Minkowski decomposition of convex lattice polygons

Ioannis Z. Emiris and Elias P. Tsigaridas

Department of Informatics and Telecommunications
National University of Athens, HELLAS {emiris,et}@di.uoa.gr

Summary. A relatively recent area of study in geometric modelling concerns toric Bézier patches. In this line of work, several questions reduce to testing whether a given convex lattice polygon can be decomposed into a Minkowski sum of two such polygons and, if so, to finding one or all such decompositions. Other motivations for this problem include sparse resultant computation, especially for the implicitization of parametric surfaces, and factorization of bivariate polynomials. Particularly relevant for geometric modelling are decompositions where at least one summand has a small number of edges. We study the complexity of Minkowski decomposition and propose efficient algorithms for the case of constant-size summands. We have implemented these algorithms and illustrate them by various experiments with random lattice polygons and on all convex lattice polygons with zero or one interior lattice points. We also express the general problem by means of standard and well-studied problems in combinatorial optimization. This leads to an improvement in asymptotic complexity and, eventually, to efficient randomized algorithms and implementations.

1 Introduction

In this paper we study the decomposition of convex polygons with integral vertices (also called lattice polygons) under the Minkowski sum, which is defined as follows:

Definition 1. *For any two subsets A and B in $\mathbb{Z}^2$, their* Minkowski sum *is $A \oplus B = \{a + b | a \in A, b \in B\}$. We call A and B the* summands *of $A \oplus B$.*

The definition of the Minkowski sum can be generalized to arbitrary dimension.

The decomposition problem has a great interest on its own. The recent work on toric Bézier patches (e.g [7, 12, 13]), in geometric modelling, motivates several questions around this problem, mainly testing whether a given lattice polygon can be written as a Minkowski sum of two such polygons and,

if so, finding one or all such decompositions. Another application in implicitization is the construction of matrices for the sparse resultant of 3 bivariate polynomials, cf. [13, sec.10.3] or [23].

One important application of general Minkowski decomposition is bivariate (and, eventually, multivariate) polynomial factorization. This is so because, given a bivariate (multivariate) polynomial, we can associate with it its Newton polytope. As first observed by Ostrowski ([15]), if the polynomial factors, then its Newton polytope decomposes.

First, we focus on Minkowski decompositions where at least one of the summands is of constant size, namely it is a line segment, a triangle or a quadrangle. These are particularly relevant when manipulating toric Bézier patches with depth. In [13], the authors "extend blossoming, degree elevation and implicitization techniques to arbitrary toric Bézier patches. [...] The key idea to each of these algorithms is to employ decompositions based on the Minkowski sum". They add [13, Sec. 10.1] that "This approach to evaluation, blossoming, and dual functionals works for any toric Bézier patch whose lattice polygon decomposes into the Minkowski sum of line segments and unit triangles". A key step in the algorithm of [23] (cf [13, Sec. 10.3]) for constructing resultant matrices for implicitization is to "decompose [Newton polygon] A into a Minkowski sum of simpler lattice polygons, typically line segments and triangles".

We estimate the hardness, from an asymptotic complexity viewpoint, and propose efficient algorithms for the case of constant-size summands. We relate Minkowski decomposition to the $k-$SUM problem, where an algorithm with time complexity $O(n^{\lceil k/2 \rceil})$ or $O(n^{\lceil k/2 \rceil} \lg n)$ exists but there are no matching lower bounds. We have implemented these algorithms and illustrated them on all lattice polygons with zero and one interior lattice points. Moreover, we performed experiments on various data sets against the algorithm of Gao and Lauder ([5]), which solves the general problem of Minkowski decomposition.

The decision problem of whether a lattice polygon admits a Minkowski decomposition is NP-complete [5]. In the same paper, a pseudo-polynomial algorithm is given with complexity in $O((nDE)^3)$, where n is the number of edges in the polygon and DE is their maximum integer length. Note that DE is exponential with respect to the bit size of the input, which is $O(n \lg (DE))$. We express the general problem by means of standard and well-studied problems in combinatorial optimization, such as the SUBSET-SUM problem. This leads to an algorithm that improves the above bound by a factor of nD. Our approach also leads immediately to approximation algorithms, to practical methods amenable to fast implementations and to a probabilistic algorithm. The implementation goes beyond the scope of the present paper.

Our paper is organized as follows. The next section defines the problem and overviews relevant work. Section 3 presents our approach for Minkowski decomposition of a lattice polygon to two summands, where at least one has a given constant number of edges. Section 4 presents the implementation of our algorithms, experiments with various random polygons and the decompo-

sition of all, up to unimodal transformations, lattice polygons with zero and one interior lattice points. In Section 5 we propose an algorithm for general decomposition of a lattice polygon, that has better time complexity than the one known so far. The last section presents our future research aims on the problem of Minkowski decomposition.

In what follows $O(\cdot)$ (resp. $O_B(\cdot)$) indicates arithmetic (resp. bit) complexity.

2 Definitions and previous work

The general problem that we deal with is:

Problem 2. MINKOWSKI-DECOMPOSITION Given a lattice polygon Q, with n vertices, decide if it is *decomposable*, that is if there are lattice polygons A and B such that $A \oplus B = Q$, where $\oplus$ denotes the Minkowski sum.

We are given a lattice polygon Q, with vertices $v_0, v_1, \ldots, v_{n-1}$, where $v_j \in \mathbb{Z}^2, 0 \leq j \leq n-1$. For every edge of the polygon we associate the vector $u_1 = (v_1 - v_0), \ldots, u_n = (v_0 - v_{n-1})$. The polygon is completely characterized by this sequence of vectors and the initial vertex v_0. In what follows edge means its vector.

Definition 3. *Let $u = (a, b)$ be a vector and $d = \gcd(a, b)$. The* primitive vector *of u is $e = (a/d, b/d)$.*

We denote the sequence of all vectors u_i as $\mathcal{U}$ and we call it the edge sequence. For every vector $u_i = (a_i, b_i)$ of Q we associate the primitive vector e_i, $1 \leq i \leq n$. We call the sequence of all primitive vectors, primitive edge sequence and denote it by $\mathcal{E}$. Additionally we call $\mathcal{A}$ the set of all possible vectors

$$\mathcal{A} = \{k_i e_i | 1 \leq i \leq n, 1 \leq k_i \leq d_i\}$$

where $d_i = \gcd(a_i, b_i)$. Let

$$D = \max\{d_1, \ldots, d_n\},$$
$$E = \max\{e_{1x}, e_{1y}, \ldots, e_{nx}, e_{ny}\},$$

where (e_{ix}, e_{iy}) are the coordinates of the primitive vector e_i. Moreover let g be the time needed for the computation of the gcd of two numbers of magnitude DE. Using the HALF-GCD algorithm ([22]) the gcd has bit complexity

$$g := O_B(\lg(DE) \lg^2 \lg(DE) \lg\lg\lg(DE))$$

The cost for computing $\mathcal{A}$ is $O_B(ng + nD\,\mathsf{M}(\max\{D, E\})) = O_B(nD\mathsf{M}(\max\{D, E\}))$, where $\mathsf{M}(\tau)$ is the time needed for the multiplication of two numbers of length τ. If we use FFT ([22]) the bit complexity of the multiplication is:

$$\mathsf{M}(\tau) = \tau \lg \tau \lg \lg \tau \tag{1}$$

In terms of arithmetic complexity the cost of computing $\mathcal{A}$ is $O(nD)$. However, when the computation of $\mathcal{A}$ is needed, its cost is dominated by other steps in the algorithms that we derive.

Lemma 4. *Consider a lattice polygon Q, such that $Q = A \oplus B$. Every edge of Q is determined uniquely as the Minkowski sum of an edge of A and a vertex of B, or as the sum of a vertex of A and an edge of B, or as a sum of two parallel edges of A and B.*

Hence, the set of the normals of Q is the union of the sets of the normals of A and B.

Using Lemma 4, it is easy to show ([5]):

Lemma 5. *A (lattice) polygon is a summand of Q if and only if its edge sequence is of the form $\{k_j e_j\}_{j \in J}$, where $J \subseteq \{1, \ldots n\}$, $0 \leq k_j \leq d_j$, $k_j \in \mathbb{Z}$ and $\sum_{j \in J} k_j e_j = (0, 0)$ (the sum of the vectors that correspond to its edges is zero).*

Theorem 6. *([5]) The decision problem of whether a lattice polygon has a Minkowski decomposition is NP-complete. There is an algorithm that decides if a polygon is decomposable, which has complexity $O(nDT)$, where T is the number of interior lattice points of the polygon. However, $T = O((nDE)^2)$ hence the complexity of the algorithm is $O(n^3 D^3 E^2)$.*

Note that, if a polygon is decomposable, there is possibly an exponential number of decompositions. The algorithm is pseudo-polynomial because its running time is polynomial in the length of the sides of the polygon rather than the logarithm of the lengths. In Section 5 we will present an algorithm that improves the time complexity by a factor of nD.

The bound $T = O(n^2 D^2 E^2)$ ([6], [8, Chap. 7]) is tight. One way that it can be achieved is as follows. Consider the lattice polygon of Fig. 1, where its edge sequence is

$$s_1 = (1, DE), s_2 = (2, DE), \ldots, s_n = (n, DE), (0, -nDE), (-\frac{n(n+1)}{2}, 0)$$

The area of the polygon is $\Theta(n^3 DE)$. If we assume that $n = \Theta(DE)$, then its area is $\Theta(n^2 D^2 E^2)$. The number of interior lattice points is asymptotically greater than the number of boundary lattice points. Also notice that #(Boundary points) $= O(n^2)$. Now, using Pick's formula

$$\text{Area} = \#(\text{Interior points}) + \frac{\#(\text{Boundary points})}{2} - 1$$

we can deduce that the number of interior lattice points is asymptotically $\Theta((nDE)^2)$.

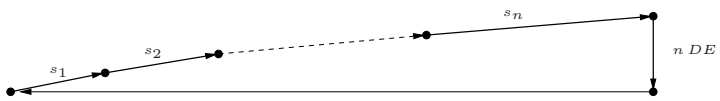

Fig. 1. A polygon with area $O((nDE)^2)$.

3 Constant-size summands

In this section we focus on the problem of decomposing a lattice polygon to two summands, where at least one has a given fixed number of edges. Remember that the input is a point sequence of cardinality n. We are dealing with two different problems:

Problem 7. Decision $k-$SUMMAND
Given a lattice polygon, decide whether there is a Minkowski decomposition to two summands, such that at least one of them has k edges.

Problem 8. Enumeration $k-$SUMMAND
Given a lattice polygon, enumerate all Minkowski decompositions of it, to two summands, where at least one of them has k edges.

In what follows, we examine in detail the cases where one summand is a segment (2−SUMMAND), a triangle (3−SUMMAND) or a quadrilateral (4−SUMMAND). The latter is generalisable to any fixed-size summand. We deal with both decision and enumeration problems. When we do not mention whether it is a decision or enumeration problem, it is clear from the context to which one we refer to.

The decision $k-$SUMMAND problem can be solved using the $k-$SUM problem. The latter is defined as follows:

Problem 9. $k-$SUM
Given a set of m integers and a goal sum S, decide whether there are k of them that add up to S.

The best known algorithm for the $k-$SUM problem has time and space complexity ([20, 21]) $O(m^{\lceil k/2 \rceil} \lg m)$ and $O(m^{\lceil k/2 \rceil})$, respectively. When k is odd the time complexity can be improved to $O(m^{\lceil k/2 \rceil})$. However, the derivation of a non-trivial lower bound for the $k-$SUM problem in the algebraic decision tree model or in the algebraic computation tree model is a major open problem. The only known result is due to Erickson ([3]), who proved an $\Omega(m^{\lceil k/2 \rceil})$ lower bound in a certain restricted variant of the linear decision tree model.

Theorem 10. *An instance of the $k-$SUMMAND problem can be transformed to an instance of $k-$SUM, such that the instance of $k-$SUMMAND has a solution if and only if the corresponding instance of $k-$SUM has a solution.*

Proof. Consider a lattice polygon Q, with n vertices. We compute $\mathcal{A}$, in $O(nD)$. Every vector in $\mathcal{A}$ is of the form $ke_i = k(e_{ix}, e_{iy})$, where $1 \leq i \leq n$ and $1 \leq k \leq d_i$. For every vector in $\mathcal{A}$ we associate the number $\alpha_{ik} = k(e_{ix} + Le_{iy})$, where $L = (k+1)DE$.

This new set of α_{ik}'s has at most nD elements. Let the target be $S = 0$. If we find k elements of this set, such that all of them correspond to different primitive vectors and that sum up to zero, then a $k-$summand exists.

Notice that the size of the instance of $k-$SUM is $O(nD)$. □

The above transformation allows us to solve the $k-$SUMMAND problem using the straight-forward algorithms of the $k-$SUM and as a consequence provides us with upper bounds for both time and space complexity. For $k = 2, 3, 4$ we have:

Decision $2-$SUMMAND can be solved in $O(nD \lg(nD))$ time and $O(nD)$ space.

Decision $3-$SUMMAND can be solved in $O(n^2D^2)$ time and $O(nD)$ space.

Decision $4-$SUMMAND can be solved in $O(n^2D^2 \lg(nD))$ time and $O(nD)$ space.

As for the general case, the decision $k-$SUMMAND problem can be solved in $O\left((nD)^{\lceil k/2 \rceil} \lg(nD)\right)$ and $O\left((nD)^{\lceil k/2 \rceil}\right)$ time, for k even and odd respectively and $O\left((nD)^{\lceil k/2 \rceil}\right)$ space.

We improve almost all the bounds in the subsequent sections.

Following [4], we give the following definition:

Definition 11. *Given two problems* PR_1 *and* PR_2 *we say that* PR_1 *is* $f(n)-$*solvable using* PR_2 *if and only if every instance of* PR_1 *of size* n *can be solved using a constant number of instances of* PR_2 *of at most linear size and* $O(f(n))$ *additional time. We denote this by*

$$\mathrm{PR}_1 \lll_{f(n)} \mathrm{PR}_2$$

Note that reduction implies, that when $f(.)$ is sufficiently small, lower bounds for the time complexity of PR_1 carry over to PR_2 and upper bounds for PR_2 hold for PR_1.

In order to prove lower bounds for the $k-$SUMMAND problem we use the following:

Theorem 12. $k-$SUM $\lll_{n \lg n}$ $k-$SUMMAND

Proof. Consider the sequence $\{a_i\}_{1 \leq i \leq n}$, where $a_i \in \mathbb{Z}$. We assume that the sequence is sorted; if it is not, then we sort it in $O(n \lg n)$ time. Let $M = \max_i |a_i|$ and $L = (k+1)M$. We form the sequence $\{s_i = a_i + L\}_{1 \leq i \leq n}$, where $0 \leq s_1 \leq \cdots \leq s_n$. Next we consider the edge sequence (see Figure 2):

$$(s_1, 1), (s_2, 1), \ldots, (s_n, 1), (0, -n), (-kL, -1), (-\sum_{i=1}^{n} a_i - (n-k)L, 1)$$

This sequence is an edge sequence of a lattice polygon, since both the sum of the ordinates and the sum of the abscissae of the vectors equal zero, and the angles of the edges are sorted, in clockwise order.

This polygon has a $k-$SUMMAND, if and only if there are k numbers in the sequence $\{a_i\}$ that sum up to zero, since in this case the edge sequence of the $k-$SUMMAND will be of form

$$(s_{i_1}, 1), (s_{i_2}, 1), \ldots, (s_{i_k}, 1), (0, -(k-1)), (-kL, -1)$$

where $i_j \in J$ and J is a subset of $\{1, \ldots, n\}$ of cardinality k. Proving the forward direction is easy. The reverse can be proven by considering the cases of summand edges and checking whether the $y-$coordinates sum to zero. □

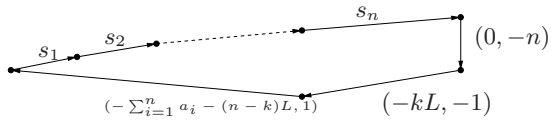

Fig. 2. Reduction of a $k-$SUM to a $k-$SUMMAND problem.

The previous reduction indicates that the $k-$SUMMAND problem is at least as hard as the $k-$SUM problem, maybe harder. Actually, this is the case when $D > 1$.

We consider as direction of a vector, the rational tangent of the angle between the positive $x-$semi-axis and the vector in a counter-clockwise orientation. For the algorithms that we will present direction and angle have the same meaning. The direction (essentially tangent) is represented by a pair of integer numbers, and each of them has magnitude at most DE. We can compare two directions in $O_B(\mathsf{M}(\lg(DE)))$ bit complexity, where $\mathsf{M}(\tau)$ is the time needed for the multiplication of two numbers of length τ (see Eq. (1)).

In what follows, we measure the algorithms' complexity using the arithmetic model (real RAM [17]). However, we can deduce the bit complexity if we multiply the derived complexities by either $\mathsf{M}(\lg(DE))$, if the comparison of directions is needed, or $\lg(DE)$, if the comparison of coordinates is needed.

Furthermore, we assume that v_0 is the bottom-left vertex, this means that v_1 is the vertex with the smallest direction. This is without loss of generality, since we can find this vertex in time $O(n)$. The key observation is that vectors in both $\mathcal{U}$ and $\mathcal{E}$ sequences are sorted in increasing order with respect to direction, for any lattice polygon. Taking this into account we deduce algorithms for the $\{2, 3, 4\}-$SUMMAND problem.

3.1 Line summand

Note that a $2-$SUMMAND exists if and only if there are at least two parallel edges. In order to decide if a line summand exists, we compute the vectors

that correspond to the edges of the lattice polygon, that is the sequence $\mathcal{U}$, in time $O(n)$.

Since $\mathcal{U}$ is sorted with respect to direction, we split it to two sequences, one that has directions in $[0, \pi)$, say $\mathcal{U}_1$ and one that has directions in $[\pi, 2\pi)$, say $\mathcal{U}_2$. We can do this in $O(n)$ time. We consider indices i and j that traverse $\mathcal{U}_1$ and $\mathcal{U}_2$, respectively. This means that i starts from the minimum direction of $\mathcal{U}_1$ and goes towards the maximum direction in $\mathcal{U}_1$ and the same for j in $\mathcal{U}_2$. If the direction of $\mathcal{U}_1[i]$ is smaller (resp. greater) than $\delta - \pi$ where δ is the direction of $\mathcal{U}_2[j]$, we advance i (resp. j). If the direction of $\mathcal{U}_1[i]$ is smaller than the direction of $\mathcal{U}_2[j]$ by π, then a line summand exists. Both the time and space complexity are $O(n)$.

If we are interested in the enumeration 2−SUMMAND problem then we have to find all the vectors with directions differing by π and for every such pair, say with indices i and j, we compute the corresponding primitive vectors, say e_i and e_j, and we output d pairs of vectors, (ke_i, ke_j), where $1 \leq k \leq d$ and $d = \min\{d_i, d_j\}$. Then we advance both indices i and j and continue the algorithm. This algorithm has time complexity $O(n + t)$, where t is the number of all possible line summands, which is at most $\frac{nD}{2}$.

The previous discussion leads to the following theorem:

Theorem 13. *There is an algorithm for the decision* 2−SUMMAND *problem that has time complexity* $O(n)$. *There is an algorithm for the enumeration* 2−SUMMAND *problem that has time complexity* $O(n+t)$. *The space complexity for both algorithms is* $O(n)$.

Both algorithms are optimal.

3.2 Triangle summand

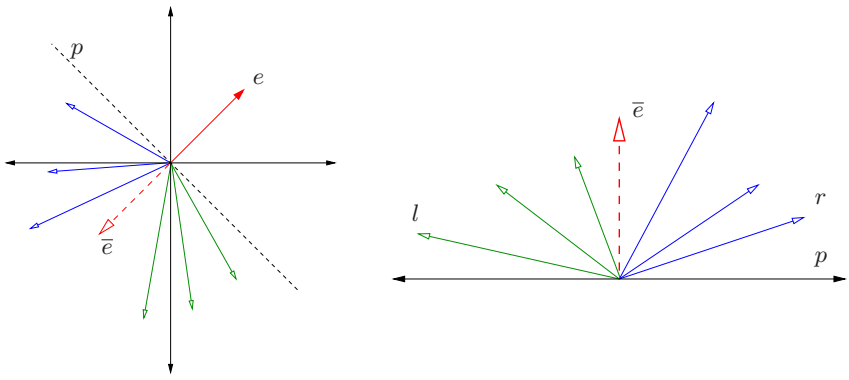

Fig. 3. Computing the triangle summands

In order to solve the decision problem for the 3−SUMMAND, first we compute the primitive edge sequence $\mathcal{E}$ and sequence $\mathcal{A}$, in time $O(nD)$. Note that

$|\mathcal{A}| = O(nD)$. Since $\mathcal{A}$ contains scalar multiples of the vectors in $\mathcal{E}$, we can assume that it is sorted in increasing order, first with respect to direction and then with respect to x and y coordinates.

If a triangle summand exists then for some primitive vector $e \in \mathcal{E}$ there are two indices r and l, where $1 \leq r, l \leq |\mathcal{A}|$, such that the direction of the vector $w = \mathcal{A}[r] + \mathcal{A}[l]$ is opposite to that of e.

Consider the case of the left half of Figure 3. Vector $e = (e_x, e_y)$ is a primitive vector and the dotted vector $\overline{e} = (\overline{e}_x, \overline{e}_y) = (-e_x, -e_y)$ is its opposite. We consider an axis perpendicular to e, this is line p in the figure, and only the vectors from $\mathcal{A}$ that lie in the same half-plane as $\overline{e}$ does. We can find these vectors in time $O(nD)$, since $\mathcal{A}$ is sorted with respect to direction. We denote this sequence of vectors also by $\mathcal{A}$. Note that this sequence is also sorted with respect to direction.

With a suitable axis rotation, the case of the left half of Figure 3 is equivalent to the one of the right half. From now on we will refer to the right half since it is more intuitive. All vectors, except $\overline{e}$, are elements of $\mathcal{A}$.

In order to find if a triangle summand exists, we start with indices $r = 1$ and $l = |\mathcal{A}|$, assuming that $\mathcal{A}$ is sorted from right to left as in Figure 3 (right). Then, we examine all vectors of $\mathcal{A}$ trying to find values for the indices r and l such that the direction of $w = \mathcal{A}[r] + \mathcal{A}[l]$ is equal to the direction $\overline{e}$. If this happens, then we check if $-\frac{w_x}{e_x}$ and $-\frac{w_y}{e_y}$ are the same integer between 1 and d. If this is the case then a triangle summand exists, otherwise we advance both r and l. If the direction of w is smaller, respectively larger, than the direction of $\overline{e}$, we advance r, respectively reduce l, by 1.

We traverse $\mathcal{A}$ in time $O(nD)$ and since we have to do this for every vector in the primitive edge sequence, the total time for the decision algorithm is $O(n^2D)$ and its space complexity is $O(nD)$.

If we are interested in the enumeration problem we advance both indices r and l, when we find direction equality, and so we enumerate all the possible triangle summands. The total time for the algorithm is $O(n^2D + t)$, where t is the number of all possible triangle summands.

The previous discussion leads to the following theorem:

Theorem 14. *There is an algorithm for the decision* 3−SUMMAND *problem that has time complexity* $O(n^2D)$. *There is an algorithm for the enumeration* 3−SUMMAND *problem that has time complexity* $O(n^2D + t)$. *The space complexity for both algorithms is* $O(nD)$.

There is an alternative algorithm for the decision problem that runs in $O(n^3)$ arithmetic complexity or $O_B(n^3g)$ bit complexity and space complexity $O(n \lg (DE))$. First we compute the primitive edge sequence $\mathcal{E}$, in time $O_B(ng)$ bit complexity. If a triangle summand exists then at least one of the $O(n^3)$ systems of Diophantine equations and inequalities

$$a_i e_{ix} + a_j e_{jx} + a_k e_{kx} = 0$$
$$a_i e_{iy} + a_j e_{jy} + a_k e_{ky} = 0$$
$$1 \leq a_i \leq d_i, 1 \leq a_j \leq d_j, 1 \leq a_k \leq d_k$$

where $1 \leq i < j < k \leq n$, must have an integer solution. As for the bit complexity of the solution of the above system, it is dominated by the computation of $\gcd(e_{ix}, e_{jx}, e_{kx})$ and $\gcd(e_{iy}, e_{jy}, e_{ky})$ and so it is $O_B(g)$.

As for the enumeration problem, we must solve all these systems and thus the algorithm has $O(n^3 + t)$ arithmetic complexity or $O_B(n^3 g + t)$ bit complexity.

From the previous discussion follows that the decision 3−SUMMAND can be solved in polynomial time arithmetic complexity. Typically, n is large compared to D, so Th. 14 is preferable, hence we do not extend this approach further.

3.3 Quadrangle summand

In order to deduce an algorithm for the decision 4−SUMMAND problem, we compute the primitive edge sequence $\mathcal{E}$ and then the sequence $\mathcal{A}$ in time $O(nD)$. We compute the sequence of all vectors that are sums of two distinct vector of $\mathcal{A}$ in time $O(n^2D^2)$. We call this sequence $\mathcal{A}_2$. We sort $\mathcal{A}_2$, first with respect to the $x-$coordinate and then with respect to the $y-$coordinate, in time bounded by $O(n^2D^2 \lg(nD))$.

For every vector in $\mathcal{A}_2$, we search $\mathcal{A}_2$ for a vector with opposite x and y coordinates. We can do the search in $O(\lg(nD))$ time. Thus the total time of this decision algorithm is $O(n^2D^2 \lg(nD))$ and its space complexity is $O(n^2D^2)$.

If we want to enumerate all the possible quadrangle summands we perform the search for every vector in $\mathcal{A}_2$. Thus the complexity for the enumeration algorithm is $O(n^2D^2 \lg(nD) + t)$, where t is the number of all possible quadrangle summands.

In practice we can eliminate the logarithmic factors since we can use a hash structure in order to keep the elements of $\mathcal{A}_2$. If we want to reduce the space requirements, we can use a special data structure that produces (in increasing or decreasing order) all possible sums of two vectors (see [20]) which has space complexity $O(nD)$ and access time $O(\lg(nD))$.

The previous discussion leads to the following theorem:

Theorem 15. *There is an algorithm for the decision* 4−SUMMAND *problem that has time complexity* $O(n^2D^2 \lg(nD))$. *There is an algorithm for enumeration* 4−SUMMAND *problem that has time complexity* $O(n^2D^2 \lg(nD)+t)$. *The space complexity for both algorithms is* $O(nD)$.

3.4 Summand with k edges

For the general $k-$SUMMAND problem we have to distinguish between two cases, when k is odd or even. As in the previous sections first we discuss the decision problem.

In both cases, first we compute the sequences $\mathcal{E}$ and $\mathcal{A}$ and then we compute all the possible sums of $\lfloor \frac{k}{2} \rfloor$ vectors of $\mathcal{A}$. Since the size of $\mathcal{A}$ is $O(nD)$, this computation can be done in $O((nD)^{\lfloor \frac{k}{2} \rfloor})$ and the space required is of the same order. We call this sequence $\mathcal{A}_{\frac{k}{2}}$.

If k is odd then we sort $\mathcal{A}_{\frac{k}{2}}$, first with respect to direction, then with respect to $x-$coordinate and finally with respect to $y-$coordinate. This can be done in $O((nD)^{\lfloor \frac{k}{2} \rfloor} \lg(nD))$. After this we proceed as in the $3-$SUMMAND case. For every primitive vector $e \in \mathcal{E}$, we traverse $\mathcal{A}_{\frac{k}{2}}$ with two pointers: One that goes from left to right and another from right to left, in order to find two vectors of $\mathcal{A}_{\frac{k}{2}}$ such that the direction of their sum is opposite to the direction of e. The time complexity of this algorithm is $O(n^{\lceil \frac{k}{2} \rceil} D^{\lfloor \frac{k}{2} \rfloor} + (nD)^{\lfloor \frac{k}{2} \rfloor} \lg(nD))$ or $O(n^{\lceil \frac{k}{2} \rceil} D^{\lfloor \frac{k}{2} \rfloor})$ if we assume that $n > \lg(nD)$.

If k is even then we proceed as in the $4-$SUMMAND case, that is we sort $\mathcal{A}_{\frac{k}{2}}$, first with respect to the $x-$coordinate and then with respect to the $y-$coordinate. This can be done in $O(n^{\lceil \frac{k}{2} \rceil} D^{\lfloor \frac{k}{2} \rfloor} \lg(nD))$. Note that since k is even $\lceil \frac{k}{2} \rceil = \lfloor \frac{k}{2} \rfloor$. Finally, for every vector of $\mathcal{A}_{\frac{k}{2}}$, we search $\mathcal{A}_{\frac{k}{2}}$, for a vector with opposite x and y coordinates. The search can be performed in $O(\lg(nD))$ time. Thus the total time for the algorithm is $O(n^{\lceil \frac{k}{2} \rceil} D^{\lfloor \frac{k}{2} \rfloor} \lg(nD))$.

As for the enumeration problem, in both cases, we continue the search when a $k-$SUMMAND is found.

The previous discussion leads to the following theorem:

Theorem 16. *There is an algorithm for the decision $k-$SUMMAND problem that has time complexity $O(n^{\lceil \frac{k}{2} \rceil} D^{\lfloor \frac{k}{2} \rfloor} \lambda)$, where $\lambda = 1$ if k is odd and $\lambda = \lg(nD)$ if k is even.*

There is an algorithm for the enumeration $k-$SUMMAND problem that has time complexity $O(n^{\lceil \frac{k}{2} \rceil} D^{\lfloor \frac{k}{2} \rfloor} \lambda + t)$, where t is the number of all possible decompositions to two summands, where at least one of them has k edges.

The space complexity for both algorithms is $O((nD)^{\lfloor \frac{k}{2} \rfloor})$.

4 Implementation and application to polygons with zero and one lattice interior point

This section sketches our implementations of the above algorithms, their application to computing all Minkowski decompositions of all polygons with one lattice interior point as well as polygons without interior lattice points and experiments with various datasets.

We implemented our algorithms in C++ and we used the geometric library CGAL ([1]). CGAL has classes that refer to points, vectors and polygons and operations on them. Additionally, there is a class for the direction (in $\mathbb{Q}$) of a vector and comparisons between them. Our code is freely available at http://www.di.uoa.gr/~et.

We performed all experiments on a 2.6GHz Pentium, with 1GB memory, running Linux, with kernel version 2.6.10. We compiled the programs with g++, v. 3.3.5, with options -O3 -DNDEBUG.

4.1 Experiments with random lattice polygons

We performed various experiments so as to check the efficiency of our algorithms for the decision $\{2,3,4\}-$SUMMAND problems. We refer to these algorithms as `ET(s)`, `ET(t)` and `ET(q)`. We also implemented in CGAL the algorithm of Gao and Lauder ([5]), which decides the general problem of Minkowski decomposition. We refer to this algorithm as `GL`. The running times of the experiments are presented in Table 1. All the times are in msec.

Columns A_k, B_k, C_k and D_k, where $k \in \{10, 20, 30, 40, 50, 60, 70\}$, refer to 500 lattice polygons with k edges, sampled in $[0, 3000] \times [0, 3000]$. The polygons in B_k, C_k and D_k, were constructed such that they have at least one segment, one triangle, one quad summand, respectively. Columns E_k, refers to 500 lattice polygons that are the convex hull of 50 random lattice points in $[0, k] \times [0, k]$.

In all cases our algorithms are considerably faster. This is the case because the `ET` algorithms are dedicated for constant-size summands and solve a polynomial problem, while `GL` is an algorithm for the general problem which is NP-complete. Special notice must be paid to the running times of `ET(s)`, which are more or less the same on all data sets. The reason is that the complexity of `ET(s)` depends linearly only on the number of edges of the polygon. Additionally, most of the time of the `GL` algorithm is spent for the computation of the integer points of the tested lattice polygon. As a consequence the running times of `GL` for the data sets A_k, B_k, C_k, D_k, where the polygons have a large number of lattice points, are not satisfactory. However in E_k, where the polygons have a small number of lattice points, the running times of `GL` are quite competitive.

Even though current experiments show the superiority of the special purpose algorithms, a more careful implementation and a more detailed experimental analysis is needed.

4.2 Lattice polygons with one lattice interior point

There are only 16 lattice polygons with one lattice interior point, modulo unimodular transformations, as proven in [18] (see also [19]). We compute all decompositions into Minkowski summands. The results are in Figure 4 and 5. These polygons are of particular interest for toric Bézier patches ([7, 12, 13]).

	A_{10}	A_{20}	A_{30}	A_{40}	A_{50}	A_{60}	A_{70}
ET(s)	0.007	0.01	0.02	0.03	0.04	0.04	0.05
ET(t)	1.1	5.1	9.6	16.5	30.1	40.3	56.2
ET(q)	1.6	6.2	11.6	19.1	34.4	46.1	65.1
GL	11150	15270	22050	23995	23370	26205	27315

	B_{10}	B_{20}	B_{30}	B_{40}	B_{50}	B_{60}	B_{70}
ET(s)	0.004	0.006	0.008	0.01	0.01	0.02	0.02
ET(t)	4.3	7.2	9.7	17.3	25.5	39.3	49.9
ET(q)	3.3	9.2	12.1	19.2	29.7	44.8	57.4
GL	27330	50105	37930	53635	46345	54205	36475

	C_{10}	C_{20}	C_{30}	C_{40}	C_{50}	C_{60}	C_{70}
ET(s)	0.003	0.006	0.008	0.01	0.01	0.02	0.02
ET(t)	1.8	3.6	10.4	16.3	27.8	37.5	53.7
ET(q)	2.6	5.3	12.7	18.2	33.0	43.2	62.1
GL	25630	27065	52810	37215	84510	86555	51465

	D_{10}	D_{20}	D_{30}	D_{40}	D_{50}	D_{60}	D_{70}
ET(s)	0.003	0.006	0.008	0.01	0.01	0.02	0.02
ET(t)	1.6	5.2	9.4	19.3	28.3	43.2	54.3
ET(q)	1.9	5.5	11.2	22.4	33.5	49.5	63.1
GL	32950	78840	72230	71240	75805	64690	73335

	E_{10}	E_{20}	E_{30}	E_{40}	E_{50}	E_{60}	E_{70}
ET(s)	0.002	0.003	0.003	0.003	0.004	0.004	0.004
ET(t)	0.1	0.4	0.3	0.4	0.4	0.5	0.5
ET(q)	0.1	0.3	0.4	0.4	0.5	0.6	0.6
GL	0.1	0.2	0.4	0.7	1.2	1.6	2.2

Table 1. Experimental results

4.3 Lattice polygons without interior lattice points

We have computed all the decompositions of lattice polygons with zero interior lattice points and area less than or equal to 3. All possible decompositions are in Figure 6.

Using the enumeration algorithms for $\{2,3,4\}-$SUMMAND of the previous section we can decompose all the polygons, up to unimodular transformations, without interior lattice points. All the decompositions are presented in Figure 7.

First we need to define all such polygons and so we state the following theorem:

Theorem 17. *[19] Any lattice polygon without interior lattice points is unimodular equivalent to a polygon $T_{m,n}$ with vertices $\{(0,0),(0,1),(m+n,0),(n,1)\}$, where $m,n \geq 0$, or to the triangle Δ_2 with vertices $\{(0,0),(2,0),(0,2)\}$.*

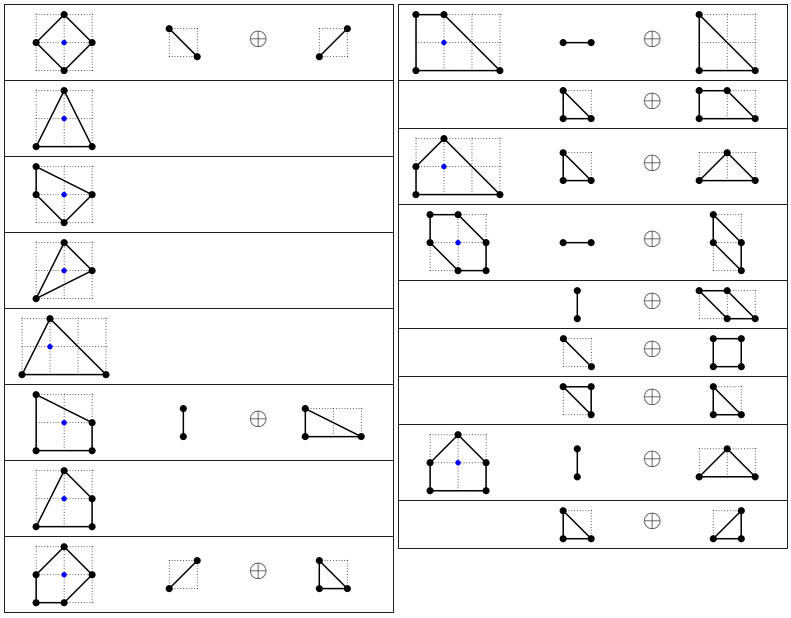

Fig. 4. Minkowski decomposition of lattice polygons, with one interior lattice point (continued in next figure).

The edge sequence of Δ_2 is $\{2(1,0), 2(-1,1), 2(0,-1)\}$. It is easy to see that Δ_2 admits a Minkowski decomposition, to two equal triangle summands. If Δ_1 is the triangle with vertices $\{(0,0),(1,0),(0,1)\}$, then $\Delta_2 = \Delta_1 \oplus \Delta_1 = 2\Delta_1$. Notice that Δ_2 and Δ_1 are homothetic. The decomposition is illustrated in the first row of Figure 7.

In order to decompose all lattice polygons $T_{m,n}$ we distinguish the following cases:

- $m \geq 1, n = 0$
  In this case $T_{1,0}$ is a triangle with vertices $\{(0,0),(m,0),(0,1)\}$ and its edge sequence is $\{(m,0),(-m,1),(0,1)\}$. It very easy to check that this triangle is irreducible with respect to Minkowski sum. We can reach the same result if we use the approach of [5, Th. 8] by checking that $\gcd(0,1,m) = 1$.
- $m = 0, n \geq 1$

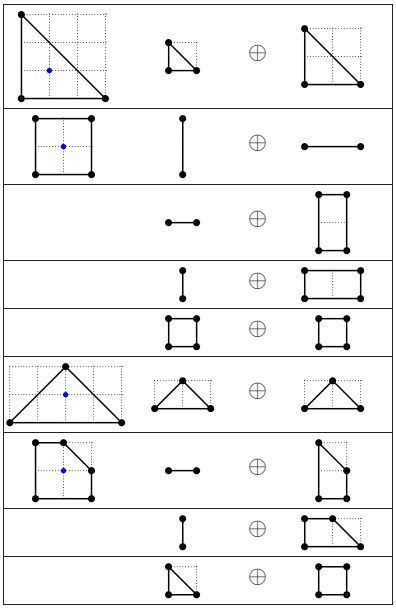

Fig. 5. Continued: Minkowski decomposition of lattice polygons, with one interior lattice point.

In this case, $T_{0,n}$ is a rectangular with vertices $\{(0,0),(n,0),(n,1),(0,1)\}$ and its edge sequence is $\{(n,0),(0,1),(-n,0),(0,-1)\}$. The Minkowski decomposition of this polygon is either two line segments (this is the first equality of the second row of Figure 7) or a line segment and a rectangle (this is the second equality of the second row of Figure 7, where $1 \leq k \leq n$). The last equality of the second row of Figure 7 presents the unique Minkowski decomposition of $T_{0,n}$ to $n+1$ irreducible summands.

- $m \geq 1, n \geq 1$
  In this case, $T_{m,n}$ is a trapezoid with vertices $\{(0,0),(m+n,0),(n,1),(0,1)\}$ and its edge sequence is $\{(m+n,0),(-m,1),(-n,0),(0,-1)\}$. We can decompose $T_{m,n}$, either to a Minkowski sum of a line segment and a trapezoid (this is the first equality of the third row of Figure 7, where $1 \leq k \leq n$) or to a Minkowski sum of triangle and a line segment (this is the second equality of the third row of Figure 7). The last equality of the third

row of Figure 7 presents the unique Minkowski decomposition of $T_{0,n}$ to irreducible summands.

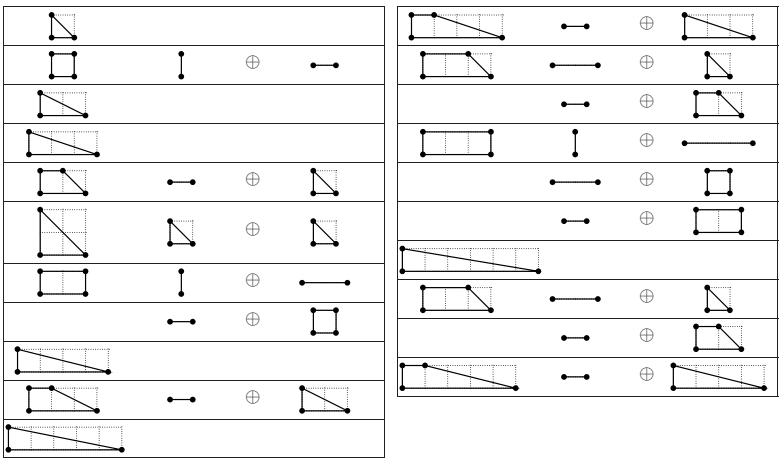

Fig. 6. Minkowski decomposition of lattice polygons, with zero interior lattice points and area less than or equal to 3.

5 Improving the general decomposition algorithm

In this section we return to the general problem, MINKOWSKI-DECOMPOSITION, and propose a different approach than the one by Gao and Lauder ([5]). This shall improve the asymptotic complexity. More importantly, we expect our approach to lead to efficient implementations in practice and to permit randomized and approximation algorithms.

The main idea is that it suffices to find combinations of vectors such that their sum is zero. Note that the sum of a subset of the vectors is zero iff both the sum of $x-$coordinates and the sum of $y-$coordinates is zero.

We need the definition of the SUBSET-SUM problem, which is an NP-complete problem:

Problem 18. SUBSET-SUM

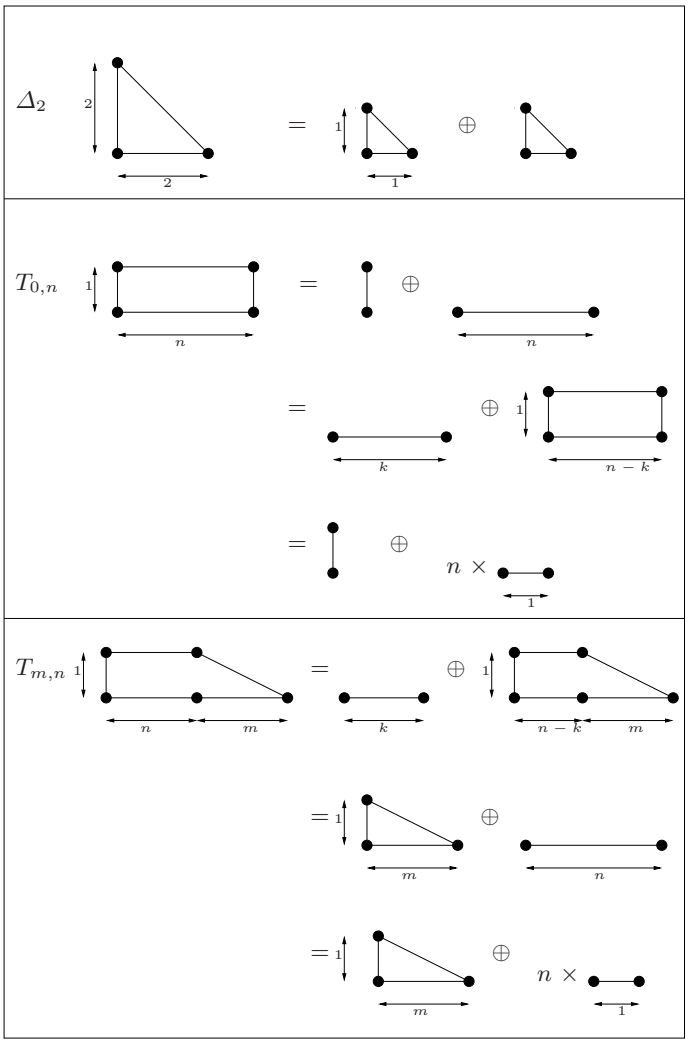

Fig. 7. Minkowski decompositions of all lattice polygons without interior lattice points.

Given a set of n positive integers and a goal sum S, decide whether there exists a subset, such that its elements add up to S.

We use the following transformation:

Lemma 19. *An instance of a* MINKOWSKI-DECOMPOSITION *problem can be transformed to an instance of a* SUBSET-SUM *problem, such that the instance of* MINKOWSKI-DECOMPOSITION *admits a solution if and only if the instance of* SUBSET-SUM *admits a solution.*

Proof. Let Q be a lattice polygon with n vertices, and also let DE be the maximum integer length of its coordinates. We compute the primitive edge sequence $\mathcal{E}$. We consider the coordinates of the primitive vectors e_i and we associate to every vector the positive number $a_i = e_{ix} + Le_{iy} + DE$, where $1 \leq i < n$ and L sufficiently large, for example $L = nDE$. We add the quantity DE to every a_i so that $a_i > 0, 1 \leq i < n$. We consider d_i copies of each a_i, thus the total number of them is $\sum_{i=1}^{n} d_i = O(nD)$.

Now our transformation is complete since the polygon Q is decomposable if and only if there is a subset of the a_i's that sums up to zero. The key idea is that if an a_i belongs to a subset that sums up to zero then its corresponding edge belongs to a summand of the polygon and vice versa.

Notice that the transformation takes $O(nD)$ time, which is also the size of the instance of the SUBSET-SUM problem. □

The time for solving SUBSET-SUM via dynamic programming is $O(N^2W)$ (see for example [9], [2]) where N is the cardinality of the set and W is an upper bound on the absolute value of every element. In our case $N = O(nD)$ and $W = O(nDE^2)$, thus the total complexity of the algorithm is $O(n^3D^3E^2)$. The complexity of this algorithm is the same as the complexity of the algorithm in [5].

However, our approach may use the dynamic programming paradigm and is completely different from the one in [5], since we completely avoid the computation of the interior lattice points of the polygon. Moreover, if we use a balancing algorithm ([16]) for solving the corresponding SUBSET-SUM problem, which is the best available and has time complexity $O(NW)$, then the complexity of our algorithm is $O(n^2D^2E^2)$. Thus, we improve the complexity by a factor nD.

So we have the following theorem

Theorem 20. *There is an algorithm for the decision* MINKOWSKI-DECOMPOSITION *problem that has $O(n^2D^2E^2)$ arithmetic complexity.*

6 Future work

In order to enumerate all possible summands of a polygon, following the approach of Section 5, we can use the various algorithms for the PARTITION problem (refer to [9]). However, a detailed experimental analysis is needed in order to decide the best approach.

Additionally, our approach of section 5 can easily lead to a randomized algorithm. We pick $L = nDE$. The quantities are of the form $a_i = e_{ix} + Le_{iy}$, of max-value $E(L+1)$, and there are $d_i \leq D$ copies of each a_i. So the maximum sum value is $nDE(L+1) = O((nDE)^2)$.

In order to check if a sum vanishes $\bmod p$, where $p > 0$ is a random integer, we have to bound the probability $Pr[failure]$, that random sum $S \in [0, nDE(L+1)]$, vanishes $\bmod p$, when $S \neq 0$, where p is a prime uniformly

distributed in $[2, \ldots, x]$. We can do this using the randomized algorithm for verifying equality of strings from [14].

Lemma 21. *[14] Let a, b be two numbers with τ bits each. If $a \neq b$, then*

$$Pr[failure] = Pr[a = b \mod p] < \frac{1}{2}$$

where p is a prime uniformly distributed in $[2, \ldots, 4\tau^2]$.

In our case $a = 0$ and $b = S$, so we need $\tau \simeq 2 \lg (nDE)$ bits to encode them. Thus we can use the previous lemma, if we choose a prime in $[2, \ldots, 4\tau^2]$, to obtain $Pr[failure] < \frac{1}{2}$.

Last, but not least, the reduction to the SUBSET-SUM problem, can lead to approximate algorithms for the Minkowski decomposition. The first step towards this direction may be the adoption of the first fully polynomial-time approximation scheme for the SUBSET-SUM problem that Ibbara and Kim suggested in [10] or the adoption of the best known, so far, approximation algorithm of Kellerer et al ([11]).

Acknowledgments

Research supported by project 70/3/7392: the project is co-funded by the European Social Fund and National Resources (EPEAEK II) – PYTHAGORAS.

References

1. CGAL: Computational geometry algorithms library. http://www.cgal.org.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.
3. J. Erickson. Lower bounds for satisfiability problems. *Chicago J. of Theoretical Computer Science*, 8, 1999. http://cjtcs.cs.uchicago.edu/articles/1999/8/contents.html
4. A. Gajentaan and M. H. Overmars. On a class of O(n^2) problems in computational geometry. *Computational Geometry, Theory and Applications*, 5(3):165–185, October 1995.
5. S. Gao and A. G. B. Lauder. Decomposition of polytopes and polynomials. *Discrete and Computational Geometry*, 26:89–104, 2001.
6. S. Gao and A. G. B. Lauder. Fast absolute irreducibility testing via Newton polytopes. *preprint*, 2004.
7. R. Goldman. *Pyramid Algorihtms: A dynamic approach to curves and surfaces for geometric modelling*. Morgan Kaufmann, 2002.
8. J. E Goodman and J. O' Rourke. *Handbook of computational geometry*. Elsevier science, Amsterdam, 1995.
9. E. Horowitz and S. Shani. Computing partitions with applications to the knapsack problem. *Journal of ACM*, 21(2):277–292, April 1974.

10. O. Ibarra and C. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.
11. H. Kellerer, R. Mansini, U. Pferschy, and M. G. Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *J. Comput. Syst. Sci.*, 66(2):349–370, 2003.
12. R. Krasauskas. Toric surface patches: Advances in geometrical algorithms and representations. *Adv. Comput. Math.*, 17(1-2):89–113, 2002.
13. R. Krasauskas and R. Goldman. Toric Bezier Patches with Depth. In R. Goldman and R. Krasauskas, editors, *Topics in Geometric Modeling and Algebraic Geometry*, volume 334, pages 65–91. AMS Mathematics of Computation, 2003.
14. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
15. A. M. Ostrowski. Über die Bedeutung der Theorie der konvexen Polyeder für die formale Algebra. *Jahresberichte Deutsche Marth. Verein* 30 (1921), 98–99.
16. D. Pisinger. *Algorithms for the Knapsack problems*. PhD thesis, Department of Computer Science, University of Kopehagen, February 1995.
17. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 3rd edition, October 1990.
18. S. Rabinowitz. A census of convex lattice polygons with at most one interior lattice point. *Ars Combinatorica*, 28:83–96, 1989.
19. J. Schicho. Simplification of surface parametrizations - a lattice polygon approach. *J of Symbolic Computation*, 36:535–554, 2003.
20. G. Woeginger. Open problems around exact algorithms. Manuscript, TU Eindhoven, 2004.
21. G. Woeginger. Exact algorithms for NP-hard problems: A survey. In M. Juenger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization - Eureka! You shrink!*, volume 2570, pages 185–207. LNCS, Springer, 2003.
22. C.K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, 2000.
23. S. Zube. The n-sided toric patches and the A-resultants. *Computer Aided Geometric Design*, 17:695–714, 2000.