Geometric optimization and sums of algebraic functions *

Antoine Vigneron †

October 21, 2011

Abstract

We present a new optimization technique that yields the first FPTAS for several geometric problems. These problems reduce to optimizing a sum of non-negative, constant description complexity algebraic functions. We first give an FPTAS for optimizing such a sum of algebraic functions, and then we apply it to several geometric optimization problems. We obtain the first FPTAS for two fundamental geometric shape matching problems in fixed dimension: maximizing the volume of overlap of two polyhedra under rigid motions, and minimizing their symmetric difference. We obtain the first FPTAS for other problems in fixed dimension, such as computing an optimal ray in a weighted subdivision, finding the largest axially symmetric subset of a polyhedron, and computing minimum-area hulls.

1 Introduction

A fundamental problem in shape matching is to find the maximum area of overlap of two polygons under rigid motions. More precisely, we want to do the following: Given two polygons A and B, find a rigid motion ρ such that the area of overlap $|A \cap \rho B|$ is maximum. (See Figure 1.) In this paper,

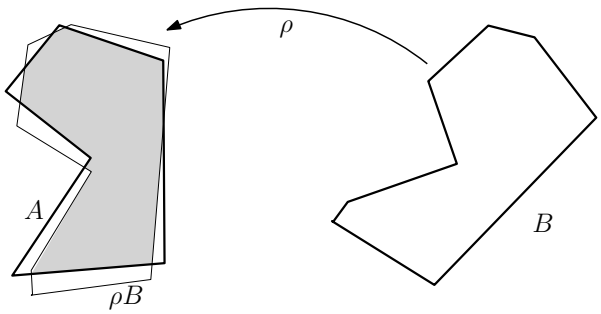

Figure 1: Given two polygons A and B, we want to maximize their area of overlap (shaded) after applying a rigid motion ρ to B.

we present a general technique that yields FPTAS for this problem and other related problems. Previously, no polynomial-time constant-factor approximation algorithm was known.

*A preliminary version of this article appeared in the proceedings of SODA 2010.

†Geometric Modeling and Scientific Visualization Center, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia. Email: antoine.vigneron@kaust.edu.sa

This overlap problem can be seen as a problem of maximizing a sum of a large number of constant description complexity rational functions (constant degree rational functions of a constant number of variables) for the following reason. Assume that the input polygons A and B have at most n edges. We first triangulate A and B, and thus A is the union of $O(n)$ triangles $A_1, A_2 \ldots$ and B is the union of $O(n)$ triangles $B_1, B_2 \ldots$ Using as parameters the sine and cosine of the angle of rotation, and the coordinates of the translation part of the rigid motion ρ, we can write the area of overlap $|A_i \cap \rho B_j|$ of any two triangles in these triangulations as a sum of a constant number of partially defined degree 4 rational functions. (See Section 4.2.) Hence, the area of overlap $|A \cap \rho B|$ can be written $\sum_{i,j} |A_i \cap \rho B_j|$, which is a sum of $O(n^2)$ partially defined degree-4 rational functions of four variables.

Several other geometric optimization problems reduce to optimizing a sum of rational functions. For instance, the generalization of our maximum area problem to arbitrary fixed dimension, where one wants to find the maximum volume of overlap of two polyhedra under rigid motions. Barequet and Rogol [8] considered the related problem of computing the largest axially symmetric subset of a polygon. Arkin et al. [7] studied generalizations of two-dimensional convex hulls, and Mahji et al. [20] studied several computational geometry problems motivated by CAD applications, that all reduce to the problem of maximizing or minimizing a sum of rational functions. Chen et al. [11, 12] considered several geometric problems that reduce to optimizing a sum of linear fractions, and in particular, the problem of finding an optimal ray in a weighted subdivision. Finally, Cheong et al. [13] considered a polygon guarding problem and the problem of maximizing the area of a Voronoi cell, which also reduce to maximizing a sum of rational functions.

A polynomial-time algorithm for the problem of optimizing a sum of constant description complexity rational functions would immediately yield polynomial-time algorithms for all the problems above. Unfortunately, it seems that no such algorithm is known. The problem is that, after summing up m constant-degree rational functions, we obtain a rational function of degree $\Omega(m)$ in the worst case. No polynomial-time algorithm seems to be known for optimizing such functions.

To overcome this difficulty, Barequet and Rogol [8], Arkin et al. [7], and Mahji et al. [20] solved numerically some optimization problems of this type. It requires finding roots of polynomials of several variables with linear degree. As was noted by Chen et al. [11], the running time of these algorithms depends on the conditioning of the roots, and thus it may not be polynomial in the size of the input. Chen et al. [11] used techniques from global optimization and obtained another algorithm, whose running time is still not known to be polynomial.

1.1 Our results and comparison with previous work

In this paper, we give a general technique that yields FPTAS for all the geometric optimization problems mentioned above. Intuitively, these problems have a constant number of degrees of freedom, and consist in optimizing a sum of lengths, areas, or volumes, in arbitrary fixed dimension. Our algorithm works in a more general setting, and in particular, it applies to sums of algebraic functions, and not just rational functions.

We first give FPTAS for optimizing a sum $f = \sum_{i=1}^{m} f_i$ of constant description complexity, non-negative, partially defined, algebraic functions $f_i : \mathbb{R}^d \to \mathbb{R}$, where $d = O(1)$. We call these functions *nice functions*; a definition can be found in Section 2.1. Our FPTAS for the maximization problem is given in Section 3.1, and an improved version when $d \leqslant 4$ is given in Section 3.3. Our FPTAS for the minimization problem are given in Section 3.4.

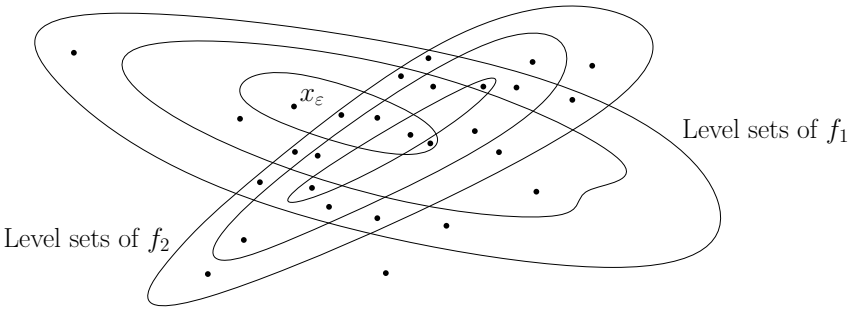

Figure 2: We sample a point in each cell of an arrangement of level sets of the functions f_i. We return the optimal point x_ε among these sample points, and we will prove that $f(x_\varepsilon)$ is an ε-approximation of the optimal value of f.

Our approach is the following: we build a collection of level sets of each function f_i. As f_i is an algebraic function of constant description complexity, each level set is given as the zero set of a polynomial of constant degree. Then we find one point in each cell of the arrangement of these level sets. Using an algorithm by Basu, Pollack, and Roy [9], we can find these points in time polynomial in the number of level sets. Among these points, we return the one that maximizes f. (See Figure 2.) In order for this approach to succeed, we need our level sets to provide an accurate discretization of the problem; we show that it suffices to take $O(\frac{1}{\varepsilon}\log\frac{m}{\varepsilon})$ level sets for each function f_i in order to get an ε relative error on $\sum f_i$. The first step for constructing these level sets is to get a factor-m approximation of the optimum of f. For the maximization problem, we just take the maximum of the maxima of all the functions f_i. For the minimization problem, we take the lowest point on the upper envelope of the functions f_i.

Then we give geometric applications of our FPTAS for optimizing sums of algebraic functions. In Section 4.1, we give an FPTAS for L_1-fitting a sphere to an n-point set, in fixed dimension. Har-Peled [16] gave a randomized FPTAS for this problem, with time bound $O(n+((1/\varepsilon)\log n)^{O(1)})$. In 2D, the exponent in the second term of this bound is somewhere between 20 and 60. Our algorithm runs in $O(n^{4+\varepsilon'} + (n/\varepsilon)^3\log^4(n/\varepsilon)\beta(n/\varepsilon))$, for any $\varepsilon' > 0$, where β is a very slowly growing function related to the inverse Ackermann function.

In Section 4.2, we give an FPTAS for maximizing the volume of overlap of two polyhedra under rigid motions, in fixed dimension. It is the first FPTAS for this problem even in $\mathbb{R}^2$. In fact, no polynomial-time, constant-factor algorithm was known. More specialized results were known: Ahn et al. [4] gave an FPTAS for convex polygons. In the non-convex case, Mount, Silverman, and Wu. [22] gave an $O(n^4)$ *exact* algorithm for maximum overlap under translations. Recently, Ahn, Cheng, and Reinbacher [3] gave an FPTAS for maximum overlap of convex polytopes under translation in arbitrary fixed dimension; In 3D, their algorithm runs in $O(n\log^{3.5} n)$ time for any $\varepsilon > 0$. For non-convex polygons under rigid motions, Cheong, Efrat, and Har-Peled [13] gave a randomized algorithm whose running time is [1] $O(n^3/E^8 \cdot \log^5 n)$ with high probability, and E is an *absolute* error bound: they allow an error of E times the area of the smallest polygon. By contrast, our algorithm is an FPTAS: it gives a *relative* error bound, which is a stronger requirement. Moreover, our algorithm is deterministic. In Section 4.3, we give an improved FPTAS for the 2D case with

[1] The time bound given in the paper [13] is smaller, because of an error at the end of the calculation.

running time $O((n^6/\varepsilon^3)\log^4(n/\varepsilon)\beta(n/\varepsilon))$.

In Section 4.4, we give an FPTAS for finding the largest axially symmetric subset of a polygon with n vertices. Its running time is $O((n^4/\varepsilon^2)\log^2(n/\varepsilon))$. Barequet and Rogol [8] gave an algorithm with running $O(n^4T(n))$, where $T(n)$ is the average time needed by a numeric algorithm to maximize some rational functions. This numeric algorithm is not known to run in worst case polynomial time, although in practice, Barequet and Rogol observed that $T(n) = O(n)$. Ahn et al. [2] gave an FPTAS for finding the largest axially symmetric subset of a *convex* polygon.

In Section 4.5, we consider the problem of minimizing the volume of the symmetric difference of two polyhedra under rigid motions, in fixed dimension. An optimal rigid motion for this problem is also optimal for the problem above of maximizing the volume of overlap, but if we are interested in ε-approximations of the optimum, then the two problems are not equivalent. We give an FPTAS for this problem in arbitrary fixed dimension. Previously, no FPTAS was known; the only previous result on this problem is by Alt et al. [5], who gave polynomial-time, constant-factor approximation algorithms for convex polygons.

In Section 4.6, we present an FPTAS for the problem of finding an optimal ray that reaches a target region in a weighted subdivision with n simplices, in arbitrary fixed dimension. In 2D, Chen et al. [11] reduced this problem to $O(n^2)$ instances of a fractional program, which they solve using practical algorithms that do not provide worst case time bounds. The only theoretical result is by Daescu and Palmer [14], who gave an $O(n^3\log^3 n)$-time algorithm in the arithmetic model of computation (as opposed to our algorithms which work in the unit cost model) for the 2D case. Their approach is based on fast algorithms for finding roots of univariate polynomials of degree n, and a geometric observation that reduces the number of variables to one. Hence, our algorithm is the first FPTAS for this problem in dimension higher than 2. In 3D, our time bound is $O(n^{5+\varepsilon'} + (n^5/\varepsilon^2)\log^2(n/\varepsilon))$ for any $\varepsilon' > 0$.

In Section 4.7, we give an FPTAS for finding the smallest star-shaped polygon containing a given polygon. Arkin et al. [7] and Chen et al. [11] gave algorithms that reduce this problem to solving $O(n^2)$ fractional programs, which are solved by algorithms that are not known to run in polynomial time. Our algorithm runs in $O(n^{4+\varepsilon'} + (n^4/\varepsilon^2)\log^2(n/\varepsilon))$ time for any $\varepsilon' > 0$.

We gave 6 geometric applications of our technique, but it also yields FPTAS for other problems, including all the problems in the papers we cited by Arkin et al. [7], Barequet and Rogol [8], Chen et al. [11], Cheong, Efrat, and Har-Peled [13], and Majhi et al. [20] Thus, our approach seems to be more general than Cheong et al.'s framework [13]. Their approach is similar, but instead of using directly the level sets of some algebraic functions, they first sample points on the input objects, and then construct an arrangement of surfaces, each surface corresponding to one sample point belonging to an object. Their approach gives time bounds with better dependency on n, but it requires careful sampling arguments for each problem, and uses randomization, while our algorithms are deterministic. It is unclear how to apply it to several of the problems we mentioned here, and in particular to the minimization problems.

Another related work is on shared camera control, by Har-Peled et al. [17]. Here, the problem reduces to optimizing a sum of piecewise linear functions. They give an exact algorithm, as well as a very fast approximation algorithm.

2 Preliminary

Let d be an integer constant. We will consider real polynomials in d variables. As no confusion is possible, we will abuse notation and identify each such polynomial P with the corresponding polynomial function $P : \mathbb{R}^d \to \mathbb{R}$. For any such polynomial $P \in \mathbb{R}[X_1, \ldots, X_d]$, we denote by $\mathrm{zer}(P)$ the zero set of P, that is, the set of points $x \in \mathbb{R}^d$ such that $P(x) = 0$. For any $r \in \mathbb{R}$ and $\mathcal{D}' \subset \mathbb{R}^d$, and for any function $g : \mathcal{D}' \to \mathbb{R}$, the *r-level set* of g, which we denote by $\mathrm{lev}(g, r)$, is defined as the set of points $x \in \mathcal{D}'$ such that $g(x) = r$:

$$\mathrm{lev}(g, r) = \{x \in \mathcal{D}' \mid g(x) = r\}.$$

An *algebraic function* $g : \mathbb{R}^d \to \mathbb{R}$ is a function that satisfies a polynomial equation of the form $P(x, g(x)) = 0$, where P is a polynomial in $d + 1$ variables. For instance, a rational function $g(x) = N(x)/D(x)$, where N and D are polynomials, is algebraic, because it is defined by the polynomial equation $N(x) - D(x) \cdot g(x) = 0$. Another example is the L_2 norm, where we have $(L_2(x_1, \ldots, x_d))^2 - x_1^2 - \cdots - x_d^2 = 0$; the L_2 norm is the positive solution to this equation.

2.1 Nice functions

The functions we want to optimize are sums of a large number of *nice functions* f_i, as defined below. A nice function is a non-negative function that coincides with an algebraic function of constant description complexity over a semialgebraic set of constant description complexity, and is equal to zero outside this semialgebraic set. In the rest of this section, we give a more formal definition of nice functions.

Let d' be an integer constant. We denote by $\mathcal{P}$ the set of polynomials in $\mathbb{R}[X_1, \ldots, X_d]$ with degree at most d'. In other words, $\mathcal{P}$ is a set of real polynomials with constant degree and a constant number of variables. We denote by $\mathcal{P}'$ the set of polynomials in $\mathbb{R}[X_1, \ldots, X_{d+1}]$ with degree at most d'.

The nice functions are defined over a *domain* $\mathcal{D} \subset \mathbb{R}^d$. This domain $\mathcal{D}$ is a semialgebraic set of $\mathbb{R}^d$ of constant description complexity, defined as follows: There exists polynomials $Q, K_1, \ldots, K_q \in \mathcal{P}$ such that $q \leqslant d'$ and

$$\mathcal{D} = \left\{ x \in \mathbb{R}^d \mid Q(x) = 0 \text{ and } K_{q'}(x) \geqslant 0 \text{ for all } 1 \leqslant q' \leqslant q \right\}.$$

We denote $d_Q = \dim(\mathrm{zer}(Q))$, and we assume that the dimension of $\mathcal{D}$ is d_Q. Intuitively, d_Q is the number of degrees of freedom of the problem. We allow Q to be the zero-polynomial, in which case $\mathrm{zer}(Q) = \mathbb{R}^d$, and $d = d_Q$.

A function $f_i : \mathcal{D} \to \mathbb{R}$ is a *nice function* with domain $\mathcal{D}$ if it has the following properties:

(i) f_i is non-negative.

(ii) There exists a semialgebraic set $\mathrm{supp}(f_i) \subset \mathcal{D}$, and an algebraic function g_i defined by a polynomial in $\mathcal{P}'$, such that $f_i(x) = g_i(x)$ for all $x \in \mathrm{supp}(f_i)$, and $f_i(x) = 0$ for all $x \in \mathcal{D} \setminus \mathrm{supp}(f_i)$.

(iii) The semialgebraic set $\mathrm{supp}(f_i)$ is defined by a boolean combination of at most d' inequalities of the form $U_{ij}(x) \geqslant 0$, where $U_{ij} \in \mathcal{P}$. The set of these polynomials U_{ij} is denoted by $\mathrm{SUPP}(f_i)$.

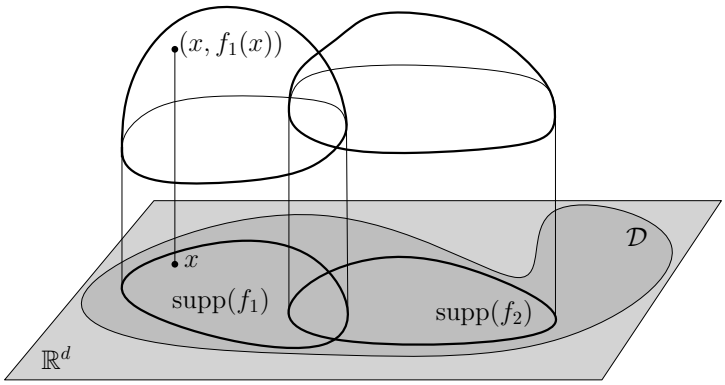

Figure 3: Two nice functions f_1 and f_2 over a domain $\mathcal{D}$. The value of f_1 outside $\text{supp}(f_1)$ is 0. Note that $\text{supp}(f_1)$ and $\text{supp}(f_2)$ may partially overlap.

(iv) The restriction of f_i to $\text{supp}(f_i)$ is continuous.

We call $\text{supp}(f_i)$ the *support* of f_i. Our definition allows f_i to be discontinuous at the boundary of $\text{supp}(f_i)$. Figure 3 illustrates our definition of nice functions.

2.2 Algorithms for arrangements

The *arrangement* of a collection of surfaces in $\mathbb{R}^d$ is the decomposition of $\mathbb{R}^d$ into maximal connected cells of dimension $0, \ldots, d$ induced by these surfaces [15, 25]. For instance, the arrangement of a collection of curves in $\mathbb{R}^2$ consists of vertices, which are endpoints or intersection points between two curves, edges, which are curve segments lying between two consecutive vertices, and faces, which are connected components of the complement of the union of the curves.

We need to be able to sample a point in each cell of an arrangement of level sets of nice functions. All these functions will have the same domain $\mathcal{D}$, so we will only consider arrangements of level sets on the fixed algebraic hypersurface $\text{zer}(Q)$. To achieve this, we use the algorithm below by Basu, Pollack, and Roy. The statement is adapted to our notations, and our special case where the degrees and the number of variables are less than a constant.

Let $\mathcal{S}$ be a set of s polynomials in $\mathcal{P}$. We denote by $\text{arr}(\mathcal{S})$ the arrangement over $\text{zer}(Q)$ of the zero sets of all the polynomials in $\mathcal{S}$.

Theorem 1 ([9]) *We can compute in time $O(s^{d_Q+1})$ a set of $O(s^{d_Q})$ points, such that each cell of $\text{arr}(\mathcal{S})$ contains at least one point of this set.*

We will also use a related result of Basu, Pollack and Roy [10, Algorithm 14.9], which allows to minimize or maximize a nice function over a semialgebraic sets. Again, the formulation is adapted to our special case, where the degree and number of variables are constant.

Theorem 2 ([10]) *We can compute in time $O(s^{2d+1})$ the minimum and the maximum of a nice function over a semialgebraic set defined by s polynomials in $\mathcal{P}$.*

The algorithm in Theorem 1 does not compute the entire structure of the arrangement $\text{arr}(\mathcal{S})$: it only samples one point in each cell. We will need to compute this structure in order to improve

the time bounds of our approximation schemes in low dimension. More precisely, we will compute a *vertical decomposition* of $\mathrm{arr}(S)$, which is a refinement of $\mathrm{arr}(S)$ into a collection of constant description complexity cells [15].

In two dimensions, this vertical decomposition has complexity $\Theta(s^2)$ [15], and it can be computed in optimal $O(s^2)$ time [6] using an algorithm by Amato, Goodrich, and Ramos.

Theorem 3 ([6]) *When $d = 2$, the vertical decomposition of $\mathrm{arr}(\mathcal{S})$ has size $O(s^2)$, and it can be computed in $O(s^2)$ time.*

In three dimensions, the complexity of $\mathrm{arr}(\mathcal{S})$ is $\Theta(s^3)$, but the best known upper bound on the complexity of its vertical decomposition is slightly higher: it is $O(s^3\beta(s))$ where $\beta(s)$ is a very slowly growing function related to the inverse Ackermann function. More precisely, $\beta(s) = \lambda(s)/s$ where $\lambda(s)$ is the maximum length of a Davenport-Schinzel sequence of order $O(1)$ with s symbols [15]. We compute $\mathrm{arr}(\mathcal{S})$ using an algorithm by Shaul and Halperin [26], which gives the following bounds. (Although the article by Shaul and Halperin focuses on arrangements of triangles, they mention that their technique handles well-behaved surface patches as well, within the same asymptotic time bound.)

Theorem 4 ([26]) *When $d = 3$, the vertical decomposition of $\mathrm{arr}(\mathcal{S})$ has size $O(s^3\beta(s))$, and it can be computed in $O(s^3 \log(s)\beta(s))$ time.*

Finally, in four dimensions, we use a result of Koltun [18]:

Theorem 5 ([18]) *When $d = 4$, and for any $\varepsilon' > 0$, the vertical decomposition of $\mathrm{arr}(\mathcal{S})$ has size $O(s^{4+\varepsilon'})$, and can be computed in time $O(s^{4+\varepsilon'})$.*

2.3 Models of computation

We use the real-RAM model of computation [24], which is the standard model in computational geometry. As in previous work on computing arrangements [18, 25] of algebraic surfaces, we assume that we can decide semialgebraic sets of constant description complexity in constant time; this corresponds to basic geometric predicates for deciding the existence of vertices, edges, and faces, and finding their relative position. We assume that, when such a semialgebraic set is non-empty, we can compute a point inside this set in constant time.

The model above assumes infinite precision arithmetic. A more realistic model that accounts for the bit-complexity of the input numbers can also be used in the algorithms by Basu, Pollack, and Roy (theorems 1 and 2). In this model, the input numbers are τ-bits integers, and the running time depends on τ. Our results still hold in this model in the sense that we still obtain FPTAS for all the problems we mention, but the running time may increase by a polynomial factor in τ, n, and $1/\varepsilon$.

3 Optimizing a sum of algebraic functions

In this section, we give FPTAS for the problem of optimizing a sum of nice functions. We first consider the maximization problem in arbitrary fixed dimension (Section 3.1 and 3.2), then we give an improved algorithm in dimension 2 to 4 (Section 3.2), and finally, we give minimization algorithms (Section 3.4.)

3.1 Maximizing a sum of algebraic functions

We consider the problem of maximizing a sum of nice functions in arbitrary fixed dimension. As we mentioned in Section 2, we will try to maximize this sum over a semialgebraic subset $\mathcal{D}$ of $\mathbb{R}^d$. Our result is the following:

Theorem 6 *Let $\varepsilon \in (0,1)$ and let m be a positive integer. Let $\mathcal{D} \subset \mathbb{R}^d$ be a d_Q-dimensional domain, and let $f = \sum_{i=1}^{m} f_i$ be a sum of m functions $f_i : \mathcal{D} \to \mathbb{R}$, where each f_i is a bounded, nice function. (See Section 2.1.) Then we can compute a point $x_\varepsilon \in \mathcal{D}$ such that $\max_{\mathcal{D}} f \leqslant (1+\varepsilon) \cdot f(x_\varepsilon)$ in time $O\left(\left(\frac{m}{\varepsilon}\right)^{d_Q+1} \log^{d_Q+1}\left(\frac{m}{\varepsilon}\right)\right)$.*

We now prove Theorem 6. We can compute the maximum $\max_{\mathcal{D}} f_i$ of each function f_i in constant time using Theorem 2. So we compute all these maxima in $O(m)$ time. If $\max_{\mathcal{D}} f_i = 0$ for some i, then $f_i(x) = 0$ for all x, so we discard f_i. Then we compute in time $O(m)$ the maximum M of these maxima, that is, $M = \max_i (\max_{\mathcal{D}} f_i)$.

Remember that each function f_i coincides with an algebraic function g_i over its support $\mathrm{supp}(f_i)$. (See Section 2.1.) We build a collection of polynomials whose zero sets give an accurate discretization of the problem. More precisely, we construct a set $\mathcal{S}$ of polynomials, such that for any $S \in \mathcal{S}$, its zero set $\mathrm{zer}(S)$ is a level set of a function g_i, or S is a polynomial in $\mathrm{SUPP}(f_i)$, or S is one of the polynomials $K_1, \ldots, K_q$ defining $\mathcal{D}$.

We now describe which polynomials are inserted into $\mathcal{S}$ for each function f_i. Let $\alpha = \varepsilon / C_\alpha$, where C_α is a large enough constant, to be specified later. Let k be the positive integer such that

$$\frac{1}{(1+\alpha)^k} \leqslant \frac{\alpha}{m} < \frac{1}{(1+\alpha)^{k-1}}. \tag{1}$$

Then we have

$$k = O\left(\frac{1}{\alpha} \log\left(\frac{m}{\alpha}\right)\right). \tag{2}$$

For each $j \in \{1, \ldots, k\}$, we insert a polynomial whose zero set is the level set $\mathrm{lev}\left(f_i, M/(1+\alpha)^j\right)$. Remember that whenever $f_i(x) \neq 0$, the function f_i is given by a polynomial $P_i \in \mathcal{P}'$ such that $P_i(x, g_i(x)) = 0$. So the level set $\mathrm{lev}\left(f_i, M/(1+\alpha)^j\right)$ is equal to $\mathrm{zer}(S_{ij}) \cap \mathrm{supp}(f_i)$, where $S_{ij}(x) = P_i\left(x, M/(1+\alpha)^j\right)$. We insert into $\mathcal{S}$ the polynomials S_{ij} for all i, j. We also insert into $\mathcal{S}$ the polynomials in $\mathrm{SUPP}(f_i)$ for all i. Finally, we insert into $\mathcal{S}$ the polynomials $K_1, \ldots, K_q$ that define $\mathcal{D}$. (See Section 2.1.) By Equation (2), the cardinality of $\mathcal{S}$ is $s = O(m/\alpha \cdot \log(m/\alpha))$. We apply Theorem 1 to $\mathcal{S}$, and thus we obtain in time $O(s^{d_Q+1})$ a set E of $O(s^{d_Q})$ points that meets every cell of the arrangement $\mathrm{arr}(\mathcal{S})$. Since $\alpha = \Theta(\varepsilon)$, this time bound can be rewritten $O\left((m/\varepsilon)^{d_Q+1} \log^{d_Q+1}(m/\varepsilon)\right)$. Then we remove from E the points that are not in $\mathcal{D}$; it can be done in constant time per point by checking the sign of the polynomials $K_1, \ldots, K_q$.

At this point, we would like to choose x_ε to be a point $x \in E$ that maximizes $f(x)$. Unfortunately, we do not know how to do it fast enough—this problem is harder than the well-known open problem of comparing two sums of square roots [23]. But in the present case, we only need an approximate answer, and the functions f_i have constant description complexity, so it is reasonable to assume that their values can be approximated well enough using standard numerical algorithms. Thus, for the rest of this proof, we will assume that we have found the point x_ε in E that maximizes f. For sake of completeness, we present in Section 3.2 a way of resolving this issue, with an explicit construction of an approximation $\hat{f}$ of f over E.

So our algorithm returns x_ε such that $f(x_\varepsilon) = \max_E f$. It remains to prove that this algorithm is correct. Let $x^* \in \mathcal{D}$ be an optimal point, that is, a point such that $f(x^*) = \max_{\mathcal{D}} f$. We want to prove that $f(x^*) \leqslant (1+\varepsilon) f(x_\varepsilon)$. Let $\mathcal{C}$ denote the cell of $\mathrm{arr}(\mathcal{S})$ that contains x^*. As the polynomials $K_1, \ldots, K_q$ that define $\mathcal{D}$ are in $\mathcal{S}$, this cell $\mathcal{C}$ is contained in $\mathcal{D}$. Then there exists a point $x_e \in E \cap \mathcal{C}$. We first need the following lemma:

Lemma 7 *For any $i \in \{1, \ldots, m\}$, we have $f_i(x^*) \leqslant \alpha \frac{M}{m} + (1+\alpha) f_i(x_e)$.*

Proof: If $f_i(x^*) \leqslant \alpha M/m$, then this lemma holds trivially. So we assume that $\alpha M/m < f_i(x^*)$. By Equation 1, we have $M/(1+\alpha)^k \leqslant \alpha M/m$. Thus, $M/(1+\alpha)^k < f_i(x^*)$, so there exists $j \in \{1, \ldots, k\}$ such that $M/(1+\alpha)^j < f_i(x^*) \leqslant M/(1+\alpha)^{j-1}$. Since x^* and x_e lie in the same cell $\mathcal{C}$ of $\mathrm{arr}(\mathcal{S})$, there exists a continuous path γ from x^* to x_e within $\mathcal{C}$. As $x^* \in \mathrm{supp}(f_i)$, and $\mathrm{SUPP}(f_i) \subset \mathcal{S}$, the path γ cannot leave $\mathrm{supp}(f_i)$. The level sets $\mathrm{lev}(f_i, M/(1+\alpha)^j)$ and $\mathrm{lev}(f_i, M/(1+\alpha)^{j-1})$ are the zero sets of $S_{ij}, S_{i(j-1)}$ restricted to $\mathrm{supp}(f_i)$, thus γ cannot cross these level sets. As f_i restricted to $\mathrm{supp}(f_i)$ is continuous, it implies that $M/(1+\alpha)^j < f_i(x_e) \leqslant M/(1+\alpha)^{j-1}$, and thus $f_i(x^*)/f_i(x_e) < 1+\alpha$. $\square$

Lemma 7 implies that $f(x^*) \leqslant \alpha M + (1+\alpha) f(x_e)$. Since $M \leqslant f(x^*)$ and $f(x_e) \leqslant f(x_\varepsilon)$, we get

$$(1-\alpha) f(x^*) \leqslant (1+\alpha) f(x_\varepsilon). \tag{3}$$

We choose $C_\alpha = 3$, and thus $\alpha = \varepsilon/3$. It implies that $f(x^*) \leqslant (1+\varepsilon) f(x_\varepsilon)$, which completes the proof of Theorem 6.

3.2 Approximating a sum of algebraic functions

In this section, we show how to resolve the problem mentioned in the proof of Theorem 6: we may not be able to find the point x_ε in E that maximizes f. To remedy this, we approximate f by a function $\hat{f}$ as follows. This approximation $\hat{f}$ also allows us to obtain a faster algorithm in low dimension. (See Section 3.3.)

We first consider each function f_i separately, for each $x \in E$. If $f_i(x) \leqslant \alpha M/m$, then we set $\hat{f}_i(x) := 0$. Otherwise, as f_i has constant description complexity, we can find in time $O(k)$ the index $j_i \in [1,k]$ such that $M/(1+\alpha)^{j_i} < f_i(x) \leqslant M/(1+\alpha)^{j_i - 1}$. Then our approximation of $f_i(x)$ is $\hat{f}_i(x) := M/(1+\alpha)^{j_i}$.

We compute $\hat{f}(x) := \sum_{i=1}^m \hat{f}_i(x)$ for each $x \in E$. For any $i \in \{1, \ldots, m\}$, we have $f_i(x) \leqslant \alpha M/m + (1+\alpha)\hat{f}_i(x)$, so by summing up over all i, we obtain

$$f(x) - \alpha M \leqslant (1+\alpha)\hat{f}(x). \tag{4}$$

Another property of $\hat{f}$ is that

$$\hat{f}(x) \leqslant f(x). \tag{5}$$

Instead of returning the point x_ε, our modified algorithm returns the point $\hat{x}_\varepsilon$ such that $\hat{f}(\hat{x}_\varepsilon) = \max_E \hat{f}$. Then $\hat{f}(x_\varepsilon) \leqslant \hat{f}(\hat{x}_\varepsilon)$, and thus by Equation (4), we have

$$f(x_\varepsilon) - \alpha M \leqslant (1+\alpha)\hat{f}(x_\varepsilon) \leqslant (1+\alpha)\hat{f}(\hat{x}_\varepsilon),$$

so Equation (3) yields

$$(1-\alpha) f(x^*) - \alpha(1+\alpha) M \leqslant (1+\alpha)^2 \hat{f}(\hat{x}_\varepsilon).$$

As $M \leqslant f(x^*)$, we obtain

$$(1 - 2\alpha - \alpha^2) f(x^*) \leqslant (1 + \alpha)^2 \hat{f}(\hat{x}_\varepsilon).$$

and thus by Equation (5)

$$(1 - 2\alpha - \alpha^2) f(x^*) \leqslant (1 + \alpha)^2 f(\hat{x}_\varepsilon).$$

Choosing $C_\alpha = 7$, and thus $\alpha = \varepsilon/7$, we conclude that $f(x^*) \leqslant (1 + \varepsilon) f(\hat{x}_\varepsilon)$, which proves that this modified algorithm is correct.

We still need to check that computing $\hat{f}(x)$ at each point of E does not make our time bound worse. As the cardinality of E is $O\left((m/\varepsilon)^{d_Q} \log^{d_Q}(m/\varepsilon)\right)$, and $k = O\left((1/\varepsilon)\log(m/\varepsilon)\right)$, it takes

$$O\left((m/\varepsilon)^{d_Q+1} \log^{d_Q+1}(m/\varepsilon)\right)$$

time to compute $\hat{f}(x)$ for all $x \in E$, which is the time bound of Theorem 6.

3.3 Improved algorithm in low dimension

In this section, we present an algorithm for maximizing a sum of nice functions with a better running time than the previous algorithm from Theorem 6, when $d \leqslant 4$ and $d = d_Q$. The function β in the statement below is a very slowly growing function related to the inverse Ackermann function, as in Theorem 4.

Theorem 8 *Let $\varepsilon \in (0, 1)$ and let m be a positive integer. Let $\mathcal{D} \subset \mathbb{R}^d$, and let $f = \sum_{i=1}^m f_i$ be a sum of m functions $f_i : \mathcal{D} \to \mathbb{R}$, where each f_i is a bounded, nice function. (See Section 2.1.) Then we can compute a point $x_\varepsilon \in \mathcal{D}$ such that $\max_{\mathcal{D}} f \leqslant (1 + \varepsilon) \cdot f(x_\varepsilon)$ in time $O\left(\left(\frac{m}{\varepsilon}\log\frac{m}{\varepsilon}\right)^2\right)$ when $d = 2$, in time $O\left(\left(\frac{m}{\varepsilon}\right)^3 \log^4\left(\frac{m}{\varepsilon}\right)\ \beta\left(\frac{m}{\varepsilon}\right)\right)$ when $d = 3$, and $O\left(\left(\frac{m}{\varepsilon}\log\frac{m}{\varepsilon}\right)^{4+\varepsilon'}\right)$ for any $\varepsilon' > 0$ when $d = 4$.*

The proof of this theorem is an extension of the proof of Theorem 6. We build the same set $\mathcal{S}$ of $O(\frac{m}{\varepsilon}\log\frac{m}{\varepsilon})$ polynomials as in Theorem 6. Instead of using the algorithm from Theorem 1 to sample a point in each cell of $\mathrm{arr}(\mathcal{S})$, we compute the vertical decomposition $\mathcal{V}$ of $\mathrm{arr}(\mathcal{S})$ using Theorem 3 (resp. Theorem 4, Theorem 5) when $d = 2$ (resp. $d = 3, d = 4$). The time bound for computing $\mathcal{V}$ dominates the running time of our algorithm, and is as stated in Theorem 8.

The *incidence graph* $\mathcal{G}$ of this vertical decomposition $\mathcal{V}$ is a graph whose nodes are the cells of $\mathcal{V}$, and such that two nodes of $\mathcal{G}$ are connected by an arc when the two corresponding cells $\mathcal{C}_1, \mathcal{C}_2$ are incident [15], that is, when $\mathcal{C}_1$ is a maximal subcell of $\mathcal{C}_2$ or $\mathcal{C}_2$ is a maximal subcell of $\mathcal{C}_1$. The algorithms from theorems 3, 4, and 5 provide this incidence graph, whose size is proportional to the size of $\mathcal{V}$.

We obtain a graph $\mathcal{G}'$ by removing from $\mathcal{G}$ the nodes that correspond to cells of $\mathrm{arr}(\mathcal{S})$ that are outside $\mathcal{D}$, and removing the edges adjacent to these nodes. This new graph $\mathcal{G}'$ may have several connected components. There is only a constant number of such connected components, because they correspond to cells in $\mathrm{arr}(\{Q, K_1, \ldots, K_q\})$, and $q = O(1)$. In the following, we consider one of these connected components $\mathcal{G}''$.

We traverse $\mathcal{G}''$, starting at an arbitrary node, and visiting each node at least once. (For instance, using depth-first search.) Within each cell of $\mathcal{V}$, the value of the approximation $\hat{f}$ of f that we presented in Section 3.2 is fixed, because $\hat{f}$ only changes when we cross a level set $\mathrm{lev}(f_i, M/(1+\alpha)^j)$.

So during our traversal of $\mathcal{G}''$, we maintain the unique value of $\hat{f}$ over the cell corresponding to the current node.

We now explain how we maintain this value when we move from a cell $\mathcal{C}_1$ of $\mathcal{V}$ to an incident cell $\mathcal{C}_2$. We use the same general position assumptions for our level sets as in previous work on vertical decompositions [15, 18, 25]. As argued in these works, it does not incur any real loss of generality. From these assumptions, when we go from $\mathcal{C}_1$ to $\mathcal{C}_2$, we enter or we leave a constant number of level sets $\mathrm{lev}(f_i, M/(1+\alpha)^j)$, so we only need to update the contribution of a constant number of functions f_i. These level sets are known from the description of $\mathcal{C}_1$ and $\mathcal{C}_2$, so we can update the value of $\hat{f}$ in constant time.

Thus, we can traverse $\mathcal{G}''$ and maintain the value of $\hat{f}$ in the current cell without increasing our time bound—we may only loose a constant factor. During the traversal of the connected components of $\mathcal{G}'$, we maintain the largest value of $\hat{f}$ found so far, as well as an arbitrary point in the corresponding cell. We return this point x_ε at the end of the traversals.

3.4 Minimizing a sum of algebraic functions

In this section, we present approximation algorithms for the minimization problem. The first one (Theorem 9) is analogous to our maximization algorithm from Theorem 6. The second algorithm (Theorem 11) is an improved version, analogous to the algorithm from Theorem 8.

Theorem 9 *Let $\varepsilon \in (0,1)$ and let m be a positive integer. Let $\mathcal{D} \subset \mathbb{R}^d$ be a d_Q-dimensional domain, and let $f = \sum_{i=1}^m f_i$ be a sum of m functions $f_i : \mathcal{D} \to \mathbb{R}$, where each f_i is a bounded, nice function. (See Section 2.1.) Then we can compute a point $x'_\varepsilon \in \mathcal{D}$ such that $f(x'_\varepsilon) \leqslant (1+\varepsilon)\min_{\mathcal{D}} f$ in time $O\left(m^{2d-2+\varepsilon'} + \left(\frac{m}{\varepsilon}\right)^{d_Q+1} \log^{d_Q+1}\left(\frac{m}{\varepsilon}\right)\right)$ for any $\varepsilon' > 0$.*

The proof of Theorem 9 is similar to the proof of Theorem 6; the main difference is the following. In the maximization problem, our discretization uses level sets of the form $f_i = M/(1+\alpha)^j$, where M is the maximum of all functions f_i. Intuitively, this approach works because M is an m-factor approximation of the optimal value. For the minimization problem, M does not provide any useful information—some functions f_i might take large values, when the minimum value of f is much smaller. So instead of using M to construct our level sets, we will use H, which is the height of the lowest point on the upper envelope of the functions f_i. The minimum of f is between H and mH, so it also gives a factor-m approximation. On the low side, it seems that we need to spend some extra time to compute H, while M was trivially obtained in $O(m)$ time.

We will first show how we compute H, and then we will complete the proof of Theorem 9.

Computing the lowest point on the upper envelope. The *upper envelope* of a set of nice functions $\mathcal{S} = \{f_1, \ldots, f_m\}$ is the set of points $(x, f_i(x))$ where $f_i(x) = \max_j f_j(x)$. Intuitively, the upper envelope is the top part of the arrangement $\mathrm{arr}(\mathcal{S})$. Our minimization algorithm will require to compute a lowest point on this upper envelope. In other words, we want to find a point $x_H \in \mathcal{D}$ that minimizes $f_i(x_H)$ under the constraint $f_i(x_H) = \max_j f_j(x)$. We denote $H = f_i(x_H)$. (See Figure 4.)

The algorithm of Theorem 2 allows us to compute x_H as follows. We handle the contribution of each function f_i separately. So we consider the set of points $x \in \mathcal{R}^d$ such that $f_i(x) \geqslant f_j(x)$ for all $j \neq i$. It is a semialgebraic set defined by $m-1$ polynomials in $\mathcal{P}$. So we can find the minimum of

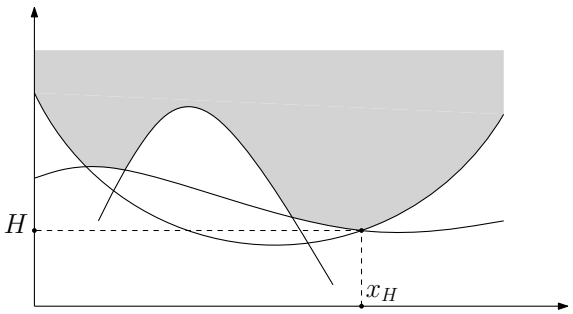

Figure 4: The point lowest point (x_H, H) on the upper envelope (shaded) of a set of functions.

f_i over this semialgebraic set in time $O(m^{2d+1})$. Then we return the minimum of these values over all $1 \leqslant i \leqslant m$ in time $O(m^{2d+2})$.

The approach above for finding x_H is sufficient to obtain an FPTAS for all our applications; however, we can obtain better time bounds as follows. When $d = 2$, we can compute the upper envelope of the functions f_i in time $O(m^{2+\varepsilon'})$ for any $\varepsilon' > 0$, using the algorithm of Agarwal, Schwarzkopf, and Sharir [1]. Then we consider the maximization diagram in $\mathbb{R}^2$, whose cells are the vertical projections of the cells of the upper envelope. We compute a vertical decomposition of this maximization diagram, and within each cell of this vertical decomposition, we apply Theorem 2 to find the lowest point (which takes constant time per cell). Overall, we still get a time bound $O(m^{2+\varepsilon'})$.

When $d \geqslant 3$, we use Koltun's construction of the vertical decomposition of an arrangement of surfaces [18]. So we obtain the upper envelope of the functions f_i in time in time $O(m^{2d-2+\varepsilon'})$ for any $\varepsilon' > 0$, and then we obtain x_H and $f(x_H)$ within the same time bound. (When $d = 3$, it is tempting to use the result of Koltun and Sharir [19] for computing the vertices, edges and 2-faces of the maximization diagram. However, we don't know how to obtain x_H efficiently from this result, as it does not provide a decomposition into cells of constant description complexity.)

Proof of Theorem 9. As in the proof of Theorem 6, we use $\alpha = \varepsilon/C_\alpha$, for a large enough constant C_α. Let k' be the positive integer such that

$$\frac{1}{(1+\alpha)^{k'}} \leqslant \frac{\alpha}{m^2} < \frac{1}{(1+\alpha)^{k'-1}}. \tag{6}$$

Then we have

$$k' = O\left(\frac{1}{\alpha}\log\left(\frac{m}{\alpha}\right)\right). \tag{7}$$

We construct a collection $\mathcal{S}'$ of polynomials as follows. For each function f_i, and for each integer $0 \leqslant j \leqslant k'$, we insert in the set $\mathcal{S}'$ the polynomial S'_{ij} corresponding to the level set $\mathrm{lev}(f_i, mH/(1+\alpha)^j)$. We also insert into $\mathcal{S}'$ the polynomials in $\mathrm{SUPP}(f_i)$ for each i. Finally, we insert the polynomials $K_1, \ldots, K_q$ that define $\mathcal{D}$.

Then we proceed as in Theorem 6: we construct a set of points E' by sampling one point in each cell of $\mathrm{arr}(\mathcal{S}')$, we discard the points that are outside $\mathcal{D}$, and we return the point x'_ε that minimizes f over E'. The time bound is the same as in Theorem 6, since our bound on k' (Equation (7)) is the

same as our bound for k (Equation (2)). We still need to check that $f(x'_\varepsilon) \leqslant (1+\varepsilon)f(x_*)$, where x_* is a point that minimizes f over $\mathcal{D}$.

Let $\mathcal{C}'$ denote the cell of $\mathrm{arr}(\mathcal{S}')$ that contains x_*. Then there exists a point $x'_e \in E' \cap \mathcal{C}'$. We first need the following lemma:

Lemma 10 *For any $i \in \{1, \ldots, m\}$, we have $f_i(x'_e) \leqslant \alpha\frac{H}{m} + (1+\alpha)f_i(x_*)$.*

Proof: First assume that $f_i(x_*) \leqslant \alpha H/(1+\alpha)m$. Then $f_i(x_*) < mH/(1+\alpha)^{k'}$. As there is a polynomial in $\mathcal{S}'$ corresponding to $\mathrm{lev}(f_i, mH/(1+\alpha)^{k'})$, and x_* lies in the same cell as x'_e, we also have $f_i(x'_e) < mH/(1+\alpha)^{k'}$. By Equation 6, it implies that $f_i(x'_e) < \alpha H/m$.

Now assume that $\alpha H/(1+\alpha)m < f_i(x_*)$. Then by Equation (6), we have $mH/(1+\alpha)^{k'+1} < f_i(x_*)$. Since $f(x_*) \leqslant mH$, we also have $f_i(x_*) \leqslant mH$, and thus there exists $j \in [0, k']$ such that $mH/(1+\alpha)^{j+1} < f_i(x_*) \leqslant mH/(1+\alpha)^j$. As $\mathcal{S}'$ contains the polynomial corresponding to the level set $mH/(1+\alpha)^j$, it follows that $f_i(x'_e) \leqslant mH/(1+\alpha)^j \leqslant (1+\alpha)f_i(x_*)$. $\square$

Lemma 10 implies that $f(x'_e) \leqslant \alpha H + (1+\alpha)f(x_*)$. Since $H \leqslant f(x_*)$ and $f(x'_\varepsilon) \leqslant f(x'_e)$, we get

$$f(x'_\varepsilon) \leqslant (1+2\alpha)f(x_*).$$

We choose $C_\alpha = 2$, and thus $\alpha = \varepsilon/2$. Then $f(x'_\varepsilon) \leqslant (1+\varepsilon)f(x_*)$, which completes the proof of Theorem 9.

Improved algorithm in low dimension. For the minimization problem in low dimension, we have the same improvement as we had for the maximization problem using the same approach. So we obtain the following result, by a straightforward modification of the proof of Theorem 8. The time bound T is the same as in Theorem 9 statement.

Theorem 11 *Let $\varepsilon \in (0,1)$ and let m be a positive integer. Let $\mathcal{D} \subset \mathbb{R}^d$, and let $f = \sum_{i=1}^m f_i$ be a sum of m functions $f_i : \mathcal{D} \to \mathbb{R}$, where each f_i is a bounded, nice function. (See Section 2.1.) Then for any $\varepsilon' > 0$, we can compute a point $x'_\varepsilon \in \mathbb{R}^d$ such that $f(x'_\varepsilon) \leqslant (1+\varepsilon)\min_{\mathcal{D}} f$ in time $O\left(m^{2+\varepsilon'} + \left(\frac{m}{\varepsilon}\log\frac{m}{\varepsilon}\right)^2\right)$ when $d = 2$, in time $O\left(m^{4+\varepsilon'} + \left(\frac{m}{\varepsilon}\right)^3 \log^4\left(\frac{m}{\varepsilon}\right)\ \beta\left(\frac{m}{\varepsilon}\right)\right)$ when $d = 3$, and in time $O\left(m^{6+\varepsilon'} + \left(\frac{m}{\varepsilon}\log\frac{m}{\varepsilon}\right)^{4+\varepsilon'}\right)$ when $d = 4$.*

4 Geometric applications

In this section, we give seven geometric applications of our FPTAS for optimizing sums of algebraic functions. The first one (Section 4.1) is a straightforward application to a L_1 sphere-fitting problem. The next application (Section 4.2) is to the problem of maximizing the overlap of two polyhedra under rigid motions. Then, we give an improved algorithm for the 2D case, where we want to maximize the area of overlap of two polygons under rigid motions (Section 4.3). The fourth application (Section 4.4) is the problem of finding the largest axially-symmetric subset of an input polygon. The fifth (Section 4.5) is the problem of minimizing the volume of the symmetric difference of two polyhedra under rigid motions. The sixth (Section 4.6) is on finding an optimal ray that hits a target region of a weighted subdivision. The last application (Section 4.7) is on computing the smallest star-shaped polygon containing an input polygon.

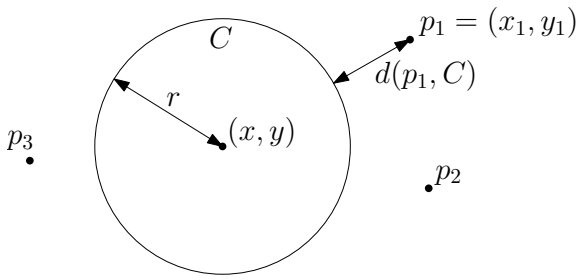

Figure 5: L_1 shape fitting. We want to minimize $d(p_1, C) + d(p_2, C) + d(p_3, C)$.

4.1 L_1 shape fitting

We consider the following shape fitting problem. Let $\delta \geqslant 2$ be a fixed integer. We are given a set Π of n points in $\mathbb{R}^\delta$. For any $(\delta-1)$-sphere $C \subset \mathbb{R}^\delta$, and any point $p_i \in \Pi$, we denote by $d(p_i, C)$ the Euclidean distance from p_i to C. Our goal is to find a sphere that minimizes $\sum_{i=1}^n d(p_i, C)$.

This problem falls within our framework. To alleviate notation, we only give details on the two dimensional case ($\delta = 2$). Then each circle C is represented by its center (x, y) and its radius r: it is represented by the point $(x, y, r) \in \mathbb{R}^3$. For each i, we denote $p_i = (x_i, y_i)$. (See Figure 5.) When $r \geqslant 0$, we set $f_i(x, y, r) = d(p_i, C)$, and thus

$$f_i(x, y, r) = \left|\sqrt{(x - x_i)^2 + (y - y_i)^2} - r\right|.$$

Our goal is to minimize over all x, y, and all $r \geqslant 0$, the function $f(x, y, r) = \sum_i f_i(x, y, r)$. So we choose the domain $\mathcal{D} = \{(x, y, r) \in \mathbb{R}^3 \mid r \geqslant 0\}$. (Hence the polynomial Q is the zero polynomial, and $K_1(x, y, r) = r$.) Observe that for all x, y and $r \geqslant 0$, we have $4r^2[(x - x_i)^2 + (y - y_i)^2] = [(x - x_i)^2 + (y - y_i)^2 + r^2 - f_i(x, y, r)^2]^2$. So we choose $\mathrm{supp}(f_i) = \mathcal{D}$, and thus f_i is algebraic over $\mathrm{supp}(f_i)$. Then the function f_i is a nice function with 3 variables.

We obtain an FPTAS by applying Theorem 9 with $m = n$ and $d = d_Q = \delta + 1$. When $\delta = 2$ or $\delta = 3$, we obtain a better time bound by applying Theorem 11 with $d = 3$ and $d = 4$, respectively.

Theorem 12 *Let $\delta \geqslant 2$ be a fixed integer. For any $\varepsilon' > 0$, there is an $O\left(n^{2\delta+\varepsilon'} + \left(\frac{n}{\varepsilon}\right)^{\delta+2} \log^{\delta+2} \frac{n}{\varepsilon}\right)$-time, $(1+\varepsilon)$-factor approximation algorithm for the L_1-sphere fitting problem in $\mathbb{R}^\delta$. When $\delta = 2$, the time bound improves to $O\left(n^{4+\varepsilon'} + \left(\frac{n}{\varepsilon}\right)^3 \log^4\left(\frac{n}{\varepsilon}\right) \beta\left(\frac{m}{\varepsilon}\right)\right)$ for any $\varepsilon' > 0$. When $\delta = 3$, the time bound improves to $O\left(n^{6+\varepsilon'} + \left(\frac{n}{\varepsilon} \log \frac{n}{\varepsilon}\right)^{4+\varepsilon'}\right)$ for any $\varepsilon' > 0$.*

As we mentioned in the introduction, Har-Peled [16] gave a randomized FPTAS for this problem, with time bound $O(n + ((1/\varepsilon) \log n)^{O(1)})$. In 2D, the exponent in the second term of this bound is somewhere between 20 and 60.

4.2 Maximizing the overlap of two polyhedra under rigid motions

In this section, we give an approximation algorithm for finding the maximum volume of overlap of two polyhedra in arbitrary fixed dimension. It is the first FPTAS for this problem. For the special case of convex polyhedra under translation, Ahn, Cheng, and Reinbacher [3] gave an FPTAS in arbitrary fixed dimension. We prove the following:

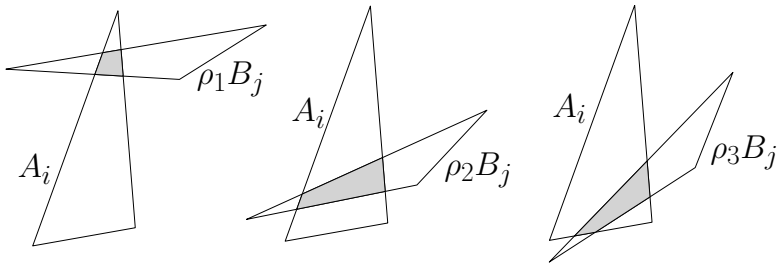

Figure 6: The intersections of A_i with $\rho_1 B_j$ and $\rho_2 B_j$ have the same combinatorial structure, but the intersection with $\rho_3 B_j$ has a different combinatorial structure.

Theorem 13 *Let δ be an integer constant, and let $\varepsilon \in (0,1)$. Let $\mathcal{R}$ be the set of rigid motions in $\mathbb{R}^\delta$. Let A and B denote two polyhedra in $\mathbb{R}^\delta$, given as unions of n_a and n_b interior-disjoint simplices, respectively. We can compute a rigid motion ρ_ε such that $(1+\varepsilon)|A \cap \rho_\varepsilon B| \geqslant \max_{\rho \in \mathcal{R}} |A \cap \rho B|$ in time $O\left(\left(\frac{n_a n_b}{\varepsilon}\right)^{\delta'} \log^{\delta'}\left(\frac{n_a n_b}{\varepsilon}\right)\right)$, where $\delta' = \delta^2/2 + \delta/2 + 1$.*

Proof: A rigid motion ρ can be specified by a matrix M and a translation vector V, such that for any $X \in \mathbb{R}^\delta$, we have $\rho(X) = MX + V$. (Here points and vectors are represented by column vectors.) Therefore ρ is specified by $\delta^2 + \delta$ real parameter, and thus we will apply Theorem 6 with $d = \delta^2 + \delta$. However, not all matrix M is the matrix of a rigid motion, and we will find a smaller value for d_Q.

Let $M = (m_{ij})$ denote a δ by δ matrix, and let $V = (v_i)$ be a δ-dimensional vector. Then the mapping $X \mapsto MX + V$ is a rigid motion if and only if $\det(M) = 1$ and $M\ {}^tM = I$, where I is the identity matrix. We denote by $\|M\ {}^tM - I\|_F$ the Frobenius norm of the matrix $M\ {}^tM - I$, which is the sum of the squares of its coefficients, and we denote $Q = \|M\ {}^tM - I\|_F + (\det(M) - 1)^2$. Then Q is a polynomial in the coefficients m_{ij} and v_j, and our mapping is a rigid motion if and only if its coefficients lie in $\mathrm{zer}(Q)$.

The time bound in Theorem 6 depends on the dimension d_Q of $\mathrm{zer}(Q)$. Here, we choose the domain $\mathcal{D}$ to be the space $\mathcal{R}$ of rigid motions, and thus $\mathcal{R} = \mathcal{D} = \mathrm{zer}(Q)$. Observe that, in order for ρ to be a rigid motion, we need M to be in the special orthonormal group, which has dimension $\delta(\delta-1)/2$. In addition, we can choose the translation vector arbitrarily, so $d_Q = \delta(\delta+1)/2$.

In order to complete the proof, we still need to argue that we can reduce our maximum overlap problem to maximizing a sum of $O(n_a n_b)$ nice functions. Remember that A is a union $\bigcup_{1 \leqslant i \leqslant n_a} A_i$ of n_a disjoint simplices. Similarly, B is a union $\bigcup_{1 \leqslant j \leqslant n_b} B_j$ of n_b disjoint simplices. For any i, j, and any $\rho \in \mathcal{R}$, we denote by $\mu_{ij}(\rho) = |A_i \cap \rho B_j|$ the volume of overlap of A_i and ρB_j. We will show that each function μ_{ij} can be written as a sum of $O(1)$ nice functions, using as parameters the coefficients (m_{ij}) of the matrix M of ρ and the coefficients (v_i) of its translation part V. Then the area of overlap $|A \cap \rho B|$ is equal to $\sum_{ij} \mu_{ij}(\rho)$, and thus it is a sum of $O(n_a n_b)$ nice functions.

We will say that two intersections of simplices $A_i \cap \rho_1 B_j$ and $A_i \cap \rho_2 B_j$ have the same combinatorial structure if their vertices correspond to the intersection of the same two faces of A_i and B_j, and if their $(d+1)$-tuples of vertices have the same orientation. See Figure 6 for an example in the plane. This notion will be helpful, because these intersections can be triangulated in the same way if they have the same combinatorial structure.

We fix a pair i_0, j_0, and we distinguish between the different combinatorial structures of $A_{i_0} \cap \rho B_{j_0}$. So we fix a combinatorial structure. For this particular combinatorial structure, we let $h_0(\rho) = \mu_{i_0 j_0}(\rho)$, and we let $h_0(\rho) = 0$ when the combinatorial structure is different. We will show that the function $\rho \mapsto |h_0(\rho)|$ is a nice function, and as there are only $O(1)$ possible combinatorial structures for this fixed pair (i_0, j_0), it will prove that $\mu_{i_0 j_0}$ is a sum of $O(1)$ nice functions.

The polytope $A_{i_0} \cap \rho B_{j_0}$ is convex, and its vertices are intersection points of δ supporting hyperplanes of A_{i_0} or ρB_{j_0}. Hence, the coordinates of these vertices are degree-δ rational functions in the parameters (m_{ij}) and (v_i). For each given combinatorial structure, we consider a triangulation of $A_{i_0} \cap \rho B_{j_0}$ into $O(1)$ simplices. The volume $h_0(\rho)$ is the sum of the volume of the simplices in this triangulation, and since the coordinates of its vertices are degree-δ rational functions of (m_{ij}) and (v_i), the volume of each simplex is a sum of $O(1)$ degree-δ^2 rational functions of (m_{ij}) and (v_i). Besides, the combinatorial structure only changes when one of these vertices appears (or disappears) on $A_{i_0} \cap \rho B_{j_0}$; there are a constant number of such conditions, and each of them reduces to solving a constant size linear system. Therefore, the support of h_0 is a constant description complexity semialgebraic set, and thus h_0 is a nice function. □

4.3 Improved algorithm for maximum overlap in the plane

In this section, we give an algorithm for maximizing the area of overlap of two polygons. It is faster than the algorithm that we gave in Theorem 13 when $\delta = 2$. The main difference is that we apply the faster algorithm from Theorem 8 for maximizing a sum of nice functions in 3D.

As we mentioned earlier, Ahn et al. [4] gave an FPTAS for convex polygons. In the non-convex case, Mount, Silverman, and Wu. [22] gave an $O(n^4)$ *exact* algorithm for maximum overlap under translations. For non-convex polygons under rigid motions, Cheong, Efrat, and Har-Peled [13] gave a randomized algorithm whose running time is $O(n^3/E^8 \cdot \log^5 n)$ with high probability, and E is an *absolute* error bound: they allow an error of E times the area of the smallest polygon. By contrast, our algorithm is an FPTAS: it gives a *relative* error bound, which is a stronger requirement.

We obtain the following result, where β is a very slowly growing function as in Theorem 4.

Theorem 14 *Let δ be a constant integer and $\varepsilon \in (0,1)$. Let A and B denote two polygons, given as unions of n_a and n_b interior-disjoint triangles, respectively. Let $\mathcal{R}$ denote the set of rigid motions in $\mathbb{R}^2$. We can compute a rigid motion ρ_ε such that $(1+\varepsilon)|A \cap \rho_\varepsilon B| \geqslant \max_{\rho \in \mathcal{R}} |A \cap \rho B|$ in time $O\left(\left(\frac{n_a n_b}{\varepsilon}\right)^3 \log^4\left(\frac{n_a n_b}{\varepsilon}\right) \ \beta\left(\frac{n_a n_b}{\varepsilon}\right)\right)$.*

Proof: Let $\rho \in \mathcal{R}$ denote the rigid motion with angle θ and whose translational part is the vector (u, v). We denote $t = \tan(\theta/2)$, then $\cos(\theta) = (1-t^2)/(1+t^2)$ and $\sin(\theta) = 2t/(1+t^2)$. Then for any point (x, y), its image $(x', y') = \rho(x, y)$ is given by the equation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{1}{1+t^2} \begin{bmatrix} 1-t^2 & -2t \\ 2t & 1-t^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} u \\ v \end{bmatrix}. \tag{8}$$

For each pair consisting of the i^{th} triangle A_i of A and the j^{th} triangle B_j of B, we define the function $\mu_{ij} : \mathcal{R} \to \mathbb{R}$ such that $\forall \rho \in \mathcal{R}$, we have $\mu_{ij}(\rho) = |A_i \cap \rho B_j|$. From the discussion in Section 4.2, and by Equation (8), each function μ_{ij} can be seen as the sum of $O(1)$ nice functions with variables (t, u, v). We complete the proof by applying Theorem 8 with $d = 3$. □

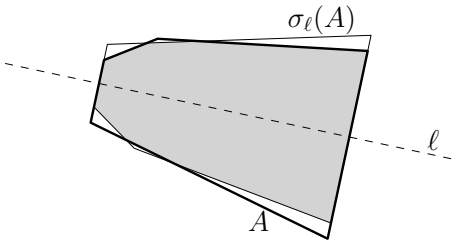

Figure 7: We want to maximize the shaded area $|A \cap \sigma_\ell(A)|$.

4.4 Largest axially symmetric subset

We consider the problem of computing a largest axially symmetric subset of a given polygon. The motivation is that the surface area of this subset is a measure of symmetry for the input polygon [8].

As we mentioned earlier, Barequet and Rogol [8] gave $O(n^4 T(n))$-time algorithm for this problem, where $T(n)$ is the average time needed by a numeric algorithm to maximize some rational functions. This numeric algorithm is not known to run in worst case polynomial time, although in practice, Barequet and Rogol observed that $T(n) = O(n)$. Ahn et al. [2] gave an FPTAS for finding the largest axially symmetric subset of a *convex* polygon

Our algorithm for finding the largest axially symmetric subset of a polygon is similar to the algorithm from Section 4.3 for maximizing the overlap of two polygons under rigid motions, but the time bound is better as this problem has only two degrees of freedom.

Theorem 15 *Let A denote a polygon with n vertices, and let $\varepsilon \in (0,1)$. In time $O\left(\frac{n^4}{\varepsilon^2}\log^2(\frac{n}{\varepsilon})\right)$, we can find a polygon $B_\varepsilon \subset A$ with axial symmetry such that*

$$\max\Big\{|B| \;\Big|\; B \text{ is axially symmetric and } B \subset A\Big\} \leqslant (1+\varepsilon)\cdot|B_\varepsilon|.$$

Proof: For any line $\ell \subset \mathbb{R}^2$, we denote by σ_ℓ the reflection with axis ℓ. Then the largest inscribed axially symmetric subset of A is the polygon $A \cap \sigma_\ell(A)$ with largest area [4, 8]. (See Fig. 7.) We use the same parameter $t = \tan(\theta/2)$ as in Section 4.2. When ℓ is the line that makes an angle $\theta/2$ with horizontal and contains the point $(0, h)$, then for any point (x, y), its image $(x', y') = \sigma_\ell(x, y)$ is given by the following equation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{1}{1+t^2}\begin{bmatrix} 1-t^2 & 2t \\ 2t & t^2-1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \frac{1}{1+t^2}\begin{bmatrix} -2th \\ 2h \end{bmatrix}. \tag{9}$$

We triangulate A, and we denote by A_i the i^{th} triangle. For each pair i, j, we denote $\sigma_{ij}(\ell) = |A_i \cap \sigma_\ell A_j|$. With the same argument as in the proof of Theorem 13, and using Equation (9), each function σ_{ij} is a sum of $O(1)$ nice functions with variables t, h. As our problem is to maximize $\sum_{i,j}\sigma_{ij}$, we conclude by applying Theorem 8. □

4.5 Minimizing the symmetric difference under rigid motions

We are given two polygons A and B with at most n vertices, and we want to find a rigid motion ρ that minimizes the area $|A \Delta \rho B|$, where $A \Delta \rho B$ denote the symmetric difference $(A \cup \rho B) \setminus (A \cap \rho B)$.

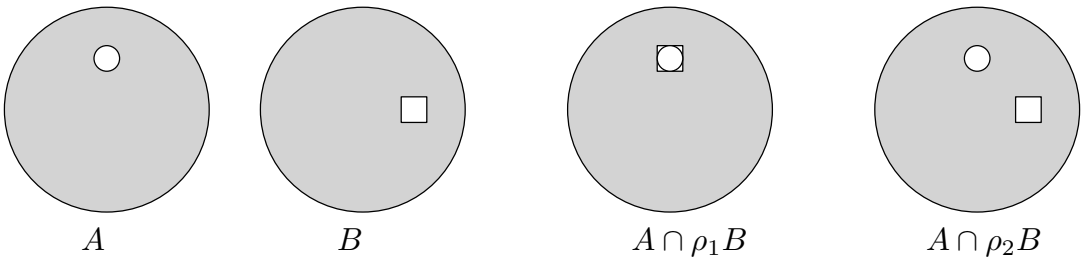

Figure 8: For these two shapes A and B, a 1.1-approximation of the symmetric difference under rigid motion would be given by the rigid motion ρ_1; note that the small disc and the small square are matched. It is not necessarily the case if we want a 1.1-approximate of the maximum overlap under rigid motion; in this case, the algorithm could return ρ_2 (right) and the small disc would not overlap the small square.

If we want an exact algorithm for this problem, then it is equivalent to maximizing the area of overlap, but the two problems are different if we only want a $(1+\varepsilon)$-approximation. (See Figure 8.)

We denote by $\mathcal{L}(\rho)$ the arrangement of the support lines of the edges of A and ρB. The *combinatorial structure* of $\mathcal{L}(\rho)$ is the set of incidence relations between the vertices, edges, and faces of $\mathcal{L}(\rho)$, as well as the cyclic order of the edges around each face. We handle separately each combinatorial structure. For a given combinatorial structure, we can use one triangulation of $A\Delta\rho B$ that works for all polygons ρB, so we are left with the problem of minimizing the sum of the areas of $O(n^2)$ triangles, which can be done using Theorem 9.

We now explain our algorithm in more details. The combinatorial structure of $\mathcal{L}(\rho)$ only changes when two support lines become equal, or three support lines intersect at one point. The coefficients in the equations of the support lines of the edges of ρB are degree-1 rational function in (t, u, v), where (t, u, v) is the same parameterization of ρ as in Section 4.3. Thus, the conditions for changing the combinatorial structure of $\mathcal{L}(\rho)$ are a set $\mathcal{S}_1$ of $O(n^3)$ polynomials in $\mathcal{P}$, with variables (t, u, v).

Using Shaul and Halperin's algorithm (Theorem 4), we construct a vertical decomposition of $\mathrm{arr}(\mathcal{S}_1)$ into $O(n^9\beta(n))$ constant complexity cells. For each cell $\mathcal{C}$ of this decomposition, we pick a rigid motion $\rho_{\mathcal{C}}$ in $\mathcal{C}$. Then we compute a triangulation of $\mathcal{L}(\rho_{\mathcal{C}})$. The combinatorial structure of this triangulation is still valid for any $\rho \in \mathcal{C}$, so we can express the area $|A\Delta\rho B|$ for any $\rho \in \mathcal{C}$ as a sum of $O(n^2)$ triangle areas, which are nice functions according to our definition. We restrict these functions to $\mathcal{C}$, using the $O(1)$ polynomial constraints that define $\mathcal{C}$. We apply Theorem 11 with $m = O(n^2)$ and $d = 3$, and so we get in time $O\left(n^{8+\varepsilon'} + \frac{n^6}{\varepsilon^3}\log^4(\frac{n}{\varepsilon})\beta(\frac{n}{\varepsilon})\right)$ an ε-approximation of the optimal rigid motion within $\mathcal{C}$. Repeating this process for each cell of the arrangement $\mathrm{arr}(\mathcal{S}_1)$, we obtain the following result:

Theorem 16 *Let $\mathcal{R}$ be the set of rigid motions in $\mathbb{R}^2$. Given two polygons A and B with at most n vertices, and $\varepsilon > 0$, we can compute a rigid motion ρ_ε such that $|A\Delta\rho_\varepsilon B| \leqslant (1+\varepsilon)\cdot\min_{\rho\in\mathcal{R}}|A\Delta\rho B|$ in time $O\left(n^{17+\varepsilon'} + \frac{n^{15}}{\varepsilon^3}\log^4(\frac{n}{\varepsilon})\beta(\frac{n}{\varepsilon})\beta(n)\right)$ for any $\varepsilon' > 0$.*

This result generalizes directly to an FPTAS for minimizing the symmetric difference of two polyhedra under rigid motion in arbitrary fixed dimension. We do not state the time bound, as it is even higher. As we mentioned in the introduction, it is the first FPTAS for this problem. The

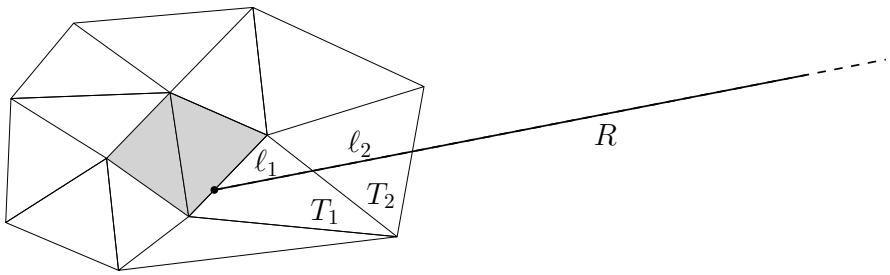

Figure 9: The target region is shaded. The cost of the ray R is $\kappa(R) = w_1\ell_1 + w_2\ell_2$, where w_1 and w_2 are the weights of simplices T_1 and T_2, respectively.

only previous result is by Alt et al. [5], who gave polynomial-time, constant-factor approximation algorithms for convex polygons.

4.6 Computing an optimal ray in a weighted subdivision

We are given a set of n interior-disjoint tetrahedra in $\mathbb{R}^3$. We use the L_p metric for some fixed integer p. Each tetrahedron T_i is associated with a positive weight w_i. The cost of a ray R is defined as

$$\kappa(R) = \sum_{i=1}^{n} w_i \|R \cap T_i\|_p,$$

where $\|R \cap T_i\|_p$ denotes the length of $R \cap T_i$ according to the L_p metric. (See Figure 9.) The optimal ray problem is to find a ray with minimum cost that starts at infinity and reaches a target region, which is the union of some of our n tetrahedra. This problem is motivated by medical applications [12]. For instance, we may want to perform a biopsy of the target region in such a way that the needle (modeled by the ray R) causes minimal damage, and hence the simplices with high weight would represent tissues that need to be preserved. Similarly, there are applications to radiation therapy.

The cost $\kappa(R)$ is minimized when the endpoint of R is at the boundary of the target region, so we can first restrict the problem to the case where the target region is a triangle. The cost function $\kappa_i(R) = w_i\|R \cap T_i\|_p$ restricted to a particular tetrahedron T_i can be written as a sum of a constant number of nice functions. Therefore, we could just apply Theorem 9 with $d = 4$. We will give a better time bound using a geometric observation that reduces the dimension to 2.

Consider a ray R_0 with endpoint r_0 in the interior of the target triangle. We first argue that we can reduce to the case where R_0 goes through an edge of a tetrahedron, using the sliding technique of Mitchell and Papadimitriou [21]. So we assume that R_0 does not cross any edge of any tetrahedron. Pick any line ℓ through r_0 and tangent to the target triangle. Let R be a ray obtained by translating R_0 along ℓ, that is, R is a ray with the same direction as R_0, and with endpoint $r \in \ell$. As was observed by Mitchell and Papadimitriou [21], when r is near r_0, the cost $\kappa(R)$ is an affine function of r, so it is non-decreasing when we move R in one of the two directions. So we move R in this direction until it first hits an edge of a tetrahedron, and thus we obtain a ray R_1 going through an edge with $\kappa(R_1) \leqslant \kappa(R_0)$.

We repeat this sliding process along the edge that intersects R_1, and we obtain a ray R_2 with cost at most $\kappa(R_0)$, such that R_2 goes through a vertex of a tetrahedron, or through two tetrahedron

edges. Hence, we have proved that the optimal ray contains a vertex, or goes through two edges. So, in order to find the optimal ray, we solve the problem separately for the rays through each vertex, and the rays through each pair of edges. We also need to consider all the possible target triangles that form the boundary of the target region. Thus, we have reduced the problem to $O(n^3)$ instances of a minimization problem with two degrees of freedom. Applying Theorem 11, we obtain the following result:

Theorem 17 *We can compute a $(1+\varepsilon)$-factor approximation of the optimal ray among n tetrahedra in time $O\left(n^{5+\varepsilon'} + \left(\frac{n^5}{\varepsilon^2}\right)\log^2\left(\frac{n}{\varepsilon}\right)\right)$ for any $\varepsilon' > 0$.*

This result generalizes to an FPTAS in arbitrary fixed dimension. We do not state the time bounds, as they would be considerably higher, and it is unclear whether there is any application in dimension higher than 3.

As we mentioned in the introduction, several results are known on this optimal ray problem. In 2D, Chen et al. [11] reduced this problem to $O(n^2)$ instances of a fractional program, which they solve using practical algorithms that do not provide worst case time bounds. The only theoretical result is by Daescu and Palmer [14], who gave an $O(n^3 \log^3 n)$-time algorithm in the arithmetic model of computation (as opposed to our algorithms which work in the unit cost model) for the 2D case. Their approach is based on fast algorithms for finding roots of univariate polynomials of degree n, and a geometric observation that reduces the number of variables to one. Hence, our algorithm is the first FPTAS for this problem in dimension higher than 2.

4.7 Minimum area hulls

Given a polygon $A \subset \mathbb{R}^2$ with n vertices, the *minimum area star-shaped hull* of A is a star-shaped polygon A^* with minimum area that contains A.

For any point $x \in \mathbb{R}^2$, we denote by $SH(x)$ the smallest polygon which is star shaped around x and contains A. This polygon $SH(X)$ is a union of at most n triangles, the vertices of each such triangle being x and two vertices of A. (See Figure 10.) Arkin et al. [7] showed that there is an

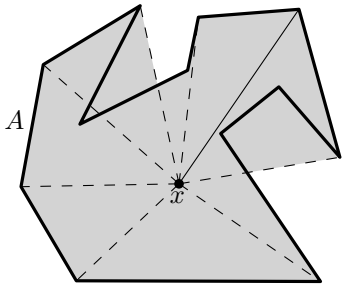

Figure 10: The shaded polygon is $SH(x)$, the smallest polygon containing the input polygon A which is star-shaped around x.

arrangement $\mathcal{A}$ of $O(n)$ lines such that, within each cell of this arrangement, the combinatorial structure of $SH(x)$ does not change.

So we triangulate this arrangement $\mathcal{A}$ and we obtain $O(n^2)$ triangles, such that in any such triangle T_i, the combinatorial structure of $SH(x)$ is fixed. For each T_i, and for each $x \in T_i$, the area $|SH(x)|$ is the sum of the areas of the at most n triangles that form $SH(x)$. Each such triangle has

two fixed vertices, the third vertex being x. Thus, $|SH(x)|$ restricted to T_i is a sum of at most n nice functions. There are only two degrees of freedom, as x is a point in the plane.

So we run the algorithm from Theorem 9 for each triangle T_i, with $d = 2$ and $m = n$. As there are $O(n^2)$ such triangles, we obtain:

Theorem 18 *Given a polygon with n vertices, we can compute a $(1+\varepsilon)$-factor approximation of its minimum area star-shaped hull in time $O\left(n^{4+\varepsilon'} + \left(\frac{n^4}{\varepsilon^2}\right)\log^2\left(\frac{n}{\varepsilon}\right)\right)$ for any $\varepsilon' > 0$.*

As we mentioned in the introduction, Arkin et al. [7] and Chen et al. [11] gave algorithms that reduce this problem to solving $O(n^2)$ fractional programs, which are solved by algorithms that are not known to run in polynomial time.

References

[1] P. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Discrete & Computational Geometry*, 15(1):1–13, 1996.

[2] H.-K. Ahn, P. Brass, O. Cheong, H.-S. Na, C.-S. Shin, and A. Vigneron. Inscribing an axially symmetric polygon and other approximation algorithms for planar convex sets. *Computational Geometry: Theory and Applications*, 33(3):152–164, 2006.

[3] H.-K. Ahn, S.-W. Cheng, and I. Reinbacher. Maximum overlap of convex polytopes under translation. In *Proc. International Symposium on Algorithms and Computation*, volume 6507 of *Lecture Notes in Computer Science*, pages 97–108, 2010.

[4] H.-K. Ahn, O. Cheong, C.-D. Park, C.-S. Shin, and A. Vigneron. Maximizing the overlap of two planar convex sets under rigid motions. *Computational Geometry: Theory and Applications*, 37(1):3–15, 2007.

[5] H. Alt, U. Fuchs, G. Rote, and G. Weber. Matching convex shapes with respect to the symmetric difference. *Algorithmica*, 21(1):89–103, 1998.

[6] N. Amato, M. Goodrich, and E. Ramos. Computing the arrangement of curve segments: divide-and-conquer algorithms via sampling. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 705–706, 2000.

[7] E. Arkin, Y. Chiang, M. Held, J. Mitchell, V. Sacristan, S. Skiena, and T.-H. Yang. On minimum-area hulls. *Algorithmica*, 21(1):119–136, 1998.

[8] G. Barequet and V. Rogol. Maximizing the area of an axially symmetric polygon inscribed in a simple polygon. *Computers & Graphics*, 31(1):127–136, 2007.

[9] S. Basu, R. Pollack, and M.-F. Roy. On computing a set of points meeting every cell defined by a family of polynomials on a variety. *Journal of Complexity*, 13(1):28–37, 1997.

[10] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer-Verlag, 2006.

[11] D. Chen, O. Daescu, Y. Dai, N. Katoh, X. Wu, and J. Xu. Efficient algorithms and implementations for optimizing the sum of linear fractional functions, with applications. *Journal of Combinatorial Optimization*, 9(1):69–90, 2005.

[12] D. Chen, O. Daescu, X. Hu, X. Wu, and J. Xu. Determining an optimal penetration among weighted regions in two and three dimensions. *Journal of Combinatorial Optimization*, 5(1):59–79, 2001.

[13] O. Cheong, A. Efrat, and S. Har-Peled. Finding a guard that sees most and a shop that sells most. *Discrete & Computational Geometry*, 37(4):545–563, 2007.

[14] O. Daescu and J. Palmer. Minimum separation in weighted subdivisions. *International Journal of Computational Geometry and Applications*, 19(1):33–57, 2009.

[15] D. Halperin. Arrangements. In J. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 2004.

[16] S. Har-Peled. How to get close to the median shape. *Computational Geometry: Theory and Applications*, 36(1):39–51, 2007.

[17] S. Har-Peled, V. Koltun, D. Song, and K. Goldberg. Efficient algorithms for shared camera control. In *Proc. ACM Symposium on Computational Geometry*, pages 68–77, 2003.

[18] V. Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. *Journal of the ACM*, 51(5):699–730, 2004.

[19] V. Koltun and M. Sharir. On overlays and minimization diagrams. *Discrete & Computational Geometry*, 41(3):385–397, 2009.

[20] J. Majhi, R. Janardan, M. Smid, and P. Gupta. On some geometric optimization problems in layered manufacturing. *Computational Geometry: Theory and Applications*, 12(3-4):219–239, 1999.

[21] J. Mitchell and C. Papadimitriou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

[22] D. Mount, R. Silverman, and A. Wu. On the area of overlap of translated polygons. *Computer Vision and Image Understanding*, 64(1):53–61, 1996.

[23] J. O'Rourke. Advanced problem 6369. *The American Mathematical Monthly*, 88(10):769, 1981.

[24] F. Preparata and I. Shamos. *Computational geometry: An introduction*. Texts and Monographs in Computer Science. Springer-Verlag, New York, 2nd edition, 1985.

[25] M. Sharir and P. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, 1995.

[26] H. Shaul and D. Halperin. Improved construction of vertical decompositions of three-dimensional arrangements. In *Proc. ACM Symposium on Computational Geometry*, pages 283–292, 2002.