Inductive Inference of Integer Sequences

Sam Tetruashvili
Advisor: Manuel Blum
samt@cmu.edu, mblum@cs.cmu.edu

May 9, 2010

Contents

List of Figures

Acknowledgements

Abstract

Inductive inference denotes hypothesizing a general rule from examples. [AS83] In this senior thesis, we give a model for *inductively inferring integer sequences*. Within this model we give algorithms that can (inductively) infer integer sequences with *high confidence*, under the assumption that all the terms of the sequence are accessible. We provide *tight* bounds on the number of sequence terms an inference algorithm needs to see before it can make an inference it is confident in. We also explore scenarios when we can confidently say that a sequence cannot be inferred by some inference algorithm.

We use The On-Line Encyclopedia of Integer Sequences [Slo] to evaluate our model. We are currently able to infer about 18.9% of the OEIS using the methods given in this thesis. We also implement a website for inferring sequences that is meant to complement the OEIS. This site can be publicly accessed at atemi.cdm.cs.cmu.edu/∼samt/sequences.html.

Chapter 1

Introduction

Inductive inferences denotes hypothesizing a general rule from examples [AS83]. In this senior thesis we will exploring the inductive inference of integer sequences. For example, consider the first few terms of the Fibonacci sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144$$

We say that an algorithm has *inferred* this sequence if when given the first few terms of the sequence the algorithm can output a general description of the entire (infinite) sequence. In this case, if an algorithm were to output the second order linear recurrence

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_{n+2} &= f_{n+1} + f_n \end{aligned}$$

we would say that it has inferred the Fibonacci sequence.

1.1 Previous Work

The problem of inductively inferring integer sequence has been attacked from a variety of perspectives. In [Ang74], Dana Angluin studied sequences that humans could easily infer. In her work she would give a subject the first few terms of an infinite integer sequence and ask them to continue the sequence. She then provides a taxonomy of integer sequences that humans are able to infer. We hope to be able infer all of the sequence classes Angluin presents in her work. From a more practical point of view, N.J.A. Sloane's Superseeker program [Slo] and the `FindSequenceFunction` procedure in Mathematica 7 have been capable of inferring various elaborate classes of integer sequences for some time now. It should be noted that these programs do not give the user any measure of how confident they should be in their results.

The problem of the inductive inference of functions (as opposed to sequences) has been studied since the mid-1960s. In [Gol67], Mark Gold gives a general model for inferring languages in the limit. The model that we present in this work is deeply inspired by the model given by Gold. Dana Angluin and Carl Smith give a very good survey of the current state of inductive inference in [AS83].

1.2 Our Results

In this senior thesis we provide a general model for inductively inferring integer sequences. We then explore the space of algorithmically generated integer sequences to find algorithms for inferring certain concept classes of sequences. We then give inference algorithms for the following concept classes

- Polynomials
- Linear Recurrences
- Decimal Expansion of Rational Numbers
- Decimal Expansions of Superpositions
- Automatic Sequences

We also give *tight* upper and lower bounds on how many sequence terms these inference algorithms need to see before they can correctly infer some (k-degree) sequence in the concepts class. We then explore the entire class of all algorithmically generated integer sequences and give strong evidence that we should not be able to infer this class in general. We then begin to scratch the surface of the study of confidence functions. At a high level these functions try to let us know how confident we are that we have seen enough sequence terms to justify some hypothesis of degree k. We give two simple properties that these confidence functions should intuitively have. We then pose a number of interesting open problems associated with confidence functions.

To evaluate the performance of our model we use the sequences generously catalogued in *The On-Line Encyclopedia of Integer Sequences* (OEIS) of N.J.A. Sloane. So far we have been able to use the methods developed in this thesis to infer about 18.9% of the sequences in this database. Much in the same spirit as the OEIS, we have built a website that uses our methods to try to infer any sequence that a user inputs. This website can be publicly accessed at

atemi.cdm.cs.cmu.edu/∼samt/sequences.html.

We encourage the reader to use this website to get an intuitive feel for this work.

Finally, we conclude by suggesting some future directions for this research. Among these are designing new algorithms to infer interesting concept classes of integer sequences that we have come across so far, searching for new (and more general) concept classes, open problems associated with confidence functions, and a number of applications of the inductive inference of integer sequences.

Chapter 2

The Evolutionary Inference Model

The scientific method has been one of civilization's most successful tools for gathering knowledge about the world we live in. The method begins with the creation of a new scientific theory. Once a scientific theory is posed, it is then barraged with an array of intricate experiments meant to push the theory to its limits. The tests continue until one of them manages to find a flaw in the theory. When this happens the scientists must go back to the drawing board and come up with another theory that accounts for the failures of the debunked theory.

The methods we propose in this chapter for inferring integer sequences are heavily inspired by the scientific method. At a very high level our model makes and tests hypotheses. When it finds that its hypothesis disagrees with the input sequence, it simply makes a new hypothesis using all of the data it has seen so far. It is important to note that these hypotheses must be *fully consistent* with the sequence terms that have been seen so far.

As you have no doubt already noticed, it is very unlikely that the scientific method will ever halt; *even if it has converged to the right theory.* Thankfully, the methods we propose let us put bounds on how long it will take for us to converge to the correct answer, provided it exists.

2.1 The Model

Definition 1. We say that an integer sequence, $\{a_n\}_{n\geq 0}$, is *algorithmically generated* if there is a Turing Machine, M, that on input $t \in \mathbb{N}$ halts with output a_t for all $t \geq 0$. For the remainder of this thesis we will use the terms sequence and algorithmically generated integer sequence interchangeably. We will refer to subsets of the set of all algorithmically generated integer sequences as *concept classes.*

Definition 2. The *Evolutionary Inference Model* for inferring integer sequences has three main components:

1. The *algorithmically generated* integer sequence, $\{a_n\}_{n\geq 0}$, that is to be inferred. We require this sequence to be represented such that at time t

of our algorithm we can efficiently retrieve a_t.

2. The *inference algorithm*, A, that we are trying to use to infer $\{a_t\}_{t\geq 0}$. This inference algorithm generates hypotheses, $\{h_t\}_{t\geq 0}$, that come from some well defined *concept class*. For example, we present an inference algorithm that tries to infer the input sequence as a linear recurrence. It should be noted that any hypothesis output by an inference algorithm should be fully consistent with all the (finite amount of) data that has been input.

3. The *confidence function*, C, that tells us how confident we are in the current hypothesis, $\{h_t\}_{t\geq 0}$, given that we are currently at time t of our algorithm. The range of these confidence functions is the interval $[0, 1]$, where 0 indicates no confidence and 1 indicates total confidence. It should be noted that C is allowed to depend on the concept class that A (our inference algorithm) is designed to infer.

Given these components, the model will then proceed to try to use the inference algorithm to find a hypothesis that it is $1 - \epsilon$ confident in, for some input $\epsilon > 0$, as shown in Figure 2.1.

```
1:  EvolutionaryInference({a_t}_{t≥0}, A, C, ε)
2:      {h_t}_{t≥0} = {0}_{t≥0}
3:
4:      for t = 0, 1, 2, ... do
5:          if h_t ≠ a_t then
6:              {h_t}_{t≥0} = A([a_0, a_1, ..., a_t])
7:          else if C({h_t}_{t≥0}, t) ≥ 1 − ε then
8:              break
9:
10:     return {h_t}_{t≥0}
```

Figure 2.1: The Evolutionary Inference Model simply keeps making new hypotheses until it finds one that it is sufficiently confident in.

In the remainder of this work we strive to design inference algorithms and confidence functions that allow us to make sure that the model does not become too confident in an incorrect hypothesis.

2.2 Inference Algorithms

2.2.1 Polynomials

This simplest class of sequences that we study are those generated by a polynomial with rational coefficients.

Definition 3. Let $\{a_t\}_{t\geq 0}$ be an integer sequence and let k be a nonnegative integer. We say that $\{a_t\}_{t\geq 0}$ *is generated by a polynomial of degree* k if there

are rational numbers $c_0, c_1, \ldots, c_k$ such that for all $t \geq 0$

$$a_t = \sum_{i=0}^{k} c_i t^i \tag{2.1}$$

To make the above equation well defined, we take $0^0 = 1$.

Let's say there is some sequence, $\{a_t\}_{t\geq 0}$, that is generated by a polynomial of degree k. The task for our inference algorithm would be to somehow find k and the coefficients $c_0, c_1, \ldots, c_k$ given some finite initial portion of the sequence. The following theorem will show that this is possible given sufficiently many sequence terms.

Theorem 1. *"Let $\{a_t\}_{t\geq 0}$ be generated by a polynomial of degree k. We can infer $\{a_t\}_{t\geq 0}$ given at least the first $k+1$ terms of the sequence. Furthermore this bound is tight."*

Proof. Say we are given exactly the first $n = k+1$ terms of the sequence: $a_0, a_1, \ldots, a_k$. Since we know that $\{a_t\}_{t\geq 0}$ is generated by a polynomial of degree k we know that for each $t \in \{0, 1, \ldots, k\}$

$$a_t = \sum_{i=0}^{k} c_i t^i \tag{2.2}$$

This gives us a system of $k+1$ equations with $k+1$ unknowns. We then rewrite this system of equations as the following matrix equation.

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1^2 & \cdots & 1^n \\ 1 & 2 & 2^2 & \cdots & 2^n \\ 1 & 3 & 3^2 & \cdots & 3^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \cdots & n^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{pmatrix} \tag{2.3}$$

This equation always has a unique solution since the matrix is a Vandermonde matrix, and thus invertible. We can find this solution using Gaussian Elimination. If we are given more than the first $k+1$ terms ($n > k+1$), then we can still solve the matrix. By equation (2.2) we know that if $i > k$ then $c_i = 0$ thus we will still be able to find the correct c_i's. Given the c_i's, we can set k to be the index of the largest non-zero c_i.

To see that this bound is tight, consider the following polynomial of degree k

$$p_x = \prod_{i=0}^{k-1} (x - i) \tag{2.4}$$

The roots of this polynomial clearly occur at $n \in \{0, 1, \ldots, k-1\}$. Thus, when given fewer then the first $k+1$ terms of $\{p_x\}_{x\geq 0}$ we are unable to distinguish it from the zero polynomial. ■

We can use the algorithm given in the proof of Theorem 1 as our inference algorithm for polynomials. We state the algorithm slightly more formally in Figure 2.2.

1 : **InferPolynomial**$([a_0, a_1, \ldots, a_{n-1}])$
2 : $\quad [c_0, c_1, \ldots, c_{n-1}] = \textbf{MatrixSolve}(M_{n-1}, [a_0, a_1, \ldots, a_{n-1}])$
3 : $\quad k = \max\{0, \max\{0 \leq i < n \mid c_i \neq 0\}\}$
4 :
5 : $\quad$ **return** $(k, [c_0, c_1, \ldots, c_k])$

Figure 2.2: The polynomial inference algorithm. Note that M_i is the matrix given in equation 2.3 with $n = i$.

2.2.2 Linear Recurrences

The next simplest class of sequences we study are those generated by a linear recurrence with rational coefficients.

Definition 4. Let $\{a_t\}_{t\geq 0}$ be an integer sequence and let k be a positive integer. We say that $\{a_t\}_{t\geq 0}$ *is generated by a linear recurrence of degree* k if there are rational numbers $c_0, c_1, \ldots, c_{k-1}$ and integers $b_0, b_1, \ldots, b_{k-1}$ such that

$$a_t = b_t \qquad \text{for } 0 \leq t < k \tag{2.5}$$

$$a_{t+k} = \sum_{i=0}^{k-1} c_i a_{t+i} \qquad \text{otherwise} \tag{2.6}$$

Say there is some sequence, $\{a_t\}_{t\geq 0}$, that is generated by a linear recurrence of degree k. The task for our inference algorithm would be to somehow infer k, the base cases $b_0, b_1, \ldots, b_{k-1}$, and coefficients $c_0, c_1, \ldots, c_{k-1}$ given some finite initial portion of the sequence. Much like for polynomials we now prove a theorem that states this type of inference is possible given sufficiently many sequence terms.

Theorem 2. *"Let $\{a_t\}_{t\geq 0}$ be generated by a linear recurrence of degree k. We can infer $\{a_t\}_{t\geq 0}$ given at least the first $2k$ terms of the sequence. Furthermore this bound is tight."*

Proof. Say we are given exactly the first $2k$ terms of the sequence: $a_0, a_1, \ldots, a_{2k-1}$. Since we know that $\{a_t\}_{t\geq 0}$ is generated by a linear recurrence of degree k we know that for $t \in \{0, 1, \ldots, k-1\}$

$$a_{t+k} = \sum_{i=0}^{k-1} c_i a_{t+i} \tag{2.7}$$

This gives us a system of k equations with k unknowns. We can rewrite this system as the following matrix equation.

$$\begin{pmatrix} a_0 & a_1 & \cdots & a_{k-1} \\ a_1 & a_2 & \cdots & a_k \\ \vdots & \vdots & \ddots & \vdots \\ a_{k-1} & a_k & \cdots & a_{2k-2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{k-1} \end{pmatrix} = \begin{pmatrix} a_k \\ a_{k+1} \\ \vdots \\ a_{2k-1} \end{pmatrix} \tag{2.8}$$

Since $\{a_t\}_{t\geq 0}$ is generated by a linear recurrence this matrix must be invertible; otherwise the coefficient vector of the linear recurrence would be undefined.

Thus we can solve this matrix equation for the c_i's. We can now take the base cases to be $b_t = a_t$ for $0 \leq t < k$. Thus in this special case we can make the correct inference.

We now reduce the case of being given the first $n > 2k$ terms of the sequence to the case of being given exactly the first $2k$ terms of the sequence. Let $j = \lfloor \frac{n}{2} \rfloor$ and consider only the first $2j$ terms of the sequence in question. Using those first $2j$ terms we make the following matrix.

$$M([a_0, \ldots, a_{2j-2}]) = \begin{pmatrix} a_0 & a_1 & \cdots & a_{j-1} \\ a_1 & a_2 & \cdots & a_j \\ \vdots & \vdots & \ddots & \vdots \\ a_{j-1} & a_j & \cdots & a_{2j-2} \end{pmatrix} \tag{2.9}$$

Since $j \geq k$ we can guarantee that the rank of this matrix is exactly k. To see why this is the case note that the last $j-k$ rows of the matrix can be represented as a linear combination of the first k rows of the matrix our sequence is linear recurrent. For example, row $k+1$ is equal to

$$\sum_{i=1}^{k} c_{i-1} \cdot \text{row } i$$

Where the c_i's are the coefficients of the underlying linear recurrence of the sequence in question.

Once we have k we can simply take the $k \times k$ sub-matrix of the matrix given in equation (2.9) and apply the algorithm that we gave for the case of being given exactly the first $2k$ terms of the sequence in question. Thus we can infer k, the base cases $b_0, b_1, \ldots, b_{k-1}$, and the coefficients $c_0, c_1, \ldots, c_{k-1}$ as required when we are given at least the first $2k$ terms of the sequence in question.

To see that this bound is tight, let k be a positive integer. Consider the following two linear recurrences of degree k

$$f_t = \begin{cases} 0 & \text{, if } t < k-1; \\ 1 & \text{, if } t = k-1 \end{cases} \qquad f_{n+k} = 0$$

$$g_t = \begin{cases} 0 & \text{, if } t < k-1; \\ 1 & \text{, if } t = k-1 \end{cases} \qquad g_{n+k} = g_n$$

We need at least $2k$ sequence terms in order to distinguish these two linear recurrences from one another, since their first $2k-1$ terms are identical. ■

We can use the algorithm given in the proof of theorem 2 as our inference algorithm for linear recurrences. We state this algorithm a bit more formally in Figure 2.3

1 : **InferLinearRecurrence**$([a_0, a_1, \ldots, a_{n-1}])$
2 : **if** $(n \equiv 1 \mod 2)$ **then**
3 : $n = n - 1$
4 :
5 : $k = \textbf{MatrixRank}(M([a_0, \ldots, a_{n-2}]))$
6 : $[c_0, c_1, \ldots, c_{k-1}] = \textbf{MatrixSolve}(M([a_0, \ldots, a_{2k-2}]), [a_k, a_{k+1}, \ldots, a_{2k-1}])$
7 : $[b_0, b_1, \ldots, b_{k-1}] = [a_0, a_1, \ldots, a_{k-1}]$
8 :
9 : **return** $(k, [b_0, b_1, \ldots, b_{k-1}], [c_0, c_1, \ldots, c_{k-1}])$

Figure 2.3: The linear recurrence inference algorithm. Note that the matrix $M([a_0, a_1, \ldots, a_{n-1}])$ is the matrix defined as in equation (2.9).

2.2.3 Decimal Expansions of Rational Numbers

The next class of sequences we study is the class of decimal expansions of rational numbers. As you probably already know, all sequences in this class are eventually periodic and are thus linear recurrent. So why is it worth studying this class of sequences? Consider the decimal expansion of $\frac{1}{503}$

$$\frac{1}{503} = 0.00198807157057654075546719681908548707753\cdots \tag{2.10}$$

Since 503 is prime we know that this decimal expansion has a period of 502. Thus this sequence can be represented as a linear recurrence of degree 502; meaning that we will need at least 1004 sequence terms before we can infer it using the technique outlined in the previous section. In this section we present a technique, given in [BBS82], to predict(infer) the terms of the $1/P$ pseudo-random number generator. This technique can be used to infer the $\frac{1}{503}$ sequence using *only 7 sequence terms!* This algorithm will allow us to infer certain sequences with exponentially fewer terms than required by our linear recurrence inference algorithm.

Definition 5. Let $\{a_t\}_{t\geq 0}$ be an integer sequence, let b be a positive integer, and let r, n be integers such that $n \neq 0$, $\gcd(r, n) = 1$, and $0 \leq r < n$. We say that $\{a_t\}_{t\geq 0}$ *is generated by a rational of degree k in base b* if a_t is the $(t+1)^{st}$ digit in the base b expansion of $\frac{r}{n}$, i.e.

$$a_t = \left\lfloor \frac{r \cdot b^t}{n} \right\rfloor \mod b \tag{2.11}$$

and $k = \lceil \log_b n \rceil$ (i.e. the number of digits required to represent n in base b).

Let's say there is some sequence, $\{a_t\}_{t\geq 0}$, that is generated by a rational of degree k in base b. Furthermore, say that we are told b. The task for our inference algorithm would be to somehow find r and n such that a_t is the $(t+1)^{st}$ term in the base b expansion of $\frac{r}{n}$. It should be noted that our algorithms are not required to infer the base! Given this we will generally take b to be 10, since we're mainly interested in decimal expansions.

Theorem 3. *"Let $\{a_t\}_{t\geq 0}$ be generated by a rational of degree k in base b. We can infer $\{a_t\}_{t\geq 0}$ given at least the first $2k+1$ terms of the sequence. Furthermore this bound is tight."*

Proof. Given in [BBS82] in Theorem 2: Problem 4. ■

Since we are mainly interested in decimal expansions of rational numbers, we will let $b = 10$ for the remainder of this thesis. It should be noted that the inference algorithm we provide is applicable to all bases.

The proof of Theorem 3 gives the following algorithm for inferring rational sequences:

```
1 : InferRational([a_0, a_1, ..., a_{n-1}], b)
2 :   f = b^{-(k+1)} sum_{i=1}^{n} a_{n-i} b^{i-1}
3 :   [c_1, c_2, ..., c_l] = ContinuedFractionExpansion(f)
4 :   for i = 1, 2, ..., l do
5 :     r/n = Convergent([c_1, ..., c_i])
6 :     if (f · b^{k+1} = ⌊b^{k+1} · r/n⌋) then
7 :       return (r, n)
8 :
9 :   return FAIL
```

Figure 2.4: The rational inference algorithm.

2.2.4 Decimal Expansions of Superpositions

We now study a slightly more general class of decimal expansions. This class is defined by a certain (basis) set of numbers whose decimal expansions are known. Given a basis, the class will consist of all rational combinations of the elements in the basis. For example, say we know the decimal expansions of the following numbers

$$B = \left\{\pi, e, \phi, \gamma, \sqrt{2}\right\} \tag{2.12}$$

our class of sequence will consist of the decimal expansions of all real numbers of the form

$$x = c_1\pi + c_2 e + c_3\phi + c_4\gamma + c_5\sqrt{2} \tag{2.13}$$

where the c_i's are rational.

Definition 6. Let n be a positive integer and let $B = \{b_1, \ldots, b_n\}$ be an n-elements subset of $\mathbb{R}^n$. *The lattice spanned by* B is

$$\mathbb{L}(B) = \left\{ \sum_{i=1}^{n} c_i \cdot b_i \;\mid\; (c_1, \ldots, c_n) \in \mathbb{Z}^n \right\} \tag{2.14}$$

Theorem 4. "*Let* $\mathbb{L} \subseteq \mathbb{R}^n$ *be a lattice with reduced basis* $\{b_1, \ldots, b_n\}$. *For every* $x \in \mathbb{L} - \{\vec{0}\}$ *we have* $|b_1|^2 \leq 2^{n-1}|x|^2$."

Proof. Given in [LLL82] Proposition 1.11. ■

[LLL82] also gives an efficient algorithm for transforming a basis into a reduced basis. We will refer to this algorithm as LLL. For the remainder of this thesis, we will only need to know that LLL outputs a basis set, $\{b_1, \ldots, b_n\}$, such that b_1 has the property given in Theorem 4.

Definition 7. Let B be an independent subset of real numbers and let $\{a_t\}_{t\geq 0}$ be an integer sequence. We say that $\{a_t\}_{t\geq 0}$ *is generated by* B if there are rationals $c_1, \ldots, c_n$ such that

$$x = \sum_{i=0}^{n} c_i \cdot b_i = \sum_{i=1}^{\infty} a_i 10^{-i} \tag{2.15}$$

In other words, our sequence is the decimal expansion of x.

Theorem 5. *"Let $B \subseteq \mathbb{R}$ and let $\{a_t\}_{t\geq 0}$ be a sequence generated by B. We can eventually infer $\{a_t\}_{t\geq 0}$."*

Proof. Since $\{a_t\}_{t\geq 0}$ is generated by B we know there are integers $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ such that

$$x = \sum_{i=1}^{n} \frac{p_i}{q_i} \cdot b_i \tag{2.16}$$

If we let $M = \operatorname{lcm}(q_1, \ldots, q_n)$ and let $c_i = M\frac{p_i}{q_i} \in \mathbb{Z}$ then

$$-M \cdot x + \sum_{i=1}^{n} c_i \cdot b_i = 0 \tag{2.17}$$

Now say we are given the first t terms of $\{a_t\}_{t\geq 0}$. We can try to infer our sequence by using LLL on the following two matrices

$$U_t = \begin{pmatrix} \lfloor b_1 \cdot 10^t \rfloor & 1 & 0 & \cdots & 0 \\ \lfloor b_2 \cdot 10^t \rfloor & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lfloor b_n \cdot 10^t \rfloor & 0 & 0 & \cdots & 1 \end{pmatrix} \tag{2.18}$$

$$V_t = \begin{pmatrix} \lfloor b_1 \cdot 10^t \rfloor & 1 & 0 & \cdots & 0 & 0 \\ \lfloor b_2 \cdot 10^t \rfloor & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \lfloor b_n \cdot 10^t \rfloor & 0 & 0 & \cdots & 1 & 0 \\ \lfloor x_t \cdot 10^t \rfloor & 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \tag{2.19}$$

Ideally almost all of the vectors in the lattice spanned by the rows of U_t are large. This should intuitively be true as the b_i's were chosen to be an independent set of real numbers. On the other hand, the lattice spanned by the rows of V_t should contain a small vector of the following form that is independent of t.

$$(\sim 0, c_1, c_2, \ldots, c_n, M) \tag{2.20}$$

If we can find this vector (or some constant multiple of it) we will have successfully made an inference. We can get close by using the LLL lattice reduction algorithm twice. We first run LLL on the rows of U_t giving us the reduced basis $u_1, \ldots, u_n$. If we let $u = u_1$ and u^* be the smallest non-zero vector in $\mathbb{L}(\text{rows of } U_t)$, then by Theorem 4 we have

$$|u^*|^2 \leq |u|^2 \leq 2^{n-1} \cdot |u^*|^2 \tag{2.21}$$

We can now run LLL on the rows of V_t to get the reduced basis $v_1, \ldots, v_n$. If we now let $v = v_1$ and v^* be the smallest non-zero vector in $\mathbb{L}$(rows of V_t), then by Theorem 4 we have

$$|v^*|^2 \leq |v|^2 \leq 2^n \cdot |v^*|^2 \tag{2.22}$$

Now note that every vector in $\mathbb{L}$(rows of U_t) can be made into a vector in the $\mathbb{L}$(rows of V_t) by appending a 0 to the vector. Also since the b_i's were chosen to be independent we know that as we get more and more sequence terms (i.e. as t increases) the size of the vector u^* will also increase. Empirical results indicate that this growth is actually very fast, i.e. $|u^*| \approx 2^t$. Thus if we are given sufficiently many sequence terms, i.e. $t > O(n)$, we will have

$$|v^*|^2 \leq |v|^2 \leq 2^n \cdot |v^*|^2 < |u^*|^2 \leq |u|^2 \leq 2^{n-1} \cdot |u^*|^2 \tag{2.23}$$

and by transitivity

$$|v|^2 \leq 2^n \cdot |v^*|^2 < |u|^2 \tag{2.24}$$

Ideally we would like $v = v^*$ or at least we would like it to be some constant multiple of v^*, so we can comfortably make an inference when

$$2^n |v|^2 < |u|^2 \tag{2.25}$$

This will eventually happen as u will continue to grow at an exponential rate while v will not change by much. Once we have a vector v, we can just check that its last coordinate is non-zero. If this is the case we have successfully inferred $\{a_t\}_{t \geq 0}$. ■

We state the algorithm given in the proof of Theorem 5 more formally in Figure 2.5

1 : **InferLLLSequence**$([a_0, a_1, \ldots, a_{n-1}], [b_1, \ldots, b_k])$
2 : $x = \sum_{i=0}^{n-1} a_i \cdot 10^{-(i+1)}$
3 : $\{u_1, \ldots, u_n\} =$ **LLLLaticeReduce**(U_t)
4 : $\{v_1, \ldots, v_n\} =$ **LLLLaticeReduce**(V_t)
5 : **if** $(|v_1|^2 \cdot 2^k < |v_1|^2) \wedge (v_1[n+1] \neq 0)$ **then**
6 : **return** v_1
7 : **else**
8 : **return** $FAIL$

Figure 2.5: The LLL Inference Algorithm. Note that the matrices U_t and V_t are defined as in equations 2.18 and 2.19 respectively

We also note that the LLL Inference Algorithm given in Figure 2.5 can be used to infer sequences other than rational combinations of the elements in our basis B. We can apply a number of transformation to any sequence we are given. For example, we can take the logarithm of any any sequence. Thus we can use this algorithm to also infer decimal expansions of $x \in \mathbb{R}$ such that

$$\ln(x) = \sum_{i=1}^{n} c_i \cdot b_i \tag{2.26}$$

for rational c_i's. We can do the same for $e^x, \ln(\ln(x))$, etc. This property makes this algorithm very useful in practice.

2.2.5 Automatic Sequences

We now study the class of automatic sequences.

Definition 8. A *partitioned deterministic finite automaton (PDFA)* is given by a six tuple

$$M = (Q, \Sigma, \delta, q_0, \Delta, \mathbf{F}) \tag{2.27}$$

where Q is a finite set of states, Σ is the input alphabet, Δ is the output alphabet, $q_0 \in Q$ is the start state, $\delta : Q \times \Sigma \to Q$ is a transition function, and $\mathbf{F} = (F_a \subseteq Q \mid a \in \Delta)$ is a partition of Q with $|\Delta|$ parts.

Definition 9. Let $M = (Q, \Sigma, \delta, q_0, \Delta, \mathbf{F})$ be a PDFA. For convenience we define $\delta^* : Q \times \Sigma^* \to Q$ as follows

$$\delta^*(q, \epsilon) = q \tag{2.28}$$

$$\delta^*(q, \sigma_1) = \delta(q, \sigma_1) \tag{2.29}$$

$$\delta^*(q, \sigma_1\sigma_2 \cdots \sigma_n) = \delta^*(\delta(q, \sigma_1), \sigma_2 \cdots \sigma_n) \tag{2.30}$$

Definition 10. Let $M = (Q, \Sigma, \delta, q_0, \Delta, \mathbf{F})$ be a PDFA. We say M *outputs* $a \in \Delta$ *when run on input* $s \in \Sigma^*$ if and only if $\delta^*(q_0, s) \in F_a$.

Example. A simple PDFA is given in Figure 2.6. For that PDFA we have $Q = \{p, q\}$, $\Sigma = \{0, 1\}$, δ is defined as shown in the transition diagram in Figure 2.6, $q_0 = p$, $\Delta = \{0, 1\}$, and $\mathbf{F} = (F_0 = \{p\}, F_1 = \{q\})$.

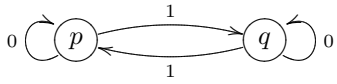


Figure 2.6: This is the PDFA that generates the famous Thue-Morse sequence. The first few terms of the sequence generated by this machine are 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1.

Definition 11. Let $M = (Q, \Sigma, \delta, q_0, \Delta, \mathbf{F})$ be a PDFA and let $p \in Q$. We say *the k-behavior of p, denoted* $[\![p]\!]$*, is the sequence generated by the PDFA* $M' = (Q, \Sigma, \delta, p, \Delta, \mathbf{F})$.

Definition 12. Let $\{a_t\}_{t\geq 0}$ be an integer sequence and let k be a positive integer. We say that $\{a_t\}_{t\geq 0}$ *is k-automatic* if and only if there is a PDFA

$$M = (Q, \Sigma, \delta, q_0, \Delta, \mathbf{F})$$

such that $\Sigma = \Sigma_k = \{0, 1, \ldots, k-1\}$ and for all $t \geq 0$

$$a_t = \delta^*(q_0, w) \tag{2.31}$$

if $w \in \Sigma^*$ is the reverse base k representation of t.

Definition 13. Let $\{a_t\}_{t\geq 0}$ be an integer sequence. We define *the k-kernel of* $\{a_t\}_{t\geq 0}$ the set of subsequences

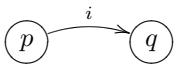
$$K = \left\{ \left\{ a_{t\cdot k^i + j} \right\}_{t\geq 0} \mid i \geq 0 \text{ and } 0 \leq j < k^i \right\} \tag{2.32}$$

For convenience we define $K_{i,j} \in K$ to be $\left\{a_{t\cdot k^i + j}\right\}_{t\geq 0}$.

Theorem 6. *"A sequence is k-automatic if and only if its k-kernel is finite."*

Proof. Given in [AS03] on page 185 Theorem 6.6.2. We present an independent proof.

Let $\{a_t\}_{t\geq 0}$ be a k-automatic sequence, as such there is a minimal PDFA that generates it. Let $M = (Q, \Sigma_k, \delta, q_0, \Delta, \mathbf{F})$ be this PDFA. The state set of this PDFA corresponds to the k-kernel of $\{a_t\}_{t\geq 0}$. To see this remember that these PDFA take input in reverse base k notation (i.e. least significant digit first). Thus you can view a transition from a state p to a state q on $i \in \Sigma_k$ as subtracting by i and then dividing by k.



(2.33)

In other words, the j^{th} term in the sequence generated by M with the start state changed to q, is just going to be the $(kj+i)^{th}$ term in the sequence generated by M with the start state changed to p. Thus given M we can simply do a breadth first search on the transition diagram to generated the k-kernel. Since the k-kernel corresponds naturally to Q we know that it is finite.

To see that the converse is true, let $\{a_t\}_{t\geq 0}$ be an integer sequence with a finite k-kernel, K. We now want to build a PDFA that computes the sequence. First make the k-kernel the state set of our machine, i.e.

$$Q = K \tag{2.34}$$

and pick Σ_k and $\Delta = \bigcup_{t\geq 0} \{a_t\}$ to be our input and output alphabets respectively. The first thing to note is that $\{a_t\}_{t\geq 0}$ must be an element of the k kernel. Choose this to be start state of the machine we are creating, i.e.

$$q_0 = \{a_t\}_{t\geq 0} \tag{2.35}$$

Define the transition relation as follows

$$\delta(K_{i,j}, \ell) = K_{i+1,\ell\cdot k^i+j} \tag{2.36}$$

We finally define our output partition by the first term in the sequence defined by each element of the k-kernel, i.e. for $\sigma \in \Delta$

$$K_{i,j} \in F_\sigma \iff a_j = \sigma \tag{2.37}$$

By construction the k-behavior of each $K_{i,j} \in Q$ will be exactly $\left\{a_{k^i\cdot t+j}\right\}_{t\geq 0}$. Specifically the k-behavior of q_0 will be $\{a_t\}_{t\geq 0}$, which means the given sequence is k-automatic. ■

We can use Theorem 6 to design an algorithm for inferring automatic sequences. At a high level the algorithm will simply use the finite initial portion of a sequence to build an approximation to its k-kernel. We can then use this approximate k-kernel to build a PDFA. But first let us define let us define our complexity measure for automatic sequences.

Definition 14. Let $\{a_t\}_{t\geq 0}$ be an integer sequence and let k, d, m be positive integers. We say that $\{a_t\}_{t\geq 0}$ *is generated by a PDFA of degree* (k, d, m) if $\{a_t\}_{t\geq 0}$ is k-automatic and the smallest PDFA that computes it has m states and an output alphabet of size d.

Let say some sequence,$\{a_t\}_{t\geq 0}$, is generated by a PDFA of degree (k, d, m). Furthermore say that we are given k and d. The task for our inference algorithm would be to somehow find m along with a m-state PDFA that generates our sequence. Once we have such a procedure, we can pick d to be the number of distinct sequence terms seen so far and then we can enumerate all plausible values of k based on the number of sequence terms we have available at our disposal.

Lemma 7. *"Let $M = (Q, \Sigma, \delta, q_0, \Delta, \mathbf{F})$ be a minimal PDFA. If each $\sigma \in \Delta$ is output by some state, then for any two states the length of the shortest word that distinguishes them is bounded above by $|Q| - |\Delta|$."*

Proof. Disrciminating words for a given machine can be generated by a standard minimization algorithm based on refining partitions. One can show that any such algorithm can take at most $m - d$ rounds. Thus our bound follows. ■

Theorem 8. *"Let $\{a_t\}_{t\geq 0}$ be generated by a PDFA of degree (k, d, m). We can infer $\{a_t\}_{t\geq 0}$ given at least the first k^{2m-d} terms of the sequence. Furthermore, this bound is tight."*

Proof. We will use the algorithm given in Theorem 6 (stated more formally in Figure 2.8) to infer $\{a_t\}_{t\geq 0}$.

Let M be the smallest PDFA that generates $\{a_t\}_{t\geq 0}$ (it should be noted that this machine exists and is unique; this can be seen via a simple extension of the quotient minimization algorithm for DFAs). In the proof of Theorem 6, we characterized the states of a PDFA to be elements of the k-kernel of the sequence that the PDFA generated. Thus we will represent the states of the PDFA we are generating as subsequences of the elements in the k-kernel of $\{a_t\}_{t\geq 0}$. In order to discover a state $K_{i,j}$ we need to have seen at least the j^{th} term of the sequence.

Now consider a state in M, say q_r, that is furthest from the start state, say it is r transitions away. Since this state is r transitions away, it is going to correspond to $K_{r,j}$ in the k-kernel for some j. In general this j can be as large as $k^r - 1$. This means that in the worst case we will have to see at least the first k^r terms of the sequence to even have a chance of finding all of the states.

Now given that we have enough terms to have potentially discovered every state, we will need to have seen enough terms to be able to distinguish any two states (k-kernel elements) from one another. So we will need to see enough terms of *each k-kernel element* to be able to tell them apart from one another. This corresponds exactly to differentiating states by their k-behavior. By Lemma 7 we know that the length of the longest differentiating word is bounded above by $m - d$. Since q_r could be the state that requires this longest differentiating word we might need to see a string of length $m - d + r$; thus requiring at least the first $k^r k^{m-d}$ sequence terms. Since we define the transition function by taking decimations, we will want to be able to have enough terms to distinguish all decimations of q_r. To do this we're going to need k times as many sequence terms as we currently have for the k-kernel element corresponding to q_r. This can be achieved by just giving k times as many terms of the original sequence. Now we can distinguish all the states from one another and soundly check if a decimated state is a new state by looking at prefixes.

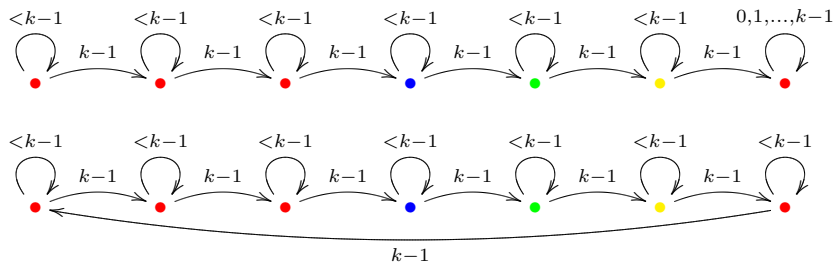


Figure 2.7: Lower Bound PDFAs: with $m = 7$ and $d = 4$. These two PDFAs agree for the first $k^{2m-d} - 1 = k^{10} - 1$ sequence terms. Note the colors of the states (red, blue, green, and yellow) indicate the output of a computation that halts in that state. Also note that the start state in each machine is the state furthest to the left.

Thus, we finally can conclude that in the worst case we will need to see at least $k^r k^{m-d} k$ sequence terms before the correct PDFA can be found. Since r is bounded above by $m-1$ we know that $\{a_t\}_{t\geq 0}$ can be inferred given at least the first k^{2m-d} sequence terms.

To see that this bound is tight, consider the two PDFAs in Figure 2.7 as an example. This example can easily be generalized to show that we always need at least k^{2m-d} sequence terms to infer a PDFA of degree (k, d, m). Both machines have $Q = \{1, \ldots, m\}, q_0 = 1, \Delta = \{1, \ldots, d\}, \Sigma = \{0, \ldots, k-1\}$, and output partition defined by

$$i \in F_d, \text{ for } 1 \leq i \leq m - d \tag{2.38}$$

$$(m - d) + i \in F_i, \text{ for } 1 \leq i \leq d \tag{2.39}$$

The transition functions for these machines will be very similar.

$$\delta_1(q, i) = \begin{cases} q+1 & \text{, if } i = k-1 \text{ and } q < m \\ q & \text{, otherwise} \end{cases} \tag{2.40}$$

$$\delta_2(q, i) = \begin{cases} q+1 & \text{, if } i = k-1 \text{ and } q < m \\ 1 & \text{, if } i = k-1 \text{ and } q = m \\ q & \text{, otherwise} \end{cases} \tag{2.41}$$

One can easily verify that these two machines will agree for the first $k^{2d-m} - 1$ sequence terms (i.e. all $0 \leq t < k^{2d-m} - 1$). Thus we will need at least k^{2d-m} sequence terms in order to tell them apart. ■

We can use the algorithm given in Theorem 8, stated more formally in Figure 2.8, as our inference algorithm for automatic sequences.

2.2.6 Turing Machines

The most general class we study is the class of algorithmically generated integer sequences.

```
1 :  InferAutomaticSequence([a_0, a_1, ..., a_{n-1}], k)
2 :    Σ = {0, 1, ..., k − 1}
3 :    Δ = ⋃_{i=0}^{n−1} {a_i}
4 :    F = (F_a = {}  |  a ∈ Δ)
5 :    A = [a_0, a_1, ..., a_{n-1}]
6 :    Q = K = {A}
7 :    while (Q is not empty) do
8 :      pick some X ∈ Q
9 :      F_{X[0]} = F_{X[0]} ∪ {X}
10 :     Q = Q − {X}
11 :     for i = 0, 1, ..., k − 1 do
12 :       D = Decimate(X, k, i)
13 :       δ(X, i) = D
14 :       if D ∉ K then
15 :         K = K ∪ {D}
16 :         Q = Q ∪ {D}
17 :
18 :   return (K, Σ, A, δ, Δ, F)
19 :
20 :  Decimate([a_0, a_1, ..., a_{n-1}], k, i)
21 :   return [a_i, a_{k+i}, a_{2k+i}, ...]
```

Figure 2.8: The automatic sequence inference algorithm.

Definition 15. We say that a sequence, $\{a_t\}_{t\geq 0}$, *is generated by a Turing Machine of degree* k, if there is a k-state Turning Machine that on input $t \in \mathbb{N}$ halts and outputs a_t for all $t \geq 0$.

We now give an impossibility result that suggests that this class of sequences cannot be inferred in general. In other words, even if we are told a particular sequence is in this class has a certain degree we can never be sure of any hypothesis we infer regardless of how many sequence terms were used to make the hypothesis.

Definition 16. They Busy-Beaver function, $\beta : \mathbb{N} \to \mathbb{N}$, is defined such that for every positive integer n, $\beta(n)$ is equal to the maximum unary output a Turning Machine on n states can print given that it halts.

Fact 9. For any increasing computable function $f : \mathbb{N} \to \mathbb{N}$ there is an $n \in \mathbb{N}$ such that $f(n) \leq \beta(n)$.

Definition 17. Let $F : \mathbb{N} \to \mathbb{N}$ be the function on input k outputs the minimum number of sequence terms any inference algorithm needs to see before it can infer any sequence generated by a Turing Machine of degree k.

Theorem 10. *"F is not computable."*

Proof. For sake of contradiction assume that F is computable and let n be a positive integer . Now let B_n be a n-state busy beaver champion, i.e. let B_n be a n state Turing Machine that outputs $\beta(n)$ in unary when run on the empty

tape. Consider the following Turing Machine sequence with B_n hard coded as follows:

```
1 :  A(t)
2 :    m = B_n() + 1
3 :    if t ≤ m then
4 :      return 0
5 :    else
6 :      return 1
```

This sequence can be implemented in the Turing Machine model using $n + c$ states for some positive integer c (we need n states for hard-coding B_n and the remaining c states are needed for the rest of the procedure). We now note that this sequence cannot be distinguished from the zero sequence, $\{0\}_{t\geq 0}$, until we have seen more than the first $\beta(n) + 1$ sequence terms. Specifically we can conclude that

$$F(n + c) > \beta(n) \tag{2.42}$$

Since we assumed F is computable we know that $g : \mathbb{N} \to \mathbb{N}$ given by

$$g(n) = F(n + c) \tag{2.43}$$

is also computable. As n was arbitrary we can conclude that for any positive integer n

$$g(n) > \beta(n) \tag{2.44}$$

which is of course a contradiction to Fact 9 (since F is clearly an increasing function). Thus F is not computable. ■

This result suggests that even if an inference algorithm exists for this class, its output cannot be trusted since we couldn't even compute how many sequence terms we would have needed to give the algorithm for it to have been able to successful make an inference of degree k.

2.3 Confidence Functions

The Evolutionary Inference Model uses confidence functions to decide when to output a particular hypothesis. Confidence functions are meant to answer the following fundamental question:

How confident am I that I have seen enough sequence terms?

In the previous section, we gave many inference algorithms with bounds on the number of sequence terms they need to see before they are guaranteed to converge to the correct inference. What's hiding behind each of these guarantees is the assumption that the particular sequence you are looking at is in your concept class and you can bound its degree. This assumption is very fundamental to the nature of inference algorithms since we have good evidence to suggest that it is very unlikely for there to be an efficient inference algorithm for inferring an arbitrary algorithmically generated integer sequence (Theorem 10). This should intuitively be the case given the theory pseudo-randomness. For example, sequences generated by the $x^2 \mod N$ generator are a proper subset of the all algorithmically generated integer sequences, but as shown in [BBS82],

this class of sequences should not be inferable under standard cryptographic assumptions. We use confidence functions in our model to *pay* for the fact that inference algorithms need to make this computational assumption.

Definition 18. Let S be a concept class and let $C : S \times \mathbb{N} \to [0, 1]$. We say that C *is a confidence function for class* S provided C has the following properties:

Property 1 (Monotonicity)**.** Say we make some hypothesis, $\{h_t\}_{t\geq 0} \in S$, at some time t and we've kept this hypothesis until some time $t' > t$, then

$$C(\{h_t\}_{t\geq 0}, t') \geq C(\{h_t\}_{t\geq 0}, t)$$

In other words, our confidence in a hypothesis cannot decrease until we've found a counterexample to the hypothesis.

Property 2 (Convergence)**.** Say we make some hypothesis, $\{h_t\}_{t\geq 0} \in S$, at some time t_0 and this hypothesis actually does generate the input sequence. Then our confidence in this hypothesis should tend to 1 as t tends to infinity. More formally

$$\lim_{t\to\infty} C(\{h_t\}_{t\geq 0}, t_0 + t) = 1$$

A simple confidence function with these properties is the fraction of sequence terms that have been used to check our current hypothesis. We can define this simple confidence function for all of our inference classes, since we can always keep track of when we made our current hypothesis in the **EvolutionaryInference** procedure. Furthermore we have have theorems for each of our inference classes (besides superpositions) that tells us how many sequence terms we need to have seen to have any confidence in our current hypothesis given that it is of a certain degree. These theorems let us define a function, $f_S(k)$, that tells us the minimum number of sequence terms we need to see before we can successfully infer any $\{a_t\}_{t\geq 0} \in S$ of degree k. We summarize this information in the Figure 2.9 for the inference algorithms given in the previous section.

Concept Class (S)	Degree Measure	f_S
Polynomials	k	$k+1$
Linear Recurrences	k	$2k$
Rationals	k	$2k+1$
Automatic Sequences	(k, d, m)	k^{2m-d}

Figure 2.9: Summary of the bounds proven in the previous section for the number of sequence terms each of our algorithms needs to see before it can successfully infer any sequence in the concept class of a certain degree.

We can uses these functions to define the following confidence function

$$C_S(\{h_t\}_{t\geq 0}, t) = \frac{(t+1) - f_S(k)}{t+1} \tag{2.45}$$

where $\{h_t\}_{t\geq 0} \in S$ and has degree k. This family of confidence functions clearly has both of the properties that we wanted. It is monotonic since if $\{h_t\}_{t\geq 0}$ doesn't change, neither does it's degree. So as t increases so will $C_S(\{h_t\}_{t\geq 0})$.

It converges to 1 when the correct hypothesis has been found since the degree stops increasing while t continues to grow.

If we would like to have a confidence of $1 - \epsilon$ in some hypothesis of degree k, then we will need to run **EvolutionaryInference** for

$$t > \frac{1}{\epsilon} \cdot f_S(k) - 1 \tag{2.46}$$

time steps. This family of confidence function seems to work well in practice, but it is by no means resistant to an adversary. Consider polynomial sequences. For any $\epsilon > 0$ that we pick an adversary can give us the sequence defined by the following polynomial

$$p(x) = \prod_{i=0}^{m} (x - i) \tag{2.47}$$

where $m = \left\lceil \frac{1}{\epsilon}(k+1) + 1 \right\rceil$. This sequence will be indistinguishable from the zero sequence until we have at least its first $m+1$ terms, but before t can reach m we will have already become sufficiently confident in our hypothesis that this is the zero sequence. Similar adversarial sequences can be found for our other concept classes. Two very compelling open problems that have come out of this research in confidence functions:

Open Problem 1. What additional properties do we want confidence functions to have?

Open Problem 2. Are there confidence functions that can perform better against an adversary?

Good answers to these question could improve the performance of our model but a huge factor.

Chapter 3

The On-Line Encyclopedia of Integer Sequences

We evaluate the quality of our model by using the data in Sloane's On-Line Encyclopedia of Integer Sequences [Slo]. This is a truly wonderful resource that contains sequences that have come up in the course of academic research. You can think of this database as containing the set of "interesting sequences." For example, there is an entry for the Busy-Beaver sequence, the sequence of stops the A train makes in New York City, the Collatz sequence, etc.

As you may have already guessed most of the sequences in this database should be fairly difficult, if not impossible, to infer using the techniques we have outlined in this work. In fact the task of inferring these sequences is sometimes isomorphic to some of the most difficult open problem in modern mathematics: for example consider sequence A001359 which is the sequence whose t^{th} term is the smaller prime in the t^{th} twin prime pair. Successfully inferring this sequence would solve the Twin Prime Conjecture.

3.1 Inferring OEIS

The task of inferring the sequences catalogued in the On-Line Encyclopedia of Integer Sequences is a bit different than the inference tasks discussed in Chapter 2. For obvious reasons, we are only given access to about the first 100-200 terms of each sequence catalogued in the OEIS. Given this restriction we never return from **EvolutionaryInference** until we have seen all of the sequence terms we are given. We output a hypothesis only if it is consistient with all of the sequence terms we are given and if we have checked the hypothesis on at least a couple of sequence terms.

So far our methods have been able to infer about 18.934 % of the sequence cataloged in the On-Line Encyclopedia of Integer Sequences. A vast majority of these inferences are made with confidence far exceeding 50 %, i.e. they've been checked on at least half of the sequence terms available to us. A brief summary of our results is given in Figure 3.1.

Concept Class	Number Inferred	Percent Inferred
Polynomial	4908	2.803 %
Linear Recurrence	29366	16.769 %
Decimal Expansions (Rationals)	1871	1.068 %
Decimal Expansions (LLL)	2352	1.343 %
Automatic Sequences	1828	1.043 %
Total	33157	**18.934 %**

Figure 3.1: Inference statistics as of April 30, 2010 when the database had 175117 integer sequences. An overwhelming majority of these inferences are made with confidence far exceeding 50%.

3.2 Our Web Service

In the same spirit as the OEIS, we have published our own service for inferring integer sequences on the web. It can be viewed at

atemi.cdm.cs.cmu.edu/~samt/sequences.html.

At this website you can input a sequence and see how well each of our inference algorithms does trying to infer it. It is our hope that this service can be a good complement to Sloane's OEIS.

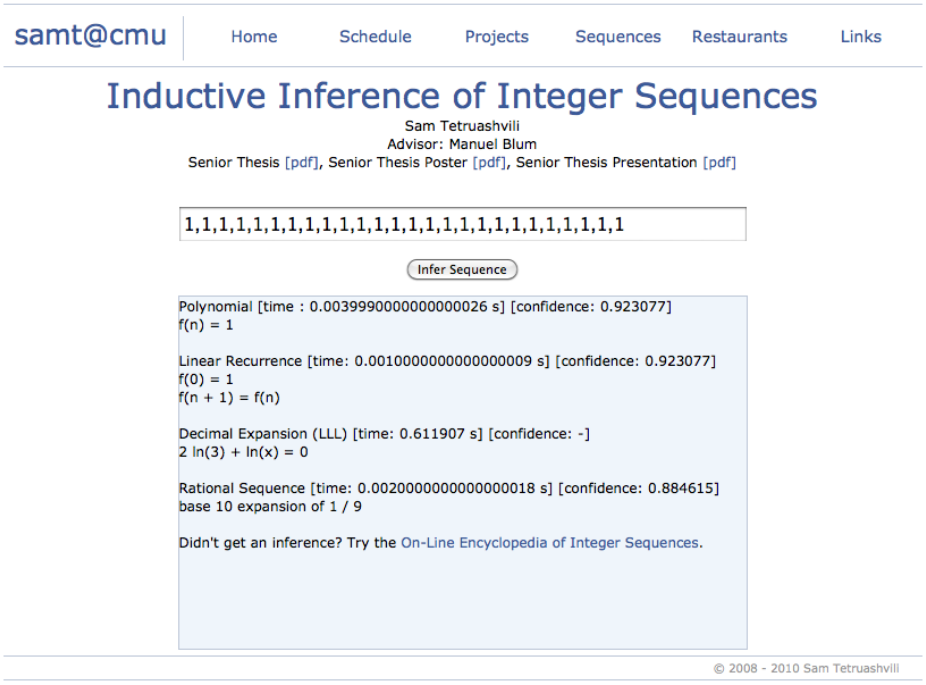


Figure 3.2: Our integer sequence inference service inferring the all 1s sequence.

Chapter 4

Future Work

4.1 Inference Algorithms

There is much more work to be done when it comes to finding more inference algorithms. We think that we have just begun to scratch the surface in this thesis. The following concept classes would be good candidates for new inference algorithms:

- k-Regular Sequences: See [AS03] chapter 16.
- Algebraic Numbers: We have an inference algorithm for these using LLL lattice reduction, but the problem is that it gives hypotheses that are polynomials. So we have no way of checking these hypothesis on future terms as required by line 5 of **EvolutionaryInference**, see Figure 2.1. Is there a way around this?
- Piecewise Linear Recurrences: such as the sequence

$$1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, \ldots \tag{4.1}$$

  Most humans can infer this sequence (as shown in [Ang74]) though the methods given in Chapter 2 are unsuccessful here. We do have a method for inferring these types of sequence; though this method needs far too many sequence terms. Thus it is not pratical. This method involves trying to break the sequence into parts, such as

$$|1|1, 2|1, 2, 3|1, 2, 3, 4|1, 2, 3, 4, 5| \ldots \tag{4.2}$$

  and then trying to infer a general for for making each of the parts. Here the rule is the i^{th} part will just contain the first i terms of the identity function.

Another interesting problem is trying to characterize the set of sequences that can be efficiently inferred. One may be able to make some progress on this problem by exploring primitive recursive functions or LOOP programs. In general these concept classes should not be efficiently inferable, though some interesting subsets of them may be. For example, consider LOOP programs in which nested loops are disallowed.

4.2 Confidence Functions

Confidence functions present many compelling open problems. We would first like to come up with more properties that these functions should have. So far as I can tell one should be able to do much better than simply asking for monotonicity and convergence. We would then like to come up with confidence functions that are more resistant to adversarial attack. By comparison, the confidence functions we gave in Chapter 2 perform very poorly in this regard.

In our research we have come up with a couple of approaches to these two problems. Our approaches have generally involved making confidence functions much more dependent on the properties of the concept class they are associated with. For example, let us consider the linear recurrence concept class. The inference algorithm given for this class in Figure 2.3 requires solving a matrix equation. The matrix used in this equation is

$$\begin{pmatrix} a_0 & a_1 & \cdots & a_{k-1} \\ a_1 & a_2 & \cdots & a_k \\ \vdots & \vdots & \ddots & \vdots \\ a_{k-1} & a_k & \cdots & a_{2k-2} \end{pmatrix} \tag{4.3}$$

Intuitively, unless we're given an adversarial sequence, the first $2k-1$ terms of a linear recurrence of degree k should look fairly chaotic. If we make the assumption that these terms are "random," then we could use the tools of probability to determine that the matrix above will have full rank with high probability. Thus a very high degree inference will be make, and in all likelihood this inference will fail to predict the next sequence term since it too is "random." For the class of linear recurrences, this intuition suggests that the marginal confidence gained by checking our inference on a single sequence term should adhere to the diminishing returns property. In other words, checking the hypothesis on just one sequence term should give us a lot of confidence since if the sequence was not linear recurrent we would most likely get a nonsensical high degree hypothesis. Furthermore, any additional check terms to give us less and less additional confidence.

Of course the downside of the diminishing returns property is that an adversary can take much more advantage of us now since we will become confident in hypothesis faster. One can try to get a handle on this problem by only increasing our confidence at certain times. For example, say the sequence of times that our previous hypotheses have failed is pretty regular. Say each hypothesis we've made so far has worked for four time steps and then failed on the fifth. In this case we would not start increasing our confidence in any hypothesis we get in the future until we see our hypothesis validated on five check terms. This can be generalized by trying to infer the sequence of times that our hypothesis have failed, and then not increasing confidence until the time we predict our hypothesis will fail. If we have a very high confidence in our inference of the failure sequence, then this technique could yield the result we desire since it would slow down the rate at which we increase our confidence. Combining this with the diminishing returns property outlined above, we could have a very useful confidence function for linear recurrence. Unfortunately, we have not been able to make the notion of inferring the failure sequence percise so there is still some work left to be done. If we can get past this hurdle we would have a very

useful confidence function for linear recurrences.

4.3 Applications

Inductive Inference has a wide array of applications. Good applications should have two fundamental properties. The first property is that there cannot be any error in the sequence that is fed to our inference algorithm. So a good application should only feed our model error-free data. The second property is that the application should need some sort of an exact answer, i.e. approximations are not good enough for these applications. We will now briefly touch on two such application to give you a flavor of the problems that inductive inference can be used to solve.

4.3.1 Finding Exact Roots of a Function

Our first application falls in the realm of Numerical Analysis. Say we are given some function $f : \mathbb{R} \to \mathbb{R}$ and we are asked to find a root of f, i.e. some $x \in \mathbb{R}$ such that $f(x) = 0$. There are currently many well known methods for finding successively better approximations to a root of f. Newton's Method is one of the most widely used such algorithms. But what if approximations are not good enough? What if we need an exact root. We can use **EvolutionaryInference** along with our decimal expansion inference algorithm to try to find an *exact* root. At a high level we will use Newton's Method to get better and better approximations to a root, i.e. Newton's method can always be used to to get another digit of the decimal expansion of an exact root. We can then feed the sequence defined by the approximate decimal expansion of the root to **EvolutionaryInference**. If the exact root happens to be in our concept class of decimal expansions, we will eventually be able to infer it given a sufficiently many terms of its decimal expansion. Once we have made an inference we can always plug it back into the given function to confirm if it is indeed an exact root (though this computation might have to be symbolic rather numerical).

We believe similar methods can be used to make an attempt at getting an exact answers to problems that we can get successively better approximations to. As such we believe that there are many other similar applications of inductive inference.

4.3.2 A Novel Spreadsheet Programming Interface

Our second application falls in the realm of Human-Computer Interaction. In current spreadsheet applications, such as Microsoft Excel, the user is forced to do a lot of cumbersome and repetitive work. Spreadsheets are much better than they used to be, but we think they can be improved with the use of inductive inference; especially for users who are less familiar with computer programming.

Consider the problem of trying to multiply two matrices that are given in a spreadsheet, see Figure 4.1. Say a user is using the standard matrix multiplication algorithm in a spreadsheet. For example, as seen in Figure 4.1, the user computes entry (1, J) in the product matrix by inputting the following formula into row (1 ,J)

$$= (\mathrm{A1} * \mathrm{E1}) + (\mathrm{B1} * \mathrm{E2}) + (\mathrm{C1} * \mathrm{E3}) \tag{4.4}$$

Say the user does this for the first row and a bit of the second row of the product matrix. An overseer program can look at the user's input and turn it into a sequence that indicates the users actions. Then our inference procedure can be used to predict the users future actions, i.e. how the user would have computed the remaining terms of the product matrix. Given this inference the overseer program can prompt the user for permission to finish off the rest of his computation for him. If the user gives their consent, then the overseer program will fill in the rest of the product matrix. Furthermore, the overseer program can remember the procedure it inferred and then let the user use it in the future. A properly implemented system like this could make the experience of using a spreadsheet more efficient and enjoyable for the user. In this manner, a user that is not very familiar with computer programming can "program" a spreadsheet to compute some simple procedure for them.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	1	0	0		3	2	1	1		3	2	1	1
2	0	0	1		4	0	1	2	=	7	2	8	1
3	-	-	-		7	2	8	1					

Figure 4.1: An example of matrix multiplication. If the user has shown us how to get the first row of the product matrix an overseeing inference algorithm should be able to learn the procedure that the user is using to multiply two matrices and then store this procedure. In the future the user should be able to ask the spreadsheet to multiply any two matrices.

References

[Ang74] Dana Angluin. Easily Inferred Sequences. Technical report, University of California at Berkeley, Department of EECS, 1974.

[AS83] Dana Angluin and Carl H. Smith. Inductive Inference: Theory and Methods. *ACM Computing Surveys*, 1983.

[AS03] Jean-Paul Allouche and Jeffrey Shallit. *Automatic Sequences: Theory, Applications, Generalizations.* Cambridge University Press, first edition, 2003.

[BBS82] Lenore Blum, Manuel Blum, and Michael Shub. Comparison of Two Pseudo-Random Number Generators. *Advances in Cryptology-Proceedings of Crypto '82*, 1982.

[Gol67] Mark E. Gold. Language Identification in the Limit. *Information and Control*, 1967.

[Knu97] Donald E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Addison-Wesley Professional, third edition, 1997.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovasz. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 1982.

[Slo] N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. http://www.research.att.com/njas/sequences/.

[Wil94] Herbert S. Wilf. *generatingfunctionology.* Academic Press, second edition, 1994.