A Framework for Robust Realistic Geometric Computations

Jeff Erickson
University of Illinois
jeffe@illinois.edu

Ivor van der Hoog
Utrecht University
i.d.vanderhoog@uu.nl

Tillmann Miltzow
Utrecht University
t.mitzow@gmail.com

Abstract

We propose a new paradigm for robust geometric computations that complements the classical fixed precision paradigm (*interval geometry*, *ε-geometry* and *stable* algorithms) and the exact geometric computation paradigm. We provide a framework where we study algorithmic problems under smoothed analysis of the input, the relaxation of the problem requirements, or the *witness* of a recognition problem. Our framework specifies a widely applicable set of prerequisites that make real RAM algorithms suitable for smoothed analysis. We prove that suitable algorithms can (under smoothed analysis) be robustly executed with expected logarithmic bit-precision. This shows in a formal way that inputs which need high bit-precision are contrived and that these algorithms are likely robust for realistic input. Interestingly our techniques generalize to problems with a natural notion of resource augmentation (geometric packing, the art gallery problem) and recognition problems (recognition of realizable order types or disk intersection graphs).

Our results have practical implications for many geometric algorithms: if their input is slightly perturbed before execution then the expected bit-precision for robust computation is logarithmic. Our results also have theoretical implications for some $\exists\mathbb{R}$-hard problems: we show that many real verification algorithms under slight perturbations of their witness require expected logarithmic bit-precision. These problems have input instances where their real verification algorithm requires at least exponential bit-precision which makes it difficult to place these $\exists\mathbb{R}$-hard problems in NP. Our results imply for a host of $\exists\mathbb{R}$-complete problems that this exponential bit-precision phenomenon comes from nearly degenerate instances.

It is not evident that problems that have a real verification algorithm belong to $\exists\mathbb{R}$. Therefore, we conclude with a real RAM analogue to the Cook-Levin Theorem, which shows that algorithmic problems belong to $\exists\mathbb{R}$, if and only if there is a real verification algorithm. This gives an easy proof of $\exists\mathbb{R}$-membership, as real verification algorithms are much more versatile than ETR-formulas.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases smoothed analysis, Existential Theory of the Reals, real-RAM, bit-precision, resource augmentation, verification algorithms, robust computations

Digital Object Identifier 10.4230/LIPIcs...

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

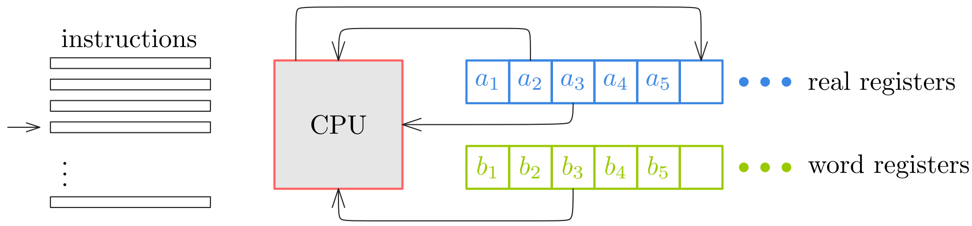

Figure 1. The dominant model in computational geometry is the real RAM. It consists of a central processing unit, which can operate on real and word registers in constant time, following a set of instructions.

1 Introduction

The RAM is a mathematical model of a computer which emulates how a computer can access and manipulate data. Within computational geometry, algorithms are often analyzed within the real RAM [48, 75, 99] (or the later Blum-Shub-Smale machine [17]) where values with infinite precision can be stored and compared in constant space and time. By allowing these infinite precision computations, it becomes possible to verify geometric primitives in constant time, which simplifies the analysis of geometric algorithms. Mairson and Stolfi [79] point out that "without this assumption it is virtually impossible to prove the correctness of any geometric algorithms." The downside of the real RAM is that it neglects the bit-precision of the underlying algorithms, although they are very important in practice. If an algorithm can be correctly executed with a limited bit-precision then the algorithm is called *robust*. Many classical examples in computational geometry are inherently nonrobust [99] and there are even examples which in the worst case require a bit-precision exponential in n in order to be correctly executed [53, 65].

Often inputs which require exponential bit-precision are contrived and do not resemble *realistic* inputs. A natural way to capture this from a theoretical perspective is smoothed analysis, which interpolates *smoothly* between worst case analysis and average case analysis [110]. Practical inputs are constructed inherently with small amount of noise and random perturbation. This perturbation helps to show performance guarantees in terms of the input size and and the magnitude of the perturbation. By now smoothed analysis is well-established, for instance Spielman and Teng received the Gödel Prize for it. However, within computational geometry its application is limited to smoothed analysis of the bit-precision of the art gallery problem [38] and order type realisability [63], and smoothed analysis of the runtime of k-means clustering [5, 80], Euclidean TSP [41, 81], and partitioning algorithms for Euclidean functionals [15].

In this paper, we apply smoothed analysis to the real RAM to show that many geometric algorithms can be correctly executed in practice with logarithmic bit-precision. This analysis has implications on several levels: (1) Many classical nonrobust geometric algorithms (computing Delaunay triangulations, convex hulls or order types) can in practice be executed using finite precision. (2) Many classical optimization problems which have a natural problem relaxation (the art gallery problem, geometric packing, computing the minimum-link path) can in practice be executed using finite precision. (3) Solutions to many classical $\exists\mathbb{R}$-complete problems (such as disk intersection-graph recognition or the Steinitz problem in fixed dimension) have polynomial size. (4) The classical sum of square roots problem can be correctly solved with a bit-precision which is linear in the number of square roots that are summed. Point (4) provides theoretical verification of an old observation that in practice this problem only requires linear bit-precision [112, Ch. 45].

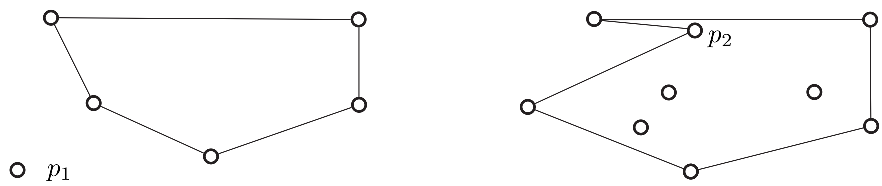

Figure 2. Examples of how computing the convex hull can fail due to imprecise computations from [67]. Left: the point p_1 is very close to the origin and thus left out. Right: the point p_2 is mistakenly added to the convex hull and produces an obvious concave chain.

RAM computations. The Random Access Machine (RAM) is a model of computation for the standard computer architecture. The precise definition of the RAM varies, but at its core the RAM has a number of *registers* and a *central processing unit* (CPU), which can perform operations on register values like reading, writing, comparisons, and arithmetic operations. The canonical model of computation within computer science is the *word RAM*, a variation on the RAM formalized by Hagerup [54] but previously considered by Fredman and Willard [49, 50] and even earlier by Kirkpatrick and Reisch [69]. The word RAM models two crucial aspects of real-life computing: (1) computers must store values with finite precision and (2) computers take more time to perform computations if the input of the computation is longer. Specifically, the word RAM supports constant-time operations on w-bit integers, where the *word size* w is a parameter of the model.

The field of computational geometry uses a different variation of the RAM called the *real RAM*, where registers may contain real numbers, instead of just integers; see Appendix A.1. This abstraction dramatically simplifies the design and analysis of geometric algorithms, at the cost of working in an physically unrealistic model. Implementations of real RAM algorithms using finite-precision data types are prone to errors. Not only because the output becomes imprecise, but because rounding errors can lead the algorithm into inconsistent states (Figure 2). Kettner [67] provides a recent overview of the problems that nonrobust geometric algorithms can bring. Nonrobust primitives are typical for naive implementations of geometric algorithms [79, 112], and unsurprisingly, there is a vast amount of research into *robust geometric computation*.

Robust geometric computation. Yao and Sharma [112, Ch. 45] present an extensive overview of techniques used to obtain robust geometric computations. They describe two broad paradigms to obtain provably robust computations. In the first paradigm, called *fixed precision approaches*, the goal is to execute a geometric algorithm using fixed bit-precision. If you have a fixed bit-precision, it can become impossible to correctly test certain geometric primitives (e.g. in-circle testing, collinearity testing) and it is therefore impossible to prove that the output is correct according to these geometric primitives. This is why approaches under this paradigm invent alternative (weaker) geometric primitives and they prove that they can construct output which satisfies the alternative primitives. The fixed paradigm includes well-known approaches such as *interval geometry* [44, 104, 105], *ε-geometry* [99], *strong algorithmic stability* [47] and the more recent *topological stability* [64, 84]. Recent examples that fall under the fixed precision paradigm are papers that discuss the construction of almost-Delaunay simplices [7], Delaunay triangulations of imprecise points [77], stability analysis of Voronoi diagram [93] and sorting of imprecise points [62]. An appealing property of this paradigm is that solutions are provably correct or stable. However, the notion of stability or correctness at times may be undesirable. For example under ε-geometry it is possible that two triples of planar points (a, b, c) and (b, c, d) are collinear even though the triple (a, b, d) is not collinear.

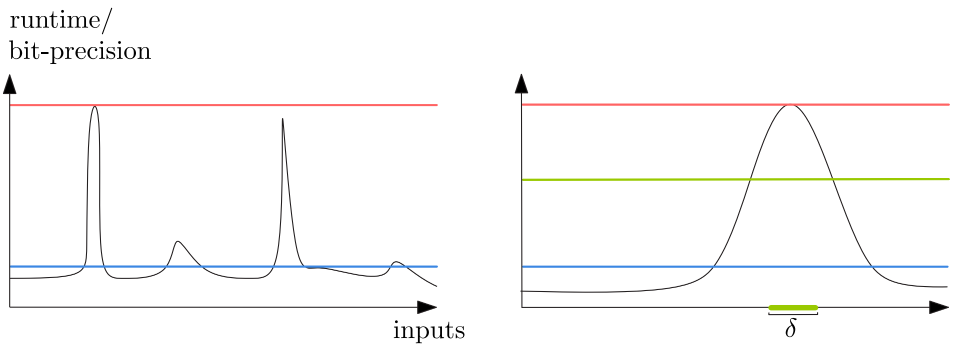

Figure 3. The x-axis symbolizes all inputs. The red line indicates the worse case running time or required bit-precision. The blue line indicates the average however, typical instances are not always average. Smoothed analysis considers the average of inputs near some worst instance (shown in green).

The second paradigm Yao and Sharma call the *exact approach*; here, geometric primitives are given a representation that allows their precision to be lazily evaluated. Such an algorithm detects if a primitive can be correctly determined with limited bit-precision and if it cannot, the algorithm can increase the bit-precision of the involved variables accordingly. This approach is sometimes referred to as *exact geometric computation* (EGC) and it is an active research field in experimental computational geometry [8, 18, 35, 48, 56, 114–116]. There are also theoretical results in this paradigm such as recent work which computes the expected running time of computing a Delaunay triangulation [22], verifies order type representation [25, 35] or a deterministic subquadratic precision bound for representing order types without coordinates [24]. Perhaps most notable is the fact that the CGAL Core library makes use of the EGC principle [18, 55, 56].

In this paper we propose a third paradigm for robust geometric computations. We apply smoothed analysis to either the input of an algorithm (Section 2 and 3), the rigidity of its geometric primitives (Section 4), or the *witness* of a recognition problem (Section 5). Our analysis implies that examples that need high bit-precision are nearly degenerate. Thus, under realistic input conditions, classical algorithms are robust with only logarithmic bit-precision.

Smoothed analysis. In *smoothed analysis*, the performance of an algorithm is studied for worst case input which is randomly perturbed by a magnitude of δ. Intuitively, smoothed analysis interpolates between average case and worst case analysis (Figure 3). The smaller δ, the closer we are to true worst case input. Correspondingly larger δ is closer to the average case analysis. The key difficulty in applying smoothed analysis is that one has to argue about both worst case and average case input. Following [33, 110] we perceive an algorithm to run in polynomial time in practice, if the expected runtime of the algorithm is polynomial in the input size n and in $1/\delta$. Spielman and Teng explain their analysis by applying it to the simplex algorithm, which was known for a particularly good performance in practice that was seemingly impossible to verify theoretically [70]. Since the introduction of smoothed analysis, it has been applied to numerous problems. For example the smoothed analysis of the Nemhauser-Ullmann algorithm [88] for the knapsack problem shows that it runs in polynomial time polynomial in n/δ [10]. A more general result that was obtained using smoothed analysis is the following: all binary optimization problems (in fact, even a larger class of combinatorial problems) can be solved in smoothed polynomial time if and only if they can be solved in pseudopolynomial time [11]. Other famous examples are the smoothed analysis of the k-means algorithm [6], the 2-OPT TSP local search algorithm [41], and the local search algorithm for MaxCut [43]. Not surprisingly, teaching material on this subject has become available [96–98]. Most relevant for us is the recent smoothed analysis of the art gallery

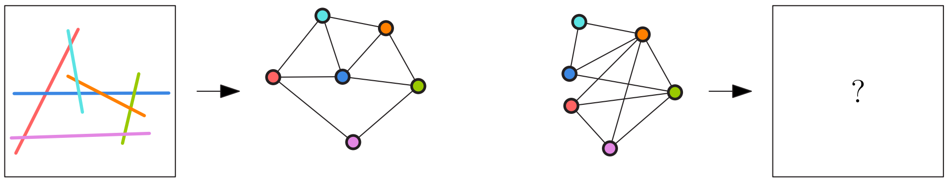

Figure 4. Left: given a set of segments S, they define a segment intersection graph G_S. Right: given a graph G, is there a set of segments S' such that $G_{S'} = G$?

problem [38] and of order types [63]. Both papers deal with the required bit-precision needed in computations under slight perturbations. The surprising similarity in the proof techniques for these two problems inspired us to generalize this work to the largest possible extent, namely to the real RAM. We formally define our model of smoothed analysis in Appendix A.2.

The Existential Theory of the Reals. The required precision of an algorithm plays an important role if we want to show that a problem lies in the class NP. It is often easy to describe a potential witness to an NP-hard problem, but the bit-precision of the witness is unknown. A concrete example is the recognition of segment intersection graphs (Figure 4): given a graph, can we represent it as the intersection graph of segments? Matoušek [82] comments on this as follows:

Serious people seriously conjectured that the number of digits can be polynomially bounded—but it cannot.

Indeed, there are examples which require an exponential number of bits in any numerical representation. This *exponential bit-precision phenomenon* occurs not only for segment intersection graphs, but also for many other natural algorithmic problems [1–3, 14, 26, 27, 37, 39, 42, 51, 66, 71, 78, 83, 85, 94, 100–102, 107–109]. It turns out that all of those algorithmic problems do not accidentally require exponential bit-precision, but are closely linked, as they are all complete for a certain complexity class called $\exists\mathbb{R}$. Thus either all of those problems belong to NP, or none of them do. Using our results on smoothed analysis, we show that for many $\exists\mathbb{R}$-hard problems the exponential bit-precision phenomenon only occurs for near-degenerate input.

The complexity class $\exists\mathbb{R}$ can be defined as the set of decision problems that are polynomial-time equivalent to deciding if a formula of the *Existential Theory of the Reals* (ETR) is true or not. An ETR formula has the form:

$$\Psi = \exists x_1, \ldots, x_n \quad \Phi(x_1, \ldots, x_n),$$

where Φ is a well-formed sentence over the alphabet

$$\Sigma = \{0, 1, x_1, \ldots, +, \cdot, =, \leq, <, \wedge, \vee, \neg\}.$$

More specifically, Φ is quantifier-free and $x_1, \ldots, x_n$ are all variables of Φ. We say Ψ is true if and only if there are real numbers $x_1, \ldots, x_n \in \mathbb{R}$ such that $\Phi(x_1, \ldots, x_n)$ is true.

Formally modeling algorithms and robustness. We define in Appendix A.1 a framework for real RAM algorithms to which we can apply smoothed analysis. A real RAM algorithm A receives a combination of real and integer input $I = (a, b) \in \mathbb{R}^n \times \mathbb{Z}^m$ and performs a series of instructions. Importantly, integer and real valued computations are never mixed. The order of the program flow is only determined by the *comparison instructions*. We say two inputs I and I' are *equivalent* under A if A makes the same decision at every comparison instruction. Note that for equivalent inputs, an algorithm A always outputs the same combinatorial solution. We say A is *robust* if for every input $I = (a, b) \in \mathbb{R}^n \times \mathbb{Z}^m$, there is an equivalent input $I' = (a', b)$, such that a' has bounded bit-precision. Let us consider the real RAM without the square root or trigonometric

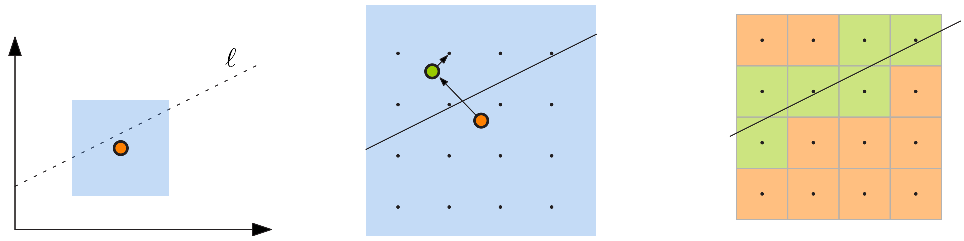

Figure 5. Given real input (a_1, a_2), we want to decide if the point (in orange) $p = (a_1, a_2)$ lies above or below the line ℓ $(y = x/2 + 1)$. If the point p lies very close to ℓ, we need a very high precision to make a correct decision. However, if p was perturbed slightly, low precision is sufficient to make a correct decision correctly with high probability and in expectation. Note that rounding the coordinates of p to w bits corresponds to snapping p to a grid width $\omega = 2^{-w}$.

operations. At all times during the computation, a real register holds a value which can be described as the quotient of two polynomials $\frac{p}{q}$ of the real input values a.

Note that adding, subtracting, or multiplying two rational functions yields another rational function, possibly of higher degree; for example, $\frac{p_1}{q_1} + \frac{p_2}{q_2} = \frac{p_1q_2+p_2q_1}{q_1q_2}$. We say an algorithm A has *algebraic-degree* Δ, if p and q always have total degree at most Δ. Similarly, A has *algebraic-dimension* d, if the number of variables in p and q are always at most d.

We assume that our real-valued input can be represented as a higher-dimensional point $a \in [0,1]^n$. This normalization assumption makes it easier to phrase our results, as the meaning of the magnitude of perturbation is with respect to the size of the involved numbers. Given a, the corresponding integer-valued input a' limited precision as follows: is the closest point to a in the scaled integer lattice $\Gamma_\omega = \omega\mathbb{Z}^d$. (We refer to this transformation as *snapping*.) We upper bound the minimal number of bits required for a correct execution of A by proving lower bounds for the scale factor ω.

Results and paper structure. In the following sections we present a framework to apply smoothed analysis to a wide class of real RAM algorithms. The real RAM and the abstract notion of an algorithm are often-debated, multifaceted concepts. The discussion of their intricacies and subtleties are technical and involved, and therefore ill-suited for the main body of this paper. We wish to highlight the generality of our framework and the wide practical and theoretical implications that it has. At the same time, we incline to not over-state our results. Therefore we defer the formal definition of the real RAM and the proofs of the theorems that rely on this definition to the appendix and instead use the main body of the paper to carefully state and explain the results. We believe one of the strengths of our results is that given the real RAM definitions, our proofs are not overly involved, which makes the framework extendable if needed.

The following sections discuss our results, their implications, *and* their limitations. Section 2 shows that for realistic input, classical geometrical algorithms are robust. Section 4 shows that resource augmentation of hard problems is a feasible model for their practical performance. Section 5 shows that a typical witness to a recognition problem can be described using logarithmic bit-precision. Section 6 shows that real verification algorithms are robust under perturbations of the witness and Section 7 shows that having a real verification algorithm is equivalent to $\exists\mathbb{R}$-membership.

2 Results of Smoothed Analysis of real RAM Algorithms

After precisely definining of both real RAM algorithms and smoothed analysis in Appendix A, we study real RAM algorithms of bounded algebraic-degree and algebraic-dimension under smoothed analysis of their input. Specifically, in Appendix B, we prove the following theorem:

► **Theorem 1.** *Consider the real RAM without square roots and a polynomial time algorithm A with algebraic-degree* Δ *and constant algebraic-dimension. Then under perturbations of the input of magnitude* δ*, A can be robustly executed with an expected input bit-precision of* $O\left(\log \frac{\Delta n}{\delta}\right)$ *and an expected intermediary precision of* $O\left(\Delta^2 \log \frac{\Delta n}{\delta}\right)$.

It is important to point out that in the proof of this theorem, we bound only the bit-precision of the real-valued input, and not the bit-precision of any intermediate results. However, it is easy to see that if we have d integers $a_1, \ldots, a_d$ represented using k bits each, then any d-variate polynomial of degree at most Δ can be evaluated using at most $O(\Delta^2 k \log d)$ bits of precision.

The proof idea (Figure 5) is to consider the algorithm A with perturbed input $I_x = (a+x, b)$. Where a and b are arbitrary input and x is a small perturbation. We model the perturbed input I_x as a high-dimensional point which we snap to a fine grid to obtain I' (input which can be described using bounded precision). We then show that, for any algorithm A that meets our prerequisites, I' and I_x are equivalent with high probability. We upper bound the probability that I' and I_x are not equivalent as follows: the input I_x is snapped to the center point of a fine grid and the center points of the grid define a Voronoi diagram. The content of a real RAM register for a specific comparison instruction, is per assumption the quotient of two polynomials whose variables depend on the input. The core argument is, that if I_x lies in a Voronoi cell which is not intersected by the variety of either of the two polynomials, then the comparison instruction will be computed correctly. We upper bound the proportion of Voronoi cells that are intersected by the variety of a polynomial with Theorem 20 in Appendix D.

Implications of Theorem 1. Theorem 1 has implications for many classically nonrobust real RAM algorithms that rely only on geometric primitives that have constant algebraic-degree and algebraic-dimension. We point out some of them in the following corollary.

► **Corollary 2** (of Theorem 1)**.** *The following problems can under perturbations of the input of magnitude* δ *be robustly solved with expected bit-precision* $O(\log(n/\delta))$ *for input and computations:*
- *Sorting real numbers.*
- *Computing the convex hull of a point set in fixed dimension.*
- *Computing Delaunay-triangulation in fixed dimension.*
- *Checking if a set of guards surveys an entire polygon.*

Let us discuss the computation of the Delaunay triangulation and the convex hull in more detail. Computing the convex hull in constant dimension is nonrobust [67] since the orientation test cannot be computed in a robust manner. Since convex hulls are one of the classical geometric structures, their robustness has been extensively studied under various conditions and in various dimensions [4, 20, 60, 76, 87, 99]. We note that orientation testing in constant dimension has constant algebraic-degree and algebraic-dimension. Thus, Theorem 1 implies that the convex hulls of *realistic* input sets can be robustly computed with only logarithmic bit-precision.

The nonrobustness of the in-circle test implies nonrobustness of computing Delaunay triangulations. Many geometric algorithms rely on Delaunay triangulations and therefore its robustness has been extensively studied. Studied topics include exact computing of triangulations [86, 91], Delaunay-like structures with more forgiving primitives [7, 8, 77], and Delaunay triangulations of imprecise as opposed to real input [21, 62, 68, 113]. The incircle test considers at most two points

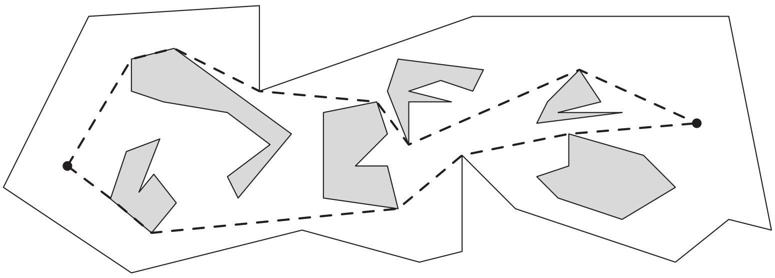

Figure 6. In a polygonal domain with holes, we want to compare the length of two given paths.

and a circle, and therefore has constant algebraic-degree and algebraic-dimension in constant dimension. It follows that Delaunay triangulations of *realistic* constant-dimensional point sets can be robustly computed with only logarithmic bit-precision.

Limitations. We want to briefly give examples of algorithmic problems that do not fit the framework that Theorem 1 provides. Algorithmic problems that have non-constant algebraic-dimension do not satisfy the conditions of Theorem 1. There are geometric questions, such as computing the average width of a point set, that are defined on input of arbitrary size; Theorem 1 cannot be applied to algorithms that use these kind of primitives. Geometric primitives that have algebraic-degree exponential in n do fit in this framework; however, the outcome of the Theorem may not be desirable. Recursive real RAM algorithms may potentially have very high algebraic-degree. Also our methods are not useful for problems that explicitly try to construct degenerate (measure zero) output. As an example consider the 3SUM problem. Under our perturbation model, with probability one, a perturbed input to 3SUM contains no three elements that sum to zero. Similarly, randomly perturbed points in the plane almost never contain three collinear points. Thus the notion of random perturbation may not be sensible for problems that explicitly deal with degeneracies. Lastly Theorem 1 is not applicable to algorithms that use square roots (unless they can be easily eliminated). The required bit-precision for the sum of square roots is an open problem within computational geometry [89, 90]. Therefore we extend our framework to specifically include square roots.

3 Results of Smoothed Analysis of real RAM Algorithms with Square Roots

In Appendix B we work under the assumption that at all times, every real register $R[i]$ contains a value that can be expressed as the quotient of two d-variate polynomials f_i, g_i of maximal degree Δ. We strengthen our analysis in Appendix C by allowing the square root operation. This means that real registers are even allowed to contain contrived expressions f such as $\sqrt{\frac{a_1^3+\sqrt{a_2}}{a_3-\sqrt{5a_4}}}$. We transform each such expression f into an *equivalent* set of polynomials $\{p, q_1, \ldots, q_s\}$. Intuitively the introduction of each square root introduces one extra *algebraic-dimension*. The algebraic-degree and the algebraic-dimension of f are inherited from $p, q_1, \ldots, q_s$.

► **Theorem 3.** *Consider the real RAM with square roots and a polynomial time algorithm A with algebraic-degree Δ, algebraic-dimension d, and extra algebraic-dimension s. Then under perturbations of the input of magnitude δ, A can be correctly executed on a real RAM if the input has an expected required bit-precision of $O\left((d+s)\log\frac{\Delta n}{\delta}\right)$.*

Implications of Theorem 3. The theorem conditions are parametrized by the abstract extra algebraic-dimension. This makes the theorem apply to a wide class of algorithmic problems that in some way or another need to evaluate square root functions, but perhaps makes it harder to see its applications. Therefore we wish to highlight two applications of the theorem. The first is on the classical sum of square roots algorithm, which takes as input the real-valued numbers $a_1, \ldots, a_n, b_1, \ldots, b_n$ and decides if $\sum_{i=1,\ldots,n} \sqrt{a_i} - \sum_{i=1,\ldots,n} \sqrt{b_i} > 0$, by computing the sums with limited bit-precision. Yap and Sharma observe that the best known theoretical bounds for the required bit-precision of the input is exponential in n [112, Ch. 45]. However, they note that in practice it has been observed that a near-linear bit-precision seems sufficient. The following corollary gives a theoretical explanation for this practical observation:

► **Corollary 4.** *Under perturbations of the input of magnitude δ, the sum of square roots can be computed on a real RAM with an expected bit-precision of $O(n \log(n/\delta))$ per input variable.*

A second implication is the computation of the shortest path in a simple polygon with holes. The bottleneck here is to compare the length of two given paths, see Figure 6.

► **Corollary 5.** *Under perturbations of the input of magnitude δ, the shortest path in a polygon can be computed on a real RAM with an expected bit-precision of $O(n \log(n/\delta))$ per path vertex variable.*

Limitations. There are two limitations of these results that we wish to mention. First of all, this theorem gives an upper bound on the expected bit-precision of the *input* variables. The difference is, that in Theorem 1 we could also prove that the precision needed for intermediate computations is bounded. In particular when one starts to recursively apply the square root operation, the required intermediate precision can explode. Second, the linear bit-precision needed for the sum of square roots and shortest path problems might be considered unpractical.

4 Results of Smoothed Analysis of Resource Augmentation

The predominant approach to find decent solutions for hard optimization problems is to compute an approximation. An alternative approach is resource augmentation, where you consider an optimal solution subject to a slightly weaker problem constraints. This alternative approach has considerably less traction in theoretical computer science than approximation algorithms have. We want to emphasize that resource augmentation algorithms find a solution which does not compromise the optimality. Using smoothed analysis, we argue that studying slight relaxations of algorithmic problems is justifyable for practical applications of the algorithm as we show that the problem conditions that make the problem hard are brittle.

An example of resource augmentation exists for the geometric packing problem (Figure 7) where an algorithm needs to pack a set of convex objects into a unit-size square container. To pack the optimal number of objects into this container is $\exists\mathbb{R}$-complete [3] and therefore a word RAM algorithm cannot hope to correctly find an optimal solution with limited time or memory. A resource augmentation algorithm looks to find a way to pack as many objects into a container C' which is larger by a factor $(1+\alpha)$ (α being the augmentation parameter). We apply smoothed analysis to resource augmentation problems, where we study these problems under a slight perturbation of the augmentation parameter. We prove in Appendix F resource augmentation problems have expected logarithmic bit-precision and give the optimal solution.

► **Theorem 6.** *Let P be a resource augmentation problem that is monotonous, moderate and smoothable. Under perturbations of the augmentation of magnitude δ, the problem P has an optimal solution with an expected bit-precision of $O(\log(n/\delta))$.*

In the proof of this theorem (Appendix F) we argue about the solution space of the problem P and we define three natural properties of this solution space. The *monotonous property* demands

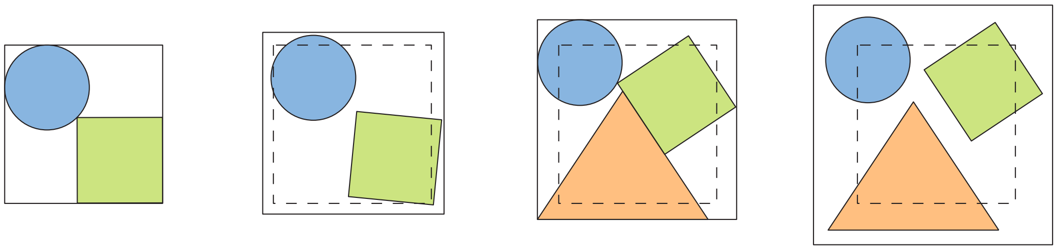

Figure 7. We augment the container from left to right. This extra space can lead to a better solution. If the optimal solution does not change, the extra space allows for a solution with low bit-precision.

that as we relax P more and more, the solution space only gains more candidate solutions. The *moderate property* demands that as we continuously relax the problem, we do not encounter more than a polynomial number of new optimums. In many hard optimization problems, the optimum is a value between 1 and n and the moderate property is then immediately implied. The *smoothable property* is the least intuitive of the three, it demands if you relax a problem P by ε, then it contains a solution which is optimal for the *original* problem and has a bit-precision of $O(\log(n/\varepsilon))$. It might appear as though the third property immediately implies the theorem, yet recall that we look for an optimal solution for the newly relaxed problem. The other two properties, together with common bounds in probability theory bound the expected bit-precision of an optimal solution to the perturbed problem.

Implications of Theorem 6. To illustrate the applicability of our findings, we give the following corollary. The first result was already shown in [38]. Recently Kostitsyna et al. showed that an optimal solution to the minimum-link path must have at least linear bit-precision [72]. The art gallery problem has been shown to be $\exists\mathbb{R}$-complete [2], and currently $\exists\mathbb{R}$-completeness of the packing problems is in preparation [3]. Our results imply that apart from near-degenerate conditions the solutions to these problems have logarithmic bit-precision.

► **Corollary 7.** *Under perturbations of the augmentation of magnitude δ, the following problems have an optimal solution with an expected bit-precision of $O(\log(n/\delta))$.*

- *the art gallery problem under perturbation of edge inflation [38].*
- *packing polygonal objects into a square container under perturbation of the container width.*
- *computing the minimum-link path in a simple polygon under perturbation of edge inflation.*

Limitations. Given an algorithmic problem, it is not clear *a priori* whether there is a way to augment resources such that it is both mathematically sound, satisfying, as well as practically plausible. For example, if we search for the smallest square container that fits a given set of items, the number of changes in the optimum is unbounded thus the moderate property does not hold.

5 Results of Smoothed Analysis of Recognition Problems

In computational geometry, we study many different geometric objects like point sets, polytopes, disks, balls, line-arrangements, segments, and rays. Many algorithms *only* use combinatorial properties of the underlying geometry. A recent impressive example is the EPTAS for the clique problem on disk intersection graphs by Bonamy et al. [19]. In the paper they first derive a set of properties for disk intersection graphs (including that they contain no two induced odd-cycles in their complement) and then they use *only* those properties to find a new EPTAS.

Suppose that a geometric problem can be solved using only precomputed combinatorial properties. Then given the combinatorial structure, we do not require real RAM computations

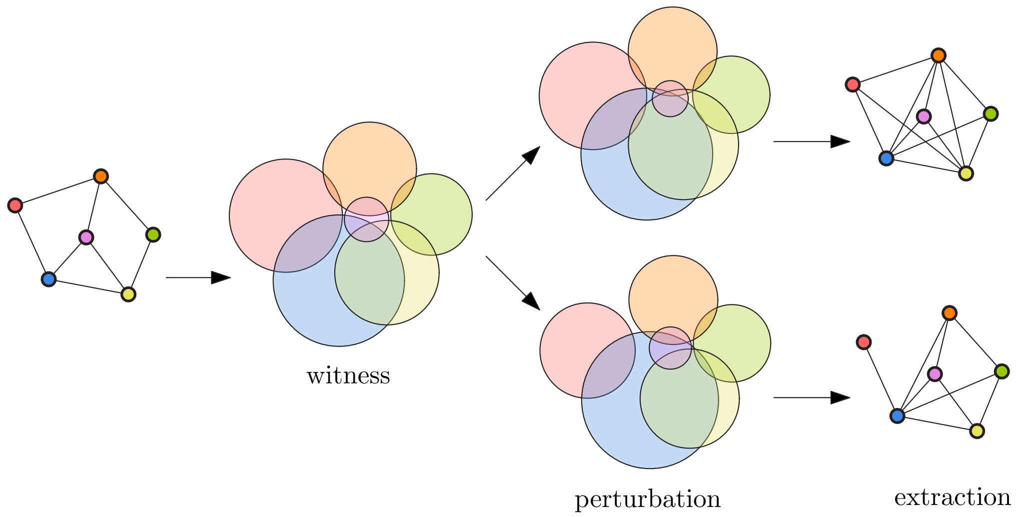

Figure 8. Given a disk intersection graph (combinatorial structure) G, there exists by definition a set of disks D (witness) representing $G = G_D$. We perturb D slightly. Each perturbation x defines a new set of disks D_x. And in turn each set D_x defines a new graph G_x. In this way, we us the perturbation on the disks to attain indirectly a probability distribution over the graphs.

and the algorithm is robust. Indeed, Bonamy et al. emphasize that their algorithm also works without the disk representation. The crux for this approach, is that we first need a family of properties that describe all geometric objects of a certain type. This is the motivation for recognition problems. The formal definition of recognition problems is presented in Appendix E but a classical example is the problem of recognizing disk intersection graphs (Figure 8). In this algorithmic problem the input is a graph G and we ask if there is a set of disks D, such that the disk intersection graph G_D equals G. In other words, we wonder whether there exists a simple characterization of the class of graphs that can arise as disk intersection graphs. The answer is likely no as it is $\exists\mathbb{R}$-complete to recognize disk intersection graphs [83]. By now, we know that the same holds for many recognition problems including the Steinitz problem, recognising visibility and unit distance graphs [23, 94, 101]. The overarching message is that we cannot forget about the geometry, when we seek for new algorithms. Our smoothed analysis shows that for typical instances at least the bit-precision of these hard recognition problems can be bounded.

In order to apply the concept of smoothed analysis, we need to define the concept of perturbation. One naive way to define perturbations for graph recognition is to alter the edges of the graph, by randomly adding or removing a δ-fraction of all the edges. There are several disadvantages to this approach. First of all, it will lead to a majority of the instances being non-representable. The reason is that if an induced subgraph $H \subset G$ cannot be represented, than also the entire graph G cannot be represented. The second reason, is that this perturbation is not geometrically motivated. In our approach, we define the perturbation in terms of the underlying geometric representation of the given combinatorial input. This in turn, gives indirectly a distribution on the combinatorial structure. We illustrate this process for disk intersection graphs. Given some graph G, we consider any disk representation D. We define a notion of perturbation on D, in a natural geometric way. Thus every perturbation x, gives rise to a new set of disks D_x, which in turn defines a new graph G_x. In this way, we have defined indirectly a model of perturbation on the graphs. The justification for this model of perturbation is the assumption that the combinatorial structures came from some geometric objects subject to noise.

► **Theorem 8.** *Let A be an extraction algorithm with a recognition problem R_A and denote by n_1 and n_2 the size of the input for A and R_A respectively. Suppose that under perturbations of the input of A, the algorithm A can be robustly executed if the input has an expected maximal bit-precision of $O(\log(n_1/\delta))$. Then the recognition problem R_A, under slight perturbation of the witness, can be robustly verified with an expected bit-precision of $O(\log(n_2/\delta))$.*

The proof of Theorem 8 follows almost immediately from the definition of an extraction algorithm, see Appendix E. together with Theorem 1. The real-valued input of an extraction algorithm is the *witness* of the recognition problem. The simplicity of its correctness contrasts its implications.

Implications of Theorem 8. The theorem applies to an array of recognition problems:

► **Corollary 9.** *The following recognition problems under perturbations of the witness of magnitude δ admit solutions with an expected bit-precision of $O(\log n/\delta)$:*

- *Recognition of realizable order types [63] (equivalent to Stretchability).*
- *Recognition of disk intersection graphs.*
- *Recognition of segment intersection graphs.*
- *Recognition of ray intersection graphs.*
- *The Steinitz Problem in fixed dimension.*

The practical implications of these corollaries are that apart from near degenerate examples, realizable order types and intersection graphs can be represented by geometric objects that are described with logarithmic bit-precision. The theoretical implications follow in the next section.

Limitations. We want to point out that we cannot handle all recognition problems. First of all it may be that the extraction algorithm does not meet the conditions of Theorem 1. Usually, this would be the case if the problem deals with unbounded dimension. We still get some bounds on the bit-precision but they may be less desirable. A concrete example is the recognition of ball intersection graphs, where the dimension of the ball is part of the input. Secondly perturbing a witness may not be a sensible idea. It does not mean that our theorems do not apply in a mathematical sense, but rather that in reality the result may be less desirable. Usually, this applies to problems that rely on degeneracies in one way or another. A concrete example is the point visibility graph recognition problem. Given a set of points P, we define a graph by making two points $p, q \in P$ adjacent if line segment pq contains no other point of P. This in turn defines a recognition problem where we are given a visibility graph G and we look for a point set P_G that realizes this visibility graph. If we perturb the real-valued witness P_G then with probability 1 there are no three collinear points. Thus, the point visibility graph will always be a clique.

6 Narrowing the Gap between NP and $\exists\mathbb{R}$

Sections 4 and 5 and the corollaries there give a logarithmic upper bound on the expected bit-precision for the input and the output of $\exists\mathbb{R}$-complete algorithmic problems (except for computing the minimum-link path). These problems have worst case inputs where their solutions are geometric objects whose description requires at least exponential bit-precision. The exponential bit-precision phenomenon is the bottleneck for NP-membership of all these $\exists\mathbb{R}$-complete problems. Our analysis shows that, in the context of the problems mentioned in Corollary 7 and 9, the gap between $\exists\mathbb{R}$ and NP (if it exists) is formed by near-degenerate input. Naturally, our analysis applies to the real verification algorithm of $\exists\mathbb{R}$-complete problems. We show in the next section that the existence of a real verification algorithm is equivalent to $\exists\mathbb{R}$-membership.

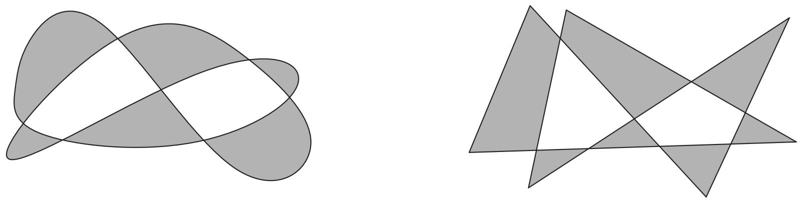

Figure 9. Given two simple closed curves in the plane it is straightforward to design an algorithm, which checks if the two curves are equivalent. But it is not straightforward to describe an ETR formula for it.

7 Algorithmic membership in ∃ℝ

The complexity class ∃ℝ is often called a "real analogue" of NP, because it deals with real-valued variables instead of Boolean variables. This analogy is correct, when we compare SAT with ETR. However, the most common way to think about NP is in terms of certificates and verification algorithms. The seminal theorem of Cook and Levin shows the equivalence of the two perspectives on NP [30, 74]. We show a similar equivalence between ETR-formulas and *real verification algorithms*. Intuitively, a real verification algorithm is a nondeterministic algorithm that runs on the *real RAM* that accepts as input both an *integer* instance I and a certificate, which may contain both integer and real components, and verifies that the certificate describes a valid solution to the instance in polynomial time. See Appendix G for a precise definition.

► **Theorem 10.** *For any discrete decision problem Q, there is a real verification algorithm for Q if and only if $Q \in \exists\mathbb{R}$.*

Our proof closely follows classical simulation arguments reducing nondeterministic polynomial-time (integer) random access machines to polynomial-size circuits or Boolean formulas, either directly [95] or via nondeterministic polynomial-time Turing machines [28–30, 40].

∃ℝ is known to be equivalent to the discrete portion of the Blum-Shub-Smale complexity class $NP^0_{\mathbb{R}}$—real sequences that can be accepted in polynomial time by a non-deterministic BSS machine *without constants*, and the equivalence of BSS machines without constants and ETR formulas is already well-known [16, 17]. However, the BSS-machine does not directly support the *integer* computations necessary for common standard programming paradigms such as indirect memory access and multidimensional arrays. The real RAM model originally proposed by Shamos [92, 106] *does* support indirect memory access through integer addresses; however, Shamos did not offer a precise definition of his model, and we are not aware of any published definition precise enough to support a simulation result like Theorem 10. We rectify this gap in Appendix A.1 by offering a precise definition of the real RAM that generalizes both the word RAM and BSS models, and which we believe formalizes the intuitive model implicitly assumed by computational geometers. Our real verification algorithms are then defined in terms of this precise model.

Theorem 10 not only strengthens the intuitive analogy between NP and ∃ℝ, but also enables much simpler proofs of ∃ℝ-membership in terms of standard geometric algorithms. Our motivation for developing Theorem 10 was Erickson's *optimal curve straightening* problem [42]: Given a closed curve γ in the plane and an integer k, is any k-vertex polygon topologically equivalent to γ? (See Figure 9.) The ∃ℝ-hardness of this problem follows from an easy reduction from stretchability of pseudolines, but reducing it directly to ETR proved much more complex; in light of Theorem 10, membership in ∃ℝ follows almost immediately from the standard Bentley-Ottman sweep-line algorithm [12]. To further illustrate the power of our technique, we also consider a new topological problem in Appendix G, which we call *optimal unknotted extension*: Given a simple polygonal *path* P in $\mathbb{R}^3$ and an integer k, can we extend P to an *unknotted* closed polygon with at most k additional vertices? In light of Theorem 10, the proof that this problem is in $\exists\mathbb{R}$ is straightforward: To verify a positive instance, guess the k new vertices and verify that the resulting knot is trivial using existing NP algorithms [58, 73].

► **Corollary 11.** *The following discrete decision problems are in $\exists\mathbb{R}$.*
- *The art gallery problem [2].*
- *The optimal curve straightening problem [42].*
- *The optimal unknotted extension problem.*

8 Conclusion

We presented an analysis of the minimal bit-precision required by polynomial time, real RAM algorithms under parametrized perturbations of their input. This parametrization can be considered a model for practical input data [110]. Our analysis generalizes prior theoretical results on robustness of algorithms under uniform random input [27, 35, 45] and provides a theoretical justification for the performance of exact computation CGAL implementations that use constant complexity geometric primitives [18, 34, 55, 56]. As our bounds are widely applicable they arguably also justify the usage of the real RAM itself as a model of computation. We used the analysis of the real RAM to show that the exponential bit-precision phenomenon for solutions of some $\exists\mathbb{R}$-complete problems comes from near-degenerate input, thereby narrowing the gap between NP and $\exists\mathbb{R}$. We wish to conclude with three open questions:

1. Are there published real RAM algorithms that even for perturbed input require exponential bit-precision, and do these algorithms possibly abuse the power of the real RAM?
2. Do all $\exists\mathbb{R}$-complete problems, under slight perturbation of their input, have a solution which can be described with expected polynomial bit-precision? Or are there indeed $\exists\mathbb{R}$-complete problems that require high bit-precision even under a reasonable model of perturbation?
3. Can we develop practical algorithms with performance guarantees for $\exists\mathbb{R}$-complete problems?

References

1 Zachary Abel, Erik Demaine, Martin Demaine, Sarah Eisenstat, Jayson Lynch, and Tao Schardl. Who needs crossings? Hardness of plane graph rigidity. *32nd International Symposium on Computational Geometry (SoCG 2016)*, 3:1–3:15, 2016.

2 Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. The art gallery problem is ∃ℝ-complete. *STOC*, 65–73, 2018.

3 Mikkel Abrahamsen, Tillmann Miltzow, and Nadja Seiferth. A framework for ∃ℝ-completeness of two-dimensional packing problems. *in Preparation* , 2020.

4 Umut Acar, Guy Blelloch, Kanat Tangwongsan, and Duru Türkoğlu. Robust kinetic convex hulls in 3d. *European Symposium on Algorithms*, 29–40, 2008.

5 David Arthur, Bodo Manthey, and Heiko Röglin. k-means has polynomial smoothed complexity. *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, 405–414, 2009.

6 David Arthur and Sergei Vassilvitskii. Worst-case and smoothed analysis of the icp algorithm, with an application to the k-means method. *FOCS 2016*, 153–164, 2006.

7 Deepak Bandyopadhyay and Jack Snoeyink. Almost-delaunay simplices: Nearest neighbor relations for imprecise points. *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, 410–419, 2004.

8 Deepak Bandyopadhyay and Jack Snoeyink. Almost-delaunay simplices: Robust neighbor relations for imprecise 3d points using cgal. *Computational Geometry* 38(1-2):4–15. Elsevier, 2007.

9 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry. ISBN-10 3-540-33098-4*. Springer, Berlin Heidelberg, 2006.

10 René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. *STOC*, 232–241, 2003.

11 René Beier and Berthold Vöcking. Typical properties of winners and losers in discrete optimization. *SIAM Journal on Computing* 35(4):855–881. SIAM, 2006.

12 Jon Bentley and Thomas Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers* C-28(9):643–647, 1979.

13 Alberto Bertoni, Giancarlo Mauri, and Nicoletta Sabadini. Simulations among classes of random access machines and equivalence among numbers succinctly represented. *Annals of Discrete Mathematics* 25:65–90, 1985.

14 Daniel Bienstock. Some provably hard crossing number problems. *Discrete & Computational Geometry* 6(3):443–459. Springer, 1991.

15 Markus Bläser, Bodo Manthey, and Raghavendra Rao. Smoothed analysis of partitioning algorithms for euclidean functionals. *Algorithmica* 66(2):397–418. Springer, 2013.

16 Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer, 1998.

17 Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: np-completeness, recursive functions and universal machines. *BulletinAmer. Math. Soc.* 21(1):1–46, 1989.

18 Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. Triangulations in cgal. *Computational Geometry* . Elsevier, 2002.

19 Marthe Bonamy, Edouard Bonnet, Nicolas Bousquet, Pierre Charbit, and Stéphan Thomassé. EPTAS for max clique on disks and unit balls. *FOCS*, 568–579, 2018. IEEE Computer Society.

20 Hervé Brönnimann, Christoph Burnikel, and Sylvain Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics* 109(1-2):25–47. Elsevier, 2001.

21 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica* 61(3):674–693. Springer, 2011.

22 Kevin Buchin and Wolfgang Mulzer. Delaunay triangulations in $o(\text{sort}(n))$ time and more. *Journal of the ACM (JACM)* 58(2):6. ACM, 2011.

23 Jean Cardinal. Computational geometry column 62. *SIGACT News* 46(4):69–78. ACM, 2015.

24 Jean Cardinal, Timothy Chan, John Iacono, Stefan Langerman, and Aurélien Ooms. Subquadratic encodings for point configurations. *34th International Symposium on Computational Geometry, SoCG*, 2018.

25 Jean Cardinal, Ruy Fabila-Monroy, and Carlos Hidalgo-Toscano. Chirotopes of random points in space are realizable on a small integer grid. *Canadian Conference on Computational Geometry*, 2019.

26 Jean Cardinal, Stefan Felsner, Tillmann Miltzow, Casey Tompkins, and Birgit Vogtenhuber. Intersection graphs of rays and grounded segments. *International Workshop on Graph-Theoretic Concepts in Computer Science*, 153–166, 2017.

27 Jean Cardinal and Udo Hoffmann. Recognition and complexity of point visibility graphs. *Discrete & Computational Geometry* 57(1):164–178. Springer, 2017.

28 Stephen Cook. Short propositional formulas represent nondeterministic computations. *Inform. Proc. Lett.* 26:269–270, 1987/88.

29 Stephen Cook and Robert Reckhow. Time bounded random access machines. *J. Comput. Syst. Sci.* 7(4):354–375, 1973.

30 Stephen A Cook. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing*, 151–158, 1971.

31 David Cox, John Little, and Donal O'Shea. *Using algebraic geometry*. Springer Science & Business Media, 2006.

32 David Cox, John Little, and Donal O'Shea. *Ideals, Varieties and Algorithms*. Springer, 2007.

33 Daniel Dadush and Sophie Huiberts. A friendly smoothed analysis of the simplex method. *STOC*, 390–403, 2018.

34 Olivier Devillers. Delaunay triangulation of imprecise points, preprocess and actually get a fast query time. *Journal of Computational Geometry* , 2011.

35 Olivier Devillers, Philippe Duchon, Marc Glisse, and Xavier Goaoc. On order types of random point sets. *arXiv:1812.08525* , 2018.

36 Edsger Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1):269–271, 1959.

37 Michael G. Dobbins, Linda Kleist, Tillmann Miltzow, and Paweł Rzążewski. $\forall\exists\mathbb{R}$-completeness and area-universality. *ArXiv 1712.05142* , 2017.

38 Michael Gene Dobbins, Andreas Holmsen, and Tillmann Miltzow. Smoothed analysis of the art gallery problem. *CoRR* abs/1811.01177, 2018.

39 Michael Gene Dobbins, Andreas Holmsen, and Tillmann Miltzow. A universality theorem for nested polytopes. *arXiv* 1908.02213, 2019.

40 Peter van Emde Boas. Machine models and similations. *Algorithms and Complexity*, chapter 1, 1, 3–66, 1990. Handbook of Theoretical Computer Science, Elsevier Science.

41 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the tsp. *SODA*, 1295–1304, 2007.

42 Jeff Erickson. Optimal curve straightening is $\exists\mathbb{R}$-complete. *arXiv:1908.09400* , 2019.

43 Michael Etscheid and Heiko Röglin. Smoothed analysis of local search for the maximum-cut problem. *ACM Transactions on Algorithms (TALG)* 13(2):25. ACM, 2017.

44 Esther Ezra and Wolfgang Mulzer. Convex hull of points lying on lines in $o(n \log n)$ time after preprocessing. *Computational Geometry* 46(4):417–434. Elsevier, 2013.

45 Ruy Fabila-Monroy and Clemens Huemer. Order types of random point sets can be realized with small integer coordinates. *XVII Spanish Meeting on Computational Geometry: book of abstracts, Alicante, June 26-28*, 73–76, 2017.

46 Steve Fisk. A short proof of Chvátal's watchman theorem. *J. Comb. Theory, Ser. B* 24(3):374, 1978.

47 Steven Fortune. Stable maintenance of point set triangulations in two dimensions. *30th Annual Symposium on Foundations of Computer Science*, 494–499, 1989.

48 Steven Fortune and Christopher Van Wyk. Efficient exact arithmetic for computational geometry. *Proceedings of the ninth annual symposium on Computational geometry*, 163–172, 1993.

49 Michael Fredman and Dan Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences* 48(3):424–436, 1993.

50 Michael Fredman and Dan Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences* 48(3):533–551, 1994.

51 Jugal Garg, Ruta Mehta, Vijay V. Vazirani, and Sadra Yazdanbod. ETR-completeness for decision versions of multi-player (symmetric) Nash equilibria. *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015), part 1*, 554–566, 2015. Lecture Notes in Computer Science (LNCS).

52 Thierry Gensane and Philippe Ryckelynck. Improved dense packings of congruent squares in a square. *DCG* 34(1):97–109, 2005.

53 Jacob Goodman, Richard Pollack, and Bernd Sturmfels. Coordinate representation of order types requires exponential storage. *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, 405–410, 1989. STOC '89, ACM.

54 Torben Hagerup. Sorting and searching on the word ram. *Proc. 15th Ann. Symp. Theoretical Aspects Comput. Sci.*, 366–398, 1998. Lecture Notes Comput. Sci. 1373, Springer Berlin Heidelberg.

55 Dan Halperin. Robust geometric computing in motion. *The international Journal of Robotics Research* 21(3):219–232. SAGE Publications Sage UK: London, England, 2002.

56 Idit Haran and Dan Halperin. An experimental study of point location in planar arrangements in cgal. *Journal of Experimental Algorithmics (JEA)* 13:3. ACM, 2009.

57 Juris Hartmanis and Janos Simon. On the power of multiplciation in random-access machines. *Proc. 15th Annu. IEEE Sympos. Switching Automata Theory*, 13–23, 1974.

58 Joel Hass, Jeffrey C. Lagarias, and Nicholas Pippenger. The computational complexity of knot and link problems. *J. ACM* 46(2):185–211, 1999.

59 Paul Heffernan and Joseph Mitchell. An optimal algorithm for computing visibility in the plane. *SIAM Journal on Computing* 24(1):184–201. SIAM, 1995.

60 Gyula Hermann. Robust convex hull-based algorithm for straightness and flatness determination in coordinate measuring. *Acta Polytechnica Hungarica* 4(4):111–120, 2007.

61 John Hershberger and Subhash Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing* 28(6):2215–2256. SIAM, 1999.

62 Ivor van der Hoog, Irina Kostitsyna, Maarten Löffler, and Bettina Speckmann. Preprocessing ambiguous imprecise points. *35th International Symposium on Computational Geometry, SoCG*, 2019.

63 Ivor van der Hoog, Tillmann Miltzow, and Martijn van Schaik. Smoothed analysis of order types. *arXiv:1907.04645* , 2019.

64 Leo Joskowicz and Yonatan Myers. Topological stability and convex hull with dependent uncertainties. *30th European Workshop on Computational Geometry*, 2014.

65 Frank Kammer, Maarten Löffler, Paul Mutser, and Frank Staals. Practical approaches to partially guarding a polyhedral terrain. *International Conference on Geographic Information Science*, 318–332, 2014.

66 Ross Kang and Tobias Müller. Sphere and dot product representations of graphs. *27th Annual Symposium on Computational Geometry (SoCG)*, 308–314, 2011.

67 Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee-Keng Yap. Classroom examples of robustness problems in geometric computations. *Computational Geometry* 40(1):61–78. Elsevier, 2008.

68 A Khanban and Abbas Edalat. Computing delaunay triangulation with imprecise input data. *Canadian Conference on Computational Geometry (CCCG)*, 2003.

69 David Kirkpatrick and Stefan Reisch. Upper bounds for sorting integers on random access machines. *Theor. Comput. Sci.* 28(3):263–276, 1984.

70 Victor Klee and George Minty. How good is the simplex algorithm. Tech. rep., Washington Univ. Seattle Dept. of Mathematics, 1970.

71 Linda Kleist. *Planar graphs and faces areas – Area-Universality*. Ph.D. thesis, Technische Universität Berlin, 2018. PhD thesis.

72 Irina Kostitsyna, Maarten Löffler, Valentin Polishchuk, and Frank Staals. On the complexity of minimum-link path problems. *Journal of Computational Geometry* 8(2):80–108, 2017.

73 Marc Lackenby. A polynomial upper bound on Reidemeister moves. *Ann. Math.* 182(2):491–564, 2015. arXiv:1302.0180.

74 Leonid Anatolevich Levin. Universal sequential search problems. *Problemy peredachi informatsii* 9(3):115–116. Russian Academy of Sciences, Branch of Informatics, 1973.

75 Chen Li, Sylvain Pion, and Chee-Keng Yap. Recent progress in exact geometric computation. *The Journal of Logic and Algebraic Programming* 64(1):85–111. Elsevier, 2005.

76 Maarten Löffler and Marc van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica* 56(2):235. Springer, 2010.

77 Maarten Löffler and Jack Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Computational Geometry* 43(3):234–242. Elsevier, 2010.

78 Anna Lubiw, Tillmann Miltzow, and Debajyoti Mondal. The complexity of drawing a graph in a polygonal region. *International Symposium on Graph Drawing and Network Visualization*, 2018.

79 Harry Mairson and Jorge Stolfi. Reporting and counting intersections between two sets of line segments. *Theoretical Foundations of Computer Graphics and CAD*, 307–325, 1988. Springer.

80 Bodo Manthey and Heiko Röglin. Improved smoothed analysis of the k-means method. *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, 461–470, 2009.

81 Bodo Manthey and Rianne Veenstra. Smoothed analysis of the 2-opt heuristic for the tsp: Polynomial bounds for gaussian noise. *International Symposium on Algorithms and Computation*, 579–589, 2013.

82 Jiří Matoušek. Intersection graphs of segments and $\exists\mathbb{R}$. , 2014. ArXiv 1406.2636.

83 Colin McDiarmid and Tobias Müller. Integer realizations of disk and segment graphs. *Journal of Combinatorial Theory, Series B* 103(1):114–143. Elsevier, 2013.

84 Wouter Meulemans, Bettina Speckmann, Kevin Verbeek, and Jules Wulms. A framework for algorithm stability and its application to kinetic euclidean msts. *Latin American Symposium on Theoretical Informatics*, 805–819, 2018.

85 Nicolai Mnëv. The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. *Topology and geometry – Rohlin seminar*, 527–543, 1988. Springer-Verlag Berlin Heidelberg.

86 Ernst Mücke. A robust implementation for three-dimensional delaunay triangulations. *International Journal of Computational Geometry & Applications* 8(02):255–276. World Scientific, 1998.

87 Takayuki Nagai, Seigo Yasutome, and Nobuki Tokura. Convex hull problem with imprecise input. *Japanese Conference on Discrete and Computational Geometry*, 207–219, 1998.

88 George Nemhauser and Zev Ullmann. Discrete dynamic programming and capital allocation. *Management Science* 15(9):494–505. INFORMS, 1969.

89 Joseph O'Rourke. Problem 33: Sum of square roots. `http://cs.smith.edu/~jorourke/TOPP/P33.html#o-6369-81`.

90 Joseph O'Rourke. Advanced problem 6369. *Amer. Math. Monthly* 88(10):769, 1981.

91 Sylvain Pion. Efficient exact geometric predicates for delaunay triangulations. *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments*, vol. 111, p. 37, 2003.

92 Franco Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Texts and Monographs in Computer Science. Springer, 1985.

93 Daniel Reem. The geometric stability of voronoi diagrams with respect to small changes of the sites. *Proceedings of the twenty-seventh annual symposium on Computational geometry*, 254–263, 2011.

94 Jürgen Richter-Gebert and Günter M. Ziegler. Realization spaces of 4-polytopes are universal. *Bulletin of the American Mathematical Society* 32(4):403–412, 1995.

95 John Robson. An $O(T \log T)$ reduction from RAM computations to satisfiability. *Theor. Comput. Sci.* 82(1):141–149, 1991.

96 Heiko Röglin. Minicourse on smoothed analysis. `https://algo.rwth-aachen.de/Lehre/SS07/VRA/Material/SmoothedAnalysis.pdf`, 2007. Online; accessed April 2019.

97 Tim Roughgarden. Smoothed Complexity and Pseudopolynomial-Time Algorithms. `https://theory.stanford.edu/~tim/f14/l/l15.pdf`, 2014. Online; accessed August 2018.

98 Tim Roughgarden. Beyond worst-case analysis. *Arxiv* 1806.09817, 2018.

99 David Salesin, Jorge Stolfi, and Leonidas Guibas. Epsilon geometry: building robust algorithms from imprecise computations. *Proceedings of the fifth annual symposium on Computational geometry*, 208–217, 1989.

100 Marcus Schaefer. Complexity of some geometric and topological problems. *Proceedings of the 17th International Symposium on Graph Drawing (GD 2009)*, 334–344, 2009. Lecture Notes in Computer Science (LNCS).

101 Marcus Schaefer. Realizability of graphs and linkages. *Thirty Essays on Geometric Graph Theory*, chapter 23, 461–482, 2013. Springer-Verlag New York.

102 Marcus Schaefer and Daniel Štefankovič. Fixed points, Nash equilibria, and the existential theory of the reals. *Theory of Computing Systems* 60(2):172–193, 2017.

103 Arnold Schönhage. On the power of random access machines. *Proc. 6th Internat. Colloq. Automata Lang. Program.*, 520–529, 1979. Lecture Notes in Computer Science., Springer-Verlag.

104 Mark Segal. Using tolerances to guarantee valid polyhedral modeling results. *ACM SIGGRAPH Computer Graphics* 24(4):105–114. ACM, 1990.

105 Mark Segal and Carlo Séquin. Consistent calculations for solids modeling. *Proceedings of the first annual symposium on Computational geometry*, 29–38, 1985.

106 Michael Ian Shamos. *Computational Geometry*. Ph.D. thesis, Yale Unviersity, 1979. ⟨`http://euro.ecom.cmu.edu/people/faculty/mshamos/1978ShamosThesis.pdf`⟩.

107 Yaroslav Shitov. A universality theorem for nonnegative matrix factorizations. , 2016, journal=ArXiv 1606.09068.

108 Yaroslav Shitov. The complexity of positive semidefinite matrix factorization. *SIAM Journal on Optimization* 27(3):1898–1909, 2017.

109 Peter Shor. Stretchability of pseudolines is NP-hard. *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, 531–554, 1991. DIMACS – Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society and Association for Computing Machinery.

110 Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)* 51(3):385–463. ACM, 2004.

111 Leonida Tonelli. Sull'integrazione per parti. *Atti della Accademia Nazionale dei Lincei* 5(18):246–253, 1909.

112 Csaba Toth, Joseph O'Rourke, and Jacob Goodman. *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017.

113 Marc Van Kreveld, Maarten Löffler, and Joseph Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM Journal on Computing* 39(7):2990–3000. SIAM, 2010.

114 Chee-Keng Yap. Towards exact geometric computation. *Computational Geometry* 7(1-2):3–23. Elsevier, 1997.

115 Chee-Keng Yap. In praise of numerical computation. *Efficient Algorithms*, 380–407, 2009. Springer.

116 Chee-Keng Yap, Vikram Sharma, and Jyh-Ming Lien. Towards exact numerical voronoi diagrams. *2012 Ninth International Symposium on Voronoi Diagrams in Science and Engineering*, 2–16, 2012.

A Preliminaries

A.1 What is the Real RAM?

The real RAM has been the standard underlying model of computation in computational geometry since the field was founded in the late 1970s [92, 106]. Despite its ubiquity, we are unfortunately unaware of *any* published definition of the model that is simultaneously precise enough to support our results and broad enough to encompass most algorithms in the computational geometry literature. The obvious candidate for such a definition is the real-computation model proposed by Blum, Shub, and Smale [16, 17]; however, this model does not support the integer operations necessary to implement even simple geometric algorithms.

Even though the real RAM is often presented, either formally or intuitively, as a random access machine that stores and manipulates only exact real numbers, countless algorithms in this model require decisions based on both exact real and finite precision integer values. Consider the following example: Given an array of n real values as input, compute their sum. Any algorithm that computes this sum must store and manipulate real numbers; however, the most straightforward algorithm also requires indirect memory access through an *integer* array index. More complex examples include call stack maintenance, discrete symbol manipulation, multidimensional array indexing and slicing, and dynamic management of pointer-based data structures.

On the other hand, real and integer operations must be combined with care to avoid unreasonable *discrete* computation power. A model that supports both exact constant-time real arithmetic and constant-time conversion between real numbers and integers, for example using the floor function, would also trivially support arbitrary-precision constant-time *integer* arithmetic. (To multiply two integers, cast them both to reals, multiply them, and cast the result back to an integer.) Including such constant-time operations allows any problem in PSPACE to be solved in polynomial time [103]; see also [13, 40, 57, 69] for similar results.

To accommodate this mixture of real and integer operations, and to avoid complexity pitfalls, we define the real RAM as an extension of the standard integer word RAM [54]. We define the real RAM in terms of a fixed parameter w, called the *word size*. A *word* is an integer between 0 and $2^w - 1$, represented as a sequence of w bits. Mirroring standard definitions for the word RAM, memory consists of two *random access arrays* $W[0 .. 2^w - 1]$ and $R[0 .. 2^w - 1]$, whose elements we call *registers*. Both of these arrays are indexed/addressed by words; for any word i, register $W[i]$ is a word and register $R[i]$ is an exact real number. (We sometimes refer to a word as an

Class	Word	Real
Constants	$W[i] \leftarrow j$	$R[i] \leftarrow 0$
		$R[i] \leftarrow 1$
Memory	$W[i] \leftarrow W[j]$	$R[i] \leftarrow R[j]$
	$W[W[i]] \leftarrow W[j]$	$R[W[i]] \leftarrow R[j]$
	$W[i] \leftarrow W[W[j]]$	$R[i] \leftarrow R[W[j]]$
Casting	—	$R[i] \leftarrow j$
	—	$R[i] \leftarrow W[j]$
Arithmetic and boolean	$W[i] \leftarrow W[j] \boxplus W[k]$	$R[i] \leftarrow R[j] \oplus R[k]$
Comparisons	if $W[i] = W[j]$ goto ℓ	if $R[i] = 0$ goto ℓ
	if $W[i] < W[j]$ goto ℓ	if $R[j] > 0$ goto ℓ
Control flow	goto ℓ	
	halt / accept / reject	

Table 1. Constant time RAM operations. The values i, j, k are constant words used for indexing.

address when it is used as an index into one of the memory arrays.)

A program on the real RAM consists of a fixed, finite indexed sequence of read-only instructions. The machine maintains an integer *program counter*, which is initially equal to 1. At each time step, the machine executes the instruction indicated by the program counter. The goto instruction modifies the program counter directly; the halt and accept and reject instructions halt execution (and possibly return a boolean result); otherwise, the program counter increases by 1 after each instruction is executed.

The input to a real RAM program consists of a pair of vectors $(a, b) \in \mathbb{R}^n \times \mathbb{Z}^m$, for some integers n and m, which are suitably encoded into the corresponding memory arrays before the program begins.[1] To maintain uniformity, we require that neither the input sizes n and m nor the word size w is known to any program at "compile time". The output of a real RAM program consists of the contents of memory when the program executes the halt instruction.

Following Fredman and Willard [49, 50] and later users of the word RAM, we assume that $w = \Omega(\log N)$, where $N = n + m$ is the total size of the problem instance at hand. This so-called *transdichotomous* assumption implies direct constant-time access to the input data. Table 1 summarizes the specific instructions our model supports. All *word* operations operate on words and produce words as output; all *real* operations operate on real numbers and produce real numbers as output. Each operation is parametrized by a small number of constant words i, j, and k. The *running time* of a real RAM program is the number of instructions executed before the program halts; each instruction requires one time step by definition.

Our model supports the following specific word operations; all arithmetic operations interpret words as non-negative integers between 0 and $2^w - 1$.

- addition: $x \leftarrow (y + z) \bmod 2^w$
- subtraction: $x \leftarrow (y - z) \bmod 2^w$
- lower multiplication: $x \leftarrow (yz) \bmod 2^w$
- upper multiplication: $x \leftarrow \lfloor yz/2^w \rfloor$
- rounded division: $x \leftarrow \lfloor y/z \rfloor$, where $z \neq 0$
- remainder: $x \leftarrow y \bmod z$, where $z \neq 0$
- bitwise nand: $x \leftarrow y \uparrow z$ (that is, $x_i \leftarrow y_i \uparrow z_i$ for every bit-index i)

(Other bitwise boolean operations can of course be implemented by composing bitwise nands.) Similarly, our model supports the following *exact* real operations.

- addition: $x \leftarrow y + z$
- subtraction: $x \leftarrow x - y$
- multiplication: $x \leftarrow y \cdot z$
- exact division: $x \leftarrow y/z$, where $z \neq 0$
- (optional) exact square root: $x \leftarrow +\sqrt{y}$, where $y \geq 0$

To avoid unreasonable computational power, our model does not allow casting real variables to integers (for example, using the floor function $\lfloor\cdot\rfloor$), or testing whether a real register actually stores an integer value, or any other access to the binary representation of a real number. However, we *do* allow casting integer variables to reals.

[1] Following standard practice, we implicitly assume throughout the paper that the integers in the input vector b are actually w-bit *words*; for problems involving larger integers, we take m to be the number of *words* required to encode the integer part of the input.

A.2 Definition of Smoothed Analysis

In this paragraph, we will formally define the smoothed complexity of an algorithm. For this paragraph, we assume that we have some algorithm fixed. Let us fix some $\delta \in [0,1]$, which describes the *magnitude of perturbation*. The variable δ describes by how much we allow to perturb the original input. In this paper, we consider an array $I = (a, b) \in \mathbb{R}^n \times \mathbb{Z}^m$ of n real numbers and m integers as the input. We assume that each real number is perturbed independently and that the integers stay as they are. We denote by $(\Omega_\delta, \mu_\delta)$ the probability space where each $x \in \Omega_\delta$ defines for each instance I a new 'perturbed' instance $I_x = (a + x, b)$. We denote by $\mathscr{C}(I_x)$ the cost of instance I_x. The smoothed expected cost of instance I equals:

$$\mathscr{C}_\delta(I) = \mathop{\mathbb{E}}_{x\in\Omega_\delta} \mathscr{C}(I_x) = \int_{\Omega_\delta} \mathscr{C}(I_x)\mu_\delta(x)\, \mathrm{d}x.$$

If we denote by Γ_n the set of all instances of size n, then the smoothed complexity equals:

$$\mathscr{C}_{\text{smooth}}(n,\delta) = \max_{I\in\Gamma_n} \mathop{\mathbb{E}}_{x\in\Omega_\delta} [\mathscr{C}(I_x)].$$

This formalizes the intuition mentioned before: not only do the majority of instances behave nicely, but actually in every neighborhood (bounded by the maximal perturbation δ) the majority of instances behave nicely. The smoothed complexity is measured in terms of n and δ. If the expected complexity is small in terms of $1/\delta$ then we have a theoretical verification of the hypothesis that worst case examples are well-spread.

A.3 Notation

We use $[a]$ to indicate the set $\{1,\ldots,a\}$. Let $z = (z_1,\ldots,z_d) \in \mathbb{Z}^d$ be a vector with integer coordinates. We denote by C_z the unit (hyper)cube, which has z as its minimal corner or formally: $C_z := \{x \in \mathbb{R}^d \mid z_i \le x_i < z_i + 1\}$. We denote by $C(d,k)$ the collection of all d-dimensional unit cubes contained in a d-dimensional cube of width k: $C(d,k) := \{C_z \mid z \in \mathbb{Z}^d \cap [0,k)^d\}$. We generalize this notation of cubes to cylinders. Given $z \in \mathbb{Z}^d$ and $s \in \mathbb{N}$, we define the *unit cylinder* $C^s_z \subset \mathbb{R}^{d+s}$ as $C^s_z = \{x \in \mathbb{R}^{d+s} \mid z_i \le x_i < z_i + 1, \forall i = 1,\ldots,d\}$ and subsequently also define $C^s(d,k) := \{C^s_z \mid z \in \mathbb{Z}^d \cap [0,k)^d\}$ as a grid of cylinders. We suppress the s in the notation, when it is clear from context. Note that for $s = 0$, cylinders and cubes are the same. We denote by $C^s(d,k)$ the collection of all $d+s$-dimensional unit cylinders: $C^s(d,k) := \{\, C_z \subset \mathbb{R}^{d+s} : z \in \mathbb{Z}^d \cap [0,k)^d \,\}$.

Let C be a cube partitioned by unit cubes $C(d,k)$. Every facet of C intuitively is a grid of (d-1)-dimensional cubes $C(d,k-1)$. In the remainder of the paper, we want to argue about varieties that intersect cubes in $C(d,k)$ and we do this via induction of the facets. Therefore, we formalize the above intuition:

Let C be a $(d+s)$-dimensional cylinder, partitioned by $(d+s)$-dimensional cylinders of equal width. We say C is *equivalent* to $C(d,k)$, denoted by $C \cong C(d,k)$, if there exists an affine transformation τ of C such that there is a one-to-one correspondence between cylinders in $\tau(C)$ and $C(d,k)$ where corresponding cylinders coincide. We give to examples of this equivalence relation that is often used in the remainder of the paper: (1) Consider any d,k and any $(d-1)$-dimensional, orthogonal hyperplane H that intersects a d-dimensional cube $C^0(d,k)$ we have that $C^0(d,k)\cap H \cong C^0(d-1,k)$. (2) Consider the d-dimensional grid Γ_ω as defined in the introduction and a cube C which has one corner on the origin and width $k\omega$. The intersection $C \cap \Gamma_\omega$ is equivalent to $C^0(d,k)$.

Following [31], we define a d-variate polynomial p in $x_1,\ldots,x_d$ with coefficients in $\mathbb{R}$ as a finite linear combination of monomials with coefficients in $\mathbb{R}$ and we will denote this by

$p \in \mathbb{R}[x_1, \ldots, x_d]$. We denote by $V(p) := \{x \in \mathbb{R}^d : p(x) = 0\}$ the *variety* of p. For any subset $S \subset R^d$, we say that p intersects S if $S \cap V(p) \neq \emptyset$. Given a set of polynomials $p_1, \ldots, p_k$, we denote their variety as $V(p_1, \ldots, p_k) = \bigcap_{i=1,\ldots,k} V(p_i)$. Let f be an expression, which defines a function, then we also use the notation $V(f) = \{x : f(x) = 0\}$, although it is not a variety. We say f *intersects* a set S, if $V(f) \cap S \neq \emptyset$. We will also need the notion of the *dimension* of a variety. We assume that most readers have some intuitive understanding, which is sufficient to follow the arguments. It is out of scope to define this formally in this paper, so we refer to the book by Basu, Pollack and Roy [9, Chapter 5]. Specifically, Lemma 22 and Lemma 24 have to be taken for granted. Given a polynomial $p \in \mathbb{R}[x_1, \ldots, x_d]$, then the linear polynomial $\ell \in \mathbb{R}[x_1, \ldots, x_d]$ is a factor of p, if there exists some $q \in \mathbb{R}[x_1, \ldots, x_d]$ such that $\ell \cdot q = p$.

B Smoothed Analysis of the real RAM

The first part of this section is devoted to prove Theorem 1, which states that a real RAM algorithm A with algebraic-degree Δ and algebraic-dimension d runs correctly (under smoothed analysis with perturbation magnitude δ) when it uses a bit-precision logarithmic in n/δ. Specifically, we prove the following lemma which implies Theorem 1. In Section C we generalize this lemma so that the operations in A may include square roots.

► **Lemma 12.** *Consider the real RAM without square roots and a polynomial time algorithm A with algebraic-degree Δ and constant algebraic-dimension. Then under perturbations of the input of magnitude δ, A can be robustly executed if the input has an expected bit-precision of:*

$$O\left(d \log \frac{d\Delta n}{\delta}\right).$$

Proof strategy and notation. We consider A with input $I = (a, b)$ where a is real-valued input in $[0,1]^n$. We assume that the word RAM has a word size w which allows us to express $2^w = \frac{1}{\omega}$ different values for each coordinate. Since A runs in polynomial time it can make at most a polynomial number of binary decisions. As we explained in the Preliminaries Section A at every such binary decision, the algorithm looks at either a real-valued register or at an integer-value register and verifies if the value at the register is 0 or strictly greater than 0. For every real-valued register R_i, per assumption the value at that register is the result of two d-variate polynomials p_i and q_i with maximum degree Δ. Let $z = (a_{i_1}, \ldots, a_{i_d})$ be the input registers that R_i depends on. Then per assumption $R_i = p_i(z)/q_i(z)$ and the evaluation of R_i depends on the evaluation of the polynomials p_i and q_i. During smoothed analysis we permute our input a into new input $(a + x)$ and we compare the execution of A with the input $(a + x)$ to its execution with the input a' which is the representation of $(a + x)$ with limited bit-precision. We are interested in the probability that the execution of A under both inputs is the same, ergo the chance that for all comparison operations, the algorithms give the same answer. First we show that with high probability a single d-variate polynomial p evaluates to the same sign under $(a + x)$ and a'. Second we apply the union bound to bound the probability that the algorithm A gives the same output under $(a + x)$ and a' for all registers. Lastly we upper bound the expected bit-precision.

For the smoothed analysis, we *snap* our permuted real-valued input $(a + x)$ onto a point with limited precision (a point in $\omega\mathbb{Z}^d \cap [0,1]^d$). For any real value in $[0,1]^d$, the natural choice to snap to is the closest point in $\omega\mathbb{Z}^d \cap [0,1]^d$. Ergo, for all $y \in \omega\mathbb{Z}^d$, all points in the Voronoi cell of y are snapped to y. Note that in this case, the Voronoi cells of all these cells are just d-dimensional cubes and we shall denote them by $C(y)$. The set of Voronoi cells over all $y \in \omega\mathbb{Z}^d \cap [0,1]^d$ forms a d-dimensional cube with the following property:

$$\Gamma_\omega = \{C(y) : y \in \omega\mathbb{Z}^d \cap [0,1]^d\} \cong C(d, 1/\omega).$$

We permute the original input a with a random value $x \in \Omega = [-\frac{\delta}{2}, \frac{\delta}{2}]^d$ to obtain a new real-valued point $(a+x)$ that must lie within a cube in Γ_ω. However the choice of a and δ limits the cubes in Γ_ω where $(a+x)$ can lie in. Specifically the range of possible locations for $(a+x)$ is a d-dimensional cube of width δ centered around a. This cube contains a d-dimensional cube of cells which is equivalent to $C(d, \lfloor \delta/\omega \rfloor)$ which we shall denote by $\Gamma_\omega(a)$ and possibly intersects an additional perimeter of cells (refer to Figure 10). We can use this observation together with Theorem 20 to estimate the probability that $\text{sign}(p(x+a)) \neq \text{sign}(p(a'))$:

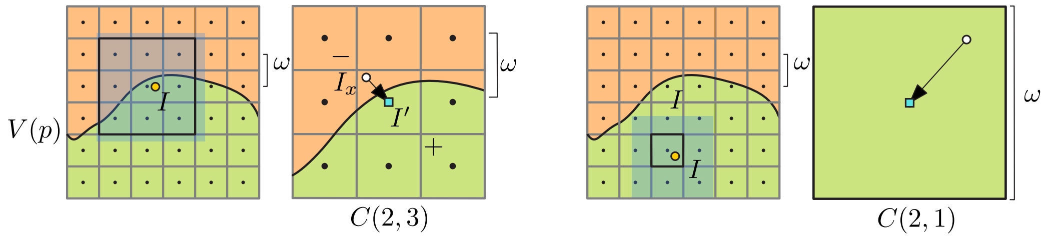

Figure 10. The variety $V(p)$ separates the region where p evaluates positively (green) and where it evaluates negatively (orange). We see two scenarios where the input I is shown as a single real-valued two-dimensional point. The permuted input I_x must lie within a cube of width δ centered around I. Depending on its size and placement, it contains a subgrid of a certain diameter.

► **Lemma 13** (Register-Snapping)**.** *Let R be some register, whose value can be described by two d-variate polynomials p, q maximum degree Δ. Let $a \in \mathbb{R}^d$ be fixed and $x \in \Omega = [-\delta/2, \delta/2]^d$ chosen uniformly at random. Assume $(a+x)$ is snapped to a point $a' \in \omega\mathbb{Z}^d \cap [0,1]^d$. Then* $\text{sign}(R(a+x)) \neq \text{sign}(R(a'))$ *with probability at most:*

$$\frac{4\,\omega\Delta 3^d(d+1)!}{\delta}.$$

Proof. Note that if both polynomials have the same sign after snapping, also R will have the same sign. Thus we are free to show that $\text{sign}(p(a+x)) \neq \text{sign}(p(a'))$ with probability at most:

$$\frac{2\,\omega\Delta 3^d(d+1)!}{\delta}.$$

The statement for q is equivalent and the union bound on these separate probabilities then upper bounds probability that the register R does not have the same sign. Whenever we have a continuous polynomial p and points $x, z \in \mathbb{R}^d$ with $p(x) < 0$ and $p(z) > 0$ then there must be a point $y \in \text{line}(x, z)$ for which $p(y) = 0$. It follows that if a cube $C \in \Gamma_\omega$ is not intersected by p, all points in C either have a positive or negative evaluation under p. If the point $(a+x)$ lies in a Voronoi cell $C(a') \in \Gamma_\omega$, then per construction it will be snapped to its center point a' and if $C(a')$ is not intersected by p then $sign(p(a+x)) = sign(p(a'))$.

Therefore we are interested in the probability that $(a+x)$ is contained in a cell that is intersected by p. As we discussed, the set of possible locations of $(a+x)$ is a cube which contains $\Gamma_\omega(a) \cong C(d, \lfloor \delta/\omega \rfloor)$ together with a perimeter of cells. We upper bound the probability that $(a+x)$ lies within a cell that is intersected by p probability in two steps: (1) We upper bound the number of cells on the perimeter and assume that they are all intersected by p. (2) we upper bound the number of cells of $\Gamma_\omega(a)$ that are intersected by p. These two numbers, divided by the total number of cells in $\Gamma_\omega(a)$ gives the upper bound we are looking for. Note that the width of $\Gamma_\omega(a)$ is equal to $k = \lfloor \frac{\delta}{\omega} \rfloor$ and that the perimeter of $\Gamma_\omega(a)$ contains $2d \cdot k^{d-1}$ cells. Theorem 20 gives an upper bound on the number of intersected cubes in $C(d, k)$ of:

$$k^{d-1}\Delta 3^d(d+1)! \leq \left\lfloor \frac{\delta}{\omega} \right\rfloor^{d-1} \Delta 3^d(d+1)!$$

There are $\lfloor \delta/\omega \rfloor^d$ cubes in $\Gamma_\omega(a)$ and it follows that:

$$\Pr\big(\text{sign}(p(a+x)) \neq \text{sign}(p(a'))\big) \leq \frac{(\lfloor \frac{\delta}{\omega} \rfloor)^{d-1}\Delta 3^d (d+1)! + 2d(\lfloor \frac{\delta}{\omega} \rfloor)^{d-1}}{\lfloor \delta/\omega \rfloor^d} \leq \frac{2\omega\Delta 3^d (d+1)!}{\delta}$$

Using this, in conjunction with the union bound finishes the proof. ◀

Lemma 13 upper bounds the probability that for input $(a+x)$ and a' a single real-valued comparison in A is different. We can use the union bound to upper bound the probability that for the whole algorithm, any real-valued comparison for $(a+x)$ and a' is different.

► **Lemma 14** (Snapping). *Let $I = (a,b)$ be the input of an algorithm A that makes $T(n)$ comparison operations and $a \in \mathbb{R}^n$ and $b \in \mathbb{Z}^n$. Let x be a permutation chosen uniformly at random in $[-\delta/2, \delta/2]$ and let $I_x = (a+x, b)$ be a permuted instance of the input. For all $\varepsilon \in [0,1]$, if I_x is snapped to a grid of width*

$$\omega \leq \frac{\varepsilon\delta}{3^d(d+1)!\Delta 4T(n)},$$

Then I_x and I' are equivalent input for the algorithm A with probability at least $1-\varepsilon$.

Proof. By E_i for $i \in [T(n)]$, we denote the event that in the i'th algorithm step the inputs I_x and I get a different outcome. Lemma 13 upper bounds the probability of E_i occuring by:

$$\Pr(E_i) \leq \frac{4\omega\Delta 3^d (d+1)!}{\delta}.$$

The probability that I_x and I' are not equivalent is equal to the probability that for at least one event E_i, the event occurs. In other words:

$$\Pr(I_x \text{ and } I' \text{ not equivalent}) = \Pr\left(\cup_{i=1}^{T(n)} E_i\right) \leq T(n) \cdot \frac{4\omega\Delta 3^d (d+1)}{\delta}.$$

In the antecedent, $Pr(I_x$ and I' not equivalent$) < \varepsilon$ is implied by $T(n) \cdot \frac{4\omega\Delta 3^d(d+1)}{\delta} < \varepsilon$ and this proves the lemma. ◀

Fianlly we use Lemma 14 to get an estimate of the expected bit-precision of the algorithm. For this estimation we need the following folklore lemma about swapping the order of integration [111].

► **Lemma 15.** *Given a function $f : \Omega \to \{1, \ldots, b\}$ and assume that $\Pr(f(x) > b) = 0$. Then it holds that*

$$\mathbb{E}[f] = \sum_{z=1}^{b} z \Pr(f(x) = z) \; = \; \sum_{z=1}^{b} \Pr(f(x) \geq z).$$

Using Lemma 15, the expected value of $\texttt{bit}(I_x)$ can be expressed as:

$$\mathbb{E}(\ \texttt{bit}(I_x)\ ) = \sum_{k=1}^{\infty} k \Pr(\ \texttt{bit}(I_x) = k\ ) \; = \sum_{k=1}^{\infty} \Pr(\texttt{bit}(I_x) \geq k).$$

We split the sum at a splitting point l:

$$\mathbb{E}(\texttt{bit}(I_x)) = \sum_{k=1}^{l} \Pr(\texttt{bit}(p_x) \geq k) \; + \sum_{k=l+1}^{\infty} \Pr(\texttt{bit}(I_x) \geq k).$$

Now we note that any probability is at most 1 therefore the left sum is at most l. Through applying Lemma 14 we note that:

$$\Pr(\texttt{bit}(I_x) \geq k) = \Pr(\text{GridWidth}(I_x) \geq 2^{-k}) \leq 2^{-k}\left(\frac{3^d(d+1)!\Delta 4T(n)}{\delta}\right),$$

which in turn implies:

$$\mathbb{E}(\texttt{bit}(I_x)) \leq \sum_{k=1}^{l} 1 \ + 4\left(\frac{3^d(d+1)!\Delta T(n)}{\delta}\right)\sum_{k=l+1}^{\infty} 2^{-k}.$$

Observe that $\sum_{k=l+1}^{\infty} 2^{-k} = 2^{-l}$. So if we choose $l = \lceil \log \frac{3^d(d+1)!\Delta T(n)}{\delta} \rceil + 2$ we get:

$$\begin{aligned}
\mathbb{E}(\texttt{bit}(I_x)) &\leq l + 4\left(\frac{3^d(d+1)!\Delta T(n)}{\delta}\right)2^{-l} \\
&\leq \left\lceil \log \frac{3^d(d+1)!\Delta T(n)}{\delta} \right\rceil + 4 + \left(\frac{3^d(d+1)!\Delta T(n)}{\delta}\right)\left(\frac{\delta}{3^d(d+1)!\Delta T(n)}\right) \\
&= \left\lceil \log \frac{3^d(d+1)!\Delta T(n)}{\delta} \right\rceil + 5 = O\left(d \log \frac{d\Delta T(n)}{\delta}\right)
\end{aligned}$$

This finishes the proof of Theorem 1.

B.1 Implications

We can apply Theorem 1 to classical geometric problems to show Corollary 2 by giving an upper bound on the algebraic-dimension and algebraic-degree of these polynomial-time algorithms.

For sorting real numbers $a_1, \ldots, a_n$, the comparison operations are of the form $a_i - a_j > 0$. This has degree $\Delta = 1$ and dimension $d = 2$.

Computing the convex hull of a point set in fixed dimension k can be done using only the order type of the pointset. The order type in dimension k corresponds to computing the determinant of a matrix with matrix-dimension k, with $k^2 - k$ real input entries. Therefore the algebraic-dimension $d = k^2 - k$ and the $\Delta \leq k$. As k is constant, so is d and Δ.

Computing a Delaunay-triangulation can be computed naively in the plane, by making an in-circle test for every triple of points, and check if a forth point is contained in the circle. The in-circle test has constant d and Δ. For dimension k, let us repeat some standard trick. We first lift the points to $\mathbb{R}^{k+1}$, then compute the convex hull C and project C back to the original space, to get the Delaunay-triangulation. Given a finite set of points $P \subset \mathbb{R}^k$, we can map them to $\varphi(P) = P' \subset \mathbb{R}^{k+1}$.

$$\varphi : (x_1, \ldots, x_k) \ \rightarrow \ (x_1, \ldots, x_k, \sum x_i^2)$$

It is well known that the orthogonal projection of the convex hull of P' onto $\mathbb{R}^k$ gives the Delaunay-triangulation. As φ and computing the convex hull has bounded algebraic-dimension and algebraic-degree, so has the Delaunay-triangulation.

Let us now consider to check if a set of guards is correctly guarding a given gallery (polygon). Again, this can be done purely in terms of order types and order types have bounded algebraic-dimension and algebraic-degree. Thus so has checking if the guards see the entire polygon.

C Square Roots

The aim of this section is to give a proof to Theorem 3. In Appendix B we worked under the assumption that at all times a real register $R[i]$ contains a value that can be expressed as the quotient of two d-variate polynomials f_i, g_i of maximal degree Δ. In this section we make the real RAM model slightly more powerful as we allow f_i and g_i to also include the square root function and we even allow recursive applications of the square root function. Formally, every real register holds an *expression* $f(x_1, \ldots, x_d)$, which contains multiplication, addition, subtraction, division and square roots. We say f is equivalent to the polynomials $p, q_1, \ldots, q_s \in \mathbb{R}[x_1, \ldots, x_d, y_1, \ldots, y_s]$ if and only if:

$$[q_1, \ldots, q_s = 0] \Rightarrow \operatorname{sign}(p) = \operatorname{sign}(f).$$

See Lemma 16, on how to find those equivalent polynomials. We call p the *evaluation polynomial* and $q_1, \ldots, q_s$ the *constraint polynomials*. We define the algebraic-degree of f as the maximum algebraic-degree of $p, q_1, \ldots, q_s$. We say the algebraic-dimension of f equals d, and s is denoted as the *extra algebraic-dimension*. We say an algorithm has algebraic-degree Δ, algebraic-dimension d and extra algebraic-dimension s, if and only if all real registers at time of comparison have those properties. We are now ready to state the main theorem of this section.

► **Theorem 3.** *Consider the real RAM with square roots and a polynomial time algorithm A with algebraic-degree Δ, algebraic-dimension d, and extra algebraic-dimension s. Then under perturbations of the input of magnitude δ, A can be correctly executed on a real RAM if the input has an expected required bit-precision of* $O\left((d+s)\log \frac{\Delta n}{\delta}\right)$.

The theorem statement in its full generality allows for recursive applications of the square root function. For example, $\sqrt{\sqrt{x_1} + \sqrt{x_2} + x_3^5}$, would be a valid expression for a real register. However we wish to note that in less general form it applies to the classical sum of square roots problem, as explained in Section C.1

Before we elaborate on the proof of Theorem 3, we first show how to transform an expression f into an equivalent set of polynomials. Then we go through the snapping and the integrating part in the same manner as with single polynomials.

► **Lemma 16.** *Let f be an expression on variables $x_1, \ldots, x_d$, with at most s square roots and divisions. Then we can transform f, into equiavlent polynomials $p, q_1, \ldots, q_s \in \mathbb{Z}[x_1, \ldots, x_d, y_1, \ldots, y_s]$.*

Proof. We transform $f = f_0$ step by step, into expressions $f_1, \ldots, f_s$ such that $f_s = p$ is a polynomial and we add in step i one additional polynomial q_i. In step i, we distinguish between two cases, and always introduce the variable y_i:

The first case is that f_{i-1} has the form $f_{i-1} = g(\sqrt{h})$, where h is a polynomial and g is some other expression. Note that g and h may also depend on other variables. We define $f_i = g(y_i)$, with the new variable y_i and add the constraint $q_i = y_i^2 - h$ which is a polynomial expression.

The second case is that f_{i-1} has the form $f_{i-1} = e(g/h)$, where g, h are a polynomials and e is some other expression. Note that e, g and h may also depend on other variables. Then we define $f_i = e(g \cdot y_i)$ and $q_i = y_i \cdot h - 1$. ◄

Let us consider similar, to the case of the real RAM without square roots, the case that a single register takes a different decision.

► **Lemma 17** (Register-Snapping)**.** *Let R be a register, which holds some expression f with maximum degree Δ, algebraic-dimension d and extra algebraic-dimension s. Let $a \in \mathbb{R}^d$ be fixed and $x \in \Omega =$*

$[-\delta/2, \delta/2]^d$ *chosen uniformly at random. Assume* $(a+x)$ *is snapped to a point* $a' \in \omega\mathbb{Z}^d \cap [0,1]^d$. *Then* $\text{sign}(f(a+x)) \neq \text{sign}(f(a'))$ *with probability at most:*

$$\frac{\omega(2\Delta)^{s+1}3^d(d+1)!}{\delta}.$$

For the forthcoming proof, we assume that the reader is familiar with the proof of Lemma 13.

Proof. As f is a continuous function, we can upper bound the probability of $f(a') \neq f(a)$, by counting the number of Voronoi cells $c \in \Gamma_\omega$, which are intersected by $V(f)$. Unfortunately, f is not a polynomial and we cannot employ Theorem 20.

Instead, we consider the polynomials $p, q_1, \ldots, q_s \in \mathbb{R}[x_1, \ldots, x_d, y_1, \ldots, y_s]$, which exist by assumption of the lemma. We denote by $\pi : \mathbb{R}^{d+s} \to \mathbb{R}^d, \quad (x_1, \ldots, x_d, y_1, \ldots, y_s) \mapsto (x_1, \ldots, x_d)$ the orthogonal projection on the first d coordinates. From the definition of $(p, q_1, \ldots, q_s)$ it follows that the orthogonal projection of their variety is equal to $V(f)$, or in other words: $V(f) = \pi(V(p, q_1, \ldots, q_s))$. We want to upper bound the number of d-dimensional Voronoi cells that are intersected by $V(f)$, and we obtain this by providing an upper bound on the number of $(d+s)$-dimensional Voronoi cells that are intersected by the higher-dimensional variety $V(p, q_1, \ldots, q_s)$. Given a cube $c \in \Gamma_\omega \subset \mathbb{R}^d$, we define (just as in Appendix A) the cylinder $\tilde{c} = \{(x, y) \in \mathbb{R}^{d+s} : x \in c\}$. Per construction of $(p, q_1, \ldots, q_s)$, it holds that:

$$V(f) \cap c = \emptyset \iff V(p, q_1, \ldots, q_s) \cap \tilde{c}.$$

In analogy to Lemma 13, we consider the d-dimensional cube partitioned by $\Gamma_\omega(a) \cong C^0(d, \lfloor \frac{\delta}{\omega} \rfloor)$. The grid $\Gamma_\omega(a)$ in our $(d+s)$-dimensional space becomes a collection of cylinders equivalent to $C^s(d, \lfloor \frac{\delta}{\omega} \rfloor)$. This now allows us to apply Theorem 23 and bound the number of cylinders that are intersected by $V(p, q_1, \ldots, q_s)$ by:

$$k^{d-1}(2\Delta)^{s+1}3^d(d+1)!,$$

Via our transformation, this implies that the number of cells in $\Gamma_\omega(a)$ that are intersected by $V(f)$ is upper bound by $k^{d-1}(2\Delta)^{s+1}3^d(d+1)!$. Just as in Lemma 13, the perimeter of $\Gamma_\omega(a)$ contains at most $2dk^{d-1}$ cells. There are k^d cubes in $\Gamma_\omega(a)$ and it follows that:

$$\begin{aligned} Pr(\text{sign}(f(a+x)) \neq \text{sign}(f(a'))) &\leq \tfrac{1}{k^d} \left(k^{d-1}(2\Delta)^{s+1}3^d(d+1)! + 2d\, k^{d-1}\right) \\ &\leq \tfrac{1}{k}\ 2(2\Delta)^{s+1}3^d(d+1)! \\ &\leq \tfrac{1}{\delta}\ \omega\, 2(2\Delta)^{s+1}3^d(d+1)! \end{aligned}$$

This finishes the proof. ◀

Lemma 17 is identical to Lemma 13, except Δ is replaced by $(2\Delta)^{s+1}$. Therefore the proof to Theorem 3 is henceforth identical to the proof of Theorem 1, except that Δ is replaced by $(2\Delta)^{s+1}$. This finishes the proof of Theorem 3.

C.1 Implications

In this section, we show the two implications of Theorem 3.

► **Corollary 18.** *Under perturbations of the input of magnitude* δ, *the sum of square roots can be computed on a real RAM with an expected bit-precision of* $O(n \log(n/\delta))$ *per input variable.*

In the of square roots algorithm, we have only one comparison, and the register holds the following expression

$$f = \sum_{i=1,\dots,n} \sqrt{x_i} - \sum_{i=n+1,\dots,2n} \sqrt{x_i}.$$

The following polynomials are equivalent: $p = y_1 + \ldots + y_n - y_{n+1} + \ldots + y_{2n}$ and $q_i = y_i^2 - x_i$. Thus $s = n, d = n, \Delta = 2$. Thus, we see that Corollary 4 is implied by Theorem 3.

► **Corollary 19.** *Under perturbations of the input of magnitude δ, the shortest path in a polygon can be computed on a real RAM with an expected bit-precision of $O(n \log(n/\delta))$ per path vertex variable.*

The fastest algorithm to compute the shortest path in a polygon with holes is by Hershberger and Suri [61]. However, it is not clear that the algorithm can be implemented using square roots alone. For our analysis, we consider the naive Dijkstra algorithm [36] on the visibility graph as follows. We define a graph $G = (V, E)$, where V is the set of vertices of the polygon together with the start and end points s, t. We say two vertices in G are adjacent if they are visible within the polygon. Each edge receives a weight, which is equal to the Euclidean distance. Now the Dijkstra algorithm on G will give us the shortest path. It is based on the comparison of length of previously computed paths. That is on the correct evaluation of the following expression:

$$f = \sum_{(i,j)\in E(P_1)} \sqrt{(x_i - x_j)^2 + (x_{i+1} - x_{j+1})^2} - \sum_{(i,j)\in E(P_2)} \sqrt{(x_i - x_j)^2 + (x_{i+1} - x_{j+1})^2}.$$

Here P_1 and P_2 denote some paths computed by the algorithm and $E(P_i)$ being their edge set. We assume for notational convenience that a_i, a_{i+1} hold the x and y-coordinate of the vertices of the polygon. Using Lemma 16, we can observe that the algebraic-dimension $d = O(n)$, the algebraic-degree $\Delta = 2$ and the extra algebraic-dimension $s = O(n)$. Now Theorem 3 implies Corollary 5.

D Polynomials Hitting Cubes

In the first part of this section, we upper bound the number of unit cubes that a d-variate polynomial p of bounded degree Δ can intersect in $C(d, k)$.

In Section D.1, we generalize Theorem 20, to deal with cylinders instead of cubes. Although the proofs are very similar, for pedagogical purposes, we keep them separate.

► **Theorem 20** (Hitting Cubes)**.** *Let $p \neq 0$ be a d-variate polynomial with maximum degree Δ and $k \geq 2\Delta + 2$. Then the polynomial p intersects at most $k^{d-1}\Delta 3^d(d+1)!$ unit cubes in $C(d, k)$.*

Our proof gives a slightly stronger, but more complicated upper bound. The proof idea is to consider the intersection between a cube $C_z \in C(d, k)$ and the polynomial p. Then either a connected component of $V(p)$ is contained in C_z or $V(p)$ must intersect one of the $(d-1)$-dimensional facets of C_z. In order to estimate how often the first situation can occur, we use a famous theorem by Oleinik-Petrovski/Thom/Milnor, in a slightly weaker form. See Basu, Pollack and Roy [9, Chapter 7] for historic remarks and related results.

► **Theorem 21** (Milnor [9])**.** *Given a set of d-variate polynomials $q_0, \ldots, q_s$ with maximal degree Δ. Then the variety $V(q_0, \ldots, q_s)$ has at most $(2\Delta)^d$ connected components.*

We use Milnor's theorem later for more than one polynomial.

We also need the following folklore lemma. See the book from Cox, Little, O'Shea [32], for more background on polynomials. Specifically Hilbert's Nullstellensatz, which can be found as Theorem 4.77 in the book by Basu, Pollack and Roy [9] is important.

► **Lemma 22** (folklore). *Let $p \in \mathbb{R}[x_1, \dots, x_d]$ be a d-variate polynomial and $H = \{x \in \mathbb{R}^d : \ell(x) = 0\}$ a $(d-1)$-dimensional hyperplane. Then $V(p) \cap H$ is the variety of a $(d-1)$-variate polynomial or ℓ is a polynomial factor of p.*

In our applications, ℓ will be of the form $x_i = a$, for some constant a.

Proof of Theorem 20. Note first that if p has a linear factor ℓ, we decompose p into $p = q \cdot \ell$ and apply the following for q and ℓ separately. This works as the maximum degree of q drops by one and the maximum degree of ℓ equals 1. Thus for the rest of the proof, we assume that p has no linear factors, which in particular, makes it possible to apply Lemma 22.

Let us define $f(d)$ as the maximum number of unit cubes of $C(d,k)$ that can be intersected by a d-variate polynomial $p \neq 0$ with maximal degree Δ. We will first show that

$$f(1) \leq \Delta. \tag{1}$$

Then we will show in a similar manner for every d, k and Δ holds that

$$f(d) \leq 2f(d-1) \cdot d(k+1) + (2\Delta)^d. \tag{2}$$

Solving the recursion then gives the upper bound of the theorem as follows: first, we show by induction that Equation 1 and Equation 2 imply $f(d) \leq (k+1)^{d-1}\Delta 2^d(d+1)!$. Equation 1 establishes the induction basis. Using $2\Delta \leq k$, the induction step goes as follows:

$$\begin{aligned}
f(d) &\leq 2f(d-1) \cdot d(k+1) + (2\Delta)^d \\
&\leq 2f(d-1) \cdot d(k+1) + (2\Delta)k^{d-1} \\
&\leq 2(k+1)^{d-2}\Delta 2^{d-1}(d)! \cdot d(k+1) + (2\Delta)k^{d-1} \\
&= (k+1)^{d-1}(2\Delta)2^{d-1}(d)! \cdot d + (2\Delta)k^{d-1} \\
&= (k+1)^{d-1}(2\Delta)(2^{d-1}(d)! \cdot d + 1) \\
&\leq (k+1)^{d-1}\Delta 2^d(d)! \cdot (d+1) \\
&= (k+1)^{d-1}\Delta 2^d(d+1)!
\end{aligned}$$

Now using $k \geq 2\Delta + 2 \geq 3$, we can deduce that

$$(k+1)^{d-1} = \frac{(k+1)^{d-1}}{k^{d-1}} \cdot k^{d-1} \leq (1.5)^d k^{d-1}$$

This implies that $f(d) \leq k^{d-1}\Delta 3^d(d+1)!$. It remains to show the validity of Equation 1 and 2.

If p is a univariate polynomial of degree Δ then its variety $V(p)$ is a set of at most Δ points and therefore p can intersect at most Δ disjoint unit intervals and this implies Equation 1.

To show the correctness of Equation 2, we refer to Figure 11. Note that there are $d(k+1)$ axis-parallel $(d-1)$-dimensional hyperplanes with integer coordinates that intersect $C(d,k)$. We denote them by $H(d,k)$. Formally, we define the hyperplane $h(i,a) = \{x \in \mathbb{R}^d : x_i = a\}$. This leads to the definition: $H(d,k) = \{h(i,a) : (i,a) \in [d] \times [k]\}$. By the comment at the beginning of the proof, we can assume that p has no linear factors and we apply Lemma 22 on p and each $h \in H(d,k)$. For all cubes in $C(d,k)$, all facets of such a cube are contained inside a $(d-1)$-dimensional hyperplane $h \in H(d,k)$. By Milnor's theorem, there are at most $(2\Delta)^d$ cubes in $C(d,k)$ which are intersected by p but whose boundary is not intersected by p. For any other cube in $C(d,k)$ that is intersected by p, the polynomial must intersect a $(d-1)$-dimensional facet of that cube.

Consider one of the $d(k+1)$ hyperplanes $h \in H(d,k)$. The set $I = h \cap C(d,k)$ is, up to affine coordinate transformations, equivalent to $C(d-1,k)$. Furthermore, by Lemma 22, $V(p)$ restricted

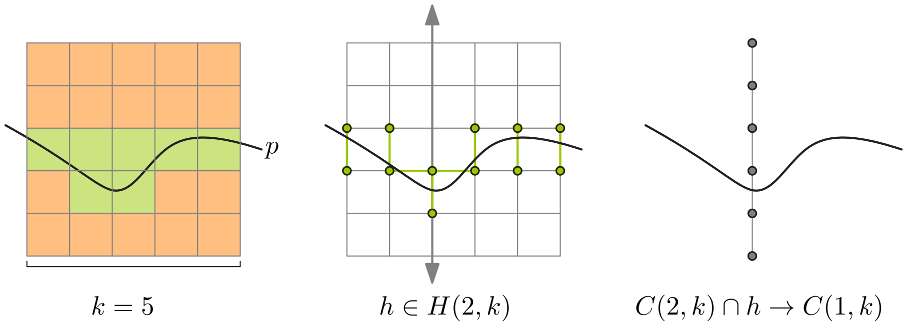

Figure 11. (Left) The polynomial p that intersects some cubes of $C(2,k)$. (Middle) The set $H(2,k)$ of axis-parallel lines and the 1-dimensional facets of $C(2,k)$ that are intersected by p. (Right) The intersection of $C(2,k)$ with a line H gives $C(1,k)$.

to h is the variety of a $(d-1)$-dimensional polynom. Thus by definition, we know that p intersects at most $f(d-1)$ cubes in I. Each of these $(d-1)$-dimensional cubes can coincide with a facet of at most two cubes in $C(d,k)$. It follows that $f(d)$ is upper bound by $(2\Delta)^d + 2 \cdot f(d-1) \cdot k(d+1)$. This shows Equation 2 and finishes the proof. ◀

D.1 Varieties hitting cylinders

In this section we generalize Theorem 20, to deal with more than one polynomial. For the application, we have in mind in Section B, we need to replace cubes by infinite cylinders.

► **Theorem 23.** *Let $q_0, \dots, q_s \in \mathbb{R}[x_1, \dots, x_d, y_1, \dots, y_s]$ be polynomials with maximum degree Δ and $k \geq 2\Delta + 1$, such that the variety $V(q_0, \dots, q_s)$ is $(d-1)$-dimensional. Then the variety $V(q_0, \dots, q_s)$ intersects at most*

$$k^{d-1}\,(2\Delta)^{s+1}\,3^d(d+1)!$$

unit cylinders in $C^s(d,k)$.

The proof of Theorem 23 follows the same line of argument as Theorem 20, but with different constants.

Similarly to Lemma 22, for polynomials, we have to state a similar lemma, for varieties.

► **Lemma 24** (folklore)**.** *Let $q_0, \dots, q_s \in \mathbb{R}[x_1, \dots, x_d, y_1, \dots, y_s]$ be polynomials with maximum degree Δ. Furthermore, the variety $V = V(q_0, \dots, q_s)$ is $(d-1)$-dimensional. Let $H = \{x \in \mathbb{R}^d : \ell(x) = 0\}$ be a $(d+s-1)$-dimensional hyperplane. Then either ℓ is a factor of all $q_0, \dots, q_s$ or $V \cap H$ is a $(d-2)$-dimensional variety.*

Proof of Theorem 23. Note first that if $q_0, \dots, q_s$ have a linear factor ℓ, we decompose $q_0, \dots, q_s$ into $q_i = q_i' \cdot \ell$ and apply the following for $q_0', \dots, q_s'$ and ℓ separately. This works as the maximum degree of $q_0, \dots, q_s$ drops by one and the maximum degree of ℓ equals 1. Thus for the rest of the proof, we assume that $q_0, \dots, q_s$ have no linear factors, which in particular, makes it possible to apply Lemma 24.

Let us define $f(d)$ as the maximum number of unit cylinders of $C^s(d,k)$ that can be intersected by $V(q_0, \dots, q_s)$ with maximum degree Δ. We will first show that

$$f(1) \leq (2\Delta)^{s+1}. \tag{3}$$

Then we will show in a similar manner for every d, s and Δ holds that

$$f(d) \leq 2f(d-1)\cdot d(k+1) + (2\Delta)^{d+s}. \tag{4}$$

Solving the recursion then gives the upper bound of the theorem as follows: first, we show by induction that Equation 3 and Equation 4 imply $f(d) \leq (k+1)^{d-1}(2\Delta)^{s+1}2^d(d+1)!$. Equation 3 establishes the induction basis. Note that $2\Delta \leq k+1$ implies $(2\Delta)^{d-1} \leq (k+1)^{d-1}$. (See Assumption of the Theorem.) The induction step goes as follows:

$$\begin{aligned}
f(d) &\leq 2d(k+1)\cdot f(d-1) + (2\Delta)^{s+d} \\
&\leq 2d(k+1)\cdot f(d-1) + (2\Delta)^{s+1}k^{d-1} \\
&\leq 2d(k+1)\cdot[(k+1)^{d-2}(2\Delta)^{s+1}2^{d-1}(d)!] + (2\Delta)^{s+1}k^{d-1} \\
&= (k+1)^{d-1}\,(2\Delta)^{s+1}\,2^{d-1}(d)!\cdot d \,+\, (2\Delta)^{s+1}k^{d-1} \\
&= (k+1)^{d-1}\,(2\Delta)^{s+1}\cdot(2^{d-1}(d)!\cdot d+1) \\
&\leq (k+1)^{d-1}\,(2\Delta)^{s+1}\,2^{d}(d)!\cdot(d+1) \\
&= (k+1)^{d-1}\,(2\Delta)^{s+1}\,2^{d}\,(d+1)!
\end{aligned}$$

We simplify this expression slightly, using $k \geq 2\Delta+2 \geq 3$. We deduce that

$$(k+1)^{d-1} = \frac{(k+1)^{d-1}}{k^{d-1}}\cdot k^{d-1} \leq (1.5)^d k^{d-1}$$

This implies that $f(d) \leq k^{d-1}(2\Delta)^{s+1}3^d(d+1)!$. It remains to show the validity of Equation 3 and 4.

If $V = V(q_0,\ldots,q_s) \subset \mathbb{R}^{1+s}$ is a 0-dimensional variety of degree Δ then it is a set of at most $t = (2\Delta)^{s+1}$ points, by Theorem 21. Therefore V can intersect at most t disjoint unit cylinders and this implies Equation 3.

Now we show correctness of Equation 4. We denote $V = V(q_0,\ldots,q_s) \subset \mathbb{R}^{d+s}$. Furthermore, we note that there are $d(k+1)$ axis-parallel $(d-1)$-dimensional hyperplanes with integer coordinates that intersect $C^s(d,k)$. We denote them by $H(d,s,k)$. Formally, we define the hyperplane $h(i,a) = \{\, x \in \mathbb{R}^{d+s} : x_i = a\,\}$. This leads to the definition:

$$H(d,s,k) = \{h(i,a) : (i,a) \in [d]\times[k]\}.$$

(This is almost identical to the case with cubes.) For all cylinders in $C^s(d,k)$, all facets of such a cylinder are contained inside a $(d-1)$-dimensional hyperplane $h \in H(d,s,k)$. By Milnor's theorem, there are at most $(2\Delta)^{s+k}$ cylinders in $C^s(d,k)$ which are intersected by V but whose boundary is not intersected by V. For any other cylinder in $C^s(d,k)$ that is intersected by V, the variety must intersect a $(d-1)$-dimensional facet of that cylinder.

Consider one of the $d(k+1)$ hyperplanes $h \in H(d,k)$. The set $I = h \cap C^s(d,k)$ is, up to coordinate transformations, equivalent to $C(d-1,k)$. Thus by definition, we know that V intersects at most $f(d-1)$ cylinders in I. (In particular, $h \not\subseteq V$, by the comment at the beginning of the proof.) Each of these of these $(d-1)$-dimensional cylinders can coincide with a facet of at most two cylinders in $C(d,k)$. It follows that $f(d)$ is upper bound by $(2\Delta)^{s+d}+2\cdot f(d-1)\cdot k(d+1)$. This shows Equation 4 and finishes the proof. ◀

E Smoothed Analysis of Recognition Problems

Recently Hoog, Miltzow and Schaik [63] applied the concept of smoothed analysis to the recognition problem of order types. We generalize their approach to general recognition problems. This section is devoted to Theorem 8.

► **Theorem 8.** *Let A be an extraction algorithm with a recognition problem R_A and denote by n_1 and n_2 the size of the input for A and R_A respectively. Suppose that under perturbations of the input of A, the algorithm A can be robustly executed if the input has an expected maximal bit-precision of $O(\log(n_1/\delta))$. Then the recognition problem R_A, under slight perturbation of the witness, can be robustly verified with an expected bit-precision of $O(\log(n_2/\delta))$.*

We first formalize all notions in the theorem and then give a proof.

We say A is an *extraction algorithm*, if it takes some geometric input $g \in [0,1]^n$ and outputs some combinatorial object $A(g) = c \in \mathbb{Z}^m$. We assume that m is polynomial in n and only dependent on n. The recognition problem R_A takes a combinatorial object c as input, and asks if there exists some geometric object g such that $A(g) = c$.

Given a recognition problem R_A and some input c, the *guessing algorithm* non-deterministically guesses a geometric object g such that $A(g) = c$, if it exists. Otherwise, it outputs NIL. The bit-precision $\mathtt{bit}(c, R_A)$ of the guessing algorithm is the number of bits to describe each g_i in $g = (g_1, \ldots, g_n)$. To be precise $\mathtt{bit}(g_i)$ equals the number of bits to represent g_i in binary and $\mathtt{bit}(g) = \max_i \mathtt{bit}(g_i)$. Now, the bit-precision of the guessing algorithm is defined by $\mathtt{bit}(c; R_A) = \min_{g:A(g)=c} \mathtt{bit}(g)$.

We are now ready to explain, smoothed analysis, specifically for recognition problems. The conceptual difficulty with smoothed analysis of recognition problems is that the model of perturbation is defined in terms of the output and not in terms of the input. The formal description is straightforward, but it is unusual to think about smoothed analysis in this way. The probability distribution of the output gives indirectly a probability distribution on the input as well, but there may be no simple way to describe this input distribution explicitly. Let us fix some extraction algorithm A, and consider some model of perturbation. We fix the magnitude of perturbation δ and denote the perturbation space by Ω_δ. We have defined a simple model of perturbation, but we want to point out that other models of perturbations are valid just as well. We define the smoothed bit-precision of the guessing algorithm on input c as

$$\mathtt{bit}_{\text{smooth}}(c, R_A, \delta) = \max_{g:A(\mathbf{g})=c} \mathtt{bit}_{\text{smooth}}(g, A, \delta).$$

Let us unwrap the last equation. We have given some input c to the combinatorial problem and there may be several g with $A(g) = c$. Let us fix the worst one possible. When we look at the smoothed bit-precision of g, we are really perturbing g according to Ω_δ and then looking at the bit-precision with respect to this perturbation. If we denote by Γ_n the set of all combinatorial objects of size n, then the smoothed bit-precision equals:

$$\mathtt{bit}_{\text{smooth}}(n, \delta) = \max_{c \in \Gamma_n} \mathtt{bit}_{\text{smooth}}(c, R_A, \delta).$$

We are now ready for the proof.

Proof. This can be shown simply by using the definition.

$$\begin{aligned}\mathtt{bit}_{\text{smooth}}(n, \delta) &= \max_{c \in \Gamma_n} \mathtt{bit}_{\text{smooth}}(c, R_A, \delta) \\ &= \max_{c \in \Gamma_n} \max_{g:A(g)=c} \mathtt{bit}_{\text{smooth}}(g, A, \delta) \\ &= \max_{g:A(g) \in \Gamma_n} \mathtt{bit}_{\text{smooth}}(g, A, \delta)\end{aligned}$$

Note that if $A(g)$ has size n, that this implies that g has size $m = \Theta(n^c)$ for some fixed constant $c > 0$. Thus we get

$$\mathtt{bit}_{\text{smooth}}(n, \delta) = \max_{g \in \Gamma_m} \mathtt{bit}_{\text{smooth}}(g, A, \delta)$$

Now we need to observe that the last line is actually the definition of the bit-precision of A. Thus we get

$$\begin{aligned}\texttt{bit}_{\text{smooth}}(n,\delta) &= \texttt{bit}_{\text{smooth}}(m,A,\delta)\\ &= O(\log m/\delta)\\ &= O(\log n^c/\delta)\\ &= O(\log n/\delta).\end{aligned}$$

This finishes the proof. ◀

F Smoothed Analysis of resource augmentation.

For the art gallery problem, Dobbins, Holmsen and Miltzow [38] showed that expanding the polygon by so-called edge-inflations, guarding will become easier. This leads to small expected bit-precision under smoothed analysis. Their proof consists of a problem specific part and a calculation of probabilities and expectations. We generalize their idea to a widely applicable framework for smoothed analysis of resource augmentation so that it remains to verify three simple problem specific properties. This section is devoted to the proof of Theorem 6 and Corollary 7. We start with an example of geometric packing before we give a formal definition of a resource augmentation problem.

Convex geometric packing. In convex geometric packing the input is a set of convex objects that need to be placed in a unit square such that the objects do not overlap. This algorithmic problem is perceived as very difficult as it is not even known how to optimally pack a set of eleven unit squares into a minimum square-sized container [52]. Recent research shows that this problem is ∃ℝ-complete [3]. A solution to the packing problem must specify both a translation and a rotation of each of the input objects and the optimal rotation or translation can be a real number with unbounded precision. Geometric packing has a natural resource augmentation where you slightly enlarge the container. Our results imply that under smoothed analysis of this enlargement, the optimal solution can be expressed with an expected logarithmic bit-precision.

Definition. Let us fix some algorithmic optimization problem P. For it to be a *resource-augmentation* problem, we assume several specific conditions explained below. First, each input consists of the actual input I and an *augmentation-parameter* $\alpha \in [0,1]$. Secondly, we assume that there is an implicitly defined *solution space* $\chi_I[\alpha] = \chi[\alpha]$. We suppress the input I in the notation, when it is clear from context. Also we denote $\chi = \chi[0]$. Furthermore, we assume that there is an evaluation function $f : \chi[\alpha] \to \mathbb{N}$. The aim is to find a solution $x^* \in \chi[\alpha]$ for which $f(x^*)$ is the maximum or minimum denoted by $\text{opt}(\chi[\alpha])$. For notational convenience, we assume for the remainder of this section that P is a maximization problem. The other case is analogue. We say that α is a *breakpoint* if $\text{opt}(\chi[\alpha-\varepsilon]) \neq \text{opt}(\chi_I[\alpha+\varepsilon])$, $\forall \varepsilon > 0$.

We assume three natural properties for resource augmentation problems.

- The *monotonous* property, which states that for all inputs I, for all $\alpha, \alpha' \in [0,1]$ if $\alpha \leq \alpha'$ then $\chi_I[\alpha] \subseteq \chi_I[\alpha']$. Note that the *mono* property implies that in a more augmented version of the problem, the optimum is at least as good.
- The *moderate* property requires that the number of breakpoints N for χ_I is upper bound by $n^{O(1)}$.
- The *smoothable* property requires that for all x optimal in $\chi_I[\alpha]$ and for all $\varepsilon > 0$, there is an $x' \in \chi_I[\alpha+\varepsilon]$ with bit-precision $\leq b = c\log n/\varepsilon$, for some $c \in \mathbb{N}$, and $f(x) \leq f(x')$. In other words, given some solution x we can increase the augmentation parameter by ε, to attain an

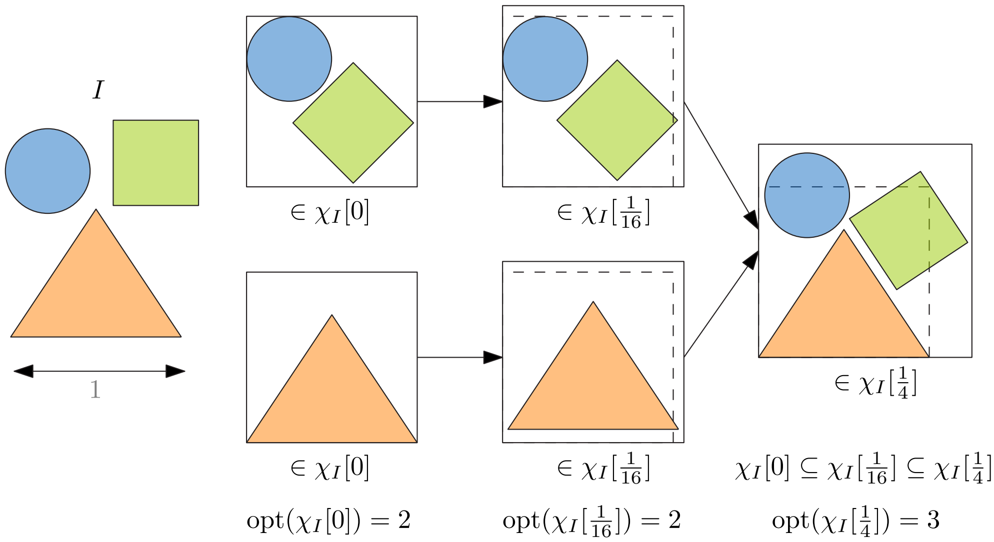

Figure 12. (left) the input for the geometric packing problem. Three convex objects and the diameter of the packing box. The non-augmented normal solution space $\chi_I[0]$ contains infinitely many solutions, as the objects can be rotated into many positions. The optimal packing might make use of a rotation which cannot be expressed using finite precision. If we augment the input, we need to pack the elements in a larger container and this allows for more solutions. The solution space $\chi_I[\frac{1}{4}]$ has a different optimum than $\chi_I[\frac{1}{16}]$ which implies that somewhere in the interval $[\frac{1}{16}, \frac{1}{4}]$ there is a breakpoint.

equally good solution $x' \in \chi[\alpha + \varepsilon]$. Furthermore, x' has low bit-precision. Note that x' is not necessarily optimal for $\chi[\alpha + \varepsilon]$.

We apply smoothed analysis to resource-augmentation problems with these three properties by choosing uniformly at random the augmentation $\alpha \in [0, \delta]$. We denote by $\mathtt{bit}(\chi[\alpha])$ the minimal bit-precision needed to express an optimal solution in the solution space $\chi[\alpha]$. We upper bound the worst case expected value $\mathbb{E}[\mathtt{bit}[\alpha]]$ for α uniform at random between $[0, \delta]$.

► **Theorem 6.** *Let P be a resource augmentation problem that is monotonous, moderate and smoothable. Under perturbations of the augmentation of magnitude δ, the problem P has an optimal solution with an expected bit-precision of $O(\log(n/\delta))$.*

Proof. Let the input I be fixed and let α be the augmentation parameter chosen uniformly at random within the perturbation range $[0, \delta]$.

Consider any value $\varepsilon > 0$. The variable α is chosen uniformly at random in the interval $(0, \delta]$, and P is moderate which implies that the probability that the interval $[\alpha - \varepsilon, \alpha]$ contains a breakpoint is upper bound by $\varepsilon N/\delta$. Assume that the interval $[\alpha - \varepsilon, \alpha]$ does not contain a breakpoint then by definition $\mathrm{opt}(\chi[\alpha - \varepsilon]) = \mathrm{opt}(\chi[\alpha])$. Let x be an optimal solution in $\chi[\alpha - \varepsilon]$. As the problem P is smoothable and $x \in \chi[\alpha - \varepsilon]$, there must be an $x' \in \chi[\alpha - \varepsilon + \varepsilon] = \chi[\alpha]$ with a bit-precision of at most $c \log(n/\varepsilon)$ and x' is also optimal for $\chi[\alpha]$. It follows that for a random $\alpha \in (0, \delta]$, the probability that there is *no* optimal solution in $\chi[\alpha]$ with a bit-precision of $c \log(n/\varepsilon)$ is upper bound by:

$$Pr\left(\mathtt{bit}(\chi[\alpha]) \geq c \log(n/\varepsilon)\right) \leq \frac{\varepsilon \cdot N}{\delta}. \tag{5}$$

Note that Equation 5 holds for every ε. We use this probability to obtain an upper bound on the expected bit-precision. Using Lemma 15, we note that for all positive integers l holds that:

$$\mathbb{E}(\mathtt{bit}(\chi[\alpha])) = \sum_{k=1}^{l} \Pr(\mathtt{bit}(\chi[\alpha] \geq k)) \;+\; \sum_{k=l+1}^{\infty} \Pr(\mathtt{bit}(\chi[\alpha] \geq k))$$

Any probability is at least 1 which means that the left sum is upper bound by l. We upper bound $\Pr(\mathtt{bit}(\chi[\alpha]) \geq k)$ by equating $k = c\log(n/\varepsilon_k) \Leftrightarrow \varepsilon_k = n/2^{(k/c)}$ and applying Equation 5:

$$\begin{aligned}
\mathbb{E}(\mathtt{bit}(\chi[\alpha])) &\leq l + \sum_{k=l+1}^{\infty} \Pr(\mathtt{bit}(\chi[\alpha] \geq k)) \\
&= l + \sum_{k=l+1}^{\infty} \Pr(\mathtt{bit}(\chi[\alpha] \geq c\log(n/\varepsilon_k))) \\
&\leq l + \sum_{k=l+1}^{\infty} \frac{\varepsilon_k \cdot N}{\delta} \\
&= l + \sum_{k=l+1}^{\infty} \frac{nN}{2^{k/c}\delta} \\
&= l + \frac{nN}{\delta} \sum_{k=l+1}^{\infty} \left(2^{1/c}\right)^{-k} \\
&= l + \frac{nN}{\delta}\, 2^{-(l+1)/c} \sum_{k'=0}^{\infty} \left(2^{-1/c}\right)^{k'} \\
&= l + \frac{nN}{\delta}\, 2^{-(l+1)/c}\, \frac{1}{1-2^{-1/c}} \\
&\leq l + 2^{-(l+1)} \frac{nN}{\delta(1-2^{-1/c})}
\end{aligned}$$

We choose $l = \lceil \log(nN/\delta) \rceil$ and note that:

$$\mathbb{E}(\mathtt{bit}(\chi[\alpha])) \leq \log(nN/\delta) \;+\; 2^{-\log(nN/\delta)} \frac{nN}{\delta(1-2^{-1/c})} + 1 \leq O(\log(n/\delta))$$

This concludes the theorem. ◀

Applying Theorem 6. Dobbins, Holmsen and Miltzow show that under smoothed analysis the $\exists\mathbb{R}$-complete art gallery problem has a solution with logarithmic expected bit-precision [38]. They show this under various models of perturbation with one being *edge-inflation*. During *edge-inflation*, for each edge e of the polygon, they shift the edge e by α outwards in a direction orthogonal to e. Our theorem generalizes the smoothed analysis result from [38].

► **Corollary 25** ([38]). *For the art gallery problem, with an α-relaxation which is an α-edge-inflation, the expected bit-precision with smoothed analysis over α is $O(\log(n/\delta))$.*

Proof. The monotonous property and the smoothable property are both shown in the short Lemmas 6 and 7 in [38]. The evaluation function f in the art gallery problem counts the number of guards of a solution. The number of guards is a natural number and any polygon can be guarded using at most $n/3$ guards [46]. This implies the moderate property. Together with the monotonous property this proves that the number of breakpoints is upper bound by $n/3$ and the corollary follows. ◀

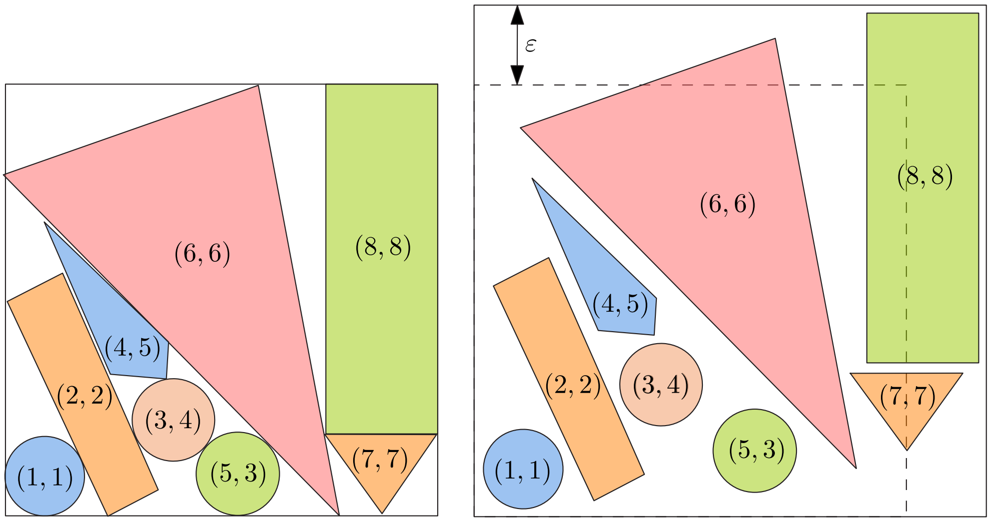

Figure 13. (Left) A set of 8 convex objects which are tightly packed in a square with diameter $(1+\alpha)$. Their enumerations specify a choice for π_x and π_y. Note that this choice is not unique. (Right) The objects packed in a square of diameter $(1+\alpha+\varepsilon)$ where an object with enumeration (i,j) is translated by $(\frac{i\varepsilon}{10}, \frac{j\varepsilon}{10})$.

We can also prove new results such as the following result about geometric packing:

► **Corollary 26.** *For geometric packing of convex objects in a unit-size container, with an α-relaxation over the container size, the expected bit-precision with smoothed analysis over α is $O(\log(n/\delta))$.*

Proof. In order to prove the corollary, we only have to prove that geometric packing with α-relaxation over the container size is monotonous, moderate and smoothable. For an input I and unit container size the solution space $\chi_I[\alpha]$ is the set of all solutions which contain a set of geometric objects from I packed within a container of size $(1+\alpha)$.

1. The monotonous property. For all $\alpha, \alpha' \in [0,1]$ if $\alpha \leq \alpha'$ then $\chi_I[\alpha]$ contains subsets of I which can be packed in a container of size $(1+\alpha)$ and therefore the elements can be packed in a container of size $(1+\alpha')$.
2. The evaluation function f counts the number of geometric objects which are correctly packed in the solution. Thus the function f can evaluate to at most n distinct values and therefore the number of breakpoints is upper bound by n.
3. Let $x \in \chi[\alpha]$ be a packing of k elements of the input in a container of size $(1+\alpha)$. Fix a value $\varepsilon > 0$ and consider x placed in a container of width $(1+\alpha+\varepsilon)$. Using the extra space and the fact that the objects are convex, we can translate the objects in x such that any two objects in x are at least $\frac{\varepsilon}{n+2}$ apart from each other and the boundary by translating them in the two cardinal directions (refer to Figure 13). We assign a linear order π_x on the n elements such that: (1) if an object dominates another in the x direction it is further in the ordering and (2) if you take an arbitrary horizontal line, the order of intersection with this line respects the order π_x. We define π_y symmetrically for vertical lines. Since the input objects are convex, such an ordering always exists. Let a convex object O in $x \in \chi_I[\alpha]$ be the i'th element in π_x and the j'th element in π_y (we start counting from 1). We translate O by $\left(\frac{i\varepsilon}{n+2}, \frac{j\varepsilon}{n+2}\right)$. Observe that (1) all objects are contained in a container of diameter $(1+\alpha+\varepsilon)$ since an element is translated by at most $\frac{n\varepsilon}{n+2}$ in any cardinal direction and (2) all objects are separated by at least $\frac{\varepsilon}{n+2}$. (Formally, we say A, B are separated by d, if and only if $\text{dist}(A,B) \geq d$.) Since any two objects are separated by at least $\frac{\varepsilon}{n+2}$, we can freely translate every object by $O(\varepsilon/n)$ and freely rotate every object by $O(\varepsilon/n)$ degrees such that all objects are still mutually

disjoint. This free rotation and translation immediately implies that there is a positioning of the objects of x where all objects are correctly packed in a container of size $(1+\alpha+\varepsilon)$ which can be represented with at most $O(\log(n/\varepsilon))$ bit-precision and this is our new solution $x' \in \chi[\alpha+\varepsilon]$. ◀

Lastly to demonstrate the wide applicability of Theorem 6 we investigate a classical problem within computational geometry which is not an ETR-hard problem: computing the minimum-link path. In this problem the input is a polygonal domain and two points contained in it and one needs to connect the points by a polygonal path with minimum number of edges. Recently, Kostitsyna, Löffler, Polishchuk and Staals [72] showed that even if the polygonal domain P is a simple polygon where the n vertices of the polygon each have coordinates with $\log n$ bits each, then still the minimal bit-precision needed to represent the minimum-link path is $O(k \log n)$ where k is the length of the path and they present a construction where $k = \Omega(n)$.

Just like the art gallery problem, the minimum link path problem has a simple polygon as input and we propose to relax the minimum link path problem in the same way as the art gallery problem was relaxed in [38]: by edge-inflation. Two points in the minimum link path may be connected if and only if they are mutually visible. Hence, with an analysis identical to 25 we can immediately state that:

► **Corollary 27.** *For computing the minimum link path, with an α-relaxation which is edge-inflation, the expected bit-precision with smoothed analysis over α is $O(\log(n/\delta))$.*

G Algorithmic Membership in $\exists\mathbb{R}$

A *discrete decision problem* is any function Q from arbitrary integer vectors to the booleans {TRUE, FALSE} (or equivalently, any language over the alphabet $\{0,1\}$). An integer vector I is called a *yes-instance* of Q if $Q(I) = $ TRUE and a *no-instance* of Q if $Q(I) = $ FALSE. Let $\circ$ denote the concatenation operator.

A *real verification algorithm* for Q is a real RAM program A that satisfies the following conditions, for some constant $c \geq 1$:

- A halts after at most N^c time steps, using word size $\lceil c \log_2 N \rceil$, given any input of total length N.
- For every yes-instance $I \in \mathbb{Z}^n$, there is a real vector x and an integer vector z, each of length at most n^c, such that A accepts input $(x, I \circ z)$.
- For every no-instance I, for every real vector x and every integer vector z, A rejects input $(x, I \circ z)$.

A *certificate* (or *witness*) for yes-instance I is any vector pair (x, z) such that A accepts $(x, I \circ z)$. Intuitively, a real verification algorithm is a *nondeterministic* polynomial-time algorithm on the real RAM that allows nondeterminism both by guessing words (or other discrete choices) and by guessing exact real numbers.

The rest of this section is devoted to proving the following theorem:

► **Theorem 10.** *For any discrete decision problem Q, there is a real verification algorithm for Q if and only if $Q \in \exists\mathbb{R}$.*

As usual, we prove this theorem in two stages. First, we describe a trivial real-verification algorithm for ETR. Second, for any discrete decision problem Q with a real-verification algorithm, we describe a polynomial-time algorithm *on the word RAM* that transforms every yes-instance of Q into a true ETR formula and transforms every no-instance of Q into a false ETR formula. The first reduction implies that every problem in $\exists\mathbb{R}$ has a real-verification algorithm; the second implies that every real-verifiable discrete decision problem is in $\exists\mathbb{R}$.

► **Lemma 28.** *ETR has a real verification algorithm.*

Proof. Let $\Phi = \exists x_1, \ldots, x_n \colon \phi(x_1, \ldots, x_n)$ be an arbitrary formula in the existential theory of the reals. The underlying predicate ϕ is a string over an alphabet of size $m + O(1)$ (the symbols $0, 1, +, \cdot, =, \leq, <, \wedge, \vee, \neg$, and the variables $x_1, \ldots, x_n$), so we can easily encode ϕ as an integer vector. Our real verification algorithm takes the (encoded) predicate ϕ and a real certificate vector $x \in \mathbb{R}^m$ as input, and evaluates the predicate $\phi(x)$ using (for example) a standard recursive-descent parser. This algorithm clearly runs in time polynomial in the length of Φ. ◄

► **Lemma 29.** *Every discrete decision problem with a real verification algorithm is in $\exists\mathbb{R}$.*

Proof. Fix a real verification algorithm A for some discrete decision problem Q. We argue that for any integer vector I, we can compute in polynomial time *on the word RAM* a corresponding ETR formula $\Phi(I)$, such that $\Phi(I)$ is true if and only if I is a yes-instance of Q. Mirroring textbook proofs of the Cook-Levin theorem, the formula Φ encodes the complete execution history of A on input $(x, I \circ z)$. The certificate vectors x and z appear as existentially quantified variables of Φ; the input integers I are hard-coded into the underlying proposition Φ.

Now fix an instance $I \in \mathbb{Z}^n$ of Q. Let $N = n + 2n^c$, let $w = \lceil c \log_2 N \rceil = 2c \log_2 n + O(1)$, and let $T = N^c = O(n^{c^2})$. Thus, w is an upper bound on the word size and T is an upper bound on the running time of A given input $(x, I \circ z)$, for any certifciates x and z of length at most n^c. Our output formula $\Phi(I)$ includes the following *register variables*, which encode the complete state of the machine at every time step t from 0 to T:

- For each address i, variable $[\![W(i,t)]\!]$ stores the value of word register $W[i]$ at time t.
- For each address i, variable $[\![R(i,t)]\!]$ stores the value of real register $R[i]$ at time t.
- Finally, variable $[\![pc(t)]\!]$ stores the value of the program counter at time t.

Altogether $\Phi(I)$ has $(2 \cdot 2^w + 1)T = O(n^{2c^2})$ register variables. These are not the only variables in $\Phi(I)$; we will introduce additional variables as needed as we describe the formula below.

Throughout the following presentation, all indexed conjunctions ($\bigwedge$), disjunctions ($\bigvee$), and summations ($\sum$) are notational shorthand; each term in these expressions appears explicitly in the actual formula $\Phi(I)$. For example, the indexed conjunction

$$\bigwedge_{b=1}^{w} \left([\![2^b]\!] = [\![2^{b-1}]\!] + [\![2^{b-1}]\!]\right)$$

is shorthand for the following explicit sequence of conjunctions

$$\left([\![2^1]\!] = [\![2^0]\!] + [\![2^0]\!]\right) \wedge \left([\![2^2]\!] = [\![2^2]\!] + [\![2^1]\!]\right) \wedge \cdots \wedge \left([\![2^w]\!] = [\![2^{w-1}]\!] + [\![2^{w-1}]\!]\right).$$

Integrality. To constrain certain real variables to have integer values, we introduce new global variables $[\![2^0]\!]$, $[\![2^1]\!]$, $[\![2^2]\!]$, …, $[\![2^w]\!]$ and equality constraints

$$\texttt{PowersOf2} := ([\![2^0]\!] = 1) \wedge \bigwedge_{b=1}^{w} ([\![2^b]\!] = [\![2^{b-1}]\!] + [\![2^{b-1}]\!])$$

The following ETR expression forces the real variable X to be an integer between 0 and 2^{w-1}:

$$\texttt{IsWord}(X) \ := \ \exists x_0, x_1, \ldots, x_{w-1} \colon \left(X = \sum_{b=0}^{w-1} x_b [\![2^b]\!]\right) \wedge \left(\bigwedge_{b=0}^{w-1} \left(x_b(x_b - 1) = 0\right)\right)$$

We emphasize that each invocation of `IsWord` requires w new variables x_b; each x_b stores the bth bit in the binary expansion of X. Our final formula $\Phi(I)$ includes the following conjunction, which forces the initial values of every word register variable to actually be a word:

$$\texttt{WordsAreWords} := \bigwedge_{i=0}^{2^w-1} \texttt{IsWord}\big(\llbracket W(i,0)\rrbracket\big)$$

Altogether this subexpression involves $w2^w$ new one-bit variables and has length $O(w2^w)$. Similarly, we can force variable X to take on a fixed integer value j by explicitly summing the appropriate powers of 2:

$$\texttt{Equals}(X,j) \ := \ \left(X = \sum_{b\,:\,j_b=1} \llbracket 2^b \rrbracket\right)$$

Input and Output. We hardcode the fixed instance I into the formula with the conjunction

$$\texttt{FixInput} \ := \ \bigwedge_{i=0}^{n-1} \texttt{Equals}\big(\llbracket W(i,0)\rrbracket,\ I[i]\big)$$

(Here $I[i]$ denotes the ith coordinate of I.) We leave the remaining initial word register variables $\llbracket W(i,0)\rrbracket$ and all initial real register variables $\llbracket R(i,0)\rrbracket$ unconstrained to allow for arbitrary certificates.

Execution. Finally, we add constraints that simulate the actual execution of A. Let L denote the number of instructions ("lines") in program A; recall that L is a constant independent from I. For each time step t and each instruction index ℓ, we define a constraint $\texttt{Update}(t,\ell)$ that forces the memory at time t to reflect an execution of line ℓ, given the contents of memory at time $t-1$. Our formula $\Phi(I)$ then includes the conjunction

$$\texttt{Execute} \ := \ \bigwedge_{t=1}^{T} \bigwedge_{\ell=1}^{L} \big((\llbracket pc(t)\rrbracket = \ell) \wedge \texttt{Update}(t,\ell)\big)$$

The various expressions $\texttt{Update}(t,\ell)$ are nearly identical. In particular, $\texttt{Update}(t,\ell)$ includes the constraints $\llbracket W(i,t)\rrbracket = \llbracket W(i,t-1)\rrbracket$ and $\llbracket R(j,t)\rrbracket = \llbracket R(j,t-1)\rrbracket$ for every word register $W[i]$ and real register $R[j]$ that are not changed by instruction ℓ. Similarly, unless instruction ℓ is a control flow instruction, $\texttt{Update}(t,\ell)$ includes the constraint

$$\texttt{Step}(t) \ := \ (\llbracket pc(t)\rrbracket = \llbracket pc(t-1)\rrbracket + 1).$$

Tables 2, 3, and 4 lists the important constraints in $\texttt{Update}(t,\ell)$ for three different classes of instructions.

- Encoding constant assignment, direct memory access, real arithmetic (including square roots), and control flow instructions is straightforward; see Table 2. For an accept instruction, we set all future program counters to 0, which effectively halts the simulation. Similarly, we encode the reject instruction as the trivially false expression $(0 = 1)$.
- Because there is no indirection mechanism in ETR itself, we are forced to encode indirect memory instructions using a brute-force enumeration of all 2^w possible addresses. For example, our encoding of the instruction $W[W[i]] \leftarrow W[j]$ actually encodes the brute-force linear scan "for all words k, if $W[i] = k$, then $W[k] \leftarrow W[j]$". See Table 3.
- Finally, Table 4 shows our encodings of arithmetic and bitwise boolean operations on words. For addition and subtraction, we store the result of the integer operation in a new variable z, and then store $z \bmod 2^w$ using a single conditional. For upper and lower multiplication, we

Instruction	Constraint
$W[i] \leftarrow j$	$\texttt{Equals}(\llbracket W(i,t)\rrbracket, j)$
$R[i] \leftarrow 0$	$\big(\llbracket R(i,t)\rrbracket = 0\big)$
$R[i] \leftarrow 1$	$\big(\llbracket R(i,t)\rrbracket = 1\big)$
$R[i] \leftarrow j$	$\texttt{Equals}(\llbracket R(i,t)\rrbracket, j)$
$W[i] \leftarrow W[j]$	$\big(\llbracket W(i,t)\rrbracket = \llbracket W(j,t-1)\rrbracket\big)$
$R[i] \leftarrow R[j]$	$\big(\llbracket W(i,t)\rrbracket = \llbracket W(j,t-1)\rrbracket\big)$
$R[i] \leftarrow W[j]$	$\big(\llbracket R(i,t)\rrbracket = \llbracket W(j,t-1)\rrbracket\big)$
$R[i] \leftarrow R[j] + R[k]$	$\llbracket R(i,t)\rrbracket = \llbracket R(j,t-1)\rrbracket + \llbracket R(k,t-1)\rrbracket$
$R[i] \leftarrow R[j] - R[k]$	$\llbracket R(i,t)\rrbracket = \llbracket R(j,t-1)\rrbracket - \llbracket R(k,t-1)\rrbracket$
$R[i] \leftarrow R[j] \cdot R[k]$	$\llbracket R(i,t)\rrbracket = \llbracket R(j,t-1)\rrbracket \cdot \llbracket R(k,t-1)\rrbracket$
$R[i] \leftarrow R[j]/R[k]$	$\llbracket R(i,t)\rrbracket \cdot \llbracket R(k,t-1)\rrbracket = \llbracket R(j,t-1)\rrbracket$
$R[i] \leftarrow \sqrt{R[j]}$	$\llbracket R(i,t)\rrbracket \cdot \llbracket R(i,t)\rrbracket = \llbracket R(j,t-1)\rrbracket$
if $W[i] = W[j]$ goto ℓ	if $(\llbracket W(i,t-1)\rrbracket = \llbracket W(j,t-1)\rrbracket)$ then $(\llbracket pc(t)\rrbracket = \ell)$ else $\texttt{Step}(t)$
if $W[i] < W[j]$ goto ℓ	if $(\llbracket W(i,t-1)\rrbracket < \llbracket W(j,t-1)\rrbracket)$ then $(\llbracket pc(t)\rrbracket = \ell)$ else $\texttt{Step}(t)$
if $R[i] = 0$ goto ℓ	if $(\llbracket R(i,t-1)\rrbracket = 0)$ then $(\llbracket pc(t)\rrbracket = \ell)$ else $\texttt{Step}(t)$
if $R[j] > 0$ goto ℓ	if $(\llbracket R(i,t-1)\rrbracket > 0)$ then $(\llbracket pc(t)\rrbracket = \ell)$ else $\texttt{Step}(t)$
goto ℓ	$\llbracket pc(t)\rrbracket = \ell$
accept	$\bigwedge_{i=t}^{T}(\llbracket pc(i)\rrbracket = 0)$
reject	$0 = 1$

Table 2. Encoding constant assignment, direct memory access, real arithmetic, and control-flow instructions as formulae; "if A then B else C" is shorthand for $(A \wedge B) \vee (\neg A \wedge C)$

define two new *word* variables u and l, declare that $u \cdot 2^w + l$ is the actual product, and then store either u or l. Similarly, to encode the division operations, we define two new word variables that store the quotient and the remainder. Finally, we encode bitwise boolean operations as the conjunction of w constraints on the one-bit variables defined by `IsWord`.

Summary. Our final ETR formula $\Phi(I)$ has the form

$$\exists[\text{variables}]: \texttt{PowersOf2} \wedge \texttt{FixInput} \wedge \texttt{WordsAreWords} \wedge \texttt{Execute} \wedge (\llbracket pc(T)\rrbracket = 0)$$

Now suppose I is a yes-instance of $\mathbb{Q}$. If we set the initial register variables to reflect the input $(x, I \circ z)$ for some certificate (x, z), then `Execute` forces the final program counter $\llbracket pc(T)\rrbracket$ to 0, at the time step when A accepts $(x, I \circ z)$). It follows that $\Phi(I)$ is true.

On the other hand, if I is a no-instance of Q, then no matter how we instantiate the remaining initial register variables, the `Execute` subexpression will include the contradiction $(0 = 1)$ at the time step when A rejects. It follows that $\Phi(I)$ is false.

Altogether, $\Phi(I)$ has $O(2^w(T + w) + TLw)$ existentially quantified variables and total length $O(2^w TL) = O(n^{2c^2})$, which is polynomial in n. Said differently, the length of $\Phi(I)$ is at most a constant times the *square* of the running time of A on input $(x, I \circ z)$, where x and z are certificate vectors of maximum possible length.

Instruction	Constraint
$W[W[i]] \leftarrow W[j]$	$\bigvee_{k=0}^{2^w-1} \Big((\llbracket W(i,t) \rrbracket = k) \wedge (\llbracket W(k,t) \rrbracket = \llbracket W(j,t-1) \rrbracket) \Big)$
$W[i] \leftarrow W[W[j]]$	$\bigvee_{k=0}^{2^w-1} \Big((\llbracket W(j,t) \rrbracket = k) \wedge (\llbracket W(i,t) \rrbracket = \llbracket W(k,t-1) \rrbracket) \Big)$
$R[W[i]] \leftarrow R[j]$	$\bigvee_{k=0}^{2^w-1} \Big((\llbracket W(i,t) \rrbracket = k) \wedge (\llbracket R(k,t) \rrbracket = \llbracket R(j,t-1) \rrbracket) \Big)$
$R[i] \leftarrow R[W[j]]$	$\bigvee_{k=0}^{2^w-1} \Big((\llbracket W(j,t) \rrbracket = k) \wedge (\llbracket R(i,t) \rrbracket = \llbracket R(k,t-1) \rrbracket) \Big)$

Table 3. Encoding indirect memory instructions as formulae

We can easily construct $\Phi(I)$ in polynomial time *on the word RAM* by brute force. We emphasize that constructing $\Phi(I)$ requires no manipulation of real numbers, only of symbols that represent existentially quantified real variables. ◀

G.1 Examples

To illustrate the usefulness of Theorem 10, we give simple proofs that three example problems are in $\exists\mathbb{R}$. For two of these problems, membership in $\exists\mathbb{R}$ was already known [2, 42]; however, our proofs are significantly shorter and follow from known standard algorithms. We introduce the third problem specifically to illustrate the mixture of real and discrete non-determinism permitted by our technique.

► **Corollary 30.** *The following discrete decision problems are in $\exists\mathbb{R}$.*
- *The art gallery problem [2].*
- *The optimal curve straightening problem [42].*
- *The optimal unknotted extension problem.*

Proof. Recall that the input to the art gallery problem is a polygon P with rational coordinates and an integer k; the problem asks whether there is a set G of k guard points in the interior of P such that every point in P is visible from at least one point in G. To verify a yes-instance, it suffices to guess the locations of the guards (using $2k$ real registers), compute the visibility polygon of each guard in $O(n \log n)$ time [59], compute the union of these k visibility polygons in $O(n^2k^2)$ time, and finally verify that the union is equal to P. We can safely assume $k < n$, since otherwise the polygon is trivially guardable, so the verification algorithm runs in polynomial time.

The optimal curve-straightening problem was introduced by the first author [42]. The input consists of an integer k and a suitable abstract representation of a closed non-simple curve γ in the plane with n self-intersections; the problem asks whether there is a k-vertex polygon P that is isotopic to γ, meaning that the image graphs of P and γ as isomorphic as plane graphs. To verify a yes-instance of this problem, it suffices to guess the vertices of the k-gon P (using $2k$ real registers), compute the image graph of P in $O((n+k)\log n)$ time using a standard sweep-line algorithm [12], and then verify by brute force that P and γ have identical crossing patterns. Again, we can safely assume that $k = O(n)$, since otherwise the curve is trivially straightenable, so the verification algorithm runs in polynomial time.

Instruction	Constraint
$W[i] \leftarrow (W[j]+W[k]) \bmod 2^w$	$\exists z\colon (z = [\![W(j,t-1)]\!] + [\![W(k,t-1)]\!]) \wedge$ $\big(\text{if } (z < [\![2^w]\!]) \text{ then } ([\![W(k,t)]\!] = z) \text{ else } ([\![W(k,t)]\!] = z - [\![2^w]\!])\big)$
$W[i] \leftarrow (W[j]-W[k]) \bmod 2^w$	$\exists z\colon (z = [\![W(j,t-1)]\!] - [\![W(k,t-1)]\!]) \wedge$ $\big(\text{if } (z \geq 0) \text{ then } ([\![W(k,t)]\!] = z) \text{ else } ([\![W(k,t)]\!] = z + [\![2^w]\!])\big)$
$W[i] \leftarrow (W[j]\cdot W[k]) \bmod 2^w$	$\exists u,l\colon \text{IsWord}(u) \wedge \text{IsWord}(l) \wedge ([\![W(i,t)]\!] = l) \wedge$ $(u \cdot [\![2^w]\!] + l = [\![W(j,t-1)]\!] \cdot [\![W(k,t-1)]\!])$
$W[i] \leftarrow \lfloor W[j]\cdot W[k]/2^w \rfloor$	$\exists u,l\colon \text{IsWord}(u) \wedge \text{IsWord}(l) \wedge ([\![W(i,t)]\!] = u) \wedge$ $(u \cdot [\![2^w]\!] + l = [\![W(j,t-1)]\!] \cdot [\![W(k,t-1)]\!])$
$W[i] \leftarrow W[j] \bmod W[k]$	$\exists q,r\colon \text{IsWord}(q) \wedge \text{IsWord}(r) \wedge ([\![W(i,t)]\!] = r) \wedge$ $(u \cdot [\![W(k,t-1)]\!] \cdot q = [\![W(j,t-1)]\!]) \wedge (r < [\![W(k,t-1)]\!])$
$W[i] \leftarrow \lfloor W[j]/W[k] \rfloor$	$\exists q,r\colon \text{IsWord}(u) \wedge \text{IsWord}(l) \wedge ([\![W(i,t)]\!] = q) \wedge$ $(u \cdot [\![W(k,t-1)]\!] \cdot q = [\![W(j,t-1)]\!]) \wedge (r < [\![W(k,t-1)]\!])$
$W[i] \leftarrow W[j] \uparrow W[k]$	$\text{IsWord}([\![W(i,t)]\!]) \wedge \text{IsWord}([\![W(j,t-1)]\!]) \wedge \text{IsWord}([\![W(k,t-1)]\!]) \wedge$ $\bigwedge_{b=0}^{w-1} \big([\![W(i,t)]\!]_b = 1 - [\![W(j,t-1)]\!]_b \cdot [\![W(k,t-1)]\!]_b\big)$

Table 4. Encoding word arithmetic and boolean instructions as formulae, where "if A then B else C" is shorthand for $(A \wedge B) \vee (\neg A \wedge C)$.

Finally, the input to the optimal unknotted extension problem consists of a polygonal path P in $\mathbb{R}^3$ with integer vertex coordinates, along with an integer k; the problem asks whether P can be extended to an *unknotted* closed polygonal curve in $\mathbb{R}^3$ with at most k additional vertices. Like the two previous problems, this problem is trivial unless $k < n$. To verify an yes-instance of this problem, it suffices to guess the coordinates of k new vertices (using $3k$ real registers), and then check that the resulting closed polygonal curve is unknotted in *nondeterministic* polynomial time (using a polynomial number of additional word registers), either using the normal-surface algorithm of Hass *et al.* [58], or by projecting to a two-dimensional knot diagram and guessing and executing an unknotting sequence of Reidemeister moves [73]. ◀