\sqrt{2}d$. For dimension k the minimum distance is $\sqrt{k}d$ and this affects the algorithm in higher dimension.

3.3. A heuristic method

The grid-based algorithm "cuts" a constant d from every factor regardless of its length. If a vector has length 10,10^3 or 10^5, the algorithm will behave the same. On the other hand, it is fast, because it does not have to check any other vector; for every vector it sees it rounds it on the spot, thus having linear time in the size of the lists. We can make a version of a circular grid, where each cell does not have a constant side length. For small vectors we create smaller cells and as the length increases so does the cell side. This way we can provide a better experimental approximation ratio. At step i we have the list E_i and we will trim it with a factor δ. We will consider the polar coordinates of the vectors. For a vector $v(\phi, r)$ let $\phi = \theta(v)$ be the angle with the x axis and $r = |v|$ its

euclidean length. Let v be a vector in E_i and, to get the list L_i, we will replace v by $v' = (\phi', r')$. First round its angle in multiple of $\delta : \phi' = \lfloor \phi/\delta \rfloor$. Next, we round its length. The idea is to round in such a way that shorter vectors are approximated better than the longer ones. Rounding to a multiple of some d does not provide that. For rounding the lengths we will construct an array A with all the acceptable rounded lengths. The entries of A are the lengths $[1, (1+\delta), \ldots, (1+\delta)^i]$ for the minimum i such that $(1+\delta)^i > M_n$. Solving this inequality we get that $i = O(n \log M_n/\epsilon)$ and this is the size of A. Now, for a vector v we just make a binary search in A for $|v|$ that returns the zone such that $(1+\delta)^k \leq |v| \leq (1+\delta)^{k+1}$ and $r' = bin\_search(A, |v|) = (1+\delta)^k$. The space is divided in $O(\delta) = O(n/\epsilon)$ angles and $O(n \log M_n/\epsilon)$ different lengths. Each E_i has size $O(n^2\epsilon^{-2} \log M_n)$; we save the quadratic factor, but add a $\log |E_i|$ for the binary search. The whole algorithm runs in time

$$O(n|E_i| \log |E_i|) = O(n^3\epsilon^{-2} \log M_n \log \frac{n^2 \log M_n}{\epsilon^2})$$

and provides the same approximation, since $\forall v \in E_i, \exists w \in Li : dist(v, w) \leq \epsilon|w|$ as before. We believe that the better behaviour of this algorithm can be suited for an average case analysis, that will prove that the algorithm provides a better approximation ratio for the majority of $v \in P_n$. Also, the binary search may be dropped by using a method to round the lengths in $O(1)$ time, dropping this way the $\log |E_i|$ factor.

4. Minkowski Decomposition using 2*D*-SS

In this section, we will describe an algorithm for approximating MinkDecomp. The algorithm takes an input polygon Q, transforms it to an instance $\{S, t\}$ of $2D$-SS-approx and calls the algorithm for $2D$-SS-approx. Then it takes the output and converts it to an approximate solution to MinkDecomp.

Let Q be the input to MinkDecomp: $Q = \{v_i \mid v_i \in \mathbb{Z}^2, 0 \leq i \leq n\}$, such that $\sum_1^n v_i = (0, 0)$. First, we create the vector set $s(Q)$ by subtracting successive vertices of Q (in clockwise order): $s(Q) = \{v_0 - v_1, v_1 - v_2, \ldots, v_n - v_0\}$. Each vector in $s(Q)$ is called an *edge vector* and $s(Q)$ is called the *edge sequence* of Q. For each edge vector in $s(Q)$ we calculate its *primitive vector*.

Algorithm 3: approx-MinkDecomp

input : Q, ϵ
output: Q'

S = primitive_edge_sequence(Q)
//get the edge sequence for the two summands
s(A) = approx-2D-SS (S,(0,0))
s(B) = S\A
A=get-points(s(A))
B=get-points(s(B))
return *Q'= A + B*

Definition 14. Let $v = (a, b)$ be a vector and $d = gcd(a, b)$. The *primitive vector of* v *is* $e = (a/d, b/d)$.

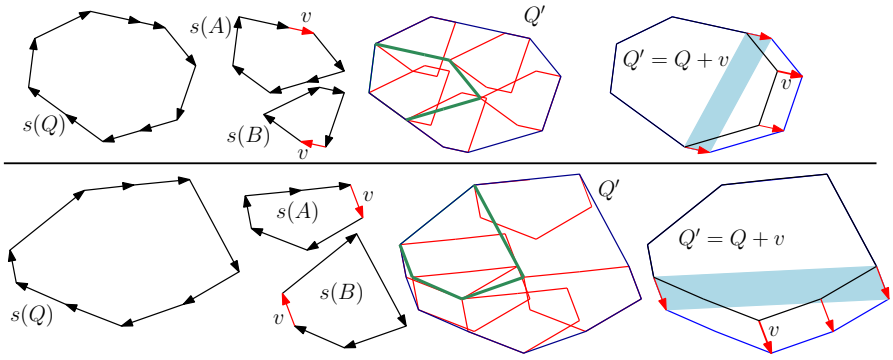

Figure 3: Two examples for two polygons Q. Their summands are shown and the red vector v is the new vector added to fill the gap. At the end, the new polygon Q' is Minkowski Sum of the two summands.

We get an edge vector $(x, y) \in s(Q)$ and calculate its primitive vector $e = (x/d, y/d)$, where $d = gcd(x, y)$. We could create the set S by adding in it d times the vector e for every $v \in s(Q)$ but this may create a set S that has exponential size to the edges of the polygon. Instead, we computer the scalars $d_1, \ldots, d_k$ by the following formulas

$$d_i = 2^i,\ i = 0, \ldots, \lfloor \log_2 d/2 \rfloor \text{ and } d_k = d - \sum_{i=1}^{\lfloor \log_2 d/2 \rfloor} d_i$$

We create the set S from the vectors $d_i e$ and repeat the procedure for all vectors $v \in s(Q)$. Notice that $\sum_1^k d_i = d$, so the primitive edge sequence also sums to $(0, 0)$. Using this construction, the primitive vectors added are at most $\log d$ for every $v \in s(Q)$ keeping the size of S polynomial with respect to the edges of Q. The primitive edge sequence uniquely identifies the polygon up to translation determined by v_0. This is a standard procedure as in [**? ?**].

The main defect in this approach is that the algorithm returns a sequence of vectors $S' \subset S$ that sum close to $(0, 0)$ but possibly not $(0, 0)$. This means the corresponding edge sequence does not form a closed polygon. To overcome this, we just add to $s(A)$ the vector v, from the last point to the first, to close the gap. If $s(A)$ sums to a point (a, b), by adding vector $v = (-a, -b)$ to $s(A)$ the edge sequence $s(A) \cup \{v\}$ now sums to $(0, 0)$. If we rearrange the vectors by their angles, they form a closed, convex polygon that is summand A. We do the same for the sequence $s(B)$. The vector added in $s(B)$ is $-v = (a, b)$ and this sequence (rearranged) also forms a closed, convex polygon. We name $s(A') = s(A) \cup \{v\}$, $s(B') = s(B) \cup \{v\}$ and take their Minkowski Sum $Q' = A' + B'$, where A' and B' are the convex polygons formed by $s(A')$ and $s(B')$. We measure how close Q' is to the input Q. Let D be the diameter of Q, the maximum distance between two vertices of Q.

Lemma 15. *Let Q be the input polygon and Q' be the polygon as discussed above. vol stands for volume. per for perimeter, i are the interior lattice points of a polygon and d_H the Huasdorff distance. We deduce that*

1. $vol(Q) \leq vol(Q') \leq vol(Q) + \epsilon D^2$
2. $per(Q) \leq per(Q') \leq per(Q) + 2\epsilon D$

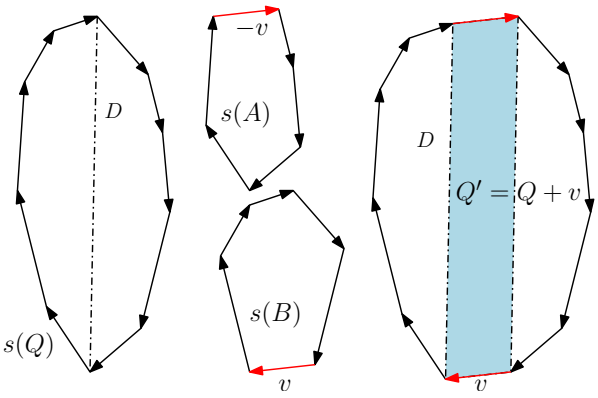

Figure 4: A worst case example where the vector v is (almost) perpendicular to the diameter D maximizing the extra volume added. More, D and v have no lattice points thus the interior points added are also maximum (D is not vertical).

3. $i(Q) \leq i(Q') \leq i(Q) + \epsilon D^2$
4. $d_H(Q, Q') \leq \epsilon/2D$

Proof. We observe that

$$s(Q') = s(A') \cup s(B') = s(A) \cup s(B) \cup \{v\} \cup \{\} \cup \{-v\} \implies$$
$$s(Q') = s(Q) \cup \{v\} \cup \{\} \cup \{-v\}.$$

This equals adding to Q a single segment s of length $|s| = |v|$ and $Q' = Q + s$. The length of vector v we add to close the gap, is the key factor to bound polygon Q'. From the guarantee of the $2D$-SS-approx algorithm we know that $s(A)$ (and respectively $s(B)$) sum to a vector with length at most $\epsilon \max\{L_n\}$. This is vector v and thus $|v| \leq \epsilon \max\{L_n\}$. Since $\max\{L_n\} \leq D$, we get $|s| = |v| \leq \epsilon D$.

From above we easily see that:

1. $per(Q) = \sum_{v \in s(Q)} |v|$, it follows $per(Q') = per(Q) + 2|v| \leq per(Q) + 2\epsilon D$.
2. $vol(Q') \leq vol(Q) + sD \leq vol(Q) + \epsilon D^2$
3. By Pick's theorem, $vol(Q) = i(Q) + b(Q)/2 - 1 \implies i(Q) = vol(Q) - b(Q)/2 + 1$. Note that $b(Q) = \sum_{\forall v \in s(Q)} d_v$ where $v = (x, y) \in s(Q)$ and $d_v = \gcd(x, y)$ as is **??**. Now, $i(Q') = i(Q) + i(sD)$ since sD is the maximum volume added and $i(sD) \leq sD - b(sD)/2 + 1 \leq sD - 1 \leq \epsilon D^2$. Thus, $i(Q') \leq i(Q) + \epsilon D^2$
4. If we "slide" Q by $s/2$ units in the direction of v, $d_H(Q, Q') = s/2 \implies d_H(Q, Q') \leq \epsilon/2D$

□

A bad example can be seen in **??**, where the added vector is (almost) perpendicular to D maximizing the extra volume and internal lattice points. **??** leads to following conclusion:

Corollary 16. *The proposed algorithm provides a $2\epsilon D$-solution for MinkDecomp-per-approx, a ϵD^2-solution for MinkDecomp-vol-approx and MinkDecomp-latt_p-approx and a $\epsilon/2D$-solution for MinkDecomp-d_H-approx.*

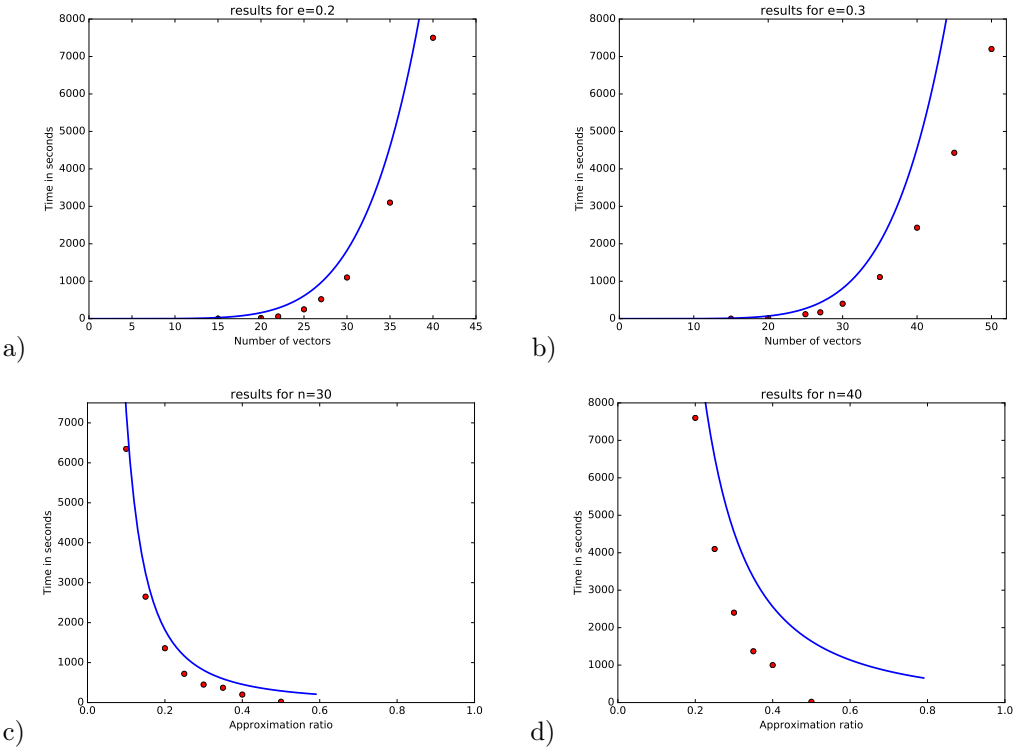

Figure 5: Experimental results for $2D$-SS-approx: a)$\epsilon = 0.2$ and b)ϵ=0.30, c)$n = 30$ and d)$n = 40$. The blue line is the expected time, the red dots our experiments.

5. Implementation and experimental results.

We implement both **????** in Python3. The code can be accessed through Github [1] and is roughly 750 lines long. We provide methods for either $2D$-SS-approx or MinkDecomp-μ-approx. To test **??** we created vectors v_i at random with $|v_i| \leq 5000$. For **??** we create random points and take their convex hull to form input polygon Q. All tests were executed in an Intel Core i5-2320 @ 3.00 GHz with 8Gb RAM, 64-bit Ubuntu GNU/Linux. Results for **??** are shown in **??** and for algorithm **??** in **??**. It is clear in **??** that our results stay below the expected time and behave analogously. In **??** the results obtained are much better than the proven bounds and in most cases volume and perimeter are almost the same and the polygons differ slightly.

6. Further work

In this work we presented some theoretical results concerning the approximation of two NP-hard problems: kD-SS and Minkdecomp. Additionally, we provide polynomial time approximation algorithms to solve these problems. The

[1] https://github.com/tzovas/Approximation-Subset-Sum-and-Minkowski-Decomposition

#vertices	#examples	vol(Q)/vol(Q')	per(Q)-per(Q')	Hausdorff	ϵ	time(secs)
3âĂŤ10	51	0,93	18,55	3,32	0,18	4,1
11âĂŤ16	45	0,977	3,43	1,81	0,33	126,4
17âĂŤ25	54	0,994	1,12	1,25	0,38	377,5

Table 1: Input polygon Q, output Q' ($per(Q) > 1000$). Gather examples by the number of their vertices. We measure volume, perimeter and Hausdorff distance and present their mean values.

next step is to associate these ideas with certain algebraic problems like approximate polynomial factoring or irreducibility testing.

Given a polynomial f_Q and its Newton polygon Q, we use the MimkDecomp approximation algorithm to find an approximation decomposition $Q' = A' + B'$. Using the irreducibility test in [**?**], we either find a binary bivariate factorization or that $f_{Q'}$ is irreducible. In the second case, we use the approximate polynomial factorization algorithm in [**?**]. All monomials of f_Q lie in the support of f'_Q therefore the corresponding coefficients should be same with the coefficients of these monomials in f'_Q. The new terms in f'_Q have coefficients, whose value is to be determined In other words, we need to determine, whether there exist valid coefficients for the monomials that correspond to lattice points in $Q' \setminus Q$.