Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 202 (2008) 101–119

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Complexity of Operators on Compact Sets

Xishun Zhao[1,2,4]

Institute of Logic and Cognition, Sun Yat-Sen University 510275 Guangzhou, P. R. China

Norbert Müller[1,3,5]

FB IV, Abteilung Informatik, Universität Trier, D-54286 Trier, Germany

Abstract

Based on oracle Turing machines, we investigate the computational complexity of operators on compact sets. For the projection and convex hull we are able to show exponential upper and lower bounds as well as a connection to the P=NP problem for special settings.

Keywords: Oracle Turing machines, compact sets, operator complexity, projection, convex hull

1 Introduction

The computability of compact sets and of operators on compact sets has already been studied in the literature, cf. [13,15,16], also the complexity of some compact sets like Julia sets was analyzed in [2,12]. However, the complexity of operators on compact sets is less explored.

To define computability and complexity of operators on compact sets, essentially two versions of Turing machines can be used: Type-2 Turing machines (cf. [13]) or oracle Turing machines (cf. [8]). From the viewpoint of computability, the approaches are equivalent. From the viewpoint of complexity theory on the real *numbers*, at least the notion of polynomial time complexity is invariant to the

[1] The authors would like to thank Professor Klaus Weihrauch for his useful comments and suggestions to write this paper, and the anonymous referees for many valuable hints and corrections.

[2] Email: hsszxs@mail.sysu.edu.cn

[3] Email: mueller@uni-trier.de

[4] Research was partially supported by the NSFC projects under Grant No. 60573011, 10410638 and a MOE project under grant number 05JJD72040122

[5] Research was partially supported by the DFG project 446 CHV 113/240/0-1

doi:10.1016/j.entcs.2008.03.011

machine model. This changes, if we look at operators on the compact sets or on continuous functions (like differentiation or integration): A computation with a Type-2 machine up to a precision n involves that a full approximation with this precision has to be written onto the output tape (or has to be read from the input tape). For compact sets or continuous functions, such approximations already have exponential size, so studying polynomial time complexity is useless in that setting. Oracle Turing machines on the other hand have a direct, non-sequential access to their oracles, they might also just write 'partial information' about the output, so we can get interesting results here.

The paper is organized as follows: Using oracle Turing machines, we show that several basic operators like scaling or rotation have a polynomial time complexity. Then we concentrate on a less trivial example, the projection of two-dimensional sets onto one of the coordinates. Here exponential upper and lower bounds hold, additionally we are able to prove a connection to the P=NP question. Similar exponential lower bounds for operators were studied extensively by Traub's school of Information-Based Complexity [10,11]. The embedding of problems of discrete complexity theory like P=NP into a real-valued context can already be found e.g. in [6,9].

On the other hand, for convex compact sets we present an algorithm working in polynomial time. It is not clear whether this result can already be derived from polynomial time algorithms for linear programming, as in our setting the compact sets can even have an empty interior.

In the last section of the paper we consider the question of computing the convex hull of a compact set. Here in general, the complexity of the operator is again exponential, and we are able to find a connection to the P=NP question even for compact sets that additionally are regular and simply connected. The proof of this connection is influenced by a simular construction in [6], where, among others, the question of finding the maximum value of a real function was considered.

Questions of the complexity of two-dimensional real sets have also been considered e.g. in [4,5]. The equivalence of the underlying computational model to the definitions in [13] has been addressed in [3], the exact relation of computational complexity in the two settings still has to be considered.

2 Computability and complexity

We start with the basic definitions based on (or cited from) [13]. Results on computability using Type-2 machines can also be found in [13], but we will formulate our own results for oracle machines instead.

Definition 2.1

- For $k \in \mathbb{N}$ let $\mathbb{K}^{(k)}$ be the set of the non-empty compact subsets of $\mathbb{R}^k$.
- $\mathbb{D} := \bigcup_{n\in\mathbb{N}} \mathbb{D}_n$ denotes the dyadic numbers, $\mathbb{D}_n := \{m \cdot 2^{-n} \mid m \in \mathbb{Z}\}$.
- For $\overline{d} = (d_1, \ldots, d_k) \in \mathbb{R}^k$ let $U_n(\overline{d}) = \{(x_1, \ldots, x_k) \mid |d_i - x_i| \leq 2^{-n}\}$ be the k-dimensional hypercube with center $\overline{d}$ and edge length 2^{1-n}.

We use a representation of arbitrary non-empty compact sets of real vectors based on definitions 5.2.1.3 and 7.4.1. of [13]:

Definition 2.2 A *grid name* ψ of a set $K \in \mathbb{K}^{(k)}$ is a pair $\psi = (h, b)$ with the following properties:

(i) The first component h is a total function

$$h : \{(n, \overline{d}) \mid n \in \mathbb{N}, \overline{d} \in \mathbb{D}_n^k\} \to \{0, 1\}$$

This function h is called the *grid indicator* and defines a sequence B_h of sets

$$B_h(n) := \{\overline{d} \in \mathbb{D}_n^k \mid h(n, \overline{d}) = 1\}$$

(ii) For each of the sets $B_h(n)$ the Hausdorff distance between $B_h(n)$ and K may not be greater than 2^{-n}, i.e.

$$(\ \forall \overline{d} \in B_h(n)\ )\ \ (\ \exists \overline{x} \in K\ )\ \ \overline{x} \in U_n(\overline{d})$$
$$(\ \forall \overline{x} \in K\ )\ \ (\ \exists \overline{d} \in B_h(n)\ )\ \ \overline{x} \in U_n(\overline{d})$$

(iii) The second component $b \in \mathbb{N}$ is called the *grid bound* for h and must satisfy

$$B_h(0) \subseteq [-2^b; 2^b]^k$$

(iv) Additionally, we require that b is minimal: If $b > 0$, then

$$B_h(0) \not\subseteq [-2^{b-1}; 2^{b-1}]^k$$

For an example of some sets $B_h(n)$ see figure 1.

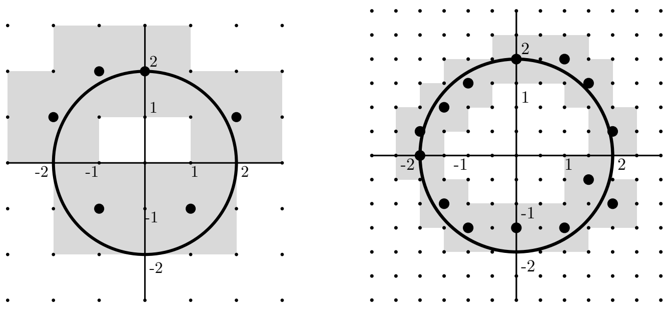

Fig. 1. Possible sets $B_h(0)$ and $B_h(1)$ for the circle $\{(x, y) \mid x^2 + y^2 = 2\}$.
The corresponding regions $U_n(\overline{d})$ are drawn in light grey.

A few remarks concerning this definition are important for our further considerations:

- The information contained in a grid name is redundant in the following sense: Knowing all elements from $B_h(0)$ we are obviously able to deduce the grid bound b. Our approach uses the grid indicator h for the access to the $B_h(n)$, i.e. there will be a 'black box' (= an oracle) just answering questions like 'Does the vector $\overline{d} \in \mathbb{D}_n^k$ belong to $B_h(n)$?'. In this setting, it is necessary to know (an upper bound for) b beforehand just to find all the vectors belonging to $B_h(0)$ via h!

- For a given compact set K, neither B_h nor b are uniquely determined; but only a *finite* number of such sets $B_h(0)$ is possible for K. For each of these possible sets, b is then uniquely determined. So only finitely many b may exist for K!
- Suppose that for a compact set K we are able to compute the grid indicator h at least for $n = 0$ and that additionally we know that $K \subseteq [-2^s, 2^s]^k$ for a given natural number s, though s might perhaps be much larger than the grid bound b. Then surely $b \leq s + 1$ and it is sufficient to compute $h(0, \overline{d})$ for all $\overline{d} \in \mathbb{D}_0^k \cap [-2^{s+1}, 2^{s+1}]^k$ in order to find all elements from $B_h(0)$ and to determine the correct value of b.

  Here we do not need to evaluate h for any n with $n > 0$, which heavily influences our definition of computational complexity below!

Names for real vectors can be defined in a much simpler way: Similar to [8], we define a name for a real vector $(x_1, .., x_k) \in \mathbb{R}^k$ to be a function $\phi : \mathbb{N} \to \mathbb{D}^k$ with $\phi(n) = (\phi_1(n), .., \phi_k(n)) \in \mathbb{D}_n^k$ and $|\phi_i(n) - x_i| \leq 2^{-n}$.

In the following we specify our model of computability and define computational complexity. In order to be sufficiently general for our later considerations, we will use a version that allows a mix of compact sets and real numbers as inputs to our machines, as we want to investigate complexity of operators from $\mathbb{R}^{k_1} \times \mathbb{K}^{(k_2)} \times \mathbb{K}^{(k_3)}$... to $\mathbb{R}^{k'}$ or $\mathbb{K}^{(k')}$

We start by describing how an oracle for $K \in \mathbb{K}^{(k)}$, or more precisely for a name $\psi = (h, b)$ of K, should work. There are two different types of queries for such an oracle:

Definition 2.3 An oracle for a compact set $K \in \mathbb{K}^{(k)}$ given via a grid name $\psi = (h, b)$ behaves as follows:

- Queries are either empty or of the form '$(n, \overline{d})$?' for $n \in \mathbb{N}$ and $\overline{d} \in \mathbb{D}_n^k$.
- For a query '$(n, \overline{d})$?' the oracle results in the value $h(n, \overline{d})$ of the grid indicator h, i.e. we see whether the vector $\overline{d}$ lies on the n-th grid defined by h (or not).
- For the empty query the oracle returns the grid bound b for h.

Oracles for $\overline{x} \in \mathbb{R}^k$, or more precisely for a name ϕ of $\overline{x}$, are much simpler, of course. There is just one type of queries:

Definition 2.4 An oracle for a real vector $\overline{x} = (x_1, .., x_k)$ given by a name ϕ behaves as follows:

- Queries are of the form 'n?' for $n \in \mathbb{N}$.
- Given a query 'n?', the oracle returns the vector $\phi(n)$, i.e. with $\phi(n) = (d_1, .., d_k)$ we have $d_i \in \mathbb{D}_n$ and $|d_i - x_i| \leq 2^{-n}$, so we get approximations for each single component x_i.

In many papers, the authors restrict their considerations to real vectors from $[0; 1]^k$ for the arguments and for the results as well. A similar approach would be to restrict ourselves to compact sets from $[0; 1]^k$. Unfortunately, this restriction would imply to intersect the *results* with $[0; 1]^k$, which in general is not a computable operation.

Instead, we define computational complexity for arbitrarily large compact sets and/or vectors. We use a setting with two parameters n and s: n is used for the precision of the grid and the dyadic approximations; s depicts a bound for the 'size' of the arguments, where this size is defined as follows:

Definition 2.5

- For compact sets $K \subseteq \mathbb{K}^{(k)}$, let size$(K)$ be the smallest $s \in \mathbb{N}$ with $K \subseteq [-2^s; 2^s]^k$.
- For vectors $\overline{x} \in \mathbb{R}^k$, let size$(\overline{x})$ be the smallest $s \in \mathbb{N}$ with $\overline{x} \in [-2^s; 2^s]^k$
- If size(c_i) is already defined for all components of a tuple $(c_1, .., c_m)$, then

$$\text{size}(c_1, .., c_m) := \max\{\text{size}(c_1), .., \text{size}(c_m)\}$$

Of course, the size defined above is not a computable function. Its only purpose is to simplify the definition below:

Definition 2.6 Let $C_1, C_2, .., C_m$ be given such that each C_i is either a subset of $\mathbb{K}^{(k_i)}$ or a subset of $\mathbb{R}^{k_i}$ for some values $k_i \in \mathbb{N}$. Let $C := C_1 \times ... \times C_m$ and let $t : \mathbb{N}^2 \to \mathbb{N}$.

Consider an operator $F : C \to C'$ with domain C, where either $C' = \mathbb{R}^{k'}$ or $C' = \mathbb{K}^{(k')}$.

F is called *computable in time t on C* if and only if there is an oracle Turing machine M such that

- M computes F on C, i.e. given oracles Ψ_i for the arguments $c_i \in C_i$, $M^{\Psi_1,..,\Psi_m}$ behaves like an oracle for $F(c_1, .., c_m) =: c' \in C'$:
  - · Case $c' \in C' = \mathbb{R}^{k'}$: For inputs of form '$n$?' the machine $M^{\Psi_1,..,\Psi_m}$ produces an output $(d'_1, .., d'_{k'})$ consistent with the real vector c', corresponding to definition 2.4.
  - · Case $c' \in C' = \mathbb{K}^{(k')}$: For inputs of form '$(n, d_1, .., d_{k'})$?' or the empty query $M^{\Psi_1,..,\Psi_m}$ produces $h(n, d_1, .., d_{k'})$ or b such that (h, b) is consistent with the compact set c', corresponding to definition 2.3.
- If s is such that $s \geq \text{size}(c_1, .., c_m)$, then the number of steps of the oracle Turing machine must be restricted by $t(n, s)$ for any of the inputs 'n?' or '$(n, d_1, .., d_k)$?' and by $t(0, s)$ for the empty query.

We will say that F is computable in polynomial (or linear) time, if there are a polynomial (or linear) p and an arbitrary(!) function $q : \mathbb{N} \to \mathbb{N}$ such that $t(n, s) \leq p(n) \cdot q(s)$, i.e. the dependency on n is strictly polynomial (or linear) in n extended by a factor depending only on the size s. In other words: We mainly study polynomial complexity parameterized by the size of the arguments.

For studying the complexity of examples, it is important to add some remarks about the notation of the numbers used in the definition above. Similar to [8], n should be given in a unary notation, i.e. as a string of length n. Also the grid bound b should be returned in unary. Furthermore, oracle queries have to be written to special query tapes and, using a query operation counting as a single step, the oracle erases the complete content of corresponding answer tapes and writes the answer to the queries onto them. Additionally, any $d = m \cdot 2^{-n} \in \mathbb{D}$ should be given as a string '$u\ v_1 \bullet v_2$': The single bit $u \in \{0, 1\}$ indicates the sign of the number d; the binary strings $v_1, v_2 \in \{0, 1\}^*$ are such that v_1 denotes the integer part of d and has no leading zeroes, and v_2 denotes the fractional part of d and may not have trailing zeroes. So if $d \in \mathbb{D}_n$, then the length of v_2 is at most n, and if $size(d) \leq s$, then the length of v_1 is at most s.

To explain some details of the definition, we prove the following instructive examples. The proofs are not difficult; the focus lies on the behavior of the parameterized complexity.

Lemma 2.7

(1) The union operators $\bigcup_k$ on compact sets are computable in linear time, where $\bigcup_k : \mathbb{K}^{(k)} \times \mathbb{K}^{(k)} \to \mathbb{K}^{(k)}$ (for $k \in \mathbb{N}$) is defined by $\bigcup_k(K_1, K_2) := K_1 \cup K_2$.

(2) The crossproduct $\times$ of compact sets is computable in linear time, where $\times : \mathbb{K}^{(k)} \times \mathbb{K}^{(k')} \to \mathbb{K}^{(k+k')}$ (for $k, k' \in \mathbb{N}$) is defined by $\times(K_1, K_2) := K_1 \times K_2$.

(3) The operator Shift is computable in linear time, where $Shift : \mathbb{K}^{(1)} \times \mathbb{R} \to \mathbb{K}^{(1)}$ is defined by $Shift(K, r) := \{x + r \mid x \in K\}$.

(4) The operator Scale is computable in polynomial time, where $Scale : \mathbb{K}^{(1)} \times \mathbb{R} \to \mathbb{K}^{(1)}$ is defined by $Scale(K, r) := \{x \cdot r \mid x \in K\}$.

(5) The operator Slant is computable in polynomial time, where $Slant : \mathbb{K}^{(2)} \times \mathbb{R} \to \mathbb{K}^{(2)}$ is defined by $Slant(K, r) := \{(x + ry, y) \mid (x, y) \in K\}$.

(6) The operator Rotate is computable in polynomial time, where $Rotate : \mathbb{K}^{(2)} \times \mathbb{R} \to \mathbb{K}^{(2)}$ is defined by
$Rotate(K, r) := \{(x\cos(r\pi) - y\sin(r\pi), x\sin(r\pi) + y\cos(r\pi)) \mid (x, y) \in K\}$.

Proof. (1) For the union we have two compact sets as arguments, i.e. $m = 2$, $k_1 = k_2 = k' = k$, and $C_1 = C_2 = C' = \mathbb{K}^{(k)}$ (corresponding to definition 2.6).

We construct a two-oracle Turing machine M as follows: Let $\psi_1 = (h_1, b_1), \psi_2 = (h_2, b_2)$ be oracles for two compact sets $K_1, K_2 \in \mathbb{K}^{(k)}$. Given these oracles, we have to consider inputs of $(n, \overline{d})$ with $n \in \mathbb{N}$ and $\overline{d} \in \mathbb{D}_n^k$ as well as the empty input.

For the empty input, M^{ψ_1,ψ_2} calls each oracle once, both with the empty query, resulting in the grid bounds b_1 and b_2. Then M^{ψ_1,ψ_2} simply returns $b := \max\{b_1, b_2\}$.

For non-empty input $(n, \overline{d})$, the machine M^{ψ_1,ψ_2} again first determines b as above and tests whether $\overline{d} \in \mathbb{D}_n^k \cap [-2^b; 2^b]^k$. If not, M^{ψ_1,ψ_2} returns 0.

Otherwise, M^{ψ_1,ψ_2} calls each oracle once more, but now with the query $(n, \overline{d})$. Let v_1 and v_2 be the two resulting bits. Then M^{ψ_1,ψ_2} returns $v := v_1 \vee v_2$.

Obviously, M^{ψ_1,ψ_2} computes the union of K_1 and K_2: b is the correct minimal bound for $B_{h_1}(0) \cup B_{h_2}(0)$ and v corresponds to correct values of the grid indicator for the union $B_{h_1}(n) \cup B_{h_2}(n)$.

Suppose $s \geq \text{size}(K_1, K_2)$. Then $b_1, b_2 \leq s + 1$. The computation time of M^{ψ_1,ψ_2} for the empty input, i.e. for b, is dominated by the computation of the maximum, so it is linear in s (because of the unary representation of b_1, b_2) and independent from n. For non-empty input the time is the sum of the time necessary to compute b, to test the input (linear in $n+s$) and then perhaps create the queries (again linear in $n+s$) and to compute v (constant time). Thus M runs in time $O(n+s)$, i.e. in 'linear time'.

(2) The result for the complexity of the crossproduct can be shown in a similar way: The only modification is that for an input $(n, (d_1, .., d_{k+k'}))$ we simply call the first oracle with $(n, (d_1, .., d_k))$ and the second one with $(n, (d_{k+1}, .., d_{k+k'}))$ instead of the identical queries mentioned above.

(3) For the shifting we have two arguments and a one-dimensional, set-valued result, i.e. $m = 2$, $k_1 = k_2 = k' = 1$. We shall construct a two-oracle Turing machine M to compute the shifting operator. Let $\psi = (h, b)$ be an arbitrary grid name for a compact set K, and let ϕ be a name of a number $r \in \mathbb{R}$.

$M^{\psi,\phi}$ has to compute the values $h'(n, d)$ (for $n \in \mathbb{N}, d \in \mathbb{D}_n$) of a grid indicator h' and a consistent grid bound $b' \in \mathbb{N}$:

In any computation, $M^{\psi,\phi}$ initially calls ϕ with query $n = 0$, yielding a result d' with $|r - d'| \leq 1$. Then $M^{\psi,\phi}$ calls the oracle ψ with the empty query to determine b. Let $\beta := b + \log_2 |d'| + 2$. Then surely $\mathit{Shift}(K, r) \subseteq [-2^\beta, 2^\beta]$

For a non-empty input (n, d), $h'(n, d)$ is computed as follows:

- If $|d| > 2^\beta$, then $h'(n, d) := 0$.
- In any other case, compute the following four values, all from $\mathbb{D}_{n+2}$:

$$T(n, d) = \{d + i \cdot 2^{-n-2} - \phi(n+2) \mid i \in \{-1, 0, 1, 2\}\}$$

  Then test whether $h(n+2, d') = 1$ for at least one $d' \in T(n, d)$. If this is the case, then $h'(n, d) := 1$, otherwise again $h'(n, d) := 0$.

We show that h' characterizes a grid for $\mathit{Shift}(K, r)$.

Suppose $h'(n, d) = 1$. Then $h(n+2, d') = 1$ for a value $d' \in T(n, d)$. So there are x, i with $x \in K$ and $|i| \leq 2$ such that $|x - d'| \leq 2^{-n-2}$ and $d' = d + i \cdot 2^{-n-2} - \phi(n+2)$. Thus, $|d - (x+r)| = |d' - i \cdot 2^{-n-2} + \phi(n+2) - (x + r)| \leq |d' - x| + |i \cdot 2^{-n-2}| + |\phi(n + 2) - r| \leq 2^{-n}$. Please note $x+r \in \mathit{Shift}(K, r)$ because $x \in K$.

On the other hand, consider any $x+r \in \mathit{Shift}(K, r)$. Then $x \in K$, hence there is $d' \in \mathbb{D}_{n+2}$ such that $h(n+2, d') = 1$ and $|x - d'| \leq 2^{-n-2}$. Additionally there is a $d \in \mathbb{D}_n$ such that $d' \in T(n, d)$. Thus, $|d - (x+r)| \leq |d - (d' - \phi(n+2))| + |d' - x| + |r - \phi(n+2)| \leq 2^{-n}$. Please note that $h'(n, d) = 1$.

For the empty query, M computes b' by explicit construction of $B'(0)$. This is possible by testing all values $d \in [-2^\beta, 2^\beta] \cap \mathbb{D}_0$ whether $h'(0, d) = 1$.

Since the subtraction of dyadic numbers costs not more than linear time, it follows that $M^{\phi,\psi}$ runs time linear in n (but exponential in s because of the computation of b').

(4)-(6) These cases can be shown with a similar construction. □

If in definition 2.6 we use the special case $m = 0$, i.e. we consider 0-ary operators, we get a notion of polynomial time complexity for real vectors and for compact sets.

For vectors, the definition is equivalent to the usual definition of polynomial computability, e.g. in [8,13]. For compact sets, the definition corresponds to definition 7.4.3(5) in [13]. The following lemma gives a few examples. As the results are quite obvious, we omit the proofs.

Lemma 2.8 (i) *A vector $\overline{x} \in \mathbb{R}^k$ is computable in polynomial time if and only if the singleton set $K := \{\overline{x}\}$ is also computable in polynomial time.*

(ii) *If a convex polygon $K \subset \mathbb{R}^2$ has vertices $\overline{x}_1, \ldots, \overline{x}_l \in \mathbb{R}^2$ that all are computable in polynomial time, then K is computable in polynomial time, too.*

A useful notion of computational complexity should be closed under composition. In the following we show that this is also true for our definition of parameterized complexity.

Theorem 2.9 *Let F, G be two operators such that range$(F) \subseteq$ dom(G). Suppose F is computable on a set C, and G is computable on a set C' with $F(C) \subseteq C'$, both in polynomial time. Then the operator $H := G \circ F$ is also computable in polynomial time on C.*

Proof. Let M_1, M_2 be oracle Turing machines computing F and G, respectively, with (monotonic increasing) complexity bounds t_1 and t_2 valid on C and C'. Using appropriate values $a, l \in \mathbb{N} \setminus \{0\}$, and $q : \mathbb{N}^2 \to \mathbb{N} \setminus \{0\}$, both $t_1(n, s)$ and $t_2(n, s)$ are bounded by $a \cdot n^l \cdot q(s)$ for $n > 0$ and by $q(s)$ for $n = 0$.

We construct a machine N as usual for the composition of oracle machines: Given $x \in C$ via an oracle Ψ for x and given some input w, N starts simulating M_2. Whenever N needs to simulate an oracle call of M_2, N simulates M_1 with oracle Ψ instead, using the oracle query as input for M_1. Obviously, N computes H in the sense of definition 2.6.

To show that N works in polynomial time, we need to consider how the dependency of the complexity on the size behaves under the composition. Essentially, we simply add the computation times, but we have to be careful about the length of the oracle queries that N has to simulate for M_1. So let $s_x := size(x)$ and let p_w be the precision necessary for input w (i.e. $p_w = 0$ for the empty query, $p_w = n$ for queries of form 'n?' or '$(n, d_1, .., d_{k'})$?').

M_1^Ψ must be able to answer the empty query as well as the query 0? in time $t_1(0, s_x) \leq q(s_x)$. Because of the restrictions to the notation of the query results, we are able to conclude $size(F(x)) \leq q(s_x)$.

The total computation time $T_N^\Psi(w)$ of N^Ψ on w is bounded by the sum of

- $t_2(p_w, q(s_x))$ for the simulation of M_2 (which is a bound for the number of queries to the oracle of M_2, i.e. simulations of M_1, and also for the length of those queries) and
- $t_1(t_2(p_w, q(s_x)), s_x)$ for each single simulation of M_1.

This leads to a total of

$$T_N^\Psi(w) \leq t_2(p_w, q(s_x)) \cdot (1 + t_1(t_2(p_w, q(s_x)), s_x))$$

$$\leq 2a^{l+2} \cdot p_w^{l^2+l} \cdot (q(q(s_x))^{l+1})q(s_x)$$

which is again polynomial in the precision parameter p_w.

□

3 Complexity of Projection

We define the projection of a compact set $K \in \mathbb{K}^{(2)}$ as:

$$Proj(K) := \{x \mid (x, y) \in K \text{ for some } y\}.$$

Theorem 3.1 *The operator Proj is computable. Its complexity has exponential upper and lower bounds.*

Proof. (1) We first show that the projection is computable and has exponential upper bounds for its complexity. Construct an oracle Turing machine M computing $Proj$ for any compact set K and any grid name $\psi = (h, b)$ of K as follows:

For any query (n, d), M^ψ first determines b and then searches whether there is a $d' \in \mathbb{D}_n \cap [-2^b, 2^b]$ such that $h(n, d, d') = 1$ by calling the oracle ψ sufficiently often. If the search is successful, then M^ψ returns 1, otherwise M^ψ returns 0. Let h' be the function defined by this part of the computation.

To show that h' is a grid indicator of $Proj(K)$, first consider (n, d) with $h'(n, d) = 1$. Then there is d' such that $h(n, d, d') = 1$, so there is $(x, y) \in K$ such that $(x, y) \in U_n(d, d')$. Hence $|d - x| \leq 2^{-n}$, ie. $x \in U_n(d)$. Please note that $x \in Proj(K)$.

On the other hand, consider any $x \in Proj(K)$. Then $(x, y) \in K$ for some y. Thus, there are d, d' such that $h(n, d, d') = 1$ (hence $h'(n, d) = 1$) and $(x, y) \in U_n(d, d')$ (hence $x \in U_n(d)$).

If, in the computation of h', the search is done in a exhaustive way for all possible $d' \in \mathbb{D}_n \cap [-2^b, 2^b]$, we get an upper bound on the complexity of M that is obviously exponential in n.

The correct grid bound b' for h' may not be larger than b, so it can be determined using h' and b, as mentioned as a remark to definition 2.2.

(2) Now suppose there were a machine N such that the worst case for the run time of N really were below $2^n \cdot q(s) \cdot \frac{1}{8 \cdot q(0)}$ for a given q. Or as an alternative: Suppose the run time were in $o(2^n) \cdot q(s)$ for an arbitrary q.

Just consider compact sets $\subseteq [-1, 1]^2$: Their size is 0, so there would be values $\overline{n}$ such that N needs less than $2^{\overline{n}-3}$ on the input $(\overline{n}, d)$ (for arbitrary d).

For such an $\overline{n}$ consider the special compact set $K = \{2 \cdot 2^{-\overline{n}}\} \times [-1, 1]$ with $Proj(K) = \{2 \cdot 2^{-\overline{n}}\}$ and consider the corresponding grid name $\psi = (h, b)$ where $b := 0$ and

$$h(n, d, d') := \begin{cases} 1 \text{ if } d' \in [-1, 1] \wedge n < \overline{n} \wedge d = 0 \\ 1 \text{ if } d' \in [-1, 1] \wedge n \geq \overline{n} \wedge d = 2 \cdot 2^{-\overline{n}} \\ 0 \text{ else} \end{cases}$$

In the following we want to define a different compact set $\overline{K}$ such that N is some-times unable to distinguish between K and $\overline{K}$. So let

$$Q_{\overline{n}} := \{d' \mid \text{there is a query } (n,d,d') \text{ during one of the computations of } N^{\psi} \text{ on } (\overline{n}, -1{\cdot}2^{-\overline{n}}), (\overline{n}, -2{\cdot}2^{-\overline{n}}) \text{ or } (\overline{n}, -3{\cdot}2^{-\overline{n}})\ \}$$

By construction of $\overline{n}$, there may be at most $2^{\overline{n}-3}$ queries to ψ for each input, hence $\#Q_{\overline{n}} \le 3\cdot 2^{\overline{n}-3}$. As $\#(\mathbb{D}_{\overline{n}} \cap [-1,1]) = 2^{\overline{n}+1}+1$, there must be a $\overline{d} \in \mathbb{D}_{\overline{n}} \cap [-1,1] \setminus Q_{\overline{n}}$.

Now consider the (compact) set $\overline{K} := K \cup \{(-2{\cdot}2^{-\overline{n}}, \overline{d})\}$ with $Proj(\overline{K}) = \{-2{\cdot}2^{-\overline{n}}, 2{\cdot}2^{-\overline{n}}\}$. Here a grid name $\overline{\psi} = (\overline{b}, \overline{h})$ valid for $\overline{K}$ is given by $\overline{b} := 0$ and

$$\overline{h}(n,d,d') := \begin{cases} 1 \text{ if } d' \in [-1,1] \wedge n < \overline{n} \wedge d = 0 \\ 1 \text{ if } d' \in [-1,1] \wedge n \ge \overline{n} \wedge d = 2{\cdot}2^{-\overline{n}} \\ 1 \text{ if } d' = \overline{d} \wedge n \ge \overline{n} \wedge d = -2{\cdot}2^{-\overline{n}} \\ 0 \text{ else} \end{cases}$$

As N^{ψ} is assumed to compute $Proj(K)$, the output of N^{ψ} must be 0 for any input of form $(\overline{n}, d)$ with $d \le 0$. By construction of $\overline{h}$, the machine N is unable to distinguish between ψ and $\overline{\psi}$ at least for the three special inputs $(\overline{n}, -1{\cdot}2^{-\overline{n}})$, $(\overline{n}, -2{\cdot}2^{-\overline{n}})$ and $(\overline{n}, -3{\cdot}2^{-\overline{n}})$, i.e. $N^{\overline{\psi}}$ must also be 0 for these three values. So the distance between $-2{\cdot}2^{-\overline{n}}$ and $B_{\overline{h}}(\overline{n})$ would be at least $2^{1-\overline{n}}$, i.e. too big. □

The construction in part (1) of the proof above shows a close connection to the classical problem whether P=NP:

Theorem 3.2 *P=NP if and only if for every polynomial time computable K, the projection $Proj(K)$ is again polynomial time computable.*

Proof. (1) Consider a fixed polynomial time computable K and let N be a corresponding Turing machine computing a grid name $\psi = (h, b)$ of K in polynomial time. As the size of K is fixed, the set $U := \{(n,d,d') \mid d, d' \in \mathbb{D}_n \wedge h(n,d,d') = 1\}$ is computable in polynomial time according to the usual definition from discrete complexity theory.

So the set $V = \{(n,d) \mid (\exists d' \in \mathbb{D}_n)\, h(n,d,d') = 1\}$ is in NP. The characteristic function of this set V is the function h' constructed at the beginning of the proof of theorem 3.1. So if NP=P, then $Proj(K)$ is also polynomial time computable.

(2) Let $A \subset \{0,1\}^+$ be an NP-complete set. Without loss of generality, A can be chosen such that there is a corresponding polynomial time computable set $U \subseteq \{w\#v \mid w, v \in \{0,1\}^+ \wedge len(w) = len(v)\}$ with $A = \{w \mid (\exists v)\, w\#v \in U\}$, where $len(w)$ denotes the length of a string w.

As an example for A we might use the satisfyability problem SAT: The number of different variables in a formula w is trivially bounded by the $len(w)$, so U may simply consist of all pairs $w\#v$, where the initial part of v describes an assignment of truth values to the variables in the formula w that satisfies w, while v is padded at the end with arbitrary bits until $len(w) = len(v)$.

In the following we construct a compact set K such that A corresponds to $Proj(K)$. We will use a similar construction as in part (2) of the proof of theorem 3.1: The strings $w\#v$ will be mapped to discrete points $(\overline{w}, \widehat{v}) \in \mathbb{D}^2$, additionally there will be (many) straight lines dividing $[-1,1]^2$ into small subregions that can be treated independently.

For any string $w \in \{0,1\}^+$ define $\underline{w}$, $\overline{w}$, and $\widehat{w}$ to be the dyadic numbers

$$\underline{w} := 2^{-len(w)} + 2^{-2len(w)} \cdot bin(w)$$
$$\overline{w} := \underline{w} + 2^{-2len(w)-1}$$
$$\widehat{w} := 2^{-len(w)-1} + 2^{-len(w)} \cdot bin(w)$$

where $bin(w) \in \mathbb{N}$ is the value of the string w using ordinary binary notation. For example, the string 101 corresponds to the three dyadic numbers $\underline{101} = +.001\,101$, $\overline{101} = +.001\,101\,1$, and $\widehat{101} = +.101\,1$.

Now consider the set $K := K_0 \cup K_1 \cup K_2$ with

$$K_0 := \{(\overline{w}, \widehat{v}) \mid w\#v \in U\}$$
$$K_1 := \{\underline{w} \mid w \in \{0,1\}^+ \} \times \{x \in \mathbb{R} \mid 0 \le x \le 1\}$$
$$K_2 := \{0,1\} \times \{x \in \mathbb{R} \mid 0 \le x \le 1\}$$

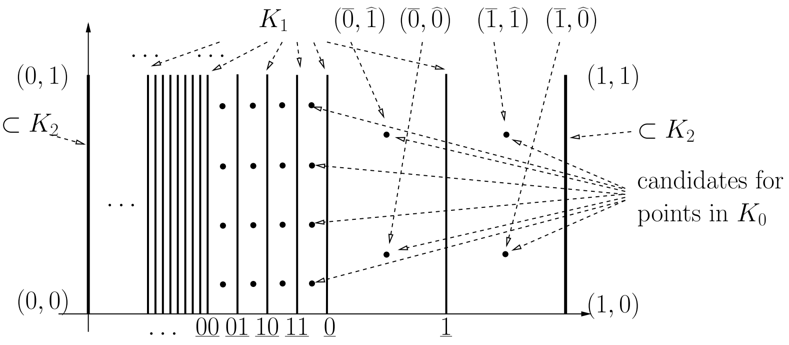
K_1
$(\overline{0},\widehat{1})$
$(\overline{0},\widehat{0})$
$(\overline{1},\widehat{1})$
$(\overline{1},\widehat{0})$
$(0,1)$
$\subset K_2$
$(1,1)$
$\subset K_2$
candidates for points in K_0
$(0,0)$
$(1,0)$
. . . $\underline{00}$ $\underline{01}$ $\underline{10}$ $\underline{11}$ $\underline{0}$
$\underline{1}$

Fig. 2. The sets K_0, K_1 and K_2

K_0 encodes U into dyadic numbers, K_1 separates points from K_0 with different x coordinates, and the purpose of K_2 is to add missing accumulation points to get a compact set and to get a proper enclosure to the set. So K is compact and

$$Proj(K) = \{0,1\} \cup \{\underline{w} \mid w \in \{0,1\}^+\} \cup \{\overline{w} \mid w \in A\}$$

Although testing the inclusion of a point in a compact set is a non-computable operation in general, the situation is different for the well-structured set $Proj(K)$: Let $\psi = (h,b)$ be a grid name for $Proj(K)$. To decide whether $w \in A$ for an $w \in \{0,1\}^+$, let $n = 2 \cdot len(w) + 2$ and check whether $h(n,d) = 1$ for at least one of the three dyadic values $d \in \{\overline{w} - 2^{-n}, \overline{w}, \overline{w} + 2^{-n}\}$.

A valid grid name for K itself is $\psi = (h, b)$ with $b = 0$ and

$$h(n,d,d') := \begin{cases} 1 \text{ if } d' \in [0,1] \wedge d \in \{0,1\} & (\text{for } K_2) \\ 1 \text{ if } d' \in [0,1] \wedge (\exists w) \;\; d = \underline{w} & (\text{for } K_1) \\ 1 \text{ if } (\exists w)(\exists v) \;\; d = \overline{w}, d' = \widehat{v}, w\#v \in U & (\text{for } K_0) \\ 0 \text{ else} \end{cases}$$

for all $n \in \mathbb{N}$ and $d, d' \in \mathbb{D}_n$. Please note that the existential quantifiers in the definition of h do not stand for a search but just for a syntactical check of the form of d and d'.

So obviously, if U is computable in polynomial time, then this also holds for K. Using the assumption that this implies polynomial time computability of $Proj(K)$, $A \in P$ would follow. □

Theorem 3.2 shows that the usual impression that a compact set naturally has some kind of inner structure is rather wrong: In the previous proof we were able to code infinitely many points (almost arbitrarily chosen) into a compact set. So an interesting question is to ask what additional properties of the compact sets K are required such that theorem 3.2 or even theorem 3.1 can no longer be shown.

Obviously, we are able to extent theorem 3.1 to connected compact sets: In the proof, we only have to connect the set K with the constructed point $(-2{\cdot}2^{-\overline{n}}, \overline{d})$ using the line segment $\overline{(-2{\cdot}2^{-\overline{n}}, \overline{d})(0, \overline{d})}$. Unfortunately, this simple modification can not be used in the proof of theorem 3.2, as the resulting set would still not be a connected set.

The following lemma shows that for e.g. convex compact sets, we are able to compute the projection efficiently (which is interesting as we are also able to rotate compact sets).

Lemma 3.3 *Restricted to convex sets, the operator Proj is computable in polynomial time.*

Proof. Let K be a convex compact set given by a grid name $\psi = (h, b)$. Because of the convexity, $Proj(K)$ is a closed interval $[l_K, r_K]$ with $l_K = \min\{x \mid (\exists y)\,(x, y) \in K\}$ and $r_K = \max\{x \mid (\exists y)\,(x, y) \in K\}$. So to determine whether a $d \in \mathbb{D}$ is sufficiently near to the $Proj(K)$, we only need to approximate l_K and r_K.

In the following we concentrate on l_K. So let $(l_K, y_K) \in K$ be an arbitrary point in K with first coordinate l_K. We describe an iterative algorithm that, for any $n \in \mathbb{N}$, finds a point $(\widehat{d'_n}, \widehat{d_n})$ with

(i) $h(n, \widehat{d'_n}, \widehat{d_n}) = 1$, so $l_K \le \widehat{d'_n} + 2^{-n}$, and

(ii) $l_K > \widehat{d'_n} - 2^{2-n}$.

Please note that it is not necessary that also $|\widehat{d_n} - y_K|$ is small.

For the initial case $n = 0$, a suitable point $(\widehat{d'_0}, \widehat{d_0})$ can be found using an exhaustive search on all $(d', d) \in \mathbb{D}_0^2 \cap [-2^b, 2^b]^2$, simply taking the smallest d' such that $h(0, d', d) = 1$.

Now consider the step from $n-1$ to n and suppose $(\widehat{d'}_{n-1}, \widehat{d}_{n-1})$ has already been found.

First we construct a small rectangle containing (l_K, y_K): Using (i) and (ii), the left edge can be chosen on the line $\{\widehat{d'}_{n-1} - 2^{3-n}\} \times \mathbb{R}$ and an even closer right edge on the line $\{\widehat{d'}_{n-1} + 2^{1-n}\} \times \mathbb{R}$. The upper and lower edges we are looking for should be such that the resulting rectangle is almost minimal: It should contain points from K near these upper and lower edges. In detail, the construction of the rectangle will be performed as follows (see figure 3):

Although $h(n-1, \widehat{d'}_{n-1}, \widehat{d}_{n-1}) = 1$, we do not know whether $h(n, \widehat{d'}_{n-1}, \widehat{d}_{n-1}) = 1$. But there must be a point (d'_n, d_n) such that $h(n, d'_n, d_n) = 1$ and $|\widehat{d'}_{n-1} - d'_n| \le 3 \cdot 2^{-n}$ as well as $|\widehat{d}_{n-1} - d_n| \le 3 \cdot 2^{-n}$.

Let $I_n = \{\widehat{d'}_{n-1} + i \cdot 2^{-n} \mid -8 \le i \le 4\}$. Then we search for two values $\underline{d}_n$ and $\overline{d}_n$ such that

- $-2^b \le \underline{d}_n \le d_n \le \overline{d}_n \le 2^b$
- $h(d', \underline{d}_n) = 1$ for at least one value $d' \in I_n$
- $h(d', \overline{d}_n) = 1$ for at least one value $d' \in I_n$
- $h(d', d) = 0$ for all values $(d', d) \in I_n \times \{\underline{d}_n - i \cdot 2^{-n} \mid i \in \{1, 2, 3\}\}$
- $h(d', d) = 0$ for all values $(d', d) \in I_n \times \{\overline{d}_n + i \cdot 2^{-n} \mid i \in \{1, 2, 3\}\}$

As $h(n, d'_n, d_n) = 1$, such values $\underline{d}_n$ and $\overline{d}_n$ do exist. The search for these values should be implemented as a binary search, where the initial borders of the search are d_n and -2^b for $\underline{d}_n$, and d_n and 2^b for $\overline{d}_n$. So the number of search steps is linear in $b + n$. Please note that we do not require $\underline{d}_n$ or $\overline{d}_n$ to be different from d_n.

For the next step in the construction, we cut the rectangle into eight stripes using seven equally spaced horizontal lines. Due to the convexity of K and the minimality of the rectangle, each of these lines must (almost) intersect the border of K. Again due to the convexity of K, the leftmost intersection point must have an x-coordinate very close to l_K.

In detail, we define nine almost equally spaced areas in $I_n \times \{d \in \mathbb{D}_n \mid \underline{d}_n \le d \le \overline{d}_n\}$ (corresponding to the lower and upper edge of the rectangle and to the seven horizontal lines) where we want to search for the new approximation $(\widehat{d'}_n, \widehat{d}_n)$:

For $1 \le \nu \le 7$ consider $d_n^{(\nu)} :=$ the value $\in \mathbb{D}_n$ closest to$(\nu \cdot \overline{d}_n + (8-\nu) \cdot \underline{d}_n)/8$ and let

$$J_n := \{\overline{d}_n, \underline{d}_n\} \cup \{d_n^{(\nu)} + i \cdot 2^{-n} \mid \nu \in \{1, ..., 7\}, i \in \{-1, 0, 1\}\}$$

So $I_n \times J_n$ consists of $13 \cdot (2 + 7 \cdot 3) = 299$ points or less, in case that $\overline{d}_n \approx \underline{d}_n$ and the areas overlap.

Finally $\widehat{d'}_n := \min\{d' \in I_n \mid (\exists d \in J_n)\, h(n, d', d) = 1\}$ and as $\widehat{d}_n$ we use one of the values where $h(n, \widehat{d'}_n, \widehat{d}_n) = 1$. By construction of $\underline{d}_n$ and $\overline{d}_n$, such a pair $(\widehat{d'}_n, \widehat{d}_n)$ does exist.

Using the convexity of K, we know that $\underline{d}_n - 2^{-n} \le y_K \le \overline{d}_n + 2^{-n}$ and that within K there must exist straight lines from (l_K, y_K) to a point in K near $I_n \times \{\underline{d}_n\}$

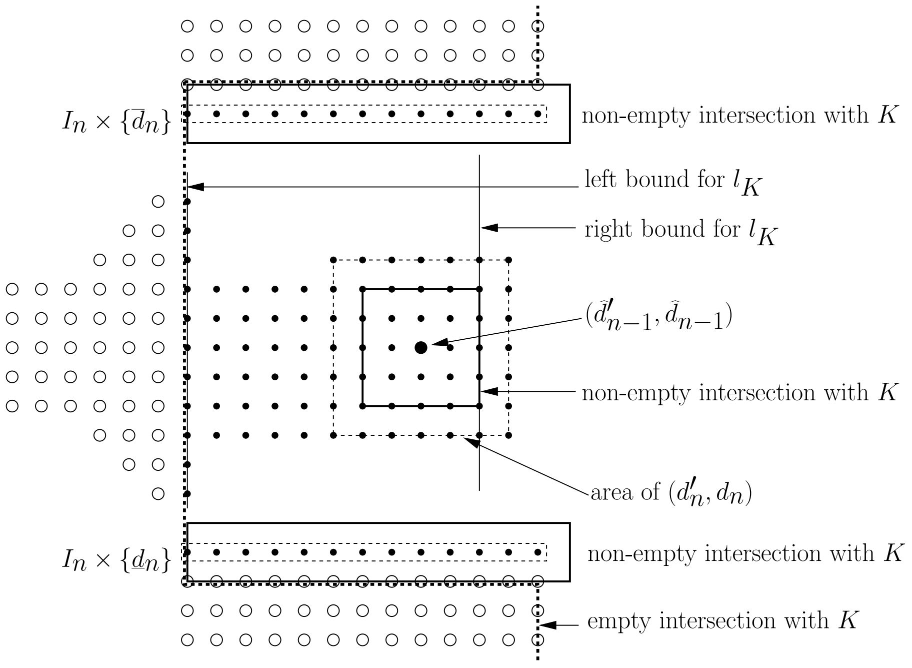
$I_n \times \{\overline{d}_n\}$
non-empty intersection with K
left bound for l_K
right bound for l_K
$(\widehat{d'}_{n-1}, \widehat{d}_{n-1})$
non-empty intersection with K
area of (d'_n, d_n)
$I_n \times \{\underline{d}_n\}$
non-empty intersection with K
empty intersection with K

Fig. 3. Projection of convex sets; constructing the enclosing rectangle from $(\widehat{d'_n}, \widehat{d_n})$.

as well as to a point in K near $I_n \times \{\overline{d}_n\}$. As the inner search areas are three points wide, any such line crossing a search area must correspond to least one point with $h(n, d, d') = 1$ in the search area and with a distance of at most 2^{-n} from the crossing point.

But, as our nine search areas are equally spaced, there must be such a crossing point with an x-coordinate c such that $|l_K - c|$ is at most one fifth of the maximal difference of y coordinates under consideration, i.e. $|l_K - c| \leq \frac{1}{5} \cdot 14{\cdot}2^{-n} < 3 \cdot 2^{-n}$. So in one of the search areas there must be a point (d', d) with $h(d', d) = 1$ and where $l_K > d' - 2^{2-n}$, which concludes our construction.

Because of the linear number of the binary search steps in each iteration, the complexity of the construction of $(\widehat{d'_n}, \widehat{d_n})$ is surely polynomial in n.

Finally, to compute a grid name $\psi' = (h', b')$ for $Proj(K)$, for query (n, d') we may approximate l_K using $\widehat{d'}_{n+3}$ and a similar approximation $\tilde{d}'_{n+3}$ for r_K: If $\widehat{d'}_{n+3} - 2^{-n-1} \leq d \leq \tilde{d}'_{n+3} + 2^{-n-1}$, then we let $h'((n, d') = 1$. b' can be computed as mentioned as a remark to definition 2.2. □

4 Convex Hull of Compact Sets

In the previous section we have seen that the projection operator in general has exponential complexity, unless the underlying set is convex. So computing a convex hull of a compact set must also be a complicated operation. Now we have a deeper look into this, especially as we were unable to extend the results for the projection to connected sets.

The convex hull operator $Chull : \mathbb{K}^{(2)} \to \mathbb{K}^{(2)}$ is defined as follows: $Chull(K)$ is the smallest convex set containing K.

Theorem 4.1 *The operator Chull is computable. Its complexity has exponential upper and lower bounds.*

Proof. (1) We construct an oracle Turing machine M to compute the convex hull of an arbitrary compact set given as an oracle: So let K be compact and let $\psi = (h, b)$ be an arbitrary grid name for K. For inputs n and $d_1, d_2 \in \mathbb{D}_n$, the machine M^ψ checks (by an exhaustive search) whether there are points $\overline{d}^{(i)} \in \mathbb{D}^2_{n+2}$ (for $i \in \{1, 2, 3\}$) such that $h(n{+}2, \overline{d}^{(i)}) = 1$ and that the distance from (d_1, d_2) to the triangle $\Delta\overline{d}^{(1)}\overline{d}^{(2)}\overline{d}^{(3)}$ is not greater than 2^{-n-2}. If so, return 1, otherwise return 0. Please notice that the check for the distance between a dyadic point and a triangle with dyadic corners can be computed in time polynomial in the length of the arguments. Thus, M runs in exponential time.

It is straightforward (so we omit details) to show that the grid indicator h' computed by M^ψ corresponds to $Chull(K)$: In the neighborhood of each of the points $\overline{d}^{(i)}$ there is a point $\overline{x}^{(i)} \in K$ (and vice versa), so for each point from $\Delta\overline{d}^{(1)}\overline{d}^{(2)}\overline{d}^{(3)}$ there is a nearby point in the triangle $\Delta\overline{x}^{(1)}\overline{x}^{(2)}\overline{x}^{(3)}$ (and vice versa). The latter triangles again determine $Chull(K)$.

(2) An exponential lower bound can be shown with a similar construction as in the proof for the projection: Now suppose N is a machine computing the convex hull. We are able to use the idea of construction of the set K from theorem 3.1 again: Let now $\overline{n}$ be such that, for K of with size 0, N needs less than $2^{\overline{n}-5}$ steps on inputs $(\overline{n}, d, d')$ (for arbitrary d, d').

Again, let K be defined as $K = \{2{\cdot}2^{-\overline{n}}\} \times [-1, 1]$ and use the same grid name $\psi = (h, b)$ as in theorem 3.1, but now consider the nine special inputs

$$I_{\overline{n}} = \{\ (\overline{n}, d, d')\ \mid\ d \in \{-1{\cdot}2^{-\overline{n}}, -2{\cdot}2^{-\overline{n}}, -3{\cdot}2^{-\overline{n}}\} \wedge d' \in \{-2^{-\overline{n}}, 0, 2^{-\overline{n}}\}\ \}$$

Let

$$\begin{aligned} Q_{\overline{n}} := \{d' \mid\ & \text{there is a query } (n, d, d') \text{ to } \psi \text{ during one} \\ & \text{of the computations of } N^\psi \text{ on inputs from } I_{\overline{n}}\ \} \end{aligned}$$

By construction of $\overline{n}$, there may be at most $2^{\overline{n}-5}$ queries to ψ for each input, hence $\#Q_{\overline{n}} \le 9 \cdot 2^{\overline{n}-5}$. As $\#(\mathbb{D}_{\overline{n}} \cap [-1, 0.5]) = 2^{\overline{n}-1} + 1$, now there must be $\overline{d}_1 \in \mathbb{D}_{\overline{n}} \cap [-1, 0.5] \setminus Q_{\overline{n}}$ and also $\overline{d}_2 \in \mathbb{D}_{\overline{n}} \cap [0.5, 1] \setminus Q_{\overline{n}}$.

Clearly the point $(-2{\cdot}2^{-\overline{n}}, 0)$ is not in $Chull(K)$, so now consider the (compact) set $\overline{K} := K \cup \{(-2{\cdot}2^{-\overline{n}}, \overline{d}_1), (-2{\cdot}2^{-\overline{n}}, \overline{d}_2)\}$ with $(-2{\cdot}2^{-\overline{n}}, 0) \in Chull(\overline{K})$ and the grid name $\overline{\psi} = (\overline{b}, \overline{h})$ with $\overline{b} := b$ and

$$\overline{h}(n,d,d') := \begin{cases} 1 \text{ if } d' \in [-1,1] \wedge n < \overline{n} \wedge d = 0 \\ 1 \text{ if } d' \in [-1,1] \wedge n \geq \overline{n} \wedge d = 2{\cdot}2^{-\overline{n}} \\ 1 \text{ if } d' = \overline{d}_1 \wedge n \geq \overline{n} \wedge d = -2{\cdot}2^{-\overline{n}} \\ 1 \text{ if } d' = \overline{d}_2 \wedge n \geq \overline{n} \wedge d = -2{\cdot}2^{-\overline{n}} \\ 0 \text{ else} \end{cases}$$

Again, h and $\overline{h}$ do not differ on $[-1,1] \times Q_{\overline{n}}$, so N^{ψ} and $N^{\overline{\psi}}$ yield the same output for the inputs from $I_{\overline{n}}$, i.e. $N^{\overline{\psi}}$ has to behave in a wrong way for $(-2{\cdot}2^{-\overline{n}}, 0) \in Chull(\overline{K})$. □

Lemma 4.2 *Suppose P=NP. Then $Chull(K)$ is poly-time computable for each poly-time computable K.*

Proof. Consider part (1) of the proof of theorem 4.1: We had to examine the set $\{(n, d_1, d_2) \mid \exists \overline{d}^{(1)}, \overline{d}^{(2)} \overline{d}^{(3)} \in \mathbb{D}^2_{n+2} \ldots\}$. For a fixed, polynomial time computable K this is obviously a problem in NP. So if P=NP, then $Chull(K)$ even is in P. □

Theorem 4.3 *If $Chull(K)$ is polynomial time computable for each polynomial time computable K, then P=NP.*

Proof. As in theorem 3.2, let $A \subset \{0,1\}^+$ be an NP-complete set such that there is a corresponding polynomial time computable set $U \subseteq \{w\#t \mid w,t \in \{0,1\}^+ \wedge len(w) = len(t)\}$ with $A = \{w \mid (\exists t)\, w\#t \in U\}$, where $len(w)$ denotes the length of a string w.

We want to construct a compact set K (based on U) such that A can be decided using $Chull(K)$. K will be computable in polynomial time, and A would be polynomial time decidable if $Chull(K)$ were also computable in polynomial time.

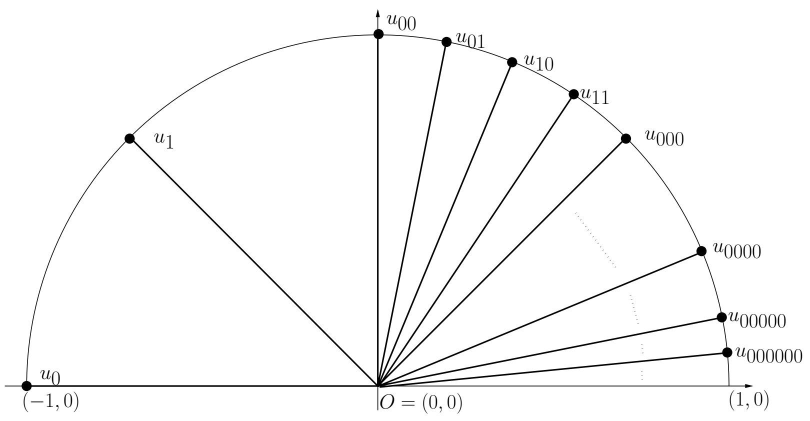

Fig. 4. Dividing the half circle into infinitely many parts.

To construct K, we divide the half circle with center $(0,0)$ and radius 1 into an infinite number of arcs, each corresponding to a string $w \in \{0,1\}^+$. To do this, we define a linear ordering $w \mapsto w^+$ on the strings: For $w = 1^n$ let $w^+ := 0^{n+1}$, and for $w \neq 1^n$ let w^+ be the lexicographically next string. Similarly, for $w \neq 0$ let w^- be the string preceding w.

For $w = 0$, let $u_w = (-1,0)$. For $w \neq 0$, let u_w be the point on the circle such that $\angle u_{w^-} O u_w = \pi \cdot 2^{-2len(w^-)}$ where $O = (0,0)$. Figure 4 shows the first few steps of this process.

Then, for each w with $n := len(w)$, divide the arc from u_w to u_{w^+} into 2^n equally sized parts, each corresponding to a string $t \in \{0,1\}^n$. That is, we use points $u_{w,t}$ given by $u_{w,0^n} = u_w$, and $\angle u_{w,t} O u_{w,t^+} = (\pi \cdot 2^{-3n})$. For simplicity, let $u_{w,(1^n)^+} := u_{w^+}$.

Each segment line $\overline{Ou_{w,t}}$ has an intersection with the segment line $\overline{u_w u_{w^+}}$, denoted by $v_{w,t}$. Please note that $v_{w,0^n} = u_{w,0^n} = u_w$, $v_{w,(1^n)^+} = u_{w,(1^n)^+} = u_{w^+}$. The case $w = 00$ is depicted in figure 5.

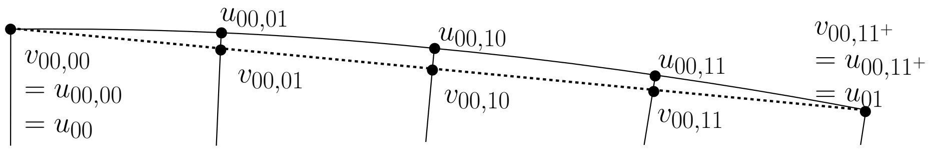

Fig. 5. Subdividing the arc from u_{00} to u_{01} into 4 parts.

If $w\#t \in A$ holds, we define $A_{w,t}$ to be the closed sector $u_{w,t} O u_{w,t^+}$; otherwise, $A_{w,t}$ is the closed triangle $\triangle v_{w,t} O v_{w,t^+}$. An example for $w = 00$ is given in figure 6.

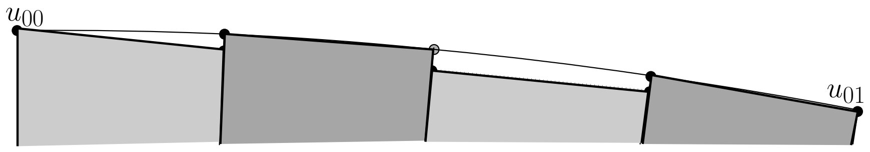

Fig. 6. Defining $A_{w,t}$, here for $00\#01, 00\#11 \in U$ and $00\#00, 00\#10 \notin U$.

Finally we define K to be the closure of $\bigcup_{w \in \{0,1\}^+} \bigcup_{t, len(t) = len(w)} A_{w,t}$. Obviously, K is compact. Since U is polynomial time decidable and the functions sine and cosine are polynomial time computable [1], we conclude that also K is computable in polynomial time.

To see the connection between $Chull(K)$ and A consider the following checkpoints z_w: With $t' := 10^{len(w)-1}$ let $u'_w := u_{w,t'}$ and $v'_w := v_{w,t'}$. Using the point z'_w of intersection between the lines $\overline{u_w u_{w,1^{len(w)}}}$ and $\overline{Ou'_w}$, we let z_w be the middle point of the line $\overline{v'_w, z'_w}$.

It is not hard to see that $w \in A$ if and only if $z_w \in Chull(K)$: If $w \in A$, then the border of $Chull(K)$ intersects the line $\overline{Ou'_w}$ between u'_w and z'_w; if $w \notin A$, then the point of intersection is v'_w. So the distance δ_w from z_w to the border of $Chull(K)$ will always be at least half the length of the line $\overline{v'_w, z'_w}$. By estimating the length

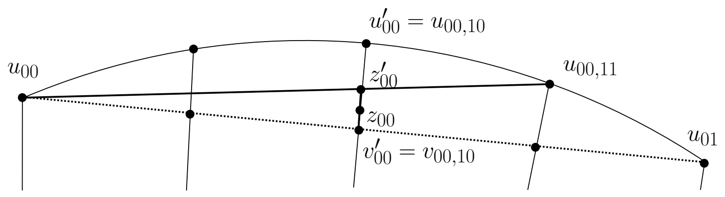

Fig. 7. Construction of the checkpoint z_w, here for $w = 00$. The arc has been enlarged for better readability.

of all concerned line segments, we can see that there is a polynomial $q(n)$ such that $\delta_w > 2^{-q(n)}$.

Now consider any grid name $\psi = (h, b)$ for $Chull(K)$. To test whether $w \in A$, i.e. to test whether there is t such that $w\#t \in U$ holds, it is sufficient to see whether $z_w \in Chull(K)$ or not: So using $m := q(n) + 2$, we first determine $(d'_w, d_w) \in \mathbb{D}_m$ such that $z_w \in U_m(d'_w, d_w)$, then we check whether $h(m, d', d)$ evaluates to 1 for at least one of the nine points in $\{(d', d) \in \mathbb{D}_m^2 \mid |d' - d'_w| \leq 2^{-m}, |d - d_w| \leq 2^{-m}\}$. With exception of the evaluation of h, all of the above steps can be performed in polynomial time.

Now suppose $\psi = (h, b)$ were computable in polynomial time: Then also A would be polynomial time decidable, i.e. NP=P. □

A similar construction with the embedding of NP-complete sets into real functions can be found already in [6], where, among others, the question of finding the maximum value of a function was considered.

The set K constructed in the previous proof is not only compact but also regular (i.e. K is the closure of its interior, $K = \overline{K^\circ}$) and simply connected. So essentially we have even shown the following slightly sharper result:

Corollary 4.4 *P=NP if and only if $Chull(K)$ is polynomial time computable for each simply connected regular compact set K computable in polynomial time.*

References

[1] Brent, R.P., Fast Multiple-Precision Evaluation of Elementary Functions, *Journal of the ACM*, **23**, No.2 (1976), 242-251.

[2] Braverman, M., *Computational Complexity of Euclidean Sets: Hyperbolic Julia Sets Are Poly-time Computable*, Master Thesis, University of Toronto, 2004

[3] Braverman, M., On the Complexity of Real Functions, *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*(2005), 155 - 164

[4] Chou, A.W. & K. Ko, Computational complexity of two-dimensional regions, *SIAM Journal on Computing*, 24:923-947, 1995

[5] Chou, A.W. & K. Ko, , On the complexity of finding paths in a two-dimensional domain I: shortest paths, *Math. Logic Quart.* 50 (2004) 551-572; preliminary version in Proc. Internat. Conf. on Computability and Complexity in Analysis, Hagen, Germany, 2003

[6] Friedman, H., The computational complexity of maximization and integration, *Adv. in Math.*, **53** (1984) 80-98.

[7] Ko, K., On Some Natural Complete Operators, *Theor. Comput. Sci.*, **37**(1985), 1-30

[8] Ko, K., *Complexity Theory of Real Functions*, Birkhäuser, Boston, 1991.

[9] Ko, K., & H. Friedman, Computational complexity of real functions, *Theoret. Comput. Sci.* , **20** (1982) 323-352.

[10] Traub, J.F. & G.W. Wasilkowski & H. Wozniakowski, Information, Uncertainty, Complexity, *Addison-Wesley, New York, 1983*

[11] Traub, J.F. & G.W. Wasilkowski & H. Wozniakowski, Information-Based Complexity, *Academic Press, New York, 1988*

[12] Rettinger, R. & K. Weihrauch, The Computational Complexity of Some Julia Sets, in *STOC03*, San Diego, California, USA, 2003.

[13] Weihrauch, K., *Computable Analysis: An Introduction*, Springer, 2000.

[14] Weihrauch, K., Computable Complexity on Computable Metric Spaces, *Math. Logic Quarterly*, **49**, No.1 (2003), 3-21.

[15] Ziegler, M., Computability on Regular Subsets of Euclidean Space, *Math. Logic Quarterly*, **48**, Suppl. 1 (2002), 157-181.

[16] Ziegler, M., Computable Operators on Regular Sets, *Math. Logic Quarterly*, **50**, No. 4/5 (2004), 392-404.