Hyperbolic Julia Sets are Poly-Time Computable

Mark Braverman [1,2]

Dept. of Computer Science
University of Toronto
Toronto, ON, Canada

Abstract

In this paper we prove that hyperbolic Julia sets are locally computable in polynomial time. Namely, for each complex hyperbolic polynomial $p(z)$, there is a Turing machine $M_{p(z)}$ that can "draw" the set with the precision 2^{-n}, such that it takes time polynomial in n to decide whether to draw each pixel. In formal terms, it takes time polynomial in n to decide for a point x whether $d(x, J_{p(z)}) < 2^{-n}$ (in which case we draw a pixel with center x), or $d(x, J_{p(z)}) > 2 \cdot 2^{-n}$ (in which case we don't draw this pixel). In the case $2^{-n} \le d(x, J_{p(x)}) \le 2 \cdot 2^{-n}$ either answer will be acceptable. This definition of complexity for sets is equivalent to the definition introduced in Weihrauch's book [16] and used by Rettinger and Weihrauch in [13].

Although the hyperbolic Julia sets were shown to be recursive, complexity bounds were proven only for a restricted case in [13]. Our paper is a significant generalization of [13], in which polynomial time computability was shown for a special kind of hyperbolic polynomials, namely, polynomials of the form $p(z) = z^2 + c$ with $|c| < 1/4$.

We show that the machine drawing the Julia set can be made independent of the hyperbolic polynomial p, and provide some evidence suggesting that one cannot expect a much better computability result for Julia sets.

We also introduce an alternative real set computability definition due to Ko, and show an interesting connection between this definition and the main definition.

Key words: computable analysis, Julia sets, computational complexity, complex dynamics.

[1] Research is partially supported by the Natural Sciences and Engineering Research Council of Canada.
[2] Email: mbraverm@cs.toronto.edu

1 Introduction

Nowadays, computers are being increasingly applied to represent mathematical objects. Computer-generated images are being extensively used in the analysis and simulations of real-life processes and their mathematical models. Our goal is to investigate a formal framework which allows us to define the computational complexity of real sets, measuring the complexity of drawing the set on a computer. Within this framework, we obtain a new result on the computability of Julia sets.

We mainly use the definition of real set complexity introduced by Weihrauch in [16] and used in [13] as the measure of complexity of some Julia sets (see also [2]).

In sections 2 and 3 we present two different definitions of computability of real sets that have been proposed, and show that they are equivalent if and only if P=NP, a result of independent interest. Theorem 3.3 can be used to prove computability of many sets for which a direct proof of computability would be hard.

Julia sets are some of the best known illustrations of a highly complicated chaotic system generated by a very simple mathematical process. These sets have been deeply studied in the framework of complex dynamics during the last century. Julia sets are not only an intriguing mathematical object, but also a major source of amazing images. Many computer programs, some of which are freely available on the web, have been written to generate these images. Algorithms for computing Julia sets have been presented and discussed in [11] and [14], for example.

It appears, however, that none of the algorithms or their implementations cope well with zooming in. With the computer using fixed-precision numbers, rounding errors significantly affect the computation when we try to zoom in. These programs also seem to work poorly near some "pathological" polynomials, for example, with $p(z) = z^2 + 1/4 + \varepsilon$, $0 < \varepsilon \ll 1$. We will return to this example in section 8.

We give the first polynomial bound on the complexity of an arbitrary hyperbolic Julia set. The class of hyperbolic polynomials is very rich. For example, in the case $p(z) = z^2 + c$, $p(z)$ is hyperbolic for all c's outside the Mandelbrot set. It is conjectured that it is also hyperbolic for all c's in the interior of the Mandelbrot set (but not on the boundary), see [9] for more information. The algorithm is outlined in sections 6 and 7. The details of the construction are mathematically involved, and many of them had to be omitted due to space constraints.

The algorithm that we present is not uniform in $p(z)$. That is, the Turing Machine computing $J_{p(z)}$ depends on $p(z)$. However, in section 8 we will show that no uniform Turing Machine computing $J_{p(z)}$ exists. Our algorithm can be modified to be uniform in the *hyperbolic* $p(z)$'s, and such a modification will remain polynomial in the precision of the computation, but might be

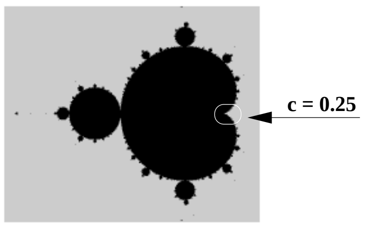

Fig. 1. The Mandelbrot set with the point $c = 1/4$ highlighted.

arbitrarily hard in the polynomial $p(z)$. Explicitly, we obtain a bound of $K(p) \cdot nM(n)$ on the running time, where $K(p)$ is a coefficient depending on the polynomial, 2^{-n} is the required precision and $M(n)$ is the complexity of multiplying two n-bit numbers.

It should be noted that the model of computation we are using is very different from the BCSS model presented in [1]. It has been shown that most Julia sets are not computable in that model (see [1] for more details).

The results of the paper can be easily generalized from hyperbolic polynomials to hyperbolic rational functions, and are very similar to the results obtained independently in [12].

2 Computability and Complexity of Bounded Subsets of $\mathbb{R}^2$

Several different computability notions for subsets of the real numbers have been proposed. It has been shown, for example, that non-trivial Julia sets are not computable in the real-RAM model (see [1]). We will use a different model introduced by Klaus Weihrauch in [16] and described in [13]. This is a very natural model when one is concerned with the complexity of "drawing" a set on the computer.

Intuitively, the definition says that the computational complexity of a set S is $t(n)$ if we can decide whether to draw a pixel of size 2^{-n} in the picture of S in time $t(n)$. To make this notion precise, we have to decide what are our expectations from a picture of S. First of all, we expect a good picture of S to cover the whole set S. On the other hand, we expect every point of the picture to be close to some point of S, otherwise the picture would have no descriptive power about S. Mathematically, we write these requirements as follows:

Definition 2.1 A set T is said to be a 2^{-n}-picture of a bounded set S if
(i) $S \subset T$, and (ii) $T \subset B(S, 2^{-n}) = \{x \in \mathbb{R}^2 \ : \ |x - s| < 2^{-n} \text{ for some } s \in S\}$.

Requirement (ii) can be also written as $d_H(S,T) \le 2^{-n}$, where d_H is the **Hausdorff distance**, defined by

$$d_H(S,T) := \inf\{r : S \subset B(T,r) \ \text{ and } \ T \subset B(S,r)\}.$$

Suppose we are trying to generate a 2^{-n}-picture of a set S using a union of round pixels of radius 2^{-n-2} with centers at all the points of the form $\left(\frac{i}{2^{n+2}}, \frac{j}{2^{n+2}}\right)$, with i and j integers. In order to draw the picture, we have to decide for each pair (i, j) whether to draw the pixel centered at $\left(\frac{i}{2^{n+2}}, \frac{j}{2^{n+2}}\right)$ or not. We want to draw the pixel if it intersects S and to omit it if some neighborhood of the pixel does not intersect S. Formally, we want to compute a function

$$f_S(n, i/2^{n+2}, j/2^{n+2}) = \begin{cases} 1, & B((i/2^{n+2}, j/2^{n+2}), 2^{-n-2}) \cap S \neq \emptyset \\ 0, & B((i/2^{n+2}, j/2^{n+2}), 2 \cdot 2^{-n-2}) \cap S = \emptyset \\ 0 \text{ or } 1, & \text{in all other cases} \end{cases} \qquad (*)$$

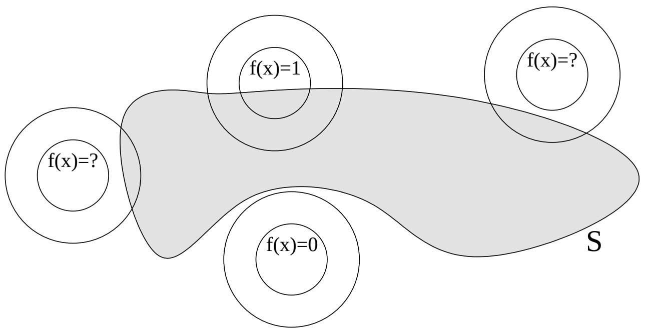

Fig. 2. Sample values of f. The radius of the inner circle is 2^{-n-2}.

Lemma 2.2 *The picture drawn according to $f_S(n, \bullet)$ is a 2^{-n}-picture of S.*

Here $\bullet$ stands for the different values of the parameters $(i/2^{n+2}, j/2^{n+2})$. The lemma illustrates the tight connection between the complexity of "drawing" the set S and the complexity of computing f. In order to reflect this connection we define the time complexity of S as follows.

Definition 2.3 A bounded set S is said to be computable in time $t(n)$ if there is a function $f(n, \bullet)$ satisfying $(*)$ which runs in time $t(n)$. We say that S is poly-time computable if there is a polynomial p, such that S is computable in time $p(n)$.

This definition is easily seen to be equivalent to the definition introduced in [16] and used in [13]. We will show that hyperbolic Julia sets are poly-time computable under this definition.

3 An Alternative Computability Definition – Ko P-Computability

We present an alternative definition of poly-time computability for sets in $\mathbb{R}^2$, which was introduced by Arthur W. Chou and Ker-I Ko in [4] (see also [8]).

While it is *not* equivalent, and is generally weaker than the definition we are using, it is still very useful due to a connection with our definition we present below.

In the model below we give x as an oracle to the machine which tries to decide whether $x \in S$. By an oracle we mean a "black box" function ϕ that on input n outputs a binary approximation $\phi(n) \in \mathbb{D}^2$ with $|\phi(n) - x| < 2^{-n}$. Here $\mathbb{D}$ denotes the set of the **dyadic** numbers $\mathbb{D} = \{k/2^l : k \in \mathbb{Z}, l \in \mathbb{N}\}$. Querying the oracle takes one time unit. A set is said to be strongly P-recognizable if there is an oracle Turing Machine, that on input x outputs 1 if $x \in S$ and 0 if x is 2^{-n}-far from S, i.e. the machine is allowed to make small one-sided errors. The definition was presented in [4] under the name of *strong P-recognizability*, we call it Ko P-computability to avoid confusion due to the fact that this definition is actually *weaker* than the other definition of computability that we are using. We summarize,

Definition 3.1 A set S is said to be **Ko P-computable** if there is an oracle TM $M^{\phi}(n)$ which runs in time polynomial in n, and outputs: (i) 1, if ϕ represents a point $x \in S$, (ii) 0, if ϕ represents a point $x \notin B(S, 2^{-n})$, (iii) 0 or 1, otherwise.

We can prove the following result, giving a simple complexity-theoretical connection between Ko P-computability and poly-time computability. See [3] for the proof.

Theorem 3.2 *Every poly-time computable set S is Ko P-computable. The converse statement (i.e. "every Ko P-computable set S is poly-time computable") holds if and only if $P = NP$.*

The following theorem makes the notion of Ko P-computability useful in the context of computing sets, in particular Julia sets. It states that Ko P-computable sets are exponential-time computable.

Theorem 3.3 *Suppose that a set S is Ko P-computable by a machine $M^{\phi}(n)$ running in time $p(n)$. Moreover, suppose that on input n, M^{ϕ} uses at most $l(n)$ first bits of x from the oracle. Then S is computable in time $t(n) = p(n) \cdot 2^{O(l)}$.*

To prove this theorem one combines the proof of the 'if' direction in theorem 3.2 with the brute force algorithm for NP-problems, which leads to an exponential upper bound on the computation time.

4 Julia Sets and Hyperbolic Julia Sets

We will give one of the equivalent definitions of the hyperbolic Julia set. More detailed information, as well as proofs and further references can be found in [9] and [10]. [10] gives a particularly good exposition of the hyperbolic Julia sets.

For the rest of the paper we fix our polynomial to be $p(z)$. Note that $p(z)$ is a polynomial with *complex* coefficients. Let $p^k(z)$ denote the k-th iteration of $p(z)$, i.e. $p^1(z) = p(z)$ and $p^{k+1}(z) = p(p^k(z))$. By a convention, $p^0(z) = z$. We define the **orbit** of z as the sequence $(z, p(z), p^2(z), \ldots)$. A point z is called **periodic** if $p^k(z) = z$ for some $k \geq 1$. The minimal such k is called the **period** of z. A periodic point z with period k and its (finite in this case) orbit $(z, p(z), \ldots, p^{k-1}(z))$ are said to be **attracting** if $|(p^k)'(z)| < 1$ and **repelling** if $|(p^k)'(z)| > 1$. Intuitively, if we iterate a point in the neighborhood of an attracting periodic point, then we will approach the attracting orbit, while if we iterate a point in the neighborhood of a repelling periodic point, we will escape the neighborhood. We say that a point c is a **critical** point of $p(z)$ if $p'(c) = 0$. We are now ready to state one of the equivalent definitions of a hyperbolic polynomial.

Definition 4.1 A polynomial $p(z)$ of degree ≥ 2 is said to be hyperbolic if every critical point of $p(z)$ converges to an attracting periodic orbit of $p(z)$ or to ∞.

Here we include ∞ as a special case to simplify matters, but in fact, by considering the Riemann sphere instead of the complex plane, we can regard ∞ as an attracting periodic point of $p(z)$, since we have $p(\infty) = \infty$ and $\lim_{n\to\infty} |p^n(z)| = \infty$ for $|z|$ large enough.

We can now give a simple definition of the Julia set in the hyperbolic case. See [10] for a proof that in the hyperbolic case this definition is equivalent to the general definition of the Julia set.

Definition 4.2 The Julia set J_p of a hyperbolic polynomial $p(z)$ is the set of all points w, such that the orbit of w does not converge to an attracting periodic orbit of $p(z)$ or to ∞. The complement of the Julia set is denoted $K_p = J_p^c$ and is called the Fatou set of the polynomial $p(z)$.

As we have mentioned above, the class of the hyperbolic polynomials is extremely robust. In particular, $z^2 + c$ is hyperbolic for all c's outside the Mandelbrot set M, and is believed to be hyperbolic for all the c's in the interior of M (see fig. 1). We summarize the most important facts about hyperbolic Julia sets we will be using in the following lemma. See [10] for details and proofs.

Lemma 4.3 *For a hyperbolic polynomial $p(z)$ the following facts hold:*

(i) The interior of J_p is empty.

(ii) $J_p = p(J_p) = p^{-1}(J_p)$.

(iii) $p(z)$ has at most deg$(p) - 1$ attracting periodic orbits (regarding an orbit as a set).

The definition itself gives a very naive "algorithm" for computing J_p. Namely, set a threshold T. To determine whether a point w is in J_p compute the first T elements of the orbit of w, $p(w), p^2(w), \ldots, p^T(w)$. If the orbit gets *close* to one of the attracting orbits, say that $w \notin J_p$, otherwise say that $w \in J_p$. In fact, many of the computer programs that draw Julia sets use this method. The problem, of course, is how to choose a good T and how to define "close". If T is not chosen properly, we might reject w's which are very close to J_p or accept w's which are far away from J_p. We will have to develop more theory in order to choose T which makes the method above work properly. The tool which we will use to control the distance between w and J_p is one of the fundamental tools in complex dynamics, called the Poincaré metric.

5 The Poincaré Metric

The Poincaré metric, known also as the hyperbolic metric, is a metric which naturally arises on hyperbolic Riemann surfaces. It is beyond the scope of this paper to discuss the metric in full generality, so we will restrict our attention to subsets of the complex plane $\mathbb{C}$. See [10] for a more comprehensive exposition. It is known that any connected open subset $S \subset \mathbb{C}$ of the complex plane which omits at least 2 points is a hyperbolic Riemann surface and has a unique (up to a multiplication by a constant) Poincaré metric d_S. We call these subsets of $\mathbb{C}$ **hyperbolic sets**.

To define the Poincaré metric, we first need to describe another fundamental mathematical concept – the notion of a covering map. While covering maps are defined in many different topological settings, we will define it for our case: the hyperbolic subsets of $\mathbb{C}$. A map $f : X \to Y$ between two hyperbolic subsets of $\mathbb{C}$ is said to be a **covering map**, if for each $y \in Y$ there is a neighborhood $N(y)$ of y such that for each connected component N' of $f^{-1}(N(y))$, the map $f : N' \to N(y)$ is a conformal (locally shape preserving) isomorphism.

In general, covering maps allow us to analyze the structure of Y using the structure of X. In particular, it is used in the definition of the Poincaré metric. We skip the details and present only the final result we use in our case. This result can be viewed as the defining property of yhr Poincaré metric. We refer the interested reader to [10].

Theorem 5.1 (Theorem of Pick) *There is one, and up to a multiplication by a constant, only one family of metrics defined on hyperbolic subsets of $\mathbb{C}$ such that for any hyperbolic subsets S and T of $\mathbb{C}$ the following holds. If $f : S \to T$ is a holomorphic map, then exactly one of the following three statements is valid:*

(i) f is a conformal isomorphism from S onto T, and maps S with its Poincaré metric isometrically to T with its Poincaré metric.

(ii) f is a covering map but is not one-to-one. In this case, it is locally but not globally a Poincaré isometry. Every smooth path $P : [0, 1] \to S$ of arclength l in S maps to a smooth path $f \circ P$ of the same length l in T.

(iii) In all other cases, f strictly decreases all non-zero distances (in the

Poincaré metric).

For every hyperbolic set S the Poincaré metric d_S has a *weight function* $p_S : S \to \mathbb{R}^+$ such that $p_S(x)$ measures the ratio between d_S and the Euclidean metric near the point x. Formally, the lenght of an arc $\gamma : [0,1] \to S$ in d_S is given by

$$l_{d_S}(\gamma) = \int_0^1 p_S(\gamma(t))|\gamma'(t)|dt.$$

We now have the basic complex-analytic background required for the construction, and we are ready to prove that the hyperbolic Julia sets are poly-time computable. Note that we can view J_p as a subset of $\mathbb{R}^2$ using the trivial identification of $\mathbb{C}$ with $\mathbb{R}^2$.

6 Hyperbolic Julia Sets are Ko P-Computable

As noted above, we will present an algorithm that uses some nonuniform information, i.e. information which depends on the polynomial $p(z)$ but not on the precision parameter n. The polynomial $p(z)$ itself is given to the algorithm as an oracle that outputs its coefficients with any required precision. Denote the polynomial $p(z) = c_m z^m + c_{m-1} z^{m-1} + \ldots + c_1 z + c_0$. We can query each c_i with precision 2^{-r} with time cost r. As per the definition of Ko P-recognizability, the input x to the algorithm is also given as an oracle. We want to decide whether $x \in J_p$.

We will now list the nonuniform information used by the algorithm. This information can be computed from the initial data (i.e. the coefficients of $p(z)$) as will be noted later, see theorem 8.4. We still list it as nonuniform to spare overcomplicated technical details from the reader.

Nonuniform constants information:

We summarize the nonuniform information used by the algorithms in the following lemma. See appendix for the explicit construction of the set $\tilde{U}$.

Lemma 6.1 *There is an explicitly presented open set $\tilde{U}$ and numbers $\tilde{d}_l > 0$, $R' > 0$, $d > 0$, $D \geq 1$, $r_c > 0$ and an integer q, such that the following hold.*

- (i) *denote $\tilde{V} = p^{-1}(\tilde{U})$, then $\tilde{V} \subset \tilde{U}$ and $d(\tilde{U}^c, \tilde{V}) \geq \tilde{d}_l$,*
- (ii) *for any critical point y_i of p, whenever $|y - y_i| < r_c$ we have $p^q(y) \notin \tilde{U}$,*
- (iii) *$J_p \subset \tilde{U} \subset B(0, R')$,*
- (iv) *d is a lower bound such that $|p'(z)| > d$ whenever $|z - y_i| > \frac{r_c}{2}$ for all critical points y_i of p,*
- (v) *D is an upper bound such that $|p'(p(y))| < D$ and $|p'(y)| < D/2$ whenever $|y| < R'$.*

$\tilde{U}$ can be thought of a huge disk with small holes poked in it around the attracting orbits. The orbit of each critical point eventually leaves $\tilde{U}$ by the definition of hyperbolic Julia sets and this happens after at most q steps. In general, the orbit of any point outside J_p converges to one of the attracting periodic orbits, and hence eventually leaves $\tilde{U}$.

Denote $U = p^{-q-1}(\tilde{U})$, $V = p^{-q-1}(\tilde{V})$. Then by the definition of r_c and q, $B(y_i, r_c) \cap U = \emptyset$ for any critical point y_i. Lemma 4.3 implies that $J_p = p^{-q-2}(J_p) \subset p^{-q-2}(\tilde{U}) = V$.

As a corollary of $\tilde{V} \subset \tilde{U}$, we conclude that $V \subset U$, and furthermore we can compute a lower bound d_l on the distance between V and U^c.

Lemma 6.2 *$d(U^c, V) \geq d_l = \frac{\tilde{d}_l}{D^{q+1}}$.*

U and V are obviously hyperbolic sets, and hence the Poincaré metric is defined on them. Denote the weight function of the Poincaré metric on U by p_U and the weight function of the Poincaré metric on V by p_V. Denote the Poincaré metrics themselves by d_U and d_V, respectively. We have the following simple lemma bounding the metric p_U. This lemma is a simple consequence of Pick's theorem (theorem 5.1). We refer the interested reader to [3] for the proof of this and other technical lemmas.

Lemma 6.3 *$p_U(z) > \frac{2}{R'}$ for all $z \in U$, and $p_U(z) < \frac{2}{d_l}$ for all $z \in V$.*

The reason that we would like to bound V from the boundary of U is that it would allow us to bound the ratio $\frac{p_V(z)}{p_U(z)}$ on V from below and bound the diameter of V in d_U from above.

We will skip the details, and only state the final results.

Lemma 6.4 *$p_V(z) \geq c \cdot p_U(z) > p_U(z)$ for all $z \in V$, where $c > 1$ is some constant computable from d_l and R' (see appendix for details and an explicit calculation of c).*

The existence of $c > 1$ as above has been known and can be easily proven using a compactness argument. Our contribution is in constructively *computing* such a c, which is much harder. This is particularly important for theorem 8.4, where we make the construction uniform.

Lemma 6.5 *For x, y in the same connected component of V, $d_U(x, y) < M_W = \frac{128R'^2}{d_l^2}$.*

Observe that by our construction V does not contain critical points of p, hence the map $p : V \to U$ is an m-fold covering map. Hence by Pick's theorem (theorem 5.1) it is a local Poincaré isometry. Formally, we obtain

Lemma 6.6 *$p_V(x) = |p'(z)| \cdot p_U(p(x))$ for all $x \in V$.*

We see that lemmas 6.4 and 6.6 imply together that $p(z)$ locally increases the metric p_U on V. As a consequence of lemmas 6.4, 6.6 and 4.3 we obtain the following key lemma.

Lemma 6.7 *Let $d_U(J_p, z)$ denote the distance between the point z and the Julia set J_p in the Poincaré metric d_U. Then $d_U(J_p, z)$ is well defined for* *all $z \in U$ and $d_U(J_p, p(x)) \geq c \cdot d_U(J_p, x)$ for all $x \in V$, where $c > 1$ is a computable constant.*

Lemma 6.7 gives us a tool to estimate the speed at which a point $x \notin J_p$ runs away from J_p in the Poincaré metric. If initially the Euclidean distance $d(J_p, x) > \varepsilon$, then by lemma 6.3, $d_U(J_p, x) > \frac{2\varepsilon}{R'}$, and assuming that the orbit of x stays in V in s steps we have by lemma 6.7 that $d_U(J_p, p^s(x)) > \frac{2\varepsilon}{R'} \cdot c^s$. But then by lemma 6.5 we have $s \leq \log_c \frac{M_W}{2\varepsilon/R'}$. If $\varepsilon = 2^{-n}$ the estimate on s is linear in n. This allows us to obtain a poly-time algorithm to Ko P-compute J_p:

```
    Algorithm 1
Input: The non-uniform input as described above,
the input $x, c_0, c_1, \ldots, c_m$ given on an oracle tape and $m, n$.
Output: 0 if $d(J_p, x) > 2^{-n}$, 1 if $x \in J_p$, either 0 or 1 otherwise.
1. Compute $c$ described above.
2. Compute a natural number $N = O(n)$ such that
   $N \geq 2 + \log_c(M_W \cdot R' \cdot 2^{n-1}) + q$.
3. Compute $p^N(x)$ within an error of $\frac{\tilde{d}_l}{4}$.
  3.1. If $p^N(x) \notin \tilde{U}$, output 0,
  3.2. If $p^N(x) \in \tilde{V}$, output 1,
  3.3. Otherwise, output 0 or 1.
```

Note that given $\tilde{U}$, and making computations with precision $\tilde{d}_l/4$ we can unambiguously decide if one of the possibilities 3.1 or 3.2 holds. Observe that $N = O(n)$, so we perform linearly many operations on x, hence we require linearly many bits of x in order to achieve the required (fixed) precision level at the end of the computation. Hence by theorem 3.3 we know that J_p is computable in time $poly(n) \cdot 2^{O(n)} = 2^{O(n)}$. Algorithm 1 *does not* compute J_p in our definition, since it might reject points $x \notin J_p$ which are very close to J_p (closer than 2^{-n-1}). In the next section we will be referring to the *exponential time* algorithm computing J_p in our sense as Algorithm 1.

7 J_p is Poly-Time Computable

We are now ready to outline the proof of the poly-time computability of the Julia set J_p. We use the result from the previous section combined with a technique very similar to the technique used in [13] to pass from an exponential to a polynomial time algorithm. Algorithm 1 does not compute J_p because it might output 0 for points x which are not in J_p, but within a very small distance of it, while the definition of computability requires output 1 in this case. To avoid this problem we need to estimate the distance from x to J_p both from above and below. We employ techniques similar to the ones used in [13].

Lemma 7.1 *There are positive constants $\alpha, \beta > 0$ computable from the initial* *data such that for any $z \in B(0, R')$ and $0 < \varepsilon < \alpha$ satisfying $d(J_p, z) \leq \varepsilon$, we have $(|p'(z)| - \beta\varepsilon) d(J_p, z) \leq d(J_p, p(z)) \leq (|p'(z)| + \beta\varepsilon) d(J_p, z)$.*

The lemma says that the change in the distance from z to J_p is locally controlled by the expansion factor $|p'(z)|$ up to some small relative error. This enables Algorithm 2 to iterate the orbit of x keeping track of the ratio between the distance $d(J_p, p^l(x))$ and $d(J_p, x)$ as long as $d(J_p, p^l(x))$ is not too big ($O(1/n)$). When $d(J_p, p^l(x))$ becomes big, we apply Algorithm 1 to estimate it. Algorithm 1 runs fast here because it needs to give an estimate with precision factor $O(1/n) = O(1/2^{\log n})$, so its running time is exponential in $\log n$. Knowing an estimate of the ratio $d(J_p, p^l(x))/d(J_p, x)$ we are able to give a good estimate on $d(J_p, x)$.

We then plug in Algorithm 2 into itself instead of Algorithm 1, to obtain Algorithm 3 which computes J_p and runs in time $O(n \cdot M(n))$, where $M(n)$ is the time complexity of multiplying two n-bit numbers. See appendix for a full exposition and pseudocode for Algorithm 2. Using the best known estimate of $O(n \log n \log\log n)$ on $M(n)$ (see [7] p. 311 and [15]) we obtain

Theorem 7.2 *For every fixed hyperbolic polynomial $p(z)$, the Julia set J_p is poly-time computable in time $O(nM(n)) \subset O(n^2 \log n \log\log n)$.*

8 Can the Result be Improved?

In this section we will try to address the question of whether the result of this paper can be improved, and by how much. The first question one might ask is whether there is a *uniform* algorithm computing J_p for an arbitrary p given as an oracle. The answer is negative. In fact, there is no such algorithm even for p of degree 2. We use the following lemma, which is very typical to computable analysis (cf. [16], p. 108, Thm 4.3.1). See section 2 for the definition of the Hausdorff metric.

Lemma 8.1 *If a function $f : \mathbb{R}^m \to K^n$, where K^n is the set of compact sets in $\mathbb{R}^n$, is computable by an oracle machine M^ϕ, then f is continuous in the Hausdorff metric.*

We use the following lemma which follows from [5], section 11, theorem 11.3:

Lemma 8.2 *The function $J : c \mapsto J_{z^2+c}$ is not continuous at $c = 0.25$ in the Hausdorff metric.*

Lemmas 8.1 and 8.2 imply together that

Theorem 8.3 *No oracle machine M^ϕ, where ϕ represents the number c, can compute the Julia set J_{z^2+c}.*

In fact, most of the programs written to draw Julia sets perform poorly for the polynomial $p(z) = z^2 + 0.25 + \varepsilon$ for small positive values of ε.

On the other hand, our construction can be made uniform over all *hyperbolic* polynomials $p(z)$. All the nonuniform information used in our construction can be extracted from the coefficients of $p(z)$. This, however, might take indefinitely long, depending on the hyperbolic polynomial p. Hence there is an algorithm, polynomial in the precision n for computing J_p, which is uniform given that $p(z)$ is hyperbolic. We will not prove it here. See [3] for a discussion of the main ideas of the proof.

Theorem 8.4 *The Julia set J_p, where the coefficients of a* **hyperbolic** *$p(z)$ are given as oracles, can be locally computed with precision 2^{-n} in time $O(nM(n))$, where the constant factor in the $O(\bullet)$ depends on $p(z)$ but not on n. In other words the time complexity of locally computing J_p is bounded by $K(p) \cdot nM(n)$ for some $K(p)$. Here, again, $M(n) \in O(n \log n \log\log n)$ is the complexity of multiplying two n-bit numbers.*

This theorem does not contradict theorem 8.3, because it only works for hyperbolic polynomials, while the polynomial $p(z) = z^2 + 0.25$ around which the discontinuity occurs in lemma 8.2 is parabolic and not hyperbolic. The uniform algorithm would never terminate on this input. Note that the point $c = 0.25$ lies on the boundary of the Mandelbrot set (see figure 1).

Further research in the area of computability of Julia sets might address the following two problems.

Problem 8.5 *Are there poly-time algorithms for other types of Julia sets? In particular, are parabolic Julia sets poly-time computable?*

Problem 8.6 *We have seen in theorem 8.3 that the problem of computing the Julia set for a general polynomial is not computable. Is there a* **particular** *polynomial $p(z)$ for which J_p is not computable? Is there such a polynomial $p(z)$ with computable coefficients?*

Acknowledgements

I would like to thank my graduate supervisor, Prof. Stephen Cook, for his insights and support during the preparation of this paper and for the many hours we spent discussing the computability of real sets.

I would like to thank Prof. Michael Yampolsky from the Department of Mathematics in the University of Toronto for helping me to learn the complex analytic background I needed for this research, suggesting that the Poincaré metric is a useful tool for tackling the problem and for the theoretical help he has provided during the preparation of this paper.

References

[1] Blum, L., F. Cucker, M. Schub, S. Smale, "Complexity and Real Computation", Springer, New York, 1998.

[2] Brattka, V., K. Weihrauch, *Computability of Subsets of Euclidean Space I: Closed and Compact Subsets*, Theoretical Computer Science, **219** (1999), pp 65-93.

[3] Braverman M., “Computational Complexity of Euclidean Sets: Hyperbolic Julia Sets are Poly-Time Computable”, Thesis, University of Toronto, 2004 (to appear).

[4] Chou, A., K. Ko, *Computational complexity of two-dimensional regions*, SIAM J. Comput. **24** (1995), pp 923-947.

[5] Douady, A., *Does a Julia set depend continuously on the polynomial?* Proc. Symposia in Applied Math.: Complex Dynamical Systems: The Mathematics Behind the Mandelbrot Set and Julia Sets, vol **49** (1994), ed R. Devaney (Providence, RI: American Mathematical Society) pp 91-138.

[6] Jost, J., “Compact Riemann Surfaces”, Second edition, Springer, 2002.

[7] Knuth, D., “The Art of Computing Programming, v. 2: Seminumerical Algorithms”, 3rd ed., Addison-Wesley, 1997.

[8] Ko, K., *Polynomial-time computability in analysis*, in ”Handbook of Recursive Mathematics”, Volume **2** (1998), Recursive Algebra, Analysis and Combinatorics, Yu. L. Ershov et al. (Editors), pp 1271-1317.

[9] McMullen, C., “Complex Dynamics and Renormalization”, Princeton University Press, Princeton, New Jersey, 1994.

[10] Milnor, J., “Dynamics in One Complex Variable - Introductory Lectures”, second edition, Vieweg, 2000.

[11] Pickover, C. A. (ed.), “Chaos and Fractals – Computer Graphical Journey, Ten Year Compilation of Advanced Research.” Elsevier, 1998.

[12] Rettinger, R., *A Fast Algorithm for Julia Sets of Hyperbolic Rational Functions*, in CCA’04, Aug 16-20, 2004, Lutherstadt Wittenberg, Germany.

[13] Rettinger, R., K. Weihrauch, *The Computational Complexity of Some Julia Sets*, in STOC’03, June 9-11, 2003, San Diego, California, USA.

[14] Saupe, D., *Efficient Computation of Julia Sets and Their Fractal Dimension.* Physica D, **28** (1987), pp 358–370.

[15] Schönhage, A., V. Strassen, *Schnelle Multiplikation grosser Zahlen*, Computing **7** (1971), pp 281–292.

[16] Weihrauch, K., “Computable Analysis”, Springer, Berlin, 2000.

[17] Zhong N., *Recursively enumerable subsets of R^q in two computable models: Blum-Schub-Smale machine and Turing machine.* Theoretical Computer Science, **197** (1998), pp 79-94.