Smoothing the gap between NP and ER

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@uu.nl

November 22, 2021

Abstract

We study algorithmic problems that belong to the complexity class of the existential theory of the reals ($\exists\mathbb{R}$). A problem is $\exists\mathbb{R}$-complete if it is as hard as the problem ETR and if it can be written as an ETR formula. Traditionally, these problems are studied in the real RAM, a model of computation that assumes that the storage and comparison of real-valued numbers can be done in constant space and time, with infinite precision. The complexity class $\exists\mathbb{R}$ is often called a real RAM analogue of NP, since the problem ETR can be viewed as the real-valued variant of SAT. The real RAM assumption that we can represent and compare arbitrary irrational values in constant space and time is not very realistic. Yet this assumption is vital, since some $\exists\mathbb{R}$-complete problems have an "exponential bit phenomenon" where there exists an input for the problem, such that the witness of the solution requires geometric coordinates which need exponential word size when represented in binary. The problems that exhibit this phenomenon are NP-hard (since ETR is NP-hard) but it is unknown if they lie in NP. NP membership is often showed by using the famous Cook-Levin theorem which states that the existence of a polynomial-time verification algorithm for the problem witness is equivalent to NP membership. The exponential bit phenomenon prohibits a straightforward application of the Cook-Levin theorem.

In this paper we first present a result which we believe to be of independent interest: we prove a real RAM analogue to the Cook-Levin theorem which shows that $\exists\mathbb{R}$ membership is equivalent to having a verification algorithm that runs in polynomial-time on a real RAM. This gives an easy proof of $\exists\mathbb{R}$-membership, as verification algorithms on a real RAM are much more versatile than ETR-formulas.

We use this result to construct a framework to study $\exists\mathbb{R}$-complete problems under smoothed analysis. We show that for a wide class of $\exists\mathbb{R}$-complete problems, its witness can be represented with logarithmic input-precision by using smoothed analysis on its real RAM verification algorithm. This shows in a formal way that the boundary between NP and $\exists\mathbb{R}$ (formed by inputs whose solution witness needs high input-precision) consists of contrived input.

We apply our framework to well-studied $\exists\mathbb{R}$-complete recognition problems which have the exponential bit phenomenon such as the recognition of realizable order types or the Steinitz problem in fixed dimension. Interestingly our techniques also generalize to problems with a natural notion of resource augmentation (geometric packing, the art gallery problem).

A prior version of this paper appeared in FOCS 2020 [29].

Acknowledgements. The second author is supported by the Netherlands Organisation for Scientific Research (NWO); 614.001.504. The third author is supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 016.Veni.192.25. We thank anonymous reviewers for valuable and detailed comments.

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 [32, 50, 63] where real values 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 [53] point out that "without this assumption it is virtually impossible to prove the correctness of any geometric algorithms."

Intuitively (for a formal definition, skip ahead to the bottom of page 2) we define the input-precision of a real RAM algorithm A with input I as the minimal word size required to express each input value in I, such that the algorithm executes the same operations in the word RAM as in the real RAM. The downside of algorithm analysis in the real RAM is that it neglects the input-precision required by the underlying algorithms for correct execution, although they are very important in practice. Many algorithms, including some verification algorithms of $\exists\mathbb{R}$-complete problems, inherently require large input-precision [63] and there are even examples which in the worst case require an input-precision exponential in the number of input variables in order to be correctly executed [36, 41].

Often inputs which require exponential input-precision are contrived and do not resemble *realistic* inputs. A natural way to capture this from a theoretical perspective is smoothed analysis, which *smoothly* interpolates between worst case analysis and average case analysis [72]. 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 the magnitude of the perturbation. By now smoothed analysis is well-established, for instance Spielman and Teng received the Gödel Prize for it.

We give a bird's eye view of the paper before providing proper definitions: we study the real RAM and its complexity class $\exists\mathbb{R}$ through the lens of smoothed analysis. We start by proving a result separate of smoothed analysis, which we believe to be of independent interest: we show a real RAM analogue of the famous Cook-Levin theorem as we prove that $\exists\mathbb{R}$ membership is equivalent to the existence of a verification algorithm that runs in polynomial-time on a real RAM. We then use the existence of this verification algorithm to construct a framework with which smoothed analysis can be applied to a wide class of $\exists\mathbb{R}$-complete problems. We show that $\exists\mathbb{R}$-complete recognition problems, with a verification algorithm in the real RAM that has polynomially bounded *arithmetic degree*, have witnesses that can be represented with a logarithmic word size under smoothed analysis. This implies that these problems have polynomial-time verification algorithms on the word RAM that succeed for all but a small set of contrived inputs. Finally, we extend our framework to include $\exists\mathbb{R}$-complete problems that have a natural notion of resource augmentation. This generalizes an earlier result of smoothed analysis of the Art Gallery problem [25].

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 [37] but previously considered by Fredman and Willard [33, 34] and even earlier by Kirkpatrick and Reisch [44]. 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.

Many portions of the algorithms community (either explicitly or implicitly) use a different variation of the RAM called the *real RAM*, where registers may contain arbitrary real numbers, instead of just integers; The usage of the real RAM is prevalent in the field of computational geometry but also in probabilistic algorithm analysis where one wants to reason about continuous perturbations of the input. The abstraction offered by the real RAM dramatically simplifies the design and analysis of algorithms, at the cost of working in a 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. Kettner [43] provides an overview of complications that arise from the unrealistic precision that the real RAM assumes.

Formally modeling real RAM algorithms. The real RAM has been the standard underlying model of computation in computational geometry since the field was founded in the late 1970s [59, 68]. Despite its ubiquity, we are 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 algorithm analysis literature. The obvious candidate for such a definition is the real-computation model proposed by Blum, Shub, and Smale [12, 13];

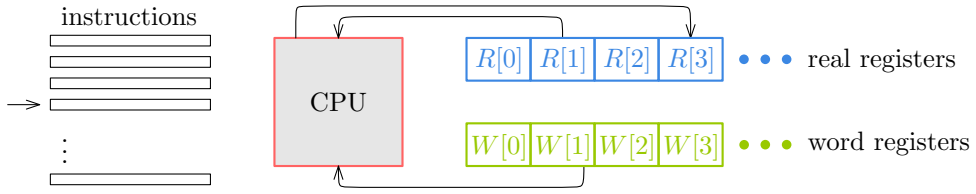

Figure 1: The dominant model in computational geometry is the real RAM. It consists of a central processing unit, which can operate on real and word registers in constant time, following a set of instructions.

however, this model does not support the integer operations necessary to implement even simple 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, and multidimensional array indexing and program slicing.

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 [67]; see also [10, 38, 44, 74] 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 [37] (refer to Figure 1). 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 *address* when it is used as an index into a memory array.)

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 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. 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. Refer to Section 6.1 for a table of the operations and further explanation of the real RAM. Crucially, we consider the real RAM without trigonometric, exponential and logarithmic operations. We do allow for the use of the square root operator for our results regarding $\exists\mathbb{R}$-completeness but not for our result on smoothed analysis.

A recent addition to the RAM models. Having detailed our proposal of RAM model, we wish to briefly mention a recent proposal made by Demaine, Hesterberg and Ku [23]. In this recent paper, the authors note similarly that the word RAM does not support operations with the appropriate degree of precision for algorithms in computational geometry. Our definition of the real RAM offers a way to theoretically model the theoretical computational power needed for constant-time real-valued computations and we obtain this model by extending the word RAM model. Demaine, Hesterberg and Ku propose their own extension of the word RAM model, which is theoretically less general but which has a more direct relation to real-life computation power.

The authors note that if a constant-size radical expression (this includes the addition, multiplication and division of square roots) depends on variables of b bits, then an algorithm can decide if the expression is

[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.

greater than 0 in $O(b)$ time. Following this observation, the authors assume a word RAM, which allows memory cells to not only store expressions of values with wordsize b, but these constant-size radical expressions. This makes their model more general than a traditional word RAM. This generalization is useful, as for example geometric operations or algorithms that require the computation of Euclidean distance are hereby supported.

We want to mention that their model of computation is not directly applicable to our results for smoothed analysis or $\exists\mathbb{R}$-completeness, as both problems per definition require a way to model constant-time arithmetic on arbitrary reals. However we found it important to note that there are recent discussions on how and when radical expressions can be allowed in RAM computations.

Formally defining bit-precision and input-precision. We say two inputs $I = (a, b) \in \mathbb{R}^n \times \mathbb{Z}^m$ and $I' = (a', b) \in \mathbb{R}^n \times \mathbb{Z}^m$ are equivalent with respect to an algorithm A on a real RAM, ($I \cong_A I'$) if every comparison operation gives the same result. In particular, this implies that at every step the program counter is at the same position. For each integer $z \in \mathbb{Z}$, we denote by $\mathtt{bit}(z)$ the length of its binary representation. For each rational number $y = p/q \in \mathbb{Q}$, with p, q irreducible, we denote by $\mathtt{bit}(y) := \mathtt{bit}(p) + \mathtt{bit}(q)$ the length of its binary representation.

First we consider $I = (a, b) \in \mathbb{Q}^n \times \mathbb{Z}^m$ as input for a real RAM algorithm A. We denote by $\mathtt{bit}_{\mathtt{IN}}(I) = \max_i \max\{\mathtt{bit}(a_i), \mathtt{bit}(b_i)\}$ the *input bit-length* of I. We denote by $C(I)$ the set of all values of all registers during the execution of A with I, and by $\mathtt{bit}(C(I)) = \max_{c \in C(I)} \{\mathtt{bit}(c)\}$ as the *execution bit-length* of I.

Now, we are ready to define the bit-precision and input-precision of *real* input. The bit-precision of A with input $I = (a, b) \in \mathbb{R}^n \times \mathbb{Z}^m$ is:

$$\mathtt{bit}(I, A) := \min\{\mathtt{bit}(C(I')) \mid I' \in \mathbb{Q}^n \times \mathbb{Z}^m, I \cong_A I'\}.$$

In the same manner, the input-precision is defined as :

$$\mathtt{bit}_{\mathtt{IN}}(I, A) := \min\{\mathtt{bit}_{\mathtt{IN}}(I') \mid I' \in \mathbb{Q}^n \times \mathbb{Z}^m, I \cong_A I'\}.$$

If there is no equivalent input $I' = (a, b) \in \mathbb{Q}^n \times \mathbb{Z}^m$, then we say $\mathtt{bit}(I, A) = \mathtt{bit}_{\mathtt{IN}}(I, A) = \infty$. It is now straightforward to *simulate* an execution of a real RAM algorithm A on input $I = (a, b) \in \mathbb{R}^n \times \mathbb{Z}^m$ on a word RAM with word size $w = O(\mathtt{bit}(I, A))$.

Arithmetic degree. Liotta, Preparata and Tamassia [51] studied the required bit-precision for real RAM proximity algorithms. To aid their analysis, they defined the *arithmetic degree* of an algorithm. We express their definition in our real RAM model and add the notion of algebraic dimension for our later analysis. It follows from our definition of real RAM and our list of operations in Table 1 that 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. Observe that adding, subtracting, or multiplying two rational functions yields another rational function, possibly of higher degree; for example, $\frac{p_1}{q_1} + \frac{p_2}{q_2} = \frac{p_1 q_2 + p_2 q_1}{q_1 q_2}$. We say an algorithm A has *arithmetic 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 is always at most d. Finally, we define the *max (integer) coefficient* c, if the absolute value of the largest coefficient of p and q is always bounded from above by c. Bounded algebraic dimension, arithmetic degree and max coefficient give an interesting relation between the input-precision and the bit-precision:

Lemma 1. *Let $I \in (a, b) \in \mathbb{R}^n \times \mathbb{Z}^m$ be some input and A be a real RAM algorithm with $\mathtt{bit}_{\mathtt{IN}}(I, A) = p$, algebraic dimension d, arithmetic degree Δ and max coefficient c. Then its bit-precision is bounded from above by $O(p\Delta^2 \log d \log c)$.*

Proof. If we multiply a number of bit-length b with a number c then the resulting number has bit complexity at most $O(b \log c)$. If we multiply Δ numbers of bit-length p, the total bit-length is at most Δp. If we sum t numbers of bit-length p, the total bit-length is at most $O(p \log t)$. In a polynomial in d variables, we have at most $O(d^\Delta)$ monomials. Thus the bit-precision is bounded from above by $O(\log c \log(d^\Delta)\Delta p) = O(p\Delta^2 \log d \log c)$. □

This allows us to only focus on input-precision in our smoothed analysis which we explain next:

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. (that is, we consider out of all inputs the input that is least-beneficial to perturb from) Intuitively, smoothed analysis interpolates between average case and worst case analysis (Figure 2). The smaller δ, the closer we are to true worst case input. Correspondingly larger δ

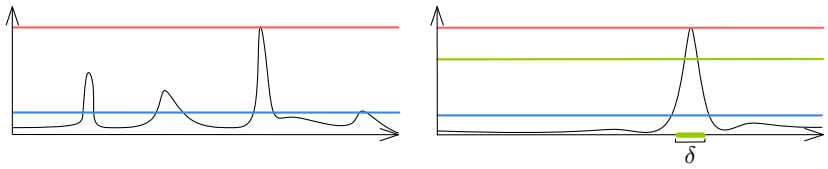

Figure 2: The x-axis represents all inputs for the algorithmic problem. The y-axis the associated cost for every input. The red line indicates the worst case cost. The blue line average cost. If we pick a parameter δ and a point a on the x-axis, then all permutations from a of at most δ form some area centered around a. We can compute the average cost of points in this area. The cost under smoothed analysis is the maximum over all points a, of this average cost and is shown by the green line.

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. 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 [45]. Since the introduction of smoothed analysis, it has been applied to numerous problems [5, 7, 27, 30, 58]. Most relevant for us is the smoothed analysis of the art gallery problem [25] and of order types [73], which we generalize in Section 4.

Formally we define smoothed analysis as follows: let us fix an algorithm A and some $\delta \in [0,1]$, which describes the *magnitude of perturbation*. We denote by $I = (a,b) \in [0,1]^n \times \mathbb{Z}^m$ the input of A. We scale the real-valued input to lie in $[0,1]$, to normalize δ. Unless explicitly stated, we assume that each real number is perturbed *independently uniformly at random* (each real number receives an offset sampled uniformly at random from $[\frac{-\delta}{2}, \frac{\delta}{2}]$). We assume that the integer input stays as it is, since we assume that it already fits into main memory. 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 $\mathscr{C}(\cdot)$ an arbitrary cost function and by $\mathscr{C}(I_x)$ the cost of instance I_x. Traditionally in smoothed analysis, this cost is the runtime required for an algorithm in order to compute its solution but in this paper we consider the input-precision and the bit-precision as the cost functions. The expected cost of instance I equals:

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

We denote by $\Lambda_{n,m}$ the set of all instances in $[0,1]^n \times \mathbb{Z}^m$. Henceforth, we implicitly assume that for all integer values b_i, $\mathtt{bit}(b_i) \leq \log\ m$ and $m = O(n)$. We drop the m from all future equations and consideration. The smoothed input-precision equals:

$$\mathscr{C}_{\text{smooth}}(\delta, n, A) = \max_{I \in \Lambda_{n,m}} \mathscr{C}_\delta(I,A).$$

This definition formalizes the intuition mentioned before: not only do we require that the majority of instances behave nicely, but actually in every neighborhood (bounded by the maximal perturbation δ) the majority of instances must behave nicely. Following [22, 72] we say an algorithm runs in polynomial cost in practice, if the smoothed cost of the algorithm is polynomial in the input size n and in $1/\delta$. If the smoothed cost is small in terms of $1/\delta$ then we have a theoretical verification of the hypothesis that worst case examples are sparse. We defined the input-precision of an algorithm A with input I as the minimum word size required for the correct execution of the algorithm on a word RAM. To model possible disparities between real RAM and word RAM execution we define an operation called *snapping* in the next paragraph.

Snapping. Our real-valued input can be represented as a higher-dimensional point $a \in [0,1]^n$. If we want to express a with only w bits, then the corresponding integer-valued input a' with limited precision is the closest point to a in the scaled integer lattice $\Gamma_\omega = \omega\mathbb{Z}^d$ with $\omega = 2^{-w}$. We call the transformation of a into a' *snapping*. We give a lower bound on the scale factor ω for which $(a,b) \cong_A (a',b)$. From this point on, whenever we refer to bounding from above the required input-precision, we refer to a lower bound on the scale factor ω such that $(a,b) \cong_A (a',b)$. Note that this lower bound, implies an bound from above on the input-precision of A. That is, ω implies a bound from above on the number of bits needed to simulate the execution of an algorithm A with input (a,b) on the word RAM. Naturally, it could be that there is a better way to represent or round (a,b) to a value (a'',b) than our snapping procedure. However, we show that our upper bound is logarithmic in the size of the real input n. A natural lower bound for the input-precision is $\Omega(\log n)$ and thus our future results can be considered tight. The algorithms that we study under this snapping operation are algorithms relevant to the complexity class $\exists\mathbb{R}$, which we discuss next.

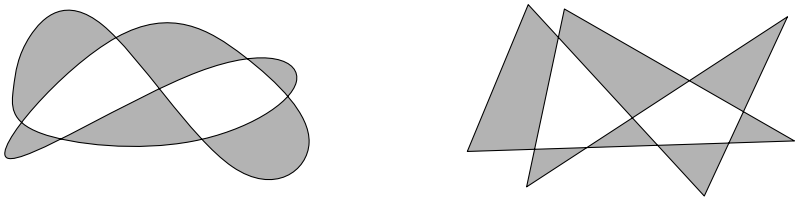

Figure 3: Given two simple closed curves in the plane it is straightforward to design an algorithm that checks if the two curves are equivalent. But it is not straightforward to describe an ETR formula for it.

The Existential Theory of the Reals. It is often sometimes to describe a real-valued witness to an NP-hard problem, but the input-precision required to verify the witness is unknown. A concrete example is the recognition of segment intersection graphs: given a graph, can we represent it as the intersection graph of segments? Matoušek [54] 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 word size in any numerical representation. This *exponential bit phenomenon* occurs not only for segment intersection graphs, but also for many other natural algorithmic problems [1–4, 11, 15, 16, 24, 26, 28, 35, 42, 46, 52, 55–57, 61, 64–66, 69–71]. It turns out that all of those algorithmic problems do not accidentally require exponential input-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 input-precision phenomenon only occurs for near-degenerate input.

The complexity class $\exists\mathbb{R}$ can be defined as the set of decision problems that can be reduced in polynomial time on a word RAM 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 the ETR formula Ψ is true if there are real numbers $x_1, \ldots, x_n \in \mathbb{R}$ such that $\Phi(x_1, \ldots, x_n)$ is true.

Paper structure. In Section 2, we give a structural result that gives an alternative definition of $\exists\mathbb{R}$ in terms of real RAM algorithms. The corresponding proofs can be found in Section 6. In Section 3, we apply smoothed analysis, first to the real RAM and then to recognition problems. The corresponding proofs can be found in Section 7 and Section 8. In Section 4, we apply smoothed analysis to resource augmentation problems. The corresponding proofs can be found in Section 9. In Section 5 we discuss the results in a broader context. Specifically, our results justify the usage of the real RAM.

2 Algorithmic membership in $\exists\mathbb{R}$

The complexity class $\exists\mathbb{R}$ is often called a "real analogue" of NP, because it deals with real-valued variables instead of Boolean variables. This is because the real-valued ETR problem plays a role which is analogous to SAT when it comes to hardness and membership. However, the most common way to think about NP-membership is in terms of certificates and verification algorithms.

The seminal theorem of Cook and Levin shows the equivalence of the two perspectives on NP-membership, see [19,49]. We show a similar equivalence between ETR-formulas and *real verification algorithms*. Intuitively, a real verification algorithm is an algorithm that runs on the real RAM that accepts as input both an *integer* instance I and a witness, and verifies that the witness describes a valid solution to the instance in polynomial-time. Note that as we define the real RAM as an extension of the word RAM, it is also possible to guess and operate with integers.

Formally we say a *discrete decision problem* is any function Q from arbitrary integer vectors to the booleans $\{\text{TRUE}, \text{FALSE}\}$. An integer vector I is called a *yes-instance* of Q if $Q(I) = \text{TRUE}$ and a *no-instance* of Q if $Q(I) = \text{FALSE}$. Let $\circ$ denote the concatenation operator. A *real verification algorithm* for Q is formally defined as a real RAM algorithm A that satisfies the following conditions, for some constant $c \geq 1$: (1) A halts after at most N^c time steps given any input of total length N where each value uses word size $\lceil c \log_2 N \rceil$. (2) 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 $(x, I \circ z)$ and (3) 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 $(x, I \circ z)$. We show the following theorem:

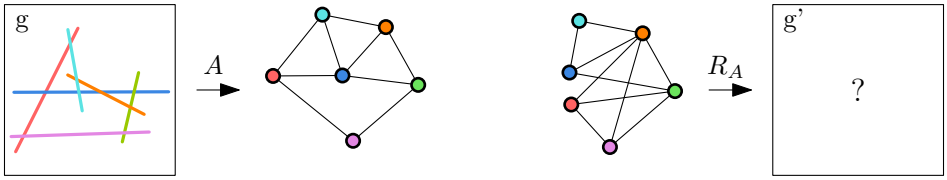

Figure 4: Left: given a set of segments g, algorithm A constructs segment intersection graph $c = A(g)$. Right: given a graph c', algorithm R_A searches for a set of segments g' such that $A(g') = c'$.

Theorem 2. *For any discrete decision problem Q, there is a real verification algorithm for $Q \Leftrightarrow Q \in \exists\mathbb{R}$.*

Our proof in Section 6 follows classical simulation arguments reducing nondeterministic polynomial-time (integer) random access machines to polynomial-size circuits or Boolean formulas, either directly [62] or via nondeterministic polynomial-time Turing machines [17–19,74]. We wish to state that the analysis in Section 6 is broad enough to support the square root operator in the real RAM.

The complexity class $\exists\mathbb{R}$ 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 [12,13]. However as we explained in the introduction, 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 [59, 68] *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 2. We rectify this gap with our precise definition of the real RAM in the previous section together with a table of operations presented in Section 6.1. Our proposal generalizes both the word RAM and BSS models, we believe it formalizes the intuitive model implicitly assumed by computational geometers.

Theorem 2 not only strengthens the intuitive analogy between NP and $\exists\mathbb{R}$, but also enables much simpler proofs of $\exists\mathbb{R}$-membership in terms of standard geometric algorithms. Our motivation for developing Theorem 2 was Erickson's *optimal curve straightening* problem [28]: Given a closed curve γ in the plane (Given as a plane 4-regular graph) and an integer k, is any k-vertex polygon topologically equivalent to γ? (See Figure 3.) The $\exists\mathbb{R}$-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 2, membership in $\exists\mathbb{R}$ follows almost immediately from the standard Bentley-Ottman sweep-line algorithm [8]. The theorem also applies to geometric packing problems [4], where the input is a set of geometric objects and a container and the output is a pairwise disjoint placement of the objects in the container. Its real verification algorithm is straightforward. To further illustrate the power of our technique, we also consider a new topological problem in Section 6, 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? Note that the path P only becomes a closed curve after we added the additional vertices. In light of Theorem 2, 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 polygon is unknotted using existing NP algorithms [39, 48].

Corollary 3. *The following discrete decision problems are in $\exists\mathbb{R}$: The art gallery problem [2], the optimal curve straightening problem [28], geometric packing, and the optimal unknotted extension problem.*

Let us remark that researchers are sometimes using the sine or cosine functions as elementary operations of the real RAM. So one may wonder whether, it is possible to extent the definition of the real RAM to include those functions as well. We want to point out that the real RAM would then easily be powerful enough to decide undecidable problems, by Richardson's Theorem [60]. Therefore, we refrain from using those functions for the real RAM.

3 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. [14]. In the paper they first derive a set of properties for disk intersection graphs and then they use *only* those properties to find a new EPTAS. More classical results include methods for robust computations

(e.g. computing the determinant for an orientation test, comparing a quadtratic polynomial for an in-circle test). Refer to a recent set of examples by Kettner et al. [43] or results on Exact Geometric Computation (EGC) such as the results by Yap [75].

The above motivates that sometimes it is nice to reason about geometric problems based on only the combinatorial properties of the geometric configuration. A related question, is whether an arbitrary combinatorial set, has an accompanying geometrical configuration. This question is the crux of a geometric recognition problem.

Formally (Figure 4), we say A is a *polynomial-time recognition verification algorithm* if it takes some real-valued geometric input $g \in [0,1]^n$ and outputs a combinatorial object $A(g) \in \mathbb{Z}^m$ in time polynomial in n (note that this implies that m is polynomial in n).

Note that there is no need to give a witness for a negative answer. Although this maybe at first confusing, recall that NP algorithms are only required to give a witness for positive answers. We define a *recognition problem* as a discrete decision problem R_A that takes a combinatorial object c as input, and returns TRUE if there exists a vector $g \in [0,1]^n$ (which we shall call a *witness*) for which $A(g) = c$. For notational convenience, we denote for a given c by $R_A(c)$ *an arbitrary* witness of c (as in the smoothed analysis that is to come, we consider the worst choice over all witnesses for c).

Total number of polynomials. When we consider our applications of smoothed analysis to recognition problems, we can observe that regardless of the input or output, the total number of polynomials that may ever be evaluated by an algorithm that constructs the output is polynomially bounded in the number of (real) input variables. To illustrate this, consider order type realizability. Given a set of n points, an algorithm may need to check the order type of some triple of points. This may be done through verifying the sign of the determinant associated with that triple of points. Since there are at most $O(n^3)$ different triples of points, there are at most $O(n^3)$ different polynomials that may be evaluated by an algorithm that identifies the order type of a point set. A similar example exists for computing the intersection graph for objects in the plane. For instance for unit disk intersection graphs, where it is sufficient to check for intersections between any pair of disks. Since there are at most $O(n^2)$ such pairs, any algorithm that identifies the disk intersection graph can evaluate only polynomials from a set of at most $O(n^2)$ polynomials. For a given algorithm where the input consists of n reals, we denote by $C(n)$ the total number of possible polynomials that may be checked by the algorithm. We say that an algorithm has a *few polynomials* if the total number of polynomials $C(n) \leq n^{O(1)}$ (note that if some algorithm has running time $T(n)$ then $C(n) \leq 2^{T(n)}$. This upper bound comes from the fact that the algorithm branches at most $T(n)$ times.)

In a previous version [29] of this article, we erroneously claimed that $C(n)$ is bounded from above by $T(n)$. To see an illustrative counter-example, consider the even-interval problem. In the even-interval problem we are given a real number $x \in [0,1]$ and $k \in \mathbb{N}$ given in unary. We ask if x is contained in one of the intervals $I_i = [\frac{2i}{k}, \frac{2i+1}{k}]$, for some i. For this problem it makes sense to measure the length of the input as the number of bits of k. This problem can easily be solved with binary search in $O(\log k)$ steps where at every step we check one polynomial that verifies if x lies in some interval I_i. Yet, depending on the value of x, the set of intervals I_i that we need to check wildly varies and is roughly $O(k)$. Hence $C(k) = O(k)$ and thus not bounded from above by the running time $O(\log k)$. Note that all of those polynomials have arithmetic degree and algebraic dimension equal to one. Specifically, the example shows that Theorem 4 is not true, without the assumption that there are few polynomials.

Defining smoothed analysis on recognition problems. Traditionally in smoothed analysis one perturbs the input of an algorithm and measures the expected cost of executing the algorithm with the new input. The real-valued geometric input g offers a straightforward way to perturb it by adding to each $g_i \in [0,1]^n$ a random offset $x_i \in [\frac{-\delta}{2}, \frac{\delta}{2}]$ chosen uniformly at random. It is not as easy to define a perturbation on the combinatorial (discrete) input that a recognition problem requires. This is why we define the perturbation in terms of the geometric witness: given any input and witness (c, g), we slightly perturb the witness g to a new geometric object $g_x = g + x$ and reconstruct the corresponding combinatorial input c_x (Figure 5). A probability distribution over possible output g_x, combined with a recognition algorithm A, implies a probability distribution over all input c_x.

We want for an instance c of R_A, to find a witness g of bounded bit-length. Note that the witness $g \in [0,1]^n \times \mathbb{Z}^0$. The witness g is thus a valid input for a real RAM program with input size n and $m = 0$. Our definition is formulated in terms of the input-precision of A:

$$\mathtt{bit}_\delta(g,A) := \mathop{\mathbb{E}}_{x \in \Omega_\delta} [\mathtt{bit}_{IN}(g+x,A)],$$

$$\mathtt{bit}_\delta(c,R_A) := \mathop{\mathbb{E}}_{x \in \Omega_\delta} [\mathtt{bit}_{IN}(g_x = R_A(c)+x,A)].$$

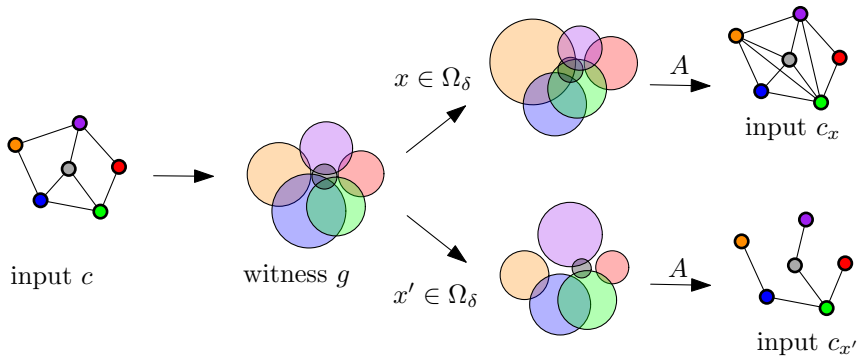

Figure 5: We define a perturbation on a combinatorial structure c through a witness g. A distribution of witnesses defines a distribution of discrete structures.

Note that the bit-complexity of c is zero in case that there is no witness. Similarly to an NP-algorithm running in constant time for a no-instance.

Next, we define the smoothed precision. We denote by Λ_m the set of all combinatorial instances c of size m (that is, all combinatorial instances c that can be described by a vector $b \in \mathbb{Z}^m$):

$$\mathtt{bit}_{\mathrm{smooth}}(\delta,n,A) = \max_{g\in[0,1]^n} \mathtt{bit}_\delta(g,A), \quad \mathtt{bit}_{\mathrm{smooth}}(\delta,m,R_A) = \max_{c\in\Lambda_m} \mathtt{bit}_\delta(c,R_A).$$

Now, we are ready to state our main theorem about smoothed analysis of recognition verification algorithms, whose proof can be found in Section 8.

Theorem 4. *Let A be a polynomial-time recognition verification algorithm with arithmetic degree Δ, algebraic dimension d and few polynomials. Under perturbations of $g \in [0,1]^n$ by $x \in \left[\frac{\delta}{2},\frac{\delta}{2}\right]^n$ chosen uniformly at random, the algorithm A has a smooth input-precision of at most:*

$$\mathtt{bit}_{smooth}(\delta,n,A) = O\left(d \log \frac{d\Delta n}{\delta}\right).$$

We stated our results only in terms of expected value, when in fact our proof also readily implies the same statement with high probability. We do this in order to follow the tradition in smoothed analysis that focuses more on expectations. Note that statements *with high probability* and *in expectation* can be linked with Chebyshev's inequality. Statements *with high probability* do usually imply statements about *expectations*, by some standard arguments.

The core idea of the proof of Theorem 4 (presented at the end of this section and illustrated by Figure 6) is to consider the recognition verification algorithm A with perturbed input $g_x = g + x$, where g is an arbitrary value in $[0,1]^n$ and x is a small perturbation chosen uniformly at random in $\left[\frac{-\delta}{2},\frac{\delta}{2}\right]^n$. We model the perturbed input g_x as a high-dimensional point which we snap to a fine grid to obtain g' (input which can be described using bounded precision). We then show that for any algorithm A that meets our prerequisites, $\mathtt{bit}_{IN}(g_x,A)$ is low with high probability. For the snapping we consider a sufficiently small ω and we snap the point g_x to a point in $\omega\mathbb{Z}^n$. The Voronoi diagram of the points in $\omega\mathbb{Z}^n$ forms a fine grid in $[0,1]^n$. As we explained in the introduction, 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 the point g_x lies in a Voronoi cell of a point with limited word size which does not intersect the variety of either of the two polynomials, then the comparison instruction will be computed correctly. We bound from above the proportion of Voronoi cells that are intersected by the variety of a polynomial in Theorem 7 in Section 7.

The algebraic proof of Theorem 7 has been placed in Section 7 to not distract from the main story line. By our assumption A has few polynomials. The union bound over the total number of polynomials bounds from above the chance that the execution of A differs for the input g' and g_x. The union bound will show that with high probability, for any perturbed input g_x, $\mathtt{bit}_{IN}(g_x,A)$ is low. The complete proof of Theorem 4 can be found in Section 8. Theorem 4 implies a result of smoothed analysis of recognition problems:

Theorem 5. *Let R_A be a recognition problem with recognition verification algorithm A. If A is a polynomial-time algorithm with constant arithmetic degree, algebraic dimension and few polynomials then*

$$\mathtt{bit}_{smooth}(\delta,m,R_A) = O(\log(m/\delta)).$$

Proof. This can be shown simply by using the definition and the result of Theorem 4.

$$\begin{aligned}\texttt{bit}_{\text{smooth}}(\delta,m,R_A) &= \max_{c\in\Lambda_m} \mathop{\mathbb{E}}_{x\in\Omega_\delta}[\texttt{bit}_{IN}(g_x,A)\mid g_x = R_A(c)+x] \\ &= \max_{g:A(g)\in\Lambda_m} \mathop{\mathbb{E}}_{x\in\Omega_\delta}[\texttt{bit}_{IN}(g_x,A)\mid g_x = g+x]\end{aligned}$$

By definition of a recognition verification algorithm, if $c = A(g)$ has size m then g has size $n = \Theta(m^{const})$ for some fixed constant $const > 0$. Thus Theorem 4 implies:

$$\begin{aligned}\texttt{bit}_{\text{smooth}}(\delta,m,R_A) &\le \max_{g\in[0,1]^n} \texttt{bit}_\delta(g,A) \\ &= \texttt{bit}_{\text{smooth}}(\delta,n,A) \\ &= O(\log(n/\delta)) = O(\log(m^{const}/\delta)) = O(\log(m/\delta)).\end{aligned}$$

□

Corollary 6. *The following recognition problems under uniform perturbations of the witness of magnitude δ have a smoothed bit-precision of $O(\log n/\delta)$: recognition of realizable order types [73], disk intersection graphs, segment intersection graphs, ray intersection graphs and the Steinitz Problem in fixed dimension.*

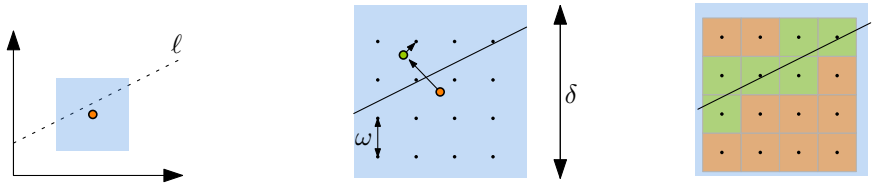

Figure 6: Given $g = (g_1, g_2) \in [0,1]^2$, we want to decide if the point g (in orange) lies above or below the line ℓ ($y = x/2+1$). If g is perturbed slightly to a point g_x, low precision is usually sufficient.

Limitations. We want to point out that we cannot handle all recognition problems. First, not all recognition verification algorithms meet the conditions of Theorem 4. For example, consider the case that the problem deals with unbounded dimension. We still get some bounds on the input-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. Second, perturbing a witness may not be sensible. 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. For example, this limitation 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.

Preliminaries for proving Theorem 4. Per definition, the word RAM has a word size w which allows us to express $2^w = \frac{1}{\omega}$ different values for each coordinate. We consider the recognition verification algorithm A with real-valued input $g \in [0,1]^n$. Since A runs in polynomial-time it can make at most a polynomial number of comparison operations (Table 1). At every such binary decision the algorithm looks at a real- or integer-value register and verifies if the value at the register is 0 or strictly greater than 0. For every real-valued register, per assumption the value at that register is the quotient of two d-variate polynomials p_i and q_i with maximum degree Δ whose variables depend only on the values in the input register. Let z be the d-dimensional vector of input variables in g that p_i and q_i depend on.

The evaluation of $p_i(z)/q_i(z)$ depends on the evaluation of the polynomials $p_i(z)$ and $q_i(z)$. During smoothed analysis we perturb our input g into new input $g_x = (g + x)$ with x a value chosen uniformly at random in $[\frac{-\delta}{2}, \frac{\delta}{2}]^m$. Thereafter, we snap g_x to g' and we are interested in the probability that the execution of A under both inputs (g_x and g') is the same, ergo the chance that for all comparison operations, the algorithms give the same answer.

Fix a vector $z = (z_1, \ldots, z_d) \in \mathbb{Z}^d$ with integer coordinates. We denote by C_z the unit (hyper)cube, which has z as its minimal corner: $C_z := [0,1]^d + z$. We denote by $C(d,k) := \{C_z \mid z \in \mathbb{Z}^d \cap [0,k)^d\}$ a $(k \times k \times \ldots \times k)$-grid of unit cubes that cover $[0,k]^d$. Let $C = [0,k]^d$ be a cube partitioned by unit cubes $C(d,k)$. Every facet

of C intuitively is a grid of $(d-1)$-dimensional cubes $C(d-1,k)$. We argue about varieties that intersect cubes in $C(d,k)$ by induction on the dimension.

To that end, we define an equivalence relation on these sets of cubes. Let C be a d-dimensional cube, partitioned by d-dimensional cubes 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 cubes in $\tau(C)$ and $C(d,k)$ where corresponding cubes coincide. We give two examples of this equivalence relation that is often used in the remainder of the paper: (1) Consider any d,k and any orthogonal $(d-1)$-dimensional, hyperplane H that intersects a d-dimensional cube $C(d,k)$ we have that $C(d,k) \cap H \cong C(d-1,k)$. (2) Consider the d-dimensional grid $\Gamma_\omega = \{C(y) : y \in \omega\mathbb{Z}^d \cap [0,1]^d\} \cong C(d,1/\omega)$, and a cube C which has one corner on the origin and width $k\omega$. The intersection $C \cap \Gamma_\omega$ is equivalent to $C(d,k)$. Using these definitions and a well-known theorem by Milnor [6] we obtain in Section 7 Theorem 7. See Theorem 7.23 in the printed second edition and Theorem 7.25 in the online version [6].

Theorem 7. *[Hitting Cubes, Section 7] Let $p \neq 0$ be a d-variate polynomial with maximum degree Δ and let $k \geq 2\Delta+2$. Then the variety $V(p)$ of p intersects at most $k^{d-1}\Delta 3^d(d+1)!$ unit cubes in $C(d,k)$.*

4 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 slightly weaker problem constraints. This alternative approach has received considerably less attention in theoretical computer science. We want to emphasize that resource augmentation algorithms find a solution which does not compromise the optimality, in the sense that it gives the optimal solution to the augmented problem. Using smoothed analysis, we argue that in practice some $\exists\mathbb{R}$-complete optimization problems have an optimal solution that can be represented with a logarithmic word size by applying smoothed analysis to the augmentation parameter. The proofs are deferred to Section 9.

For the art gallery problem, Dobbins, Holmsen and Miltzow [25] showed augmenting the polygon by so-called edge-inflations, makes guarding the polygon easier. This leads to small expected input-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.

Definition. Let us fix an algorithmic optimization problem P with real verification algorithm A. For it to be a *resource-augmentation* problem, we require three specific conditions: First, each input consists of an $I \in [0,1]^n \times \mathbb{Z}^m$ together with an *augmentation-parameter* $\alpha \in [0,1]$. Secondly, we assume that there is an implicitly defined *solution space* $\chi_I[\alpha] = \chi[\alpha]$ where each element in $\chi[\alpha]$ is considered a solution for the problem P (with input I and augmented by α) which does not have to be optimal. We require that for each α, α' with $\alpha < \alpha'$ it holds that $\chi[\alpha] \subset \chi[\alpha']$. For example, in the art gallery problem I is the real-valued vertices of a simple polygon and an augmentation can be an inflation of the polygon by α (see [25]). The set $\chi_I[\alpha]$ is the set of all guard placements which guard the inflated polygon. Thirdly, we assume that there exists an evaluation function $f : \chi[1] \to \mathbb{N}$. The aim, given α, is to find a solution $x^* \in \chi[\alpha]$ for which $f(x^*)$ is the maximum or minimum denoted by $\text{opt}(\chi[\alpha])$. Henceforth, for notational convenience, we assume that P is a maximization problem.

Smoothed analysis of resource augmentation. For any x, we intuitively consider the minimal word size required for each coordinate in x such that the verification algorithm A of P can verify if $x \in \chi_I[\alpha]$ on the word RAM. We denote by $\texttt{bit}(\chi_I[\alpha]) = \min\{\texttt{bit}_{IN}(x,A) \mid f(x) = \text{opt}(\chi_I[\alpha]), x \in \chi_I[\alpha]\}$ the minimal precision needed to express an optimal solution in the solution space $\chi_I[\alpha]$. For the smoothed analysis of resource augmentation, we choose α uniform at random in $[0,\delta]$ (since we assume that a negative augmentation is not well-defined). Just as in the previous section, we first define the expected cost of a given input: $\texttt{bit}_\delta(I,P) = \mathop{\mathbb{E}}_{\alpha\in\Omega_\delta}[\texttt{bit}(\chi_I[\alpha])]$ and the smoothed cost:

$$\texttt{bit}_{\text{smooth}}(\delta,n,m,P) = \max_{I\in[0,1]^n\times\mathbb{Z}^m} \texttt{bit}_\delta(I,P).$$

In this paper, we study resource augmentation problems with three additional but natural properties:

- 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 this property implies that in a more augmented version of the problem, the optimum is at least as good.

- We define a breakpoint as an $\alpha \in [0,1]$ such that $\forall \varepsilon > 0,\ \mathrm{opt}(\chi_I[\alpha - \varepsilon]) \neq \mathrm{opt}(\chi_I[\alpha + \varepsilon])$. Then the *moderate* property requires that there are at most $n^{O(1)}$ breakpoints, for all inputs.
- 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 input-precision with respect to its real verification algorithm of $\leq c \log(n/\varepsilon)$, for some $c \in \mathbb{N}$, and $f(x) \leq f(x')$ (recall that we assumed that P is a maximization problem. Else the statement is symmetric). Ergo, given some x we can increase the augmentation parameter by ε, to obtain an equally good solution $x' \in \chi_I[\alpha + \varepsilon]$. Furthermore, x' has low input-precision. Note that x' is not necessarily optimal for $\chi_I[\alpha + \varepsilon]$.

We apply smoothed analysis to resource-augmentation problems with these three properties by choosing uniformly at random the augmentation $\alpha \in [0, \delta]$. In Section 9 we prove the following theorem:

Theorem 8. *Let P be a resource augmentation problem that is monotonous, moderate and smoothable and $m \leq O(n^c)$ then $\mathtt{bit}_{smooth}(\delta, n, m, P)$ is bounded from above by $O(\log(n/\delta))$.*

Implications of Theorem 8. To illustrate the applicability of our findings, we give the following two corollaries. The first result was already shown in [25]. 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 [4]. Our results imply that apart from near-degenerate conditions the solutions to these problems have logarithmic input-precision.

Corollary 9. *Under perturbations of the augmentation of magnitude δ, the following problems have an optimal solution with an expected input-precision of $O(\log(n/\delta))$: (1) the art gallery problem under perturbation of edge inflation [25], (2) packing polygonal objects into a square container under perturbation of the container side-length.*

5 Discussion

Here, we discuss the implications of our results to the real RAM and to the gap between $\exists\mathbb{R}$ and NP.

The real RAM In Section 6, we define formally a model of the real RAM. This model corresponds to the model that is intuitively used by researchers in computational geometry for decades. Yet, it is still conceivable that one can design a polynomial time algorithm for SAT on the real RAM, even if P $\neq$ NP. Yet, this paper gives several arguments that support the intuition that those pitfalls will not happen. Although, the real RAM was invented purely in order to avoid to deal with cumbersome precision issues, its role in Theorem 2 is completely different. In particular, suddenly all algorithms analyzed on the real RAM help us now to establish $\exists\mathbb{R}$-membership, exactly because they do allow real inputs. Furthermore, assume that we have an algorithm SAT-SOLVE that can determine in polynomial time on a real RAM if a given SAT formula is true, then Theorem 2 implies co-NP $\subseteq \exists\mathbb{R}$. In addition, if there were an algorithm that would solve true quantified Boolean formulas (TQBF) in polynomial time on a real RAM then Theorem 2 would even imply $\exists\mathbb{R} =$ PSPACE. Still, it seems difficult to rule out a polynomial time algorithm for SAT on the real RAM, as we can also not rule out a SAT algorithm on the word RAM.

Smoothing the gap between NP and $\exists\mathbb{R}$. In Section 2 we strengthened the intuitive analogy between NP and $\exists\mathbb{R}$ by showing that for both of them membership is equivalent to the existence of a polynomial-time verification algorithm in their respective RAM. It is well-known that NP $\subseteq \exists\mathbb{R}$, but it is unknown if the two complexity classes are the same. The gap between the two complexity classes could be formed by inputs for $\exists\mathbb{R}$-complete problems, for which the witness cannot (to the best of our knowledge) be represented using polynomial input-precision. In Section 3 and Section 9 we show that for a wide class of $\exists\mathbb{R}$-complete problems that have such an "exponential input-precision phenomenon" their witness can, under smoothed analysis, be presented using logarithmic input-precision. Thus, the gap between $\exists\mathbb{R}$ and NP (if it exists) is formed by near-degenerate input.

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

This section is devoted to proving the following theorem:

Theorem 2. *For any discrete decision problem Q, there is a real verification algorithm for $Q \iff Q \in \exists\mathbb{R}$.*

To this end, we finish the formalization of the real RAM that we started in the Introduction.

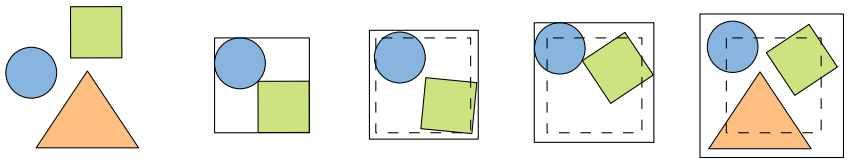

Figure 7: We augment the container from left to right. This extra space can lead to a better solution. If the optimal solution does not change, the extra space allows for a solution with low input-precision.

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.

6.1 What is the Real RAM?

We explained in the introduction that the input to a real RAM algorithm 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 algorithm begins. To maintain uniformity, we require that neither the input sizes n and m nor the word size w is known to any algorithm at "compile time". The output of a real RAM algorithm is the contents of memory when the algorithm executes the halt instruction.

Following Fredman and Willard [33, 34] 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 algorithm is the number of instructions executed before its 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.

6.2 Proofs of Algorithmic Membership

This section is dedicated to prove Theorem 2. 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 10. *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 $n + O(1)$ (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 11. *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 $w = \lceil c \log_2 N \rceil = c^2 \log_2 n + O(1)$, and let $T = N^c = O(n^{c^2})$. Recall that the constant c stems from the definition of a verifier. 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 certificates 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 $[\![2^0]\!]$, $[\![2^1]\!]$, $[\![2^2]\!]$, $\ldots$, $[\![2^w]\!]$ 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}:

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

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. Although, this is a fairly lengthy way to check if a variable is an integer in a certain range, recall that ETR only has the constants $0, 1$ in their alphabet. Our final formula $\Phi(I)$ includes the following conjunction, which forces the initial values of every word register variable to actually be a word:

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

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:

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

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

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

(Here $I[i]$ denotes the ith coordinate of I.) We leave the remaining initial word register variables $[\![W(i,0)]\!]$ and all initial real register variables $[\![R(i,0)]\!]$ 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 of 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

$$\texttt{Execute} := \bigwedge_{t=1}^{T} \bigwedge_{\ell=1}^{L} \big(([\![pc(t)]\!] = \ell) \Rightarrow \texttt{Update}(t, \ell) \big)$$

The various expressions $\texttt{Update}(t, \ell)$ are nearly identical. In particular, $\texttt{Update}(t, \ell)$ includes the constraints $[\![W(i,t)]\!] = [\![W(i,t-1)]\!]$ and $[\![R(j,t)]\!] = [\![R(j,t-1)]\!]$ 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

$$\texttt{Step}(t) := ([\![pc(t)]\!] = [\![pc(t-1)]\!] + 1).$$

Tables 2 to 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.

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

- 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 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

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

Now suppose I is a yes-instance of 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 $[\![pc(T)]\!]$ 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. (Recall that L denotes the number of instruction lines in the program of A.) 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.

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. □

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

Table 3: Encoding indirect memory instructions as formulae

6.3 Examples

To illustrate the usefulness of Theorem 2, we give simple proofs that four example problems are in $\exists\mathbb{R}$. Recall the problems as stated in Corollary 3.

1. The art gallery problem
2. Optimal straight curve straightening problem
3. Geometric packing
4. Optimal unknotted extension problem.

For two of these problems, membership in $\exists\mathbb{R}$ was already known [2,28]; however, our proofs are significantly shorter and follow from known standard algorithms. We introduce the fourth problem specifically to illustrate the mixture of real and discrete non-determinism permitted by our technique.

Proof of Corollary 3. 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 [40], 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 [28]. 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 are 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 [8], 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.

Recall that the input for geometric packing is a container, the corresponding pieces and a number k of how many pieces we wish to pack. The real witness is a description of the corresponding movement of each piece. The verification algorithm places all pieces and checks that there are no intersections between pieces and all pieces are contained in the container. This can be done with some standard sweep-line algorithm [9].

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.* [39], or by projecting to a two-dimensional knot diagram and guessing and executing an unknotting sequence of Reidemeister moves [48]. □

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

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

7 Polynomials Hitting Cubes

In the first part of this section, we bound from above the number of unit cubes that a d-variate polynomial p of bounded degree Δ can intersect in $C(d,k)$. In the second part of this section we finish the proof of Theorem 4 by applying a well-known lemma for the sum of probabilities.

Following Cox et al. [20] we define a d-variate polynomial p in $x_1,\ldots,x_d$ with real coefficients as a finite linear combination of monomials with real coefficients. Let $p \in \mathbb{R}[x_1,\ldots,x_d]$ denote the set of such polynomials.

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 $V(p_1,\ldots,p_k) = \bigcap_{i=1,\ldots,k} V(p_i)$. For any expression f which defines a function, we also use the notation $V(f) = \{x : f(x) = 0\}$, although it is not necessarily a variety. We say f *intersects* a set S, if $V(f) \cap S \neq \emptyset$. We 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 [6, Chapter 5]. Specifically, Lemma 13 has to be taken for granted. Given a polynomial $p \in \mathbb{R}[x_1,\ldots,x_d]$, 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 $\ell \cdot q = p$.

Theorem 7. *[Hitting Cubes, Section 7] Let $p \neq 0$ be a d-variate polynomial with maximum degree Δ and let $k \geq 2\Delta + 2$. Then the variety $V(p)$ of 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 [6, Chapter 7] for historic remarks and related results.

Theorem 12 (Milnor [6])**.** *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. For more background on polynomials, see the book from Cox, Little, O'Shea [21]. Specifically Hilbert's Nullstellensatz, which can be found as Theorem 4.77 in the book by Basu, Pollack and Roy [6] is important.

Lemma 13 (folklore)**.** *Let $p \in \mathbb{R}[x_1,\ldots,x_d]$ 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 7. 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 13.

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 Equations (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).

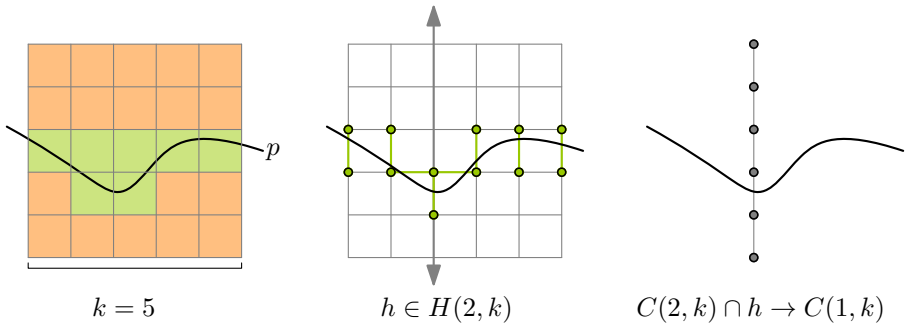

Figure 8: (Left) The polynomial p whose variety intersects some cubes of $C(2,k)$. (Middle) The set $H(2,k)$ of axis-parallel lines and the 1-dimensional facets of $C(2,k)$ that are intersected by p. (Right) The intersection of $C(2,k)$ with a line H gives $C(1,k)$.

To show the correctness of Equation (2), we refer to Figure 8. 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 13 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 the variety of p, the variety of p 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 13, $V(p)$ restricted 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 bound from above by $(2\Delta)^d + 2\cdot f(d-1)\cdot k(d+1)$. This shows Equation (2) and finishes the proof. □

8 Smoothing the real RAM

This section is dedicated to prove the following theorem, which we first recall.

Theorem 4. *Let A be a polynomial-time recognition verification algorithm with arithmetic degree Δ, algebraic dimension d and few polynomials. Under perturbations of $g \in [0,1]^n$ by $x \in \left[\frac{\delta}{2}, \frac{\delta}{2}\right]^n$ chosen uniformly at*

random, the algorithm A has a smooth input-precision of at most:

$$\mathtt{bit}_{smooth}(\delta, n, A) = O\left(d \log \frac{d\Delta n}{\delta}\right).$$

For the smoothed analysis, we *snap* our perturbed real-valued input $g_x = (g + x)$ onto a point with limited precision g', 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. The Voronoi cells of all these cells (apart from those belonging to grid points on the boundary of $[0,1]^d$) are just d-dimensional cubes and we shall denote a cube centered at y by $C(y)$. We denote $\Gamma_\omega = \{C(y) : y \in \omega\mathbb{Z}^d \cap [0,1]^d\} \cong C(d, 1/\omega)$.

The perturbed point g_x must lie within some cube $C(y)$ in Γ_ω. However the choice of original g and δ limits the cubes in Γ_ω where g_x can lie in. Specifically (Figure 9) the range R of possible locations for g_x is a cube of width δ centered around g. The boundary of this range cube R likely does not align with the boundaries of grid cells in Γ_w. Therefore we make a distinction between two types of cells: the range cube R must contain a cube of cells which is equivalent to $C(d, \lfloor \delta/\omega \rfloor)$ and this sub-cube of R we shall denote by $\Gamma_\omega(g)$. All others cells which are intersected by R but not contained in $\Gamma_\omega(g)$ we call the *perimeter cells*. We can use this observation together with Theorem 7 to estimate the probability that $\text{sign}(p(g_x)) \neq \text{sign}(p(g'))$:

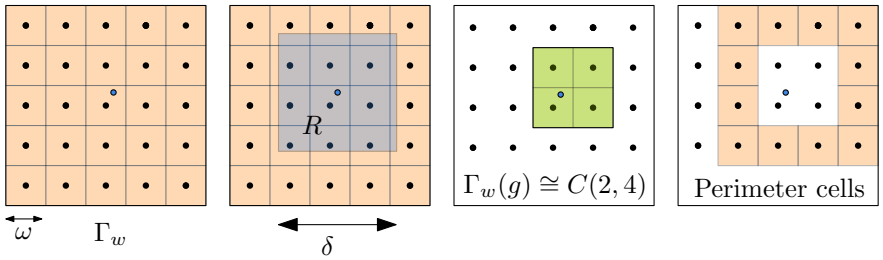

Figure 9: We depict the range R of width δ around a point g. The block of green cells in $\Gamma_w(g)$ and the remaining cells that are intersected by R are the perimeter cells.

Lemma 14. *Let p, q be two d-variate polynomials with maximum degree Δ. Let $g \in \mathbb{R}^d$ be fixed and $x \in \Omega = [-\delta/2, \delta/2]^d$ chosen uniformly at random. Assume $g_x = g + x$ is snapped to a point $g' \in \omega\mathbb{Z}^d \cap [0,1]^d$. Then $\text{sign}\left(\frac{p(g_x)}{q(g_x)}\right) \neq \text{sign}\left(\frac{p(g')}{q(g')}\right)$ with probability at most:*

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

Proof. It suffices to show that $\text{sign}(p(g_x)) \neq \text{sign}(p(g'))$ 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 bounds from above the probability that their division does not have the same sign. For any polynomial p and points $x, z \in \mathbb{R}^d$ with $p(x) < 0$ and $p(z) > 0$, there must be a point $y \in \text{segment}(x, z)$ for which $p(y) = 0$. It follows that if a cube $C \in \Gamma_\omega$ does not intersect the variety of p, all points in C either have a positive or negative evaluation under p. Let g' be the closest point to g_x in $\omega\mathbb{Z} \cap [0,1]^d$ and let $C(g')$ be its Voronoi cell. If $C(g')$ does not intersect the variety of p then $\text{sign}(p(g_x)) = \text{sign}(p(g'))$. Therefore we are interested in the probability that g_x is contained in a Voronoi cell that is intersected by the variety of p.

As we discussed, the set of possible locations of g_x is a cube which contains $\Gamma_\omega(g) \cong C(d, \lfloor \delta/\omega \rfloor)$ together with a collection of perimeter cells.

We bound from above the probability that g_x lies within a cell that is intersected by p in two steps: (1) We bound from above the number of perimeter cells and make the worst case assumption that they are all intersected by the variety of p. (2) We bound from above the number of cells of $\Gamma_\omega(g)$ that intersect the variety of p. These two numbers, divided by the total number of cells in $\Gamma_\omega(g)$ gives the upper bound we are looking for. Note that the width of $\Gamma_\omega(g)$ is equal to $k = \lfloor \frac{\delta}{\omega} \rfloor$ and that the perimeter of $\Gamma_\omega(g)$ contains $2d \cdot k^{d-1}$ cells. Theorem 7 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 $k = \lfloor \delta/\omega \rfloor^d$ cubes in $\Gamma_\omega(a)$ and it follows that:

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

Using this, in conjunction with the union bound over the signs of $p(g_x)$ and $q(g_x)$ respectively, finishes the proof. □

Lemma 14 is used to bound from above the probability that snapping the perturbed input g_x changes a single real-valued comparison in A. We can use the union bound to bound from above the probability that for the whole algorithm, any real-valued comparison for g_x and g' is different.

Recall that we denote by $C(n)$ the total number of possible polynomials with which a given algorithm A performs a comparison operation. As remarked in Section 3, it holds that $C(n)$ is bounded from above by $2^{T(n)}$, where $T(n)$ denotes the running time. The theorem statement of Theorem 4 assumes $C(n)$ to be polynomial in the input-size.

Lemma 15 (Snapping). *Let $g \in [0,1]^n$ be the input of an algorithm A that makes at most $C(n)$ potential real RAM comparison operations. Let x be a perturbation chosen uniformly at random in $[-\delta/2, \delta/2]^n$ and let $g_x = g + x$ be a perturbed instance of the input. For all $\varepsilon \in [0,1]$, if g_x is snapped to a grid of width*

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

then the algorithm A makes for g_x and g' the same decision at each comparison instruction with probability at least $1-\varepsilon$.

Proof. By E_i for $i \in [C(n)]$, we denote the event that in the i'th polynomial on the inputs g_x and g' get a different outcome. Lemma 14 bounds from above the probability of E_i occurring by:

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

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

$$\Pr(g_x \text{ and } g' \text{ not equivalent}) = \Pr\left(\cup_{i=1}^{C(n)} E_i\right) \leq C(n) \cdot 4\omega\Delta 3^d (d+1)/\delta.$$

In the antecedent, $Pr(g_x$ and g' not equivalent$) < \varepsilon$ is implied by the inequality

$$C(n) \cdot \frac{4\omega\Delta 3^d (d+1)}{\delta} < \varepsilon,$$

as stated above. □

Now we are ready to bound from above the *expected* input-precision of A. By definition it holds that

$$\mathbb{E}(\,\mathtt{bit}_{IN}(g_x, A)\,) = \sum_{k=1}^{\infty} k \Pr(\,\mathtt{bit}_{IN}(g_x, A) = k\,)$$

Recall the following well-known lemma from Tonelli regarding the sum of expectations. Given a function $f : \Omega \to \{1, 2, \ldots\}$ 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 the lemma with $b \to \infty$, the expected value of $\mathtt{bit}_{IN}$ can be expressed as:

$$\mathbb{E}(\,\mathtt{bit}_{IN}(g_x, A)\,) = \sum_{k=1}^{\infty} k \Pr(\,\mathtt{bit}_{IN}(g_x, A) = k\,) \;= \sum_{k=1}^{\infty} \Pr(\,\mathtt{bit}_{IN}(g_x, A) \geq k\,).$$

We split the sum at a splitting point l:

$$\mathbb{E}(\,\mathtt{bit}_{IN}(g_x, A)\,) = \sum_{k=1}^{l} \Pr(\,\mathtt{bit}_{IN}(g_x, A) \geq k\,) \;+ \sum_{k=l+1}^{\infty} \Pr(\,\mathtt{bit}_{IN}(g_x, A) \geq k\,).$$

Now we note that any probability is at most 1 therefore the left sum is at most l. Through applying Lemma 15 we note that:

$$\Pr(\mathtt{bit}_{IN}(g_x,A) \geq k) = \Pr(\text{GridWidth}(g_x) \geq 2^{-k}) \leq 2^{-k}\left(\frac{3^d(d+1)!\Delta 4C(n)}{\delta}\right),$$

which in turn implies:

$$\mathbb{E}(\mathtt{bit}_{IN}(g_x,A)) \leq \sum_{k=1}^{l} 1 \ + 4\left(\frac{3^d(d+1)!\Delta C(n)}{\delta}\right)\sum_{k=l+1}^{\infty} 2^{-k}.$$

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

$$\begin{aligned}
\mathbb{E}(\mathtt{bit}_{IN}(g_x,A)) &\leq l + 4\left(\frac{3^d(d+1)!\Delta C(n)}{\delta}\right)2^{-l} \\
&\leq \left\lceil \log \frac{3^d(d+1)!\Delta C(n)}{\delta} \right\rceil + 2 + \\
&+ \left(\frac{3^d(d+1)!\Delta C(n)}{\delta}\right)\left(\frac{\delta}{3^d(d+1)!\Delta C(n)}\right) \\
&= \left\lceil \log \frac{3^d(d+1)!\Delta C(n)}{\delta} \right\rceil + 3 = O\left(d \log \frac{d\Delta C(n)}{\delta}\right)
\end{aligned}$$

Since this holds for arbitrary g with $g_x = g + x$, this bounds from above the input-precision. This finishes the proof of Theorem 4.

9 Smoothed Analysis of resource augmentation

This section is devoted to proving Theorem 8 and its corollaries. We briefly recall that for any x, we consider the minimal word size required for each coordinate in x before a verification algorithm A can verify if $x \in \chi_I[\alpha]$ on the word RAM. We denote by $\mathtt{bit}(\chi_I[\alpha]) = \min\{\mathtt{bit}_{IN}(x,A) \mid f(x) = \text{opt}(\chi_I[\alpha]), x \in \chi_I[\alpha]\}$ the minimal precision needed to express an optimal solution in the solution space $\chi_I[\alpha]$. We suppress I in the notation when it is clear from context.

Theorem 8. *Let P be a resource augmentation problem that is monotonous, moderate and smoothable and $m \leq O(n^c)$ then $\mathtt{bit}_{smooth}(\delta, n, m, P)$ is bounded from above by $O(\log(n/\delta))$.*

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 bounded from above by $\varepsilon N/\delta$ for some choice of $N = n^{O(1)}$. Assume that the interval $[\alpha - \varepsilon, \alpha]$ does not contain a breakpoint then by definition $\text{opt}(\chi[\alpha - \varepsilon]) = \text{opt}(\chi[\alpha])$. 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 an input-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 an input-precision of $c\log(n/\varepsilon)$ is bounded from above by:

$$Pr\left(\mathtt{bit}(\chi[\alpha]) \geq c\log(n/\varepsilon)\right) \leq \frac{\varepsilon \cdot N}{\delta}. \tag{3}$$

Note that Equation (3) holds for every ε. We use this probability to obtain an upper bound on the expected input-precision. Using the lemma by Tonelli, 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 most 1 which means that the left sum is bound from above by l. We bound from above $\Pr(\mathtt{bit}(\chi[\alpha]) \geq k)$ by equating $k = c\log(n/\varepsilon_k) \Leftrightarrow \varepsilon_k = n/2^{(k/c)}$ and applying Equation (3):

$$\sum_{k=l+1}^{\infty} \Pr(\mathtt{bit}(\chi[\alpha] \geq k)) = \sum_{k=l+1}^{\infty} \Pr(\mathtt{bit}(\chi[\alpha] \geq c\log(n/\varepsilon_k))) \leq \sum_{k=l+1}^{\infty} \frac{\varepsilon_k \cdot N}{\delta}$$

$$= \frac{nN}{\delta} 2^{-(l+1)/c} \frac{1}{1-2^{-1/c}} \leq 2^{-(l+1)} \frac{nN}{\delta(1-2^{-1/c})}$$

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 8. Dobbins, Holmsen and Miltzow show that under smoothed analysis the $\exists\mathbb{R}$-complete art gallery problem has a solution with logarithmic expected input-precision [25]. 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 [25].

Corollary 16 ([25]). *For the art gallery problem, with an α-augmentation which is an α-edge-inflation, the expected input-precision with smoothed analysis over α is at most $O(\log(n/\delta))$.*

Proof. The monotonous property and the smoothable property are both shown in the short Lemmas 6 and 7 in [25]. 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 [31]. This implies the moderate property. Together with the monotonous property this proves that the number of breakpoints is bounded from above by $n/3$ and the corollary follows. □

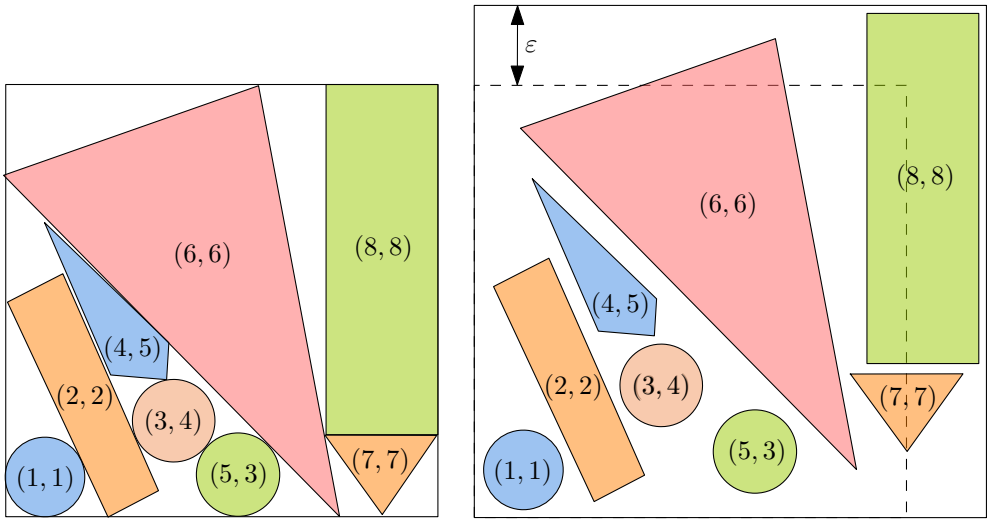

Figure 10: (Left) A set of 8 convex objects which are tightly packed in a square with diameter $(1+\alpha)$. Their enumerations specify a choice for π_x and π_y. Note that this choice is not unique. (Right) The objects packed in a square of diameter $(1+\alpha+\varepsilon)$ where an object with enumeration (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 17. *For geometric packing of convex objects in a unit-size container, with an α-augmentation over the container size, the expected input-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-augmentation 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 $(1+\alpha)$. First, we show that this algorithmic problem has the monotonous property.

Specifically, we show that for all $\alpha, \alpha' \in [0,1]$ with $\alpha \leq \alpha'$, $\chi_I[\alpha]$ 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 $(1+\alpha')$.

Next, we show the problem is moderate. 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 bounded from above by n.

Lastly we show smoothable property. We show that for all optimal packings x in $\chi_I[\alpha]$ and for all $\varepsilon > 0$, there is a solution $x' \in \chi_I[\alpha+\varepsilon]$ with input-precision with respect to its real verification algorithm of $O(\log(n/\varepsilon))$ and $f(x) \leq f(x')$ (since packing is a maximization problem). In other words: given an optimal packing x, that packs k objects in a container of size $(1+\alpha)$ we must show that there is a packing x' in a container of size $(1+\alpha+\varepsilon)$, that packs at least k objects *and* that the solution x' can be specified using $O(\log(n/\varepsilon))$ bits per coordinate.

To this end, let $x \in \chi[\alpha]$ be an optimal 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 will 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 10).

Specifically, 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 a linear extension to π_x must always exist (Figure 11). 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 $\frac{\varepsilon}{n+2}$ due to the translation. Indeed, since all objects were originally disjoint their distance was originally at least zero. Now they are separated by at least $\frac{\varepsilon}{n+2}$ in both the x and y direction.

This proves that we can place the packing x in a container of size $(1+\alpha+\varepsilon)$, such that any two objects are separated by at least $\frac{\varepsilon}{n+2}$. What remains to show, to prove that we can always specify this packing using logarithmic bits per coordinate.

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. We claim that we can now move and rotate every object in x by at most $O(\varepsilon/n)$, such that all objects are still pairwise disjoint and such that their rotation and translation can be described with $O(\log n/\varepsilon)$ bits. For translation, this is immediately true (simply snap one of the points of the convex object to a corresponding nearest gridpoint). When describing a rotation however, it is not immediately obvious that one can describe a suitable rotation with $O(\log n/\varepsilon)$ bits. As we are not aware of any publication that specifies how to describe rotations with limited bit-precision, we give an example of how to do this in Section 9.1.

Thus, we showed that given any optimal packing $x \in \chi[\alpha]$ there exists a packing $x' \in \chi[\alpha+\varepsilon]$, that packs at least as many objects and can be described with $O(\log n/\varepsilon)$ bits per objects. Therefore, geometric packing is smoothable. □

Lastly to demonstrate the wide applicability of Theorem 8 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 [47] 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 input-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 augment the minimum link path problem in the same way as the art gallery problem was augment in [25]: 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 Corollary 16 we can immediately state that:

Corollary 18. *For computing the minimum link path, with an α-augmentation which is edge-inflation, the expected input-precision with smoothed analysis over α is $O(\log(n/\delta))$.*

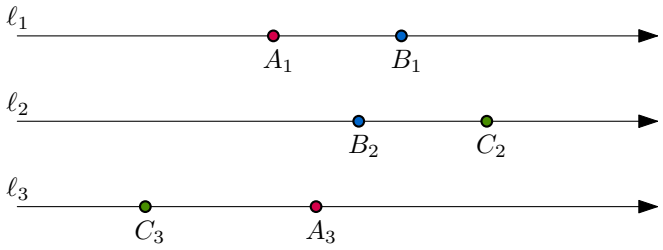

Figure 11: Consider for the sake of contradiction three planar pairwise disjoint convex objects A, B, C with $\pi_x(A) < \pi_x(B)$, $\pi_x(B) < \pi_x(C)$ and $\pi_x(C) < \pi_x(A)$. Then, without loss of generality (due to symmetry), there exist three lines ℓ_1, ℓ_2, ℓ_3 as shown in the figure, where A intersects ℓ_1 before B in the intersection point A_1, B intersects ℓ_2 before C in the intersection point B_2 and C intersects ℓ_3 before A in the point C_3. Since A is convex, there exists a segment connecting A_1 and A_3 contained in A. This segment has to cross ℓ_2, left of B_2 (else this segments intersects the segment between B_1 and B_2, and via therefore the convex object B). It follows that the segment connecting A_1 and A_3 intersects the segment connecting C_2 and C_3 which contradicts that these objects were pairwise disjoint.

9.1 Describing a rotation using limited input-precision

To finalise the argument for Corollary 17, we claimed that for every object in a packing x, we could slightly rotate that object so that its rotation can be described with a limited number of bits. For completeness, we dedicate this section to providing a way to describe such a rotation.

In computer applications (and specifically applications that contain geometry), it is not uncommon to want to rotate geometric objects. However, describing a rotated geometric object is nontrivial on a real RAM and problematic on a word RAM. More often than not, rotations are the result of user input and from an application perspective it would be nice if a user could specify an angle ϕ and obtain a geometric object that is rotated by ϕ degrees. However, this is infeasible in both the word RAM *and* real RAM. Rotating a vector v by ϕ degrees is equal to multiplying the vector v with the matrix:

$$R = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$$

This rotation matrix R is a matrix with two key properties: (1) the transposed matrix R^T is equal to the inverse of R,i.e., $R^T = R^{-1}$ and (2) the determinant of R equals 1. These two properties ensure that the rotated version of a vector still has the same length and that adjacent edges that get rotated still have the same angle between them. In order to compute R, given ϕ, requires the computation of the cos and sin function, which cannot be simulated by the real RAM, as explained in Section 6.1. If we use the word RAM, we can compute an approximation of cos and sin, but the resulting matrix, will only in exceptional cases have properties (1) and (2). However, note that we can specify any rotation R by providing a unit vector V_R with:

$$R \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = v_R \quad \Rightarrow \quad \begin{pmatrix} a & -b \\ b & a \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} = v_R.$$

Clearly the vector v_R might have real coefficients and therefore a rotation that can be expressed in the real RAM may not always be used in the word RAM. However, we show that for any w and any desired rotation $v_R \in \mathbb{R}^2$ we can express a rotation matrix R' which uses an input-precision of w bits such that: $||R \cdot (1,0) - R'(1,0)|| < 2^{-w}$. That is the goal of this subsection, we show that it is possible to obtain such a R' with logarithmic *input-precision* such that the resulting rotation that we obtain is not far off.

Let $v_R = (a, b)$ be the desired rotation and let v_R not be an orthonormal vector (if it is, then the desired rotation can be expressed using a constant-complexity rotation matrix). V_R must lie in one of the four quantiles of the plane (We chose the quantiles to lie diagonally to the x and y axis, see Figure 12). Given v_R, we consider the unique orthonormal vector that lies in the quantile opposite of V_R. Let without loss of generality v_R be a vector lying in the left quantile, then the orthonormal vector that we consider is $(1,0)$. We now assume that v_R lies above the x-axis, the argument for below the x-axis is symmetrical.

We round the coordinates (a, b) to the closest point (a', b') that lies above the x-axis, outside of the unit circle *and* a' and b' can be described using w bits each. Observe that since $a, b \in [0,1]$, it must be that $||(a,b) - (a',b')|| \leq 2 \cdot 2^{-w}$.

Lemma 19. *Consider the line through* (a', b') *and* $(1,0)$ *and its point of intersection with the unit circle* (x, y)*. It must be that:*

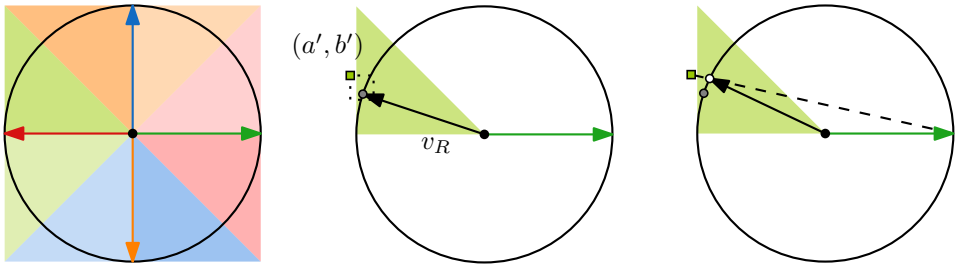

Figure 12: (Left) the four quantiles that we consider in orange, green, blue and red. If the desired vector v_R lies in a quantile, we construct a rotation using the orthonormal vector in the opposite quantile. (Middle) We round $v_R = (a, b)$ to the closest top-right point that can be expressed with w bits. (Right) The points $(1, 0)$ and (a', b') and their point of intersection on the unit circle.

1. $||(a, b) - (x, y)|| \leq 2 \cdot 2^{-w}$.
2. *The rotation matrix $R'(1, 0) = (x, y)$ can be described using limited input-precision.*

Proof. The proof is illustrated by Figure 12. Consider the square C centered around (a, b) with (a', b') as its top left corner. Both of its right corners must be contained within the unit disk. The line ℓ through $(1, 0)$ and (a', b') is equal to $y = \frac{b'(x-1)}{a'-1}$. Per construction, the values a' and b' can be expressed using w bits, therefore the line can be described by $y = \alpha x - \beta$ with α and β having $O(w)$ bits. Per construction, the slope α is between 0 and -1 (since (a', b') must lie within the same quantile as (a, b) and, above the x-axis). It follows that the line ℓ intersects C in its right facet and therefore the point of intersection (x, y) must be contained within C which proves the first item.

Item two: The unit circle is given by $y^2 + x^2 = 1$. The point of intersections between the line and the circle is:

$$x = \frac{\pm\sqrt{\alpha^2 - \beta^2 + 1} - \alpha\beta}{\alpha^2 + 1}, \quad y = \frac{\beta \pm \alpha\sqrt{\alpha^2 - \beta^2 + 1}}{\alpha^2 + 1}$$

We know that one point of intersection is $(1, 0)$. Given that α and β can both be realized by $O(w)$ bits, this implies that the values $\sqrt{\alpha^2 - \beta^2 + 1}$ and $\sqrt{\alpha^2 - \beta^2 + 1}$ can also be realized by $O(w)$ bits. □

References

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

[2] Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. The art gallery problem is $\exists\mathbb{R}$-complete. In *STOC*, pages 65–73, 2018.

[3] Mikkel Abrahamsen, Linda Kleist, and Tillmann Miltzow. Training neural networks is er-complete. *CoRR*, abs/2102.09798, 2021.

[4] Mikkel Abrahamsen, Tillmann Miltzow, and Nadja Seiferth. A framework for $\exists\mathbb{R}$-completeness of two-dimensional packing problems. *in Preparation*, 2020.

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

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

[7] René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. In *STOC*, pages 232–241. ACM, 2003.

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

[9] Jon Louis Bentley and Thomas A Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Computer Architecture Letters*, 28(09):643–647, 1979.

[10] 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.

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

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

[13] 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.

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

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

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

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

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

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

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

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

[22] Daniel Dadush and Sophie Huiberts. A friendly smoothed analysis of the simplex method. In *STOC*, pages 390–403. ACM, 2018.

[23] Erik D Demaine, Adam C Hesterberg, and Jason S Ku. Finding closed quasigeodesics on convex polyhedra. In *36th International Symposium on Computational Geometry (SoCG 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

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

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

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

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

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

[29] Jeff Erickson, Ivor van der Hoog, and Tillmann Miltzow. Smoothing the gap between np and er. In *FOCS 2020*, 2020.

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

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

[32] Steven Fortune and Christopher Van Wyk. Efficient exact arithmetic for computational geometry. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 163–172. ACM, 1993.

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

[34] 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.

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

[36] Jacob Goodman, Richard Pollack, and Bernd Sturmfels. Coordinate representation of order types requires exponential storage. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 405–410, New York, NY, USA, 1989. ACM.

[37] Torben Hagerup. Sorting and searching on the word ram. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *Proc. 15th Ann. Symp. Theoretical Aspects Comput. Sci.*, number 1373 in Lecture Notes Comput. Sci., pages 366–398. Springer Berlin Heidelberg, 1998.

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

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

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

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

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

[43] 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, 2008.

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

[45] Victor Klee and George Minty. How good is the simplex algorithm. Technical report, Washington Univ. Seattle Dept. of Mathematics, 1970.

[46] Linda Kleist. *Planar graphs and faces areas – Area-Universality*. PhD thesis, Technische Universität Berlin, 2018. PhD thesis.

[47] 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.

[48] Marc Lackenby. A polynomial upper bound on Reidemeister moves. *Ann. Math.*, 182(2):491–564, 2015.

[49] Leonid Anatolevich Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973.

[50] 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, 2005.

[51] Giuseppe Liotta, Franco P Preparata, and Roberto Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM Journal on Computing*, 28(3):864–889, 1998.

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

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

[54] Jiří Matoušek. Intersection graphs of segments and $\exists\mathbb{R}$. *ArXiv 1406.2636*, 2014.

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

[56] Tillmann Miltzow and Reinier F. Schmiermann. On classifying continuous constraint satisfaction problems. *CoRR*, abs/2106.02397, 2021.

[57] Nicolai Mnëv. The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In Oleg Y. Viro, editor, *Topology and geometry – Rohlin seminar*, pages 527–543. Springer-Verlag Berlin Heidelberg, 1988.

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

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

[60] Daniel Richardson. Some undecidable problems involving elementary functions of a real variable. *The Journal of Symbolic Logic*, 33(4):514–520, 1969.

[61] 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.

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

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

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

[65] Marcus Schaefer. Realizability of graphs and linkages. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*, chapter 23, pages 461–482. Springer-Verlag New York, 2013.

[66] 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.

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

[68] Michael Ian Shamos. *Computational Geometry*. PhD thesis, Yale Unviersity, 1979.

[69] Yaroslav Shitov. A universality theorem for nonnegative matrix factorizations. *ArXiv 1606.09068*, 2016.

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

[71] Peter Shor. Stretchability of pseudolines is NP-hard. In Peter Gritzmann and Bernd Sturmfels, editors, *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, DIMACS – Series in Discrete Mathematics and Theoretical Computer Science, pages 531–554. American Mathematical Society and Association for Computing Machinery, 1991.

[72] 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, 2004.

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

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

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