Journal of Machine Learning Research 9 (2008) 1823-1852 Submitted 11/07; Revised 6/08; Published 8/08

Learning Balls of Strings from Edit Corrections*

Leonor Becerra-Bonache LEONOR.BECERRA-BONACHE@YALE.EDU
Department of Computer Science
Yale University
51 Prospect Street
New Haven, CT, 06511, USA

Colin de la Higuera CDLH@UNIV-ST-ETIENNE.FR
Jean-Christophe Janodet JANODET@UNIV-ST-ETIENNE.FR
Frédéric Tantini FREDERIC.TANTINI@UNIV-ST-ETIENNE.FR
Universities of Lyon
Laboratoire Hubert Curien
18 rue du Professeur Benoît Lauras
42000 Saint-Étienne, France

Editor: Rocco Servedio

Abstract

When facing the question of learning languages in realistic settings, one has to tackle several problems that do not admit simple solutions. On the one hand, languages are usually defined by complex grammatical mechanisms for which the learning results are predominantly negative, as the few algorithms are not really able to cope with noise. On the other hand, the learning settings themselves rely either on too simple information (text) or on unattainable one (query systems that do not exist in practice, nor can be simulated). We consider simple but sound classes of languages defined via the widely used edit distance: the balls of strings. We propose to learn them with the help of a new sort of queries, called the correction queries: when a string is submitted to the Oracle, either she accepts it if it belongs to the target language, or she proposes a correction, that is, a string of the language close to the query with respect to the edit distance. We show that even if the good balls are not learnable in Angluin's MAT model, they can be learned from a polynomial number of correction queries. Moreover, experimental evidence simulating a human Expert shows that this algorithm is resistant to approximate answers.

Keywords: grammatical inference, oracle learning, correction queries, edit distance, balls of strings

1. Introduction

Do you know how many *Nabcodonosaur* were kings of Babylon? And do you know when *Arnold Shwartzeneger* was born? Just two decades ago, you would have had to consult encyclopedias and Who's Who dictionaries in order to get answers to such questions. At that time, you may have needed this information in order to participate to quizzes and competitions organized by famous magazines during the summers, but because of *these* questions, you might possibly have missed the very first prize. Why?... Nowadays, everything has changed: you naturally use the Web, launch

*. This paper is an extended version of "Learning Balls of Strings with Correction Queries" presented at the 2007 European Conference in Machine Learning (ECML'07).

your favorite search engine, type two keywords, follow three links and note down the answers. In this particular case, you discover...that no king of Babylon was called *Nabcodonosaur* but two *Nab***u**c**h***odonos****o****r*'s reigned there many centuries ago. Again, the day *Arnold Shwartzeneger* was born is not clear, but it is easy to check that *Arnold S***c***hwa***rz***ene***g***ger* was born in 1947, July 30^{th}.

So you would probably win today the great competitions of the past. Indeed, the actual search engines are able to propose *corrections* when a keyword is not frequent. Those corrections are most often reliable because the reference dictionary is built from the billions of web pages indexed all over the world. Hence, a search engine is playing the role of an imperfect but powerful Oracle, able to validate a relevant query by returning relevant documents, but also to correct any suspect query. Such an Oracle is able to answer to what we shall call *correction queries*.

The first goal of this paper is to show, from a theoretical standpoint, that the concept of correction query allows one to get new challenging results in the field of Active Learning. In this framework, developed by Angluin in the 80's (Angluin, 1987b), a Learner (He) has access to an Oracle (She) that knows a concept he must discover. To this purpose, he submits different kinds of queries (e.g., Correction Queries) and she has to answer without lying. The game ends when he guesses the concept. Query-based Learners are often interesting from a practical viewpoint. For instance, instead of requiring a human expert to label huge quantities of data, this expert could be asked by the Learner, in an interactive situation, to provide a small amount of targeted information.

The second goal of this paper is to provide evidence that correction queries are suitable for this kind of real-life applications. However, assuming that the Oracle is a human expert introduces new constraints. On one hand, it is inconceivable to ask *a polynomial* number of queries: this may still be too much for a human. So the learning algorithm should aim at minimizing the number of queries even if we must pay for it with a worse time complexity. On the other hand, a human being (or even the Web) is fallible. Therefore the learning algorithm should also aim at learning functions or languages from *approximate* corrections.

In the above Web example, the search engine uses the frequency of words to propose corrections. In consequence, correct words (e.g., *malophile* = someone who loves apples) are sometimes subject to a correction (e.g., *halophile* = a cell which thrives in environments with high concentrations of salt). Another key point is the distance used to find a closest correct string; it is a variant of the *edit distance*, also called the *Levenshtein distance*, which measures the minimum number of deletion, insertion or substitution operations needed to transform one string into another (Levenshtein, 1965; Wagner and Fisher, 1974). This distance have been used in many fields including Computational Biology (Gusfield, 1997; Durbin et al., 1998), Language Modelling (Amengual et al., 2001; Amengual and Dupont, 2000) and Pattern Recognition (Navarro, 2001; Chávez et al., 2001).

Edit distance appears in specific Grammatical Inference problems, in particular when one wants to learn languages from noisy data (Tantini et al., 2006). In this domain, the classes of languages studied are not defined following the Chomsky Hierarchy. Indeed, even the easiest level of this hierarchy, the class of regular languages, is not at all robust to noise, since it includes all the parity functions, which can be defined as regular languages and are not learnable in the presence of noise (Kearns and Li, 1993). In order to avoid this difficulty, we shall consider only special finite languages, that may seem elementary to formal language theoreticians, but are relevant for topologists and complex for combinatorialists: the *balls of strings*.

Balls of strings are formed by choosing one specific string, called the centre, and all its neighbours up to a given length for the edit distance, called the radius. From a practical standpoint, balls of strings appear in a variety of settings: in approximate string matching tasks, the goal is to find all

close matches to some target string (Navarro, 2001; Chávez et al., 2001); in noisy settings, garbled versions of an unidentified string are given and the task is to recover the original string (Kohonen, 1985); when using dictionaries, the task can be described as that of finding the intersection between two languages, the dictionary itself and a ball around the target string (Schulz and Mihov, 2002); in the field of bioinformatics, extracting valid models from large data sets of DNA or proteins can involve looking for substrings at distance less than some given bound, and the set of these approximate substrings can also be represented by balls (Sagot and Wakabayashi, 2003).

Hence, in this paper, we study the problem of identifying balls of strings from correction queries. In Section 2, we present the motivations of our work; we discuss why noise is a problem (2.1), which queries should be used to learn languages (2.2), and the relevance of a fallible Oracle in real applications (2.3). Definitions are given in Section 3, where we pay special attention to the definitions of edit distance (3.1), balls of strings (3.2), and correction queries (3.3). On one hand, we prove that the balls are not learnable with Angluin's membership and equivalence queries, and on the other hand, that the deterministic finite automata are not learnable with correction queries.

The main result of the paper is shown in Section 4. It consists of a polynomial time algorithm that infers any ball from correction queries. We explain some technical results (4.1), and we present the algorithm (4.2). An important question is raised concerning the fact that only good balls can be learned with a polynomial number of correction queries (4.3). In Section 5, we study the effectiveness of our algorithm in more practical situations. First, we are concerned with the case where the Oracle is fallible (5.1). Next, we try to minimize the number of queries asked, considering the fact that the expensive resource is the expert playing the part of the Oracle, not the machine making the computations (5.2). We conclude in Section 6.

2. Motivations and Related Work

Several questions need to be addressed before tackling the core of the problem.

2.1 Why is it Hard to Learn Languages in Noisy Settings?

Languages can either be generated, recognized or defined by mechanisms like regular expressions, finite state automata or formal grammars (Harrison, 1978; Sakarovich, 2004; Salomaa, 2006). Alternatively equations can define properties that the strings in the language should verify (Clark et al., 2006). The techniques enabling to learn such formalisms are known as grammatical inference, and new algorithms are developed all the time. But there is one issue that is systematically problematic for such algorithms: that of dealing with noise.

Results obtained in the field of grammatical inference show that learning in noisy situations is hard (de la Higuera, 2006). Some attempts to deal with this problem can be found, for example, in the GOWACHIN (Lang et al., 1998) and GECCO competitions (Lucas, 2004), where the problem of learning DFA from noisy examples was the main challenge.

Noise over strings can, in the simplest case, just affect the labels: a string in the language will be classified as not belonging, whereas a string can be labeled inside the language when it is not. It is known since (Trakhtenbrot and Barzdin, 1973; Angluin, 1978) that with even small doses of noise, learning automata is hard.

The second sort of noise that one may encounter with strings, which is possibly most characteristic here, consists in having the strings slightly modified through some noisy channel. This type of

noise is invariably described by the edit distance (Levenshtein, 1965): individual symbols appear or disappear, or even are transformed into different ones.

Again, for the edit noise, the typical classes of languages belonging to the Chomsky hierarchy (Chomsky, 1957; Sakarovich, 2004) are far from robust. Consider for instance the set of strings over the alphabet $\{0,1\}$ whose parity of 0's is identical to the parity of 1's (see Figure 1). This typical

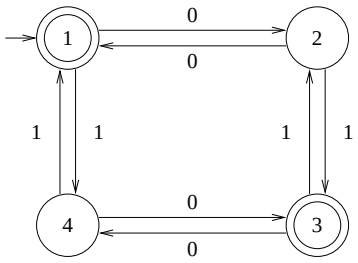

Figure 1: An automaton recognizing $\{w \in (0+1)^* : |w|_0 \bmod 2 = |w|_1 \bmod 2\}$.

regular language is clearly very sensitive to the noise: if any symbol is inserted or deleted in a string, the string will cease to belong to the language; and conversely, any string out of the language will be transformed into a string from the language, as the parity of either of the letters will change.

Unfortunately, the picture is even less clear with other regular languages such as 0^*1^* and $(010)^*$, or higher languages in the Chomsky hierarchy such as $\{ww : w \in (0+1)^*\}$ or the set of palindromes or $\{0^n1^n2^n : n \geq 0\}$. Indeed, these languages are sparse in the set of all strings, so trying to learn them from noisy data is like looking for a needle in a haystack: no string seems to belong to the target anymore.

The reader may think that probably, all these textbook languages are not relevant in practice. In which case, studying their learnability in the presence of noise would not be significative. Nevertheless, concerning randomly drawn regular languages, the picture is not better: the website developed by Coste et al. (1998) shows that despite a decade of efforts, no convincing solution has been yet found to take into account the noise during the learning process.

Therefore, if we are to learn languages in a noisy setting where the noise is modelled through the edit distance, we think that it is necessary to consider other classes of languages that could be much better adapted to this type of noise. The balls of strings are an example of such languages.

2.2 What Queries Should we Use?

Learning with queries was introduced by Angluin in order to provide a firm mathematical background to machine learning in a non statistical setting (Angluin, 1987b). In this paradigm, both positive and negative results are relevant. Indeed, if one cannot learn using a polynomial number of questions, then one cannot do it from data that one gets without choice from the environment. In this setting the questions are called queries and they are asked to a perfect abstract machine, called Oracle.

Several types of queries have been studied, and some classes were proved to be learnable from specific combination of queries (see Angluin, 2004, for a survey). The best known and most important of such positive results is that deterministic finite state automata are polynomially learnable

from a combination of membership queries and strong equivalence queries (Angluin, 1987a). The corresponding definitions will be given in Section 3.3.

We argue that equivalence queries are not realistic for the intended applications, and we choose to use the recently introduced correction queries instead (Becerra-Bonache and Yokomori, 2004). When making a correction query, we submit a string to the Oracle who answers YES if the string is in the target language, and if it is not then the Oracle returns a string from the language that is closest to the query. This string is called the correction.

In order to give an introductory intuition, let us consider the case where we want to learn disks in the plane using the Euclidean distance. Instead of learning from examples (with the possibility of them being labeled), let us suppose we have access to an Oracle that will answer the following query: a point is proposed, and is returned either the answer YES or a correction of this point, that is, the closest point in the disk.

Then we can proceed in three stages to learn a disk of centre O and radius R with correction queries as shown in Figure 2:

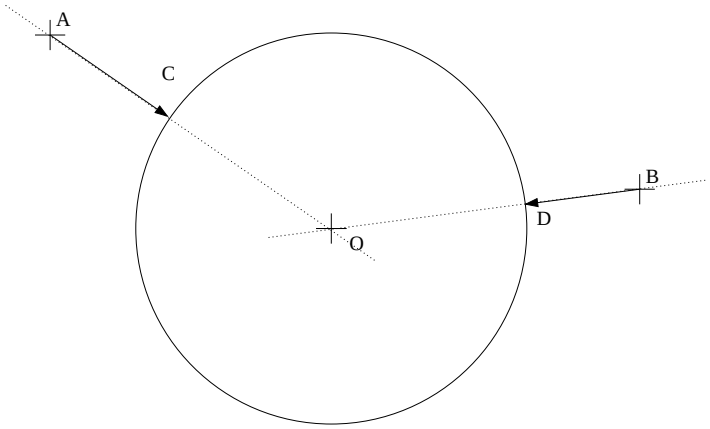

Figure 2: Three stages are sufficient to learn the disks of $\mathbb{R}^2$ with correction queries: (1) find two points A and B outside of the disk haphazardly using correction queries; (2) ask the Oracle for the corrections of A and B, which will result in C and D, respectively; (3) use a ruler to deduce the centre O and a compass to draw the circle.

1. We start by finding two points A and B outside of the disk we want to identify. Looking for them haphazardly by asking to the Oracle if such or such a point is in the disk is enough: intuitively, we are going to find them with very few queries.

2. We ask the Oracle for the corrections of A and B. Concerning A, the Oracle is going to return a point C inside the disk, as close as possible to A. Clearly, this point is at the intersection of the segment $[OA]$ and the boundary circle of the target disk. Likewise, let D be the correction of B.

3. We draw the lines (AC) and (BD) with a ruler: they intersect in O. Then we can draw the circle with a compass. We get the radius by measuring the distance between O and C.

Hence, it is easy to learn the balls of $\mathbb{R}^2$ with very few correction queries. Now, focusing on balls of strings, we may hope that the previous approach is good and try to reproduce it.

However, building the centre of a ball from strings on its periphery is difficult for at least two reasons. On one hand, (Σ^*, d) is a metric space with no vector space as underlying structure. This is similar to the case where we were trying to learn the disks of the plane with just a compass but no ruler.[1] On the other hand, the problem is formally *hard*:

Theorem 1 (de la Higuera and Casacuberta 2000) *Given a finite set of strings $W = \{w_1, \ldots, w_n\}$ and a constant K, deciding whether a string $u \in \Sigma^*$ exists such that $\max_{w \in W} d(u,w) < K$ (respectively $\sum_{w \in W} d(u,w) < K$) is $\mathcal{NP}$-complete.*

Therefore, we will have to study the balls in more detail and make the best possible use of the correction queries, so as not to build the centres from scratch.

2.3 Why might the Oracle be Fallible?

Above we argued that the active learning model was based on the strong assumption of a perfect Oracle. This corresponded to a reasonable assumption when dealing with mathematics and with the objective of being in a favorable setting in which negative results could be blamed on the complexity of the task and not on the adversarial nature of the Oracle.

But in recent years, the active learning setting (corresponding to learning from an Oracle) has been accepted as a plausible setting for real applications. Indeed we are faced with huge quantities of unlabeled data. Choosing which data is to receive attention by an expert (human or machine) is a difficult question. Interactive learning sessions, where the learning algorithm asks for specific information during runtime, is an interesting alternative to deal with such problems.

A typical example is system SQUIRREL (Carme et al., 2007) which induces a web wrapper through interaction with a human user. Another case is that of testing hardware (Hagerer et al., 2002): the specifications of the software correspond to the Oracle which can then allow to check if the constructed item obeys to the specifications. In both examples the Oracle is fallible: in the second one because testing equivalence is done through sampling.

A third situation in which active learning can be useful corresponds to that of rendering intelligible some black box learned through some statistical machine learning method. Indeed, even if hyper-planes (Clark et al., 2006) or recurrent neural networks (Giles et al., 2001) are difficult to interpret, one can try to use the learned numerical models as Oracles in an active learning algorithm whose result might be some rule based classifier (de la Higuera, 2006).

3. Definitions

An *alphabet* Σ is a finite nonempty set of symbols called *letters*. For the sake of clarity, we shall use $0, 1, 2, \ldots$ as letters in our examples and write $a, b, c, \ldots$ to denote variables for letters in an alphabet. A *string* $w = a_1 \ldots a_n$ is any finite sequence of letters. We write Σ^* for the set of all strings over Σ and λ for the empty string. Let $a \in \Sigma$, $|w|$ be the length of w and $|w|_a$ the number of occurrences of

1. Actually, this is still possible: a theorem due to Mohr (1672), rediscovered by Mascheroni (1797), states that every construction with a ruler and a compass can also be done with a compass only. We know an algorithm that uses 14 circles to learn a disk of the plane. If the reader knows a better method, please contact us!

a in w. We say that a string u is a *subsequence* of v, denoted $u \preceq v$, if $u = a_1 \dots a_n$ and there exist $u_0, \dots, u_n \in \Sigma^*$ such that $v = u_0 a_1 u_1 \dots a_n u_n$. A *language* is any subset $L \subseteq \Sigma^*$. Let $\mathbb{N}$ be the set of non negative integers. For all $k \in \mathbb{N}$, let $\Sigma^k = \{w \in \Sigma^* : |w| = k\}$ and $\Sigma^{\leq k} = \{w \in \Sigma^* : |w| \leq k\}$. Let $\mathbb{R}$ denote the set of real numbers. We say that a real number $\rho \in \mathbb{R}$ is *irrational* if $|\rho| \neq \frac{p}{q}$ for all $p, q \in \mathbb{N}$.

3.1 Edit Distance

The *edit distance* $d(w, w')$ between two strings w and w' is the minimum number of *edit operations* needed to transform w into w' (Levenshtein, 1965).

More precisely, we say that *w rewrites to w' in one step*, written $w \to w'$, if either

1. $w = uav$ and $w' = uv$ (*deletion* of a letter), or
2. $w = uv$ and $w' = uav$ (*insertion* of a letter), or
3. $w = uav$ and $w' = ubv$ (*substitution* of a letter by another letter),

where $u, v \in \Sigma^*$, $a, b \in \Sigma$ and $a \neq b$.

Let $\xrightarrow{k}$ denote a *rewriting derivation* made of k rewriting steps. The edit distance $d(w, w')$ is the minimum $k \in \mathbb{N}$ such that $w \xrightarrow{k} w'$. For instance, $d(0100, 001) = 2$ since $0\underline{1}00 \to 00\underline{0} \to 001$ and rewriting 0100 into 001 cannot be achieved with less than two steps. Notice that $d(w, w')$ can be computed in time $O(|w| \cdot |w'|)$ by means of dynamic programming (Wagner and Fisher, 1974).

The following basic property states that $d(w, w')$ is at least the number of insertions needed to equalize the lengths of w and w':

Proposition 2 *For all $w, w' \in \Sigma^*$, $d(w, w') \geq \big||w| - |w'|\big|$. Moreover, $d(w, w') = \big||w| - |w'|\big|$ iff ($w \preceq w'$ or $w' \preceq w$).*

In all the parts of this paper but in Section 5.2.1, we shall use the standard edit distance defined above. However, for practical reasons, people often use variants of this definition. Sometimes, new edit operations are defined such as the exchange of two adjoining letters in a string. And often, the edit operations are weighted. We shall give more details when needed.

3.2 Balls of Strings

It is well-known that the edit distance is a *metric* (Crochemore et al., 2007), so it conveys to Σ^* the structure of a *metric space*.

Definition 3 (Ball of Strings) *The ball of centre $o \in \Sigma^*$ and radius $r \in \mathbb{N}$, denoted $B_r(o)$, is the set of all the strings whose distance to o is at most r:*

$$B_r(o) = \{w \in \Sigma^* : d(o, w) \leq r\}.$$

For instance, if $\Sigma = \{0, 1\}$, then $B_1(10) = \{0, 1, 00, 10, 11, 010, 100, 101, 110\}$ and $B_r(\lambda) = \Sigma^{\leq r}$ for all $r \in \mathbb{N}$.

The previous example illustrates the fact that the number of strings in a ball grows exponentially with the radius. Experimentally (see Table 1), we clearly notice that for center strings of fixed length, the average number of strings is more than twice larger when the radius is incremented by 1. This

Length of the centre	Radius					
	1	2	3	4	5	6
0	3.0	7.0	15.0	31.0	63.0	127.0
1	6.0	14.0	30.0	62.0	126.0	254.0
2	8.6	25.6	56.5	119.7	246.8	501.6
3	10.8	41.4	101.8	222.8	468.6	973.0
4	13.1	61.4	173.8	402.9	870.9	1850.8
5	16.3	91.0	285.1	698.5	1584.4	3440.9
6	17.9	125.8	441.2	1177.5	2771.3	6252.9
7	21.2	166.9	678.0	1908.8	4835.8	11233.5
8	24.3	200.2	1034.2	3209.9	8358.1	19653.6
9	26.0	265.4	1390.9	5039.6	13677.8	34013.1

Table 1: Average number of strings in a ball. The alphabet has 2 letters. Each value is computed over 20 random centres (possibly the same).

combinatorial explosion occurs as soon as $|\Sigma| \geq 2$, although we leave open the question of finding a general formula that would assess the volume of any ball $B_r(o)$.

The combinatorial explosion noted before raises the problem of the representation scheme that we should use to learn the balls, that is to say, the format of the output space of any learning algorithm. Basically, we need representations whose size is "reasonable", which is not the case of an exhaustive enumeration. An alternative representation could be based on automata, since the balls of strings are finite and thus regular languages.

It is not difficult to see that every ball $B_r(o)$ is recognized by a *non deterministic finite automaton with λ-transitions* having $O(|o| \cdot r)$ states. However, the non deterministic automata are bad candidates from the learning standpoint. Indeed, they are not learnable in most paradigms (Angluin and Kharitonov, 1995; de la Higuera, 1997).

The corresponding *deterministic finite automata* (DFA) do not have this drawback. However, experiments show that these DFA often have an exponential number of states. More precisely, several efficient algorithms exist to build a DFA that recognizes $B_r(o)$ (Ukkonen, 1985; Melichar, 1995; Schulz and Mihov, 2002). For instance, Schulz and Mihov (2002) have recently introduced the so-called *Levenshtein automaton*. Denoting by $n(o,r)$ the number of states of this automaton, they state: $n(o,1) = O(5 \cdot |o|)$, $n(o,2) = O(30 \cdot |o|)$, $n(o,3) = O(180 \cdot |o|)$, $n(o,4) = O(1353 \cdot |o|)$. Basically, $n(o,r)$ is linear in $|o|$ but exponential in r (In their construction, the size of the alphabet only plays a role in the number of transitions, not in the number of states).

Unfortunately, proving that the minimal DFA has the same property is a challenging combinatorial problem. So we only claim here:

Conjecture 4 *The minimal* DFA *recognizing the ball $B_r(o)$ has $\Omega(2^r \cdot |o|)$ states in the worst case.*

On the other hand, why not represent the ball $B_r(o)$ by the pair (o,r) itself? Indeed, its size is $|o| + \log r$, which is reasonable (Garey and Johnson, 1979). Besides, deciding whether $w \in B_r(o)$ or not is immediate: one only has to (1) compute $d(o,w)$ and (2) check whether this distance is $\leq r$,

which is achievable in time $O(|o| \cdot |w| + \log r)$. Finally, when the alphabet has at least two letters, (o, r) is a unique thus *canonical* representation of $B_r(o)$:

Theorem 5 *If $|\Sigma| \geq 2$ and $B_{r_1}(o_1) = B_{r_2}(o_2)$, then $o_1 = o_2$ and $r_1 = r_2$.*

Proof

- *Claim 1: if $B_{r_1}(o_1) = B_{r_2}(o_2)$, then $|o_1| + r_1 = |o_2| + r_2$.* Indeed, let $w \in \Sigma^{r_1}$, then $d(o_1, o_1 w) = |w| = r_1$ by Proposition 2. So $o_1 w \in B_{r_1}(o_1)$, thus $o_1 w \in B_{r_2}(o_2)$, that is to say, $d(o_1 w, o_2) \leq r_2$. Now $d(o_1 w, o_2) \geq |o_1 w| - |o_2|$ from Proposition 2. So we deduce that $r_2 \geq |o_1 w| - |o_2| = |o_1| + r_1 - |o_2|$. The same reasoning yields $|o_1| + r_1 \geq |o_2| + r_2$.
- *Claim 2: if $|\Sigma| \geq 2$ and $o_2 \not\preceq o_1$, then there exists $w \in \Sigma^*$ such that (1) $|w| = r_1 + |o_1|$ and (2) $o_1 \preceq w$ and (3) $o_2 \not\preceq w$.* Indeed, assume that $\Sigma = \{0, 1, \ldots\}$ and o_2 begins with an 0. Then we define $w = 1^{r_1} o_1$ and get the result.

Theorem itself: Assume that $o_1 \neq o_2$. Then either $o_1 \not\preceq o_2$, or $o_2 \not\preceq o_1$. Suppose that $o_2 \not\preceq o_1$, without loss of generality. By Claim 2, there exists a string w such that (1) $|w| = r_1 + |o_1|$ and (2) $o_1 \preceq w$ and (3) $o_2 \not\preceq w$. As $o_1 \preceq w$, Proposition 2 yields $d(o_1, w) = |w| - |o_1| = r_1$. So $w \in B_{r_1}(o_1)$. On the other hand, $o_2 \not\preceq w$, so Proposition 2 yields $d(o_2, w) > \big||w| - |o_2|\big| = \big|r_1 + |o_1| - |o_2|\big| = r_2$, so $w \notin B_{r_2}(o_2)$. In consequence, $B_{r_1}(o_1) \neq B_{r_2}(o_2)$, that is impossible. Therefore, $o_1 = o_2$, and by Claim 1, $r_1 = r_2$. ■

Notice however that if $\Sigma = \{0\}$, then $B_2(0) = B_3(\lambda) = \{\lambda, 0, 00, 000\}$, for instance.

3.3 Queries

Query learning is a paradigm introduced by Angluin (1987b). Her model brings a Learner and an Oracle into play. The goal of the Learner is to identify the representation of an unknown language, by submitting queries to the Oracle. The latter knows the target language and answers properly to the queries (she does not lie). The Learner is bounded by efficiency constraints at each step of the learning process: the runtime of the Learner to make its next query must be polynomial in the size of the target representation and in the length of the information returned by the Oracle up to that point. Notice that certain types of queries require answers that may be of unbounded length (examples or counter-examples). In that case, it is impossible not to take into account the length of this information in the amount of time and queries the Learner is allowed.

Between the different combinations of queries, one, called MAT (Minimally Adequate Teacher), is sufficient to learn the DFA (Angluin, 1987a). Two kinds of queries are used:

Definition 6 (Membership and Equivalence Queries) *Let Λ be a class of languages on Σ^* and $L \in \Lambda$ a target language known by the Oracle, that the Learner aims at guessing.*

In the case of membership queries, the Learner submits a string $w \in \Sigma^*$ to the Oracle; her answer, denoted by $\mathrm{MQ}(w)$, is either YES if $w \in L$, or NO if $w \notin L$.

In the case of equivalence queries, the Learner submits (the representation of) a language $K \in \Lambda$ to the Oracle; her answer, denoted by $\mathrm{EQ}(K)$, is either YES if $K = L$, or a string belonging to the symmetric difference $\big((K \setminus L) \cup (L \setminus K)\big)$ if $K \neq L$.

Although membership queries and equivalence queries have established themselves as a standard combination, there are real grounds to believe that equivalence queries are too powerful to

exist or even be simulated. From a cognitive point of view, we may imagine that a child could ask to his mother whether some sentence is correct or not (that would be a membership query), but not whether he knows English or not (that would be an equivalence query). As suggested by Angluin (1987a), in practice, we may be able to substitute the equivalence queries with a random draw of strings that are then submitted as membership queries (*sampling*). However, in many cases, sampling is not possible because the relevant distribution is unknown and/or inaccessible (de la Higuera, 2006).

Besides, we will not consider membership queries and equivalence queries together because they do not help to learn balls:

Theorem 7 *Assume* $|\Sigma| \geq 2$. *Let* $m,n \in \mathbb{N}$ *and* $\mathcal{B}_{\leq m,n} = \{B_r(o) : r \leq m, o \in \Sigma^*, |o| \leq n\}$. *Any algorithm that identifies every ball of* $\mathcal{B}_{\leq m,n}$ *with equivalence queries and membership queries necessarily uses* $\Omega(|\Sigma|^n)$ *queries in the worst case.*

Proof Following Angluin (1987b), we describe a malevolent Oracle who forces any method of exact identification using membership and equivalence queries to make $\Omega(|\Sigma|^n)$ queries in the worst case. The Oracle is an Adversary: she changes the target ball during the process of identification in order to penalize the Learner. However, all her answers will have to be consistent with the final ball. Technically, she maintains a set S of all the possible balls. At the beginning, $S = \mathcal{B}_{\leq m,n}$. As long as S contains at least two balls, she proceed as follows: her answer to the equivalence query $L = B_r(o)$ is the counterexample o; her answer to the membership query o is NO; in other words, she always declares that o is not in the target ball. After such an answer, every ball of S that contains o cannot be a possible target anymore, so she eliminates them from S. At this point, many balls might disappear, but only one of centre o and radius 0. As there are $\Omega(|\Sigma|^n)$ such balls in $\mathcal{B}_{\leq m,n}$, the Learner will need $\Omega(|\Sigma|^n)$ queries to identify one of them. ■

It should be noted that if the Learner is given one string from the ball, then he can learn using a polynomial number of membership queries.[2] We shall see that the *correction queries*, introduced below, allow to get round these problems:

Definition 8 (Correction Queries) *Let L be a target language known by the Oracle and w a string submitted by the Learner to the Oracle. Her answer, denoted* $\text{CQ}(w)$*, is either* YES *if $w \in L$, or a correction of w with respect to L if $w \notin L$, that is a string $w' \in L$ at minimum edit distance from w:*

$$\text{CQ}(w) = \textit{one string of } \left\{w' \in L : d(w,w') \textit{ is minimum}\right\}.$$

Notice that other milder definitions of correction queries have been proposed in the literature such as Becerra-Bonache et al. (2006) and Kinber (2008). However, the correction queries defined above can easily be simulated knowing the target language. Moreover, we have seen in the introduction that they naturally exist in real-world applications such as the search engines of the Web. Also, we can note that the correction queries are relevant from a cognitive point of view: there is growing evidence that corrective input for grammatical errors is widely available to children (Becerra-Bonache, 2006).

And last but not least, the correction queries as well as the balls rely on a distance, that foreshadows nice learning results. This is not the case for every class of languages:

2. More precisely, the best algorithm we know uses $O\big(|\Sigma|(|o|+r)\big)$ membership queries.

Theorem 9 *Assume $|\Sigma| \geq 2$. Let $n \geq 2$ and $\mathcal{D}_{\leq n}$ the set of all* DFA *with fewer than n states. Any algorithm that identifies every* DFA *of $\mathcal{D}_{\leq n}$ with correction queries necessarily uses $\Omega(|\Sigma|^n)$ queries in the worst case.*

Proof Remember that the number of states of a DFA is a reasonable measure of its size. Let A_w denote the minimal DFA that recognizes $\Sigma^* \setminus \{w\}$. The reader may check that A_w has $|w|+2$ states (see Figure 3 for an example). So basically, $\{A_w : w \in \Sigma^{n-2}\} \subseteq \mathcal{D}_{\leq n}$. Following Angluin (1987b) again, we describe an Adversary that maintains a set S of all the possible DFA. At the beginning, $S = \mathcal{D}_{\leq n}$. Each time the correction of any string w is demanded, the Adversary answers YES and eliminates $\mathcal{A}_w$ from S (and a lot of other DFA) in order to be consistent. As there are $\Omega(|\Sigma|^n)$ such DFA in $\mathcal{D}_{\leq n}$, identifying one of them requires $\Omega(|\Sigma|^n)$ queries. ■

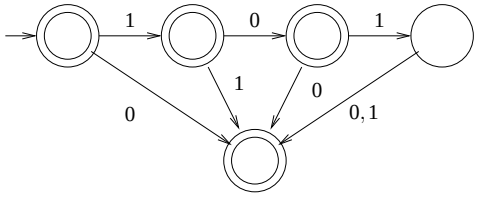

Figure 3: The minimal DFA A_{101} that recognizes $\Sigma^* \setminus \{101\}$ has $5 (= |101|+2)$ states.

4. Identifying Balls of Strings using Corrections

In this section, we propose an algorithm that learns the balls of strings using correction queries. We follow the method described for the disks of the plane. However, several details distinguish the balls of strings and the balls in $\mathbb{R}^2$.

4.1 Technicalities

In this section we introduce four related mathematical results. The first is an analysis of the corrections the Oracle can make. The second corresponds to the definition of the set of the longest strings in a ball (what we call the *upper border* of the ball). The third result is an algorithm allowing to extract the centre of the ball if we are given some elements from this upper border. And finally we explain how to find a string from the upper border using corrections.

4.1.1 A CHARACTERIZATION OF THE CORRECTIONS

When the Learner submits a string outside of a ball to the Oracle, she answers with a string that belongs to the 'circle' delimiting the ball. However, a string often has a lot of different possible corrections, contrarily to what happens in the plane. For instance, the possible corrections for the string 0000 with respect to the ball $B_2(11)$ are $\{00, 001, 010, 100, 0011, 0101, 0110, 1001, 1010, 1100\}$.

By the definition of a correction query, the Oracle will choose one of them arbitrarily, possibly the worst one from the Learner's point of view. Nevertheless, the Oracle's potential malevolence is limited by the following result, that characterizes the set of *all* the possible corrections for a string:

Lemma 10 *Let $B_r(o)$ be a ball and $v \notin B_r(o)$. Then the set of possible corrections of v is exactly $\{u \in \Sigma^* : d(o,u) = r \text{ and } d(u,v) = d(o,v) - r\}$.*

Proof Let $k = d(o,v)$ and consider a derivation from o to v of minimum length: $o \xrightarrow{k} v$. As $v \notin B_r(o)$, we get $k > r$, so this derivation passes through a string w_0 such that $o \xrightarrow{r} w_0 \xrightarrow{k-r} v$. Let us define the set $W = \{w \in \Sigma^* : d(o,w) = r \text{ and } d(w,v) = k-r\}$. Basically, $w_0 \in W$, so $W \neq \emptyset$. Moreover, $W \subseteq B_r(o)$. Now let U denote the set of all the possible corrections of v. We claim that $U = W$. Indeed, let $u \in U$ and $w \in W$. If $d(u,v) > d(w,v)$, then w is a string of $B_r(o)$ that is closer to v than u, so u cannot be a correction of v. On the other hand, if $d(u,v) < d(w,v)$, then as $d(o,v) \leq d(o,u) + d(u,v)$, we deduce that $d(o,u) \geq d(o,v) - d(u,v) > d(o,v) - d(w,v)$. As $d(o,v) = k$ and $d(w,v) = k-r$, we get $d(o,u) > r$, which is impossible since $u \in U \subseteq B_r(o)$. Hence, $d(u,v) = d(w,v) = k-r$. In consequence, all the strings $w \in W$ and corrections $u \in U$ are at the same distance from v, thus $W \subseteq U$. Moreover, we have $d(o,v) \leq d(o,u) + d(u,v)$, so $k \leq d(o,u) + k - r$, thus $d(o,u) \geq r$. As $u \in B_r(o)$, we deduce that $d(o,u) = r$. Finally, as we have stated that $d(u,v) = k-r$, we can conclude that $U \subseteq W$. ■

Here is a geometric interpretation of the result above. Let us define the *segment* $[o,v] = \{w \in \Sigma^* : d(o,w) + d(w,v) = d(o,v)\}$ and the *circle* $C_r(o) = \{w \in \Sigma^* : d(o,w) = r\}$. Lemma 10 states that a string u is a possible correction of v *iff* $u \in [o,v] \cap C_r(o)$. The fact that v has several possible corrections shows that the geometry of Σ^* is very different from that of $\mathbb{R}^2$.

4.1.2 THE BORDERLINE STRINGS OF MAXIMUM LENGTH

We begin by distinguishing the longest strings of any ball:

Definition 11 (Upper Border) *The upper border of a ball $B_r(o)$, denoted $B_r^{max}(o)$, is the set of all the strings that belong to $B_r(o)$ and are of maximum length:*

$$B_r^{max}(o) = \{u \in B_r(o) : \forall w \in B_r(o), |w| \leq |u|\}.$$

For instance, given $\Sigma = \{0,1\}$, we get $B_1^{max}(10) = \{010, 100, 101, 110\}$.

The strings of $B_r^{max}(o)$ are remarkable because they are all built from the centre o by doing r insertions. So from a string $w \in B_r^{max}(o)$, one 'simply' has to guess the inserted letters and delete them to find o again. We get:

Proposition 12 *$w \in B_r^{max}(o)$ iff ($o \preceq w$ and $d(o,w) = |w| - |o| = r$).*

Proof Let us assume that $o \preceq w$ and $d(o,w) = |w| - |o| = r$. Then $w \in B_r(o)$. Let w' be a string such that $|w'| > |w|$. Then, by Proposition 2, $d(o,w') \geq |w'| - |o| > |w| - |o| = r$, so $w' \notin B_r(o)$. Therefore, $w \in B_r^{max}(o)$. Conversely, let $w \in B_r^{max}(o)$. Consider an arbitrary letter $a \in \Sigma$ and the string $a^r o$. Basically, $d(o,a^r o) = r$, so $a^r o \in B_r(o)$. As $w \in B_r^{max}(o)$, we deduce that $|w| \geq |a^r o| = |o| + r$. Therefore, by Proposition 2, $d(o,w) \geq |w| - |o| \geq r$. On the other hand, $r \geq d(o,w)$ holds since $w \in B_r^{max}(o)$. So we deduce that $d(o,w) = |w| - |o| = r$, that also brings $o \preceq w$, by Proposition 2. ■

4.1.3 FINDING THE CENTRE GIVEN STRINGS FROM THE UPPER BORDER

Some strings of $B_r^{max}(o)$ are even more informative. Indeed, let $a \in \Sigma$ be an arbitrary letter. Then $a^r o \in B_r^{max}(o)$. So, if we know r, we can easily deduce o. We claim that the correction queries allow us to get hold of $a^r o$ from any string $w \in B_r^{max}(o)$ by swapping the letters. This is the goal of EXTRACT_CENTRE (see Algorithm 1).

Let us run this procedure on an example. Consider the ball $B_2(11)$. Then it is easy to check that $B_2^{max}(11) = \{0011, 0101, 0110, 0111, 1001, 1010, 1011, 1100, 1101, 1110, 1111\}$. Running EXTRACT_CENTRE on the string $w = 0110$ and radius $r = 2$ transforms, at each loop, the i^{th} letter of w to a 0 that is put at the beginning and then submits it to the Oracle. We get:

i	w	w'	CQ(w')	w changes
1	$\underline{0}110$	0110	YES	yes
2	$0\underline{1}10$	0010	0110	no
3	$01\underline{1}0$	0010	0110	no
4	$011\underline{0}$	0011	YES	yes

Therefore, EXTRACT_CENTRE stops with $w = 0011$ and returns $o = 11$ (since $r = 2$).

Algorithm 1 EXTRACT_CENTRE

Require: A string $w = a_1 \dots a_n \in B_r^{max}(o)$, the radius r
Ensure: The centre o of the ball $B_r(o)$

```
x ← a_n        (* x is an arbitrary letter *)
for i = 1 to n do
    Assume w = a_1 ... a_n and let w' = x a_1 ... a_{i-1} a_{i+1} ... a_n
    if CQ(w') = YES then w ← w' end if
end for
Assume w = a_1 ... a_n and return a_{r+1} ... a_n
```

Proposition 13 *Given $w \in B_r^{max}(o)$ and the radius r, Algorithm* EXTRACT_CENTRE *returns o using $O(|o| + r)$ correction queries.*

Proof (Sketch) Let us show that the swapping operation is correct. Consider the string $w = a_1 \dots a_n \in B_r^{max}(o)$ and let $w' = x a_1 \dots a_{i-1} a_{i+1} \dots a_n$ for some $0 \leq i \leq n$. If there exists at least one derivation $o \xrightarrow{r} w$ where the letter a_i of w comes from an insertion in o, then deleting a_i and doing the insertion of a x in front of o yields a string w' that satisfies $o \preceq w'$ and $|w'| = |w|$. By Proposition 2, we get $d(o, w') = |w'| - |o| = |w| - |o| = r$, so by Proposition 12, $w' \in B_r^{max}(o)$ and CQ(w') = YES. On the other hand, if there is no derivation where a_i is introduced by an insertion, then deleting a_i and inserting a x yields a string w' such that $o \not\preceq w'$. By Proposition 2, we get $d(o, w') > |w'| - |o|$. As $|w'| = |w|$, we deduce that $d(o, w') > r$. So $w' \notin B_r^{max}(o)$ and CQ(w') $\neq$ YES. ■

4.1.4 FINDING ONE BORDERLINE STRING OF MAXIMUM LENGTH

Hence, we are now able to deduce the centre of a ball as soon as we know its radius and a string from its upper border. The following technical result is a step towards finding this string (although we have no information about r and $|o|$ yet):

Proposition 14 *Suppose that $\Sigma = \{a_1, \ldots, a_n\}$ and consider the string $v = (a_1 \ \ldots \ a_n)^k$ with $k \geq |o| + r$. Then every correction of v belongs to $B_r^{max}(o)$.*

Proof Let U be the set of all the possible corrections of v. Let us show that $U = B_r^{max}(o)$. As $v = (a_1 \ldots a_n)^k$ with $k \geq |o| + r$, we get $o \preceq v$, so $d(o,v) = |v| - |o|$, by Proposition 2. Let $w \in B_r^{max}(o)$. By Proposition 12, we get $o \preceq w$ and $d(o,w) = |w| - |o| = r$. Moreover, as $v = (a_1 \ldots a_n)^k$ with $k \geq |o| + r$, we get $w \preceq v$. So $d(w,v) = |v| - |w| = |v| - |o| - r = d(o,v) - r$ by Proposition 2. As $d(o,w) = r$ and $d(w,v) = d(o,v) - r$, Lemma 10 yields $B_r^{max}(o) \subseteq U$. Conversely, let $u \in U$. We get $d(o,u) = r$, again by Lemma 10. If $o \preceq u$, then $u \in B_r^{max}(o)$ by Proposition 12. If $o \not\preceq u$, then Proposition 2 yields $d(o,u) > |u| - |o|$, that is to say, $|u| < |o| + r$. But then, $d(u,v) \geq |v| - |u| > |v| - |o| - r = d(w,v)$ for all $w \in B_r^{max}(o)$, so $u \notin U$, that is impossible. Therefore, $U \subseteq B_r^{max}(o)$. ■

If one submits $(a_1 \ldots a_n)^k$ with a sufficiently large k, then one is sure to get a string of $B_r^{max}(o)$. So the last problem is to find such an interesting k. The following lemma states that if one asks to the Oracle the correction of a string made of a lot of 0's, then this correction contains precious informations about the radius and the number of occurrences of 0's in the centre:

Lemma 15 *Consider the ball $B_r(o)$. Let $a \in \Sigma$ be any letter and $j \in \mathbb{N}$ an integer such that $a^j \notin B_r(o)$. Let w denote a correction of a^j. If $|w| < j$, then $|w|_a = |o|_a + r$.*

Proof Let $a^j \notin B_r(o)$ and $w = \mathrm{CQ}(a^j)$. By Lemma 10, we get $d(o,w) = r$ and $d(w,a^j) = d(o,a^j) - r$. As $|w| < |a^j|$, the computation of $d(w,a^j)$ consists in (1) substituting all the letters of w that are not a's, thus doing $|w| - |w|_a$ substitutions, and (2) completing this string with further a's in order to reach a^j, thus doing $j - |w|$ insertions of a's. So we deduce that $d(w,a^j) = |w| - |w|_a + j - |w| = j - |w|_a$. Let us compute $d(o,a^j)$. Clearly, if $|o| \leq |w| < j$, then we can use the same arguments as before and get $d(o,a^j) = j - |o|_a$. Finally, since $d(w,a^j) = d(o,a^j) - r$, we deduce that $j - |w|_a = j - |o|_a - r$, *that is*, $|w|_a = |o|_a + r$.

Now suppose that $|o| > |w|$. Then we cannot use the same arguments as before, because it is possible that $|o| \geq |a^j|$, thus that deletions are needed to compute $d(o,a^j)$. However, this case is impossible. Indeed, consider a derivation $o \xrightarrow{r} w$. Since $|o| > |w|$, there is at least one deletion along this derivation. Now, instead of deleting a letter, suppose that we substitute it by an a and do not change anything else. This leads us to a new derivation $o \xrightarrow{r} w'$ (or $o \xrightarrow{r-1} w'$ if the deleted letter was an a) with $|w'| = |w| + 1$ and $|w'|_a = |w|_a + 1$. Moreover, $d(o,w') \leq r$, thus $w' \in B_r(o)$. Finally, as $|w| < j$, we get $|w'| \leq j$, so with the same arguments as before, only substitutions and insertions are necessary to compute $d(w',a^j)$. More precisely, we have $d(w',a^j) = (|w'| - |w'|_a) + (j - |w'|) = -|w|_a - 1 + j = d(w,a^j) - 1$, thus $d(w',a^j) < d(w,a^j)$. Since $w' \in B_r(o)$, w cannot be a correction of a^j. ■

Finally, let us assume that the alphabet is $\Sigma = \{a_1, \ldots, a_n\}$ and let $j_1, \ldots, j_n \in \mathbb{N}$ be large integers. If we define $k = \sum_{i=1}^n |\mathrm{CQ}(a_i^{j_i})|_{a_i}$, then Lemma 15 brings $k = \sum_{i=1}^n (|o|_{a_i} + r) = |o| + |\Sigma| \cdot r \geq |o| + r$. So we can plug k into Proposition 14 and get a string $w = \mathrm{CQ}\left((a_1 \ldots a_n)^k\right) \in B_r^{max}(o)$. Moreover, we have $|w| = |o| + r$ and $k = |o| + |\Sigma| \cdot r$. So, we deduce that the radius is $r = (k - |w|)/(|\Sigma| - 1)$.

4.2 An Algorithm to Learn Balls from Correction Queries

Let us summarize, by assuming that $\Sigma = \{a_1, \ldots, a_n\}$ and that the target is the ball $B_r(o)$. (1) For each letter a_i, the Learner asks for the correction of a_i^j where j is sufficiently large to get a correction whose length is smaller than j; (2) the Learner sets $k = \sum_{i=1}^{n} |\text{CQ}(a_i^{j_i})|_{a_i}$ and gets the correction w of the string $v = (a_1 \ldots a_n)^k$; (3) from k and $|w|$, he deduces r; (4) he uses EXTRACT_CENTRE on w and r, and he gets o. In other words, he is able to guess the balls with correction queries (see Algorithm IDF_BALL and Proposition 16).

Algorithm 2 IDF_BALL

Require: The alphabet $\Sigma = \{a_1, \ldots, a_n\}$
Ensure: The representation (o, r) of the target ball $B_r(o)$
1: $j \leftarrow 1; k \leftarrow 0$
2: **for** $i = 1$ **to** n **do**
3: **while** $\text{CQ}(a_i^j) = \text{YES}$ **or else** $|\text{CQ}(a_i^j)| \geq j$ **do**
4: $j \leftarrow 2 \cdot j$
5: **end while**
6: $k \leftarrow k + |\text{CQ}(a_i^j)|_{a_i}$
7: **end for**
8: $w \leftarrow \text{CQ}((a_1 a_2 \ldots a_n)^k)$
9: $r \leftarrow (k - |w|)/(|\Sigma| - 1)$
10: $o \leftarrow \text{EXTRACT\_CENTRE}(w, r)$
11: **return** (o, r)

For instance, consider the ball $B_2(11)$ defined over $\Sigma = \{0, 1\}$. IDF_BALL begins by looking for the corrections of 0^j and 1^j with a sufficiently large j. We might observe: $\text{CQ}(0) = \text{YES}$, $\text{CQ}(0^2) = \text{YES}$, $\text{CQ}(0^4) = 0011$, $\text{CQ}(0^8) = 0110$, $\text{CQ}(1^8) = 1111$. So $k = |0110|_0 + |1111|_1 = 2 + 4 = 6$. Then $\text{CQ}\left((01)^6\right) = \text{CQ}(010101010101) = 0110$, for instance, so $r = (6-4)/(2-1) = 2$. Finally, $\text{EXTRACT\_CENTRE}(0110, 2)$ returns 11. So the algorithm returns $(11, 2)$, which is the representation of the target ball.

Proposition 16 *Given any fixed ball $B_r(o)$, the Algorithm* IDF_BALL *returns the representation (o, r) using $O(|\Sigma| + |o| + r)$ correction queries.*

Proof The correction of IDF_BALL is clear. Concerning the number of queries, the corrections of all the strings a_i^j require $O(|\Sigma| + \log(|o| + r))$ correction queries (lines 2-5). Indeed, $O(\log(|o| + r))$ queries are necessary to get long enough corrections, plus one query per letter, thus $|\Sigma|$ queries. Then EXTRACT_CENTRE needs $O(|o| + r)$ correction queries (line 8) to find the centre, by Proposition 13. So the total amount of queries is $O(|\Sigma| + |o| + r)$. ■

4.3 Only the Good Balls are Learnable with IDF_BALL

We now have an algorithm that guesses all the balls $B_r(o)$ with $O(|\Sigma| + |o| + r)$ correction queries. Is this result relevant?

In Section 3, we have decided to represent every ball $B_r(o)$ by the pair (o,r). The size of this representation, $|o|+\log r$, is basically related to the number of bits needed to encode this representation. Notice that $|o|+r$ *is not* a correct measure of the size: this would correspond to an encoding in base 1 of the radius r, which is not considered reasonable (Garey and Johnson, 1979). Therefore, the number of correction queries used by IDF_BALL is exponential in $\log r$. In consequence, if r is too large with respect to $|o|$, (e.g., $r > 2^{|o|}$), then our algorithm is not able to identify efficiently the target ball.

In order to avoid this problem, we introduce the following definition that allows us to rewrite Proposition 16 in a more relevant way:

Definition 17 (Good Balls)

- *Let $q()$ be any fixed polynomial. We say that a ball $B_r(o)$ is* $q()$-good *if $r \leq q(|o|)$.*
- *We say that $B_r(o)$ is* very good *if $r \leq |o|$.*

A very good ball is thus a $q()$-good ball for the polynomial $q(x) = x$.

Then, Proposition 16 yields:

Theorem 18

- *Let $q()$ be any fixed polynomial. The set of all $q()$-good balls $B_r(o)$ is identifiable with an algorithm that uses $O(|\Sigma| + |o| + q(|o|))$ correction queries and a polynomial amount of time.*
- *The set of all very good balls is identifiable with a linear number $O(|\Sigma| + |o|)$ of correction queries and a polynomial amount of time.*

Finally, the reader may wonder if a better learnability result could be established, which would include the huge balls. Unfortunately, there is not a unique answer to this question. On one hand, if the number of correction queries authorized to learn can also depend on the length of the longest correction provided by the Oracle during a run of IDF_BALL, then the answer is positive: all the balls are learnable. On the other hand, in de la Higuera et al. (2008), it has been proved that the set of all the balls was not polynomially identifiable in the limit, nor in most relevant online learning paradigms, whereas positive results were established for the good balls in most paradigms. From this point of view, Theorem 18 is satisfying.

5. Learning in a Realistic Environment

The setting of learning with queries itself occurs in many circumstances: when a human being is asked to provide data for a learning program, an alternative to have the human expert labeling huge quantities of data can be to have the learning system interact with the human expert, who then only labels those items required. Nevertheless, assuming that the Oracle is a human expert introduces new constraints. On one hand, asking billions of queries is unacceptable: there is no chance to get enough answers in reasonable time. So the learning algorithm should aim at minimizing the number of queries even if we must pay for it with a worse time complexity. On the other hand, a human (or even the Web) is fallible. Therefore, the learning algorithm should aim at learning functions or languages that are robust from corrections that may not be ideal, thus approximative.

These issues are discussed in de la Higuera (2006). Some examples of this alternative approach (imperfect Oracle) are: system SQUIRREL, which makes use of queries to a human expert to allow

wrapper induction (Carme et al., 2007); learning test sets (Bréhélin et al., 2001) and testing hardware (Hagerer et al., 2002), where the actual electronic device can be physically tested by entering a sequence, and the device will then be able to answer a membership query (note that in that setting equivalence queries will be usually simulated by sampling). Another typical question occurs when learning some non intelligible function, defined perhaps by a complex kernel (Clark et al., 2006) or neural networks (Giles et al., 2001): a representation of these complex functions in another setting can be obtained if we use the complex function as an Oracle to learn from.

5.1 Faced with a Fallible Oracle

The algorithm IDF_BALL has been designed in an ideal setting, where we have assumed that the Oracle was a perfect machine: her answers were so precise that we could scrupulously characterize them (see Lemma 10). However, as described in the introduction, in practice, an Oracle is often an expert, thus a human being, or is simulated through sampling. In such settings, our assumption is no longer correct. Indeed, computing the correction of $(101)^{127}$ *w.r.t.* the ball $B_{217}((1011)^{32})$ is probably out of the cognitive capacities of any human being. So our algorithm should not believe unwisely the answers he gets since they can be approximate. In this section, we would like to show, with a series of experiments, that our algorithm withstands such approximate (that is to say, inaccurate, noisy) answers.

5.1.1 DESIGNING THE APPROXIMATE ORACLE

We want here to design an approximate Oracle that might look like a human being. So let us consider a string w and a ball $B_r(o)$. Let $\mathrm{CQ}_h(w)$ denote the answer of the approximate Oracle, and $\mathrm{CQ}(w)$ the answer that would be returned by a perfect Oracle (as before).

Firstly, we assume that an expert can easily determine whether an example fulfills a concept or not, thus here, whether w belongs to $B_r(o)$ or not. So we assume that if $\mathrm{CQ}(w) = \mathrm{YES}$, then $\mathrm{CQ}_h(w) = \mathrm{YES}$. Secondly, what is really hard for the expert is to *compute* the best correction of w when $w \notin B_r(o)$, and more precisely, a string of the ball that is *as close to w as possible*. Again, $\mathrm{CQ}_h(w)$ will probably be inside the ball rather than on its frontier.

Staying a step ahead, let $X = d(w, \mathrm{CQ}_h(w)) - d(w, \mathrm{CQ}(w))$ measure the distance between an approximate correction and a perfect one. Intuitively, an approximate but strong Oracle will often provide corrections such that $X = 0$, sometimes $X = 1$ and rarely $X \geq 2 \ldots$ To formalize this idea, we introduce a confidence parameter $0 < p \leq 1$, called the *accuracy level of the Oracle*, that translates the quality of her answers, and use a geometric distribution: $Pr(X = k) = (1-p)^k p$, for all $k \in \mathbb{N}$.

Therefore, with probability $(1-p)^k p$, the correction $\mathrm{CQ}_h(w)$ of a string w will be in the target ball, at distance k of $\mathrm{CQ}(w)$. Basically, we get $E(X) = (1/p) - 1$. So when the Oracle is very accurate, say $p = 0.8$, then the average distance between an approximate and a perfect correction is low (0.25). Conversely, an expert with limited computation capacities, say $p = 0.4$, will often provide inaccurate corrections, at distance 1.5 on average.

Our model of approximate Oracle is simple. For instance, we do not suppose that she has any memory, thus by submitting twice every string w, we would probably get 2 different corrections that could be used to correct the corrections! We want here to study the resistance of IDF_BALL to approximate answers, not to design the best possible algorithm, able to beat the approximate Oracle. So from this standpoint, our basic model is sufficient.

5.1.2 BEHAVIOR OF THE ALGORITHM FACED WITH AN APPROXIMATE ORACLE

Following Theorem 18, IDF_BALL systematically guesses the target ball with the help of a perfect Oracle. But of course, he is sometimes going to fail in front of an approximate Oracle. So, in order to assess the resistance of IDF_BALL to approximate corrections, we conduct the following experiment. We randomly choose a set of 100 balls $B_r(o)$ such that $|o| + r = 200$. More precisely, we make the radius r vary between 10 and 190 by step of 20, and randomly choose 10 centres o of length $200 - r$ for each radius. Then, for every accuracy level $0.5 \leq p \leq 1$, we ask IDF_BALL to learn all of them and we compute the percentage of balls he is able to retrieve, that we call the *precision of the algorithm*. We show the result in Figure 4. We notice that IDF_BALL is able to identify about 75% of the balls faced with an accuracy level of $p = 0.9$. Of course, as one can expect, with lower levels of accuracy, his performances quickly drop (15% for $p = 0.5$).

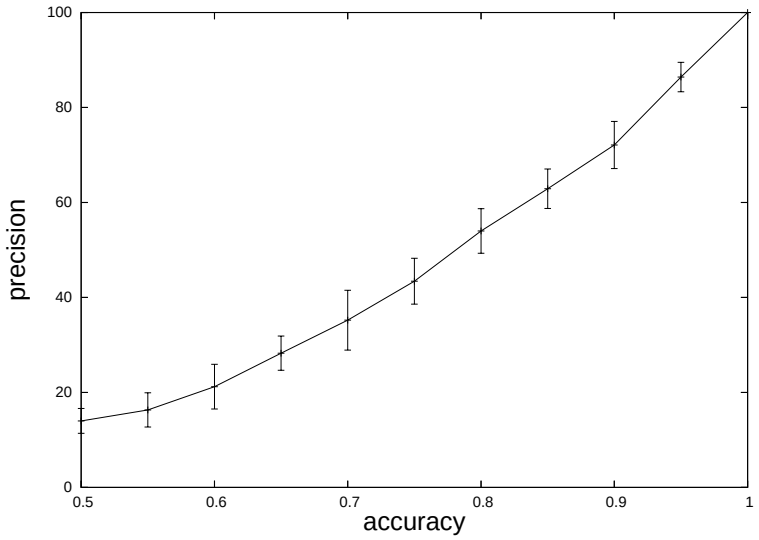

Figure 4: Precision of IDF_BALL faced with an approximate Oracle in function of the accuracy level p. Each point is assessed on 100 balls.

We also show, in Figure 5, the average distances between the centres of the target balls and the centres of the the learnt balls when he fails to retrieve them. We observe that these distances are not that important: even with an accuracy level of $p = 0.5$, the difference is less than 3. The last curve in Figure 6 is the difference between the radii, that basically follow the same trend.

5.1.3 IMPROVING THE PRECISION WITH *a posteriori* HEURISTICS

We have seen that IDF_BALL was able to assimilate the approximations of the Oracle up to a certain level of accuracy. Moreover, the centres and the radii returned by the algorithm are generally not far from the target. Therefore, it is reasonable to think that we could improve the precision by exploring the neighborhood of the learnt centre, using local edit modifications. This kind of approaches has been pioneered by Kohonen (1985) and is surveyed in Martínez-Hinarejos et al. (2000).

Suppose that the learnt ball is $B_k(u)$ and let Q be the set of all the corrections returned by the Oracle during the process of IDF_BALL. The heuristics is composed of two steps:

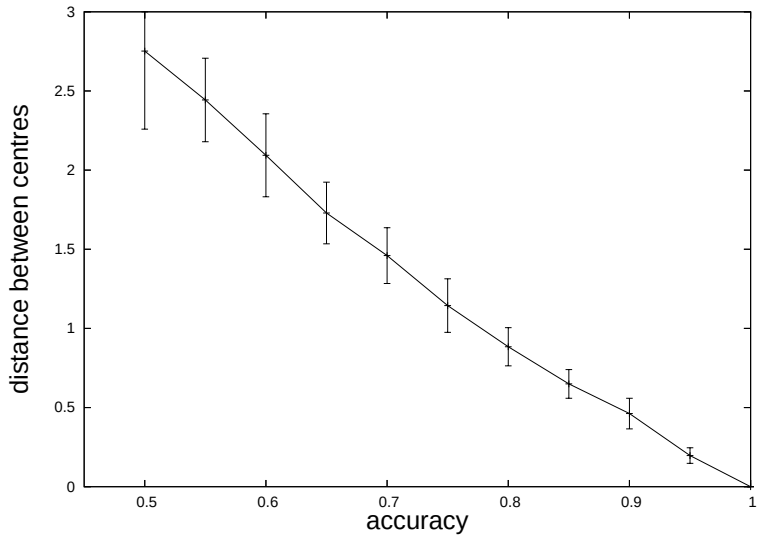

Figure 5: Average distances (and standard deviation) between the centres of the target balls and the centres of the learnt balls, when IDF_BALL fails in retrieving them.

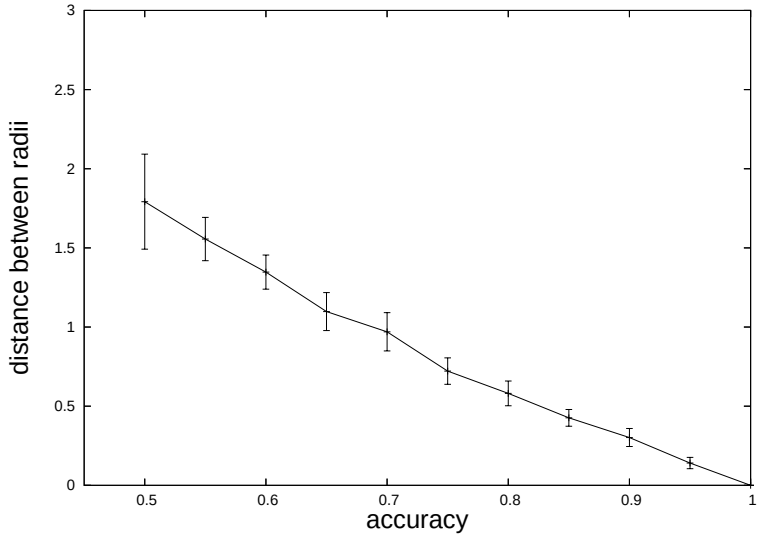

Figure 6: Average difference (and standard deviation) between the radii of the target balls and the radii of the learnt balls, when IDF_BALL fails in retrieving them.

1. We test each neighbor u' (at distance 1) of the learnt centre u and examine if it is a better centre with respect to Q, that is to say, if there exists $k' \in \mathbb{N}$ such that $k' < k$ and $Q \subseteq B_{k'}(u')$. Then we keep the representations (u', k') of the smallest balls that contain all the corrections seen so far.

2. From this set, we select all the pairs (u', k') that maximize the number of corrections (of Q) at distance k', in such a way as to get the maximum number of corrections on the border of the new ball. Then we randomly choose and return one of them.

This heuristics will be very good each time u is at distance 1 from the target centre. But as soon as this distance grows, IDF_BALL will fail again. In order to enhance the one-step heuristics, we iterate the process and design a second until-convergence heuristics by repeating the local search described above, until the size of the ball cannot decrease anymore.

In order to show that the balls learnt by IDF_BALL can be corrected *a posteriori*, we compare, in a series of experiments, the precision of the algorithm without any post-treatment, with the one-step heuristics and with the until-convergence heuristics. We fix $|o| + r = 200$ and make the radius vary from 10 to 190. For each radius, we randomly draw 50 centres of length $200 - r$. Then, we make the accuracy level vary from 0.5 to 1. For each pair (accuracy, radius), we ask IDF_BALL to retrieve the 50 balls and note the precision. In order to be able to reduce the variance due to the approximations of the Oracle, we repeat the experiment 10 times using the same set of balls and finally plot the average precisions in Figure 7.

We can remark that whatever the accuracy level, using the until-convergence heuristics is never worse than the one-step heuristics, which is never worse than no post-treatment at all. However, our heuristics do not always improve the precision of the algorithm: this depends on the ratio between the radius of the target ball and the length of its centre. In order to detail this, we have extracted two transverse sections, shown in Figures 8 and 9, where we fix the radii.

Figure 8 describes the precision of IDF_BALL for target balls such that $r = 170$ and $|o| = 30$. In this case, we gain little using the heuristics. This is probably due to the fact that the size of the set Q, which is used to control the heuristics, is incomparably smaller than the volume of such balls. In other words, the heuristics are not sufficiently guided by Q towards the targets, because Q is not informative enough.

On the other hand, Figure 9 describes the precision for target balls such that $r = 10$ and $|o| = 190$. Basically, our heuristics outperform the precision with respect to the algorithm without any post-treatment, whatever the accuracy level of the Oracle. Moreover, the benefit is all the more important as the accuracy level is bad. For instance, when $p = 0.6$, the until-convergence heuristics is able to dramatically boost the precision from 12% to 86%.

So in this setting, with no further enhancement, IDF_BALL produces balls that are so close to the targets that they can easily be improved using only basic local modifications.

5.2 Using Less Correction Queries

We have seen that the good balls were identifiable with $O(|\Sigma| + |o| + r)$ correction queries. However, as discussed in the introduction, such a number of queries is excessive if the Oracle is a human being. Moreover, if the reader thinks of what happens in the plane (see Figure 2), then very few queries are needed to identify the disks. Hence, our result might seem to be a bit disappointing.

If one takes a closer look at IDF_BALL, one can notice that the first part of the identification, that is to say, the search for a string of $B_r^{max}(o)$, is done with $O(|\Sigma| + \log(|o| + r))$ correction queries

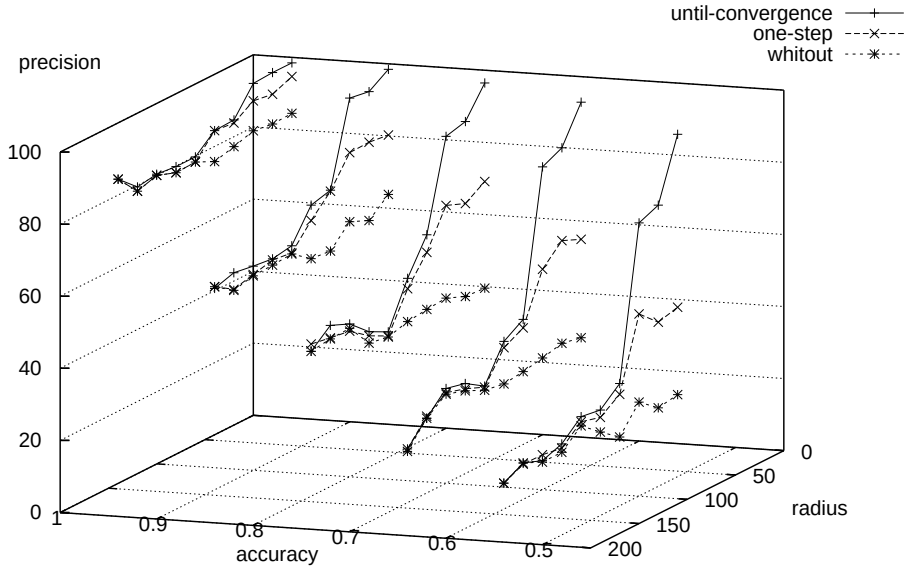

Figure 7: Precision of IDF_BALL with and without heuristics in function of accuracy and radius when $|o| + r = 200$. For each pair (accuracy, radius), we compute the average over 50 balls.

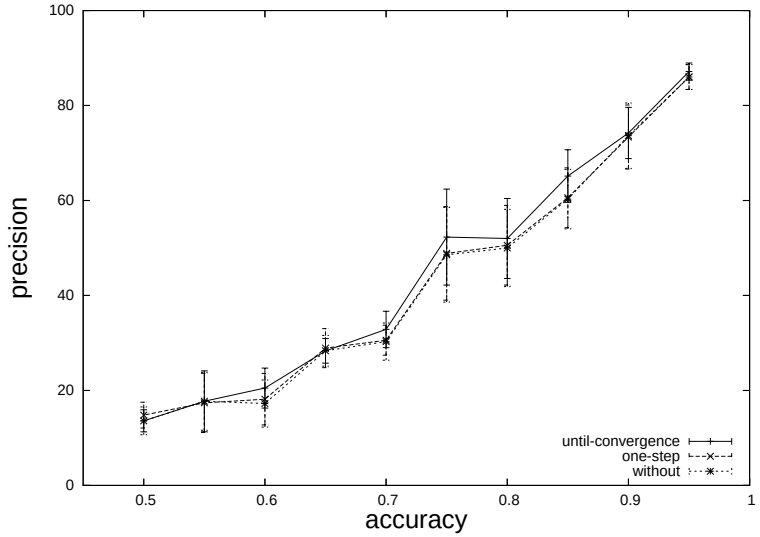

Figure 8: Precision of IDF_BALL when $|o| + r = 200$ for $r = 170$. For each accuracy, we compute the average over 50 balls. We run the experiment 10 times in order to reduce the variance.

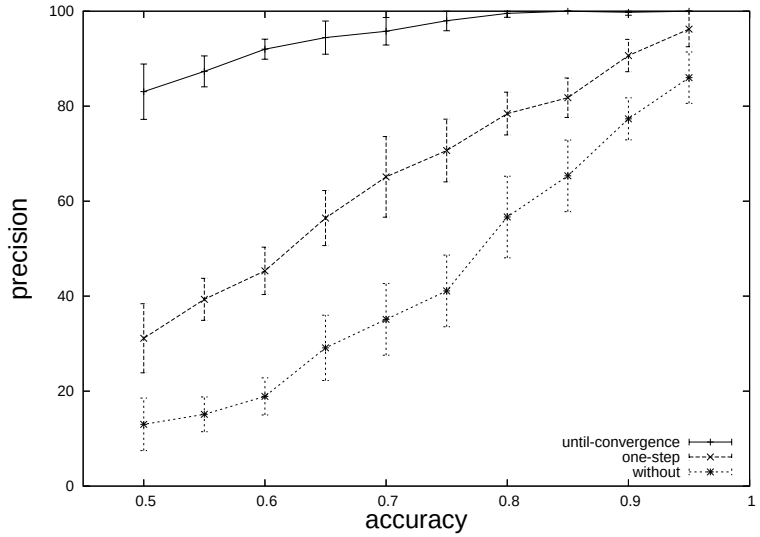

Figure 9: Precision of IDF_BALL when $|o| + r = 200$ for $r = 10$. For each accuracy, we compute the average over 50 balls. We run the experiment 10 times in order to reduce the variance.

(thus, a logarithmic number). What is really expensive is to find the centre of the ball using the function EXTRACT_CENTRE. We are going to show below that this function can be eliminated from the learning stage, and thus, that the complexity can be dramatically reduced, but in a slightly different setting.

For reasons that we shall develop later, we now suppose that the alphabet has at least three letters: $\Sigma = \{a_1, \ldots, a_n\}$ with $n \geq 3$.

5.2.1 THE USE OF A WEIGHTED EDIT DISTANCE

Up to now, we have considered the standard edit distance defined by Levenshtein (1965). However, for practical reasons, people often use variants of this definition where the edit operations are weighted. In this case, every derivation $w \xrightarrow{k} w'$ has a weight which is the sum of the weights of the edit operations along the derivation. Then the *weighted edit distance* $d(w, w')$ is the minimum weight of every derivation transforming w into w'. Clearly, if the weight of all the edit operations is 1, then we get the standard edit distance.

The different combinations of weights will impose alternative algorithms when using correction queries. As we aim at showing that the number of correction queries can be dramatically reduced, we assume that:

1. the weight of every insertion and every deletion is 1 (as before),

2. the weight of every substitution is an irrational number ρ such that $0 < \rho < 1$.

For instance, the weight of the substitution could be $\rho = \frac{\pi}{4} \simeq 0.7854$.

It is worth noting that the low cost of a substitution operation is usual from a linguistic point of view. For instance, works on Phonology make this assumption in order to enforce the alignment of phonetically similar segments (Albright and Hayes, 2003).

Nevertheless, the fact that ρ is not rational may be confusing for the reader. Actually, from the Learner standpoint, we will see that he never needs to compute the weighted edit distance (that is probably not the case of the Oracle). So the fact that ρ is not a fraction will not be a problem.

We can show that this set of weights induces an edit distance that can be computed using dynamic programming.[3] Moreover, Proposition 2, stating that (1) $d(w,w') \geq \big||w|-|w'|\big|$ and (2) $d(w,w') = \big||w|-|w'|\big|$ *iff* ($w \preceq w'$ or $w' \preceq w$), still holds, because the weight of the insertions and deletions is 1. Finally, the fact that ρ is irrational allows us to establish strong properties:

Proposition 19 *Let $o,w,w' \in \Sigma^*$ be three strings. The following statements are equivalent:*

1. *There exists a derivation of minimum weight from w to w' that uses $x \in \mathbb{N}$ insertions and deletions, and $y \in \mathbb{N}$ substitutions;*
2. *$d(w,w') = x + \rho y$;*
3. *All the derivations of minimum weight from w to w' use $x \in \mathbb{N}$ insertions and deletions, and $y \in \mathbb{N}$ substitutions.*

In consequence, if $d(o,w) = d(o,w')$, then all the derivations from o to w and from o to w' use the same number of insertions and deletions, and the same number of substitutions.

Proof

- 3. $\Longrightarrow$ 1.: straightforward.
- 1. $\Longrightarrow$ 2.: since the weight of the insertions and deletions is 1, and the weight of the substitutions is ρ, and the derivation has a minimum weight, we get $d(w,w') = x + \rho y$.
- 2. $\Longrightarrow$ 3.: consider another derivation from w to w' of minimum weight that uses $x' \in \mathbb{N}$ insertions and deletions, and $y' \in \mathbb{N}$ substitutions. Then we get $d(w,w') = x' + \rho y' = x + \rho y$, so $x - x' = \rho(y' - y)$. As ρ is irrational and x, x', y, y' are integers, we deduce that $y' - y = 0$ and $x - x' = 0$, thus $x' = x$ and $y' = y$.

■

Of course, this result would not hold if ρ was a rational number, for instance $\rho = \frac{1}{2}$, because two substitutions would have the same weight as one insertion, which might induce two very different derivations of minimum weight between two strings.

5.2.2 THE NEW GOOD BALLS AND CORRECTIONS

Basically, changing the edit distance also changes the balls. For instance, using the standard edit distance, we get $B_1(011) = \{01, 11, 001, 010, 011, 111, 0011, 0101, 0110, 1011, 0111\}$. But the use of the weighted edit distance with $\rho = \frac{\sqrt{2}}{4} \simeq 0.3536$ adds $\{000, 101, 110\}$ as new strings.

3. Indeed, its restriction to $\Sigma \cup \{\lambda\}$ is a distance, so following Crochemore et al. (2007), the standard dynamic programming algorithm of Wagner and Fisher (1974) can be used to compute the weighted edit distance over Σ^*.

The reader may also wonder whether the radius of the balls should still be an integer or not. Actually, we shall not consider balls whose radius is not an integer, because otherwise, the balls $B_r(o)$ and $B_{r+\frac{\rho}{2}}(o)$ might represent the same set. In other words, Theorem 5, that states the uniqueness of the representation, would not hold anymore. Conversely, if we only consider balls with an integer radius, then the reader can check that Theorem 5 still holds (because Proposition 2 still holds).

Concerning the corrections, their properties become more intricate due to the weights. In particular, Lemma 10 was stating that the set of possible corrections of any string $v \notin B_r(o)$ was exactly $\{u \in \Sigma^* : d(o,u) = r \text{ and } d(u,v) = d(o,v) - r\}$. This result does not hold anymore. Indeed, consider the ball $B_1(011)$ when $\rho = \frac{\sqrt{2}}{4}$. Then any correction u of the string 100 is in $\{000, 101, 110\}$. Basically, $d(011,u) = 2\rho < 1$ and $d(u,100) = \rho > d(011,100) - 1 = 3\rho - 1$. In other words, a correction is not necessarily on the circle that delimits the ball.

Nevertheless, we get a more sophisticated result that characterizes the set of all the possible corrections:

Lemma 20 *Let $B_r(o)$ be a ball and $v \notin B_r(o)$. Given any $\alpha \in \mathbb{R}$, we define*

$$C_\alpha = \{u \in \Sigma^* : d(o,u) = \alpha \text{ and } d(u,v) = d(o,v) - \alpha\}.$$

All the nonempty C_α define concentric arcs of circles of strings around the centre o. Let α_0 be the radius of biggest one inside the ball of strings:

$$\alpha_0 = \max_{0 \leq \alpha \leq r} \{\alpha : C_\alpha \neq \emptyset\}.$$

Then the set of possible corrections of v is exactly C_{α_0}.

Proof The proof is the same as that of Lemma 10, except that W is replaced by C_{α_0} and r is replaced by α_0. The key point is that W could be empty with the weighted edit distance whereas C_{α_0} cannot, by definition. ■

5.2.3 THE BORDERLINE STRINGS OF MAXIMUM LENGTH

Let us tackle the problem of learning the balls. As in Section 4, we study the longest strings of $B_r(o)$ since they are very informative. Indeed, we are going to show as in Lemma 15, that if one asks for the correction w of a string made of a lot of 0's, then $|w|_0 = |o|_0 + r$. In addition, in our setting, we also get $w \in B_r^{max}(o)$ directly. Nevertheless, we must pay for it by assuming that we know the polynomial $q()$ for which $B_r(o)$ is a good ball.

Lemma 21 *Let $q()$ be a fixed polynomial with coefficients in $\mathbb{N}$. Consider the $q()$-good ball $B_r(o)$, a letter $a \in \Sigma$ and an integer $j \in \mathbb{N}$ such that $a^j \notin B_r(o)$. Let $u = \mathrm{CQ}(a^j)$ and $v = \mathrm{CQ}(a^{j+q(j)})$. If $|u| < j$, then $v \in B_r^{max}(o)$ and $|v|_a = |o|_a + r$.*

This subsection aims at proving this lemma, using two intermediate results:

Proposition 22 *Consider the ball $B_r(o)$, a letter $a \in \Sigma$ and an integer $j \in \mathbb{N}$ such that $a^j \notin B_r(o)$. Let $u = \mathrm{CQ}(a^j)$. If $|u| < j$, then $|o| < j$.*

Proof Let us show that $|o| \leq |u|$; as $|u| < j$, we shall get the result. Hence, suppose that $|u| < |o|$ and consider a rewriting derivation $o \xrightarrow{k} u$ of minimum weight $d(o,u) = x + \rho y$. Since $|o| > |u|$, there is at least one deletion along this derivation. Suppose that, instead of deleting a letter, we substitute it by an a and do not change anything else. This leads us to a new derivation $o \xrightarrow{k} u'$ (or $o \xrightarrow{k-1} u'$ if the deleted letter was an a) with $|u'| = |u| + 1$ and $|u'|_a = |u|_a + 1$. Moreover, $d(o,u') \leq (x-1) + \rho(y+1) = d(o,u) - 1 + \rho$. Since $d(o,u) \leq r$, we deduce that $d(o,u') < r$, thus $u' \in B_r(o)$. Finally, as $|u| < j$, we get $|u'| \leq j$, so only substitutions and insertions are necessary to compute both $d(u,a^j)$ and $d(u',a^j)$. More precisely, we have $d(u',a^j) = (j - |u'|) + \rho(|u'| - |u'|_a) = (j - |u| - 1) + \rho(|u| - |u|_a) = d(u,a^j) - 1$, thus $d(u',a^j) < d(u,a^j)$. As $u' \in B_r(o)$, u cannot be a correction of a^j, which is a contradiction. So $|u| \geq |o|$, thus $|o| < j$. ■

Proposition 23 *Consider the ball $B_r(o)$, a letter $a \in \Sigma$ and an integer $\ell \in \mathbb{N}$ such that $a^\ell \notin B_r(o)$. Let $v = \mathrm{CQ}(a^\ell)$. If $|o| + r < \ell$, then $v \in B_r^{max}(o)$ and $|v|_a = |o|_a + r$.*

Proof As $|o| + r < \ell$, we have $|o| < \ell$, so the computation of $d(o,a^\ell)$ necessarily uses $\ell - |o|$ insertions of a's and $|o| - |o|_a$ substitutions by a's. Let us define a reference derivation from o to a^ℓ, where the $\ell - |o|$ insertions are performed first at the beginning of o, and then the $|o| - |o|_a$ substitutions by a's in o: $o \xrightarrow{\ell-|o|} a^{\ell-|o|}o \xrightarrow{|o|-|o|_a} a^{\ell-|o|}a^{|o|} = a^\ell$. As $\ell - |o| > r$, the string $a^{\ell-|o|}o$ is not in the ball, so this derivation passes through the string $a^r o$ before reaching $a^{\ell-|o|}o$. In other words, the reference derivation looks as follows: $o \xrightarrow{r} a^r o \xrightarrow{\ell-|o|-r} a^{\ell-|o|}o \xrightarrow{|o|-|o|_a} a^\ell$. Now consider Lemma 20. Basically, $d(o,a^r o) = r$ and $d(a^r o,a^\ell) = \ell - |o| - r + \rho(|o| - |o|_a) = d(o,a^\ell) - r$, so $C_r \neq \emptyset$. Therefore, as $v = \mathrm{CQ}(a^\ell)$, we deduce that $d(o,v) = r$ and $d(v,a^\ell) = d(o,a^\ell) - r$. We claim that only insertions of a's can occur along the derivation $o \xrightarrow{r} v$. Indeed, we have $d(o,v) = r \in \mathbb{N}$, so by Proposition 19, no substitution occurs. Moreover, no deletion occurs since any minimal derivation from o to a^ℓ only uses insertions of a's and substitutions by a's. In consequence, $v \in B_r^{max}(o)$ and $|v|_a = |o|_a + r$. ■

Proof [of Lemma 21] By Proposition 22, we get $|o| < j$. Then we have $|o| + r \leq |o| + q(|o|)$. Moreover, as $|o| < j$ and all the coefficients of $q()$ are in $\mathbb{N}$, we deduce that $|o| + r < j + q(j)$. So plugging $\ell = j + q(j)$ in Proposition 23 yields the result. ■

5.2.4 LEARNING THE BALLS LOGARITHMICALLY

As a consequence of Lemma 21, the correction of a long string of 0's leads to a string of $B_r^{max}(o)$. But we get more properties, if the alphabet has at least three letters, say $0,1,2\ldots$ Indeed, let $u_0 = \mathrm{CQ}(0^j)$ with $|u_0| < j$, and $v_0 = \mathrm{CQ}(0^{j+q(j)})$. Thanks to Lemma 21, v_0 is obtained from o with r insertions of 0's. So all the letters in v_0, but the occurrences of 0, are those of o and appear in the correct order.

More formally, let E_a be the function that erases every occurrence of any letter $a \in \Sigma$ in a string:

1. $E_a(\lambda) = \lambda$,
2. $E_a(a.z) = E_a(z)$, and

3. $E_a(b.z) = b.E_a(z)$, for all $b \neq a$.

Then, for every $a \in \Sigma$, $E_a(v_a) = E_a(o)$.

For instance, consider the ball $B_1(o)$ with $o = 10302$. If the corrections of the strings 0^ℓ, 1^ℓ and 2^ℓ (with ℓ big enough) are $v_0 = 103020$, $v_1 = 101302$ and $v_2 = 103202$ respectively, then $E_0(v_0) = E_0(o) = 132$, $E_1(v_1) = E_1(o) = 0302$ and $E_2(v_2) = E_2(o) = 1030$.

Furthermore, we can easily deduce o by *aligning* the strings $E_0(o)$ and $E_1(o)$ and $E_2(o)$:

$E_0(o)$	1	·	3	·	2
$E_1(o)$	·	0	3	0	2
$E_2(o)$	1	0	3	0	·
o	1	0	3	0	2

This procedure does not use any new correction query and runs in time $O(|o|)$ which is clearly more efficient than EXTRACT_CENTRE. Notice that if $|\Sigma| > 3$, we only need three corrections to align and deduce the center. So we finally obtain Algorithm 3 and Theorem 24.

Algorithm 3 IDF_WEIGHTED_BALLS

Require: The alphabet $\Sigma = \{a_1, \ldots, a_n\}$ with $n \geq 3$, and the polynomial $q()$
Ensure: The representation (o, r) of the target $q()$-good ball $B_r(o)$

$j \leftarrow 1$
for $i = 1$ to 3 **do**
  while $\text{CQ}(a_i^j) = \text{YES}$ **or else** $|\text{CQ}(a_i^j)| \geq j$ **do**
    $j \leftarrow 2 \cdot j$
  end while
  $v_i \leftarrow \text{CQ}\left(a_i^{j+q(j)}\right)$
  $e_i \leftarrow E_{a_i}(v_i)$
end for
$o \leftarrow \text{ALIGN}(e_1, e_2, e_3)$
$r \leftarrow |v_1| - |o|$
return (o, r)

Theorem 24 *Assume* $|\Sigma| \geq 3$.

- *Let $q()$ be any fixed polynomial with coefficients in* $\mathbb{N}$. *The set of all $q()$-good balls $B_r(o)$ is identifiable with an algorithm that uses $O(\log(|o| + q(|o|)))$ correction queries and a polynomial amount of time.*
- *The set of all very good balls is identifiable with a logarithmic number $O(\log |o|)$ of correction queries and a polynomial amount of time.*

Therefore, assuming that the weight of the substitutions is an irrational < 1 allows us to reduce dramatically the complexity of the learning stage. Of course, this gain is not possible with all weighted distances, which leaves room for further research. Moreover, if the Learner does not know the polynomial $q()$, we believe that learning is still possible.

6. Discussion and Conclusion

In this work, we have used correction queries to learn a particular class of languages from an Oracle. The intended setting is that of an inexact Oracle, and experiments show that the proposed algorithm can learn a language sufficiently close to the target for simple local modifications (with no extra queries). In order to do this, the languages we consider are good balls of strings defined with the edit distance. Studying them allowed us to catch a glimpse of the geometry of sets of strings, which is very different from the Euclidean geometry. A number of questions and research directions are left open by this work.

A first question concerns the distance we use. We have chosen to work with the unitary edit distance, but in many applications, the edit operations can have different weights. Preliminary work has allowed us to notice that the geometry of sets of strings, thus the algorithmics, could change considerably depending on the sorts of weights we used: with the substitutions costing less than the other two operations, a much faster algorithm exists, requiring only $O(\log(|o|+r))$ correction queries. Alternative conditions over the weights require new interesting learning algorithms.

A second question concerns the inaccuracy model we are using: as noticed in Section 5.1, with the current model it would be possible to repeat the same query various times, getting different corrections, but possibly being able, through some majority vote scheme, to get the adequate correction with very little extra cost. Just asking for *persistent* corrections is not enough to solve this problem: a good model should require that if one queries from a close enough string (a^{999} instead of a^{1000}) then the corrections should also remain close. Topologically, we would expect the Oracle to be k-Lipschitz continuous (with $0 < k < 1$).

A third more challenging problem then arises: our choice here was to learn supposing the Oracle was exact, and correcting later. But a more direct approach might be better, by taking into account the inexactitude of the Oracle when interpreting the correction.

Acknowledgments

The authors wish to thank Jose Oncina for his help in proving Theorem 5, Rémi Eyraud for fruitful discussions about this paper, Dana Angluin for constructive comments and Baptiste Gorin for his helpful pointers towards the Mohr-Mascheroni constructions. We would also like to thank the anonymous referees that have carefully read this manuscript and allowed us to improve the results based on the weighted edit distance. Their remarks led us to formulate Conjecture 4 that was discussed with Ron Greensberg, Borivoj Melichar, Klaus Schultz and Stoyan Mihov. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2006-216886, and by a Marie Curie International Fellowship within the 6th European Community Framework Programme. This publication only reflects the authors' views.

References

A. Albright and B. Hayes. Rules vs. analogy in English past tenses: A computational/experimental study. *Cognition*, 90:119–161, 2003.

J.-C. Amengual and P. Dupont. Smoothing probabilistic automata: An error-correcting approach. In *Proc. of the 5th International Colloquium in Grammatical Inference (ICGI'00)*, pages 51–64.

LNAI 1891, 2000.

J.-C. Amengual, A. Sanchis, E. Vidal, and J.-M. Benedí. Language simplification through error-correcting and grammatical inference techniques. *Machine Learning Journal*, 44(1-2):143–159, 2001.

D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987a.

D. Angluin. Queries revisited. *Theoretical Computer Science*, 313(2):175–194, 2004.

D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39: 337–350, 1978.

D. Angluin. Queries and concept learning. *Machine Learning Journal*, 2(4):319–342, 1987b.

D. Angluin and M. Kharitonov. When won't membership queries help? *Journal of Computer and System Sciences*, 50(2):336–355, 1995.

L. Becerra-Bonache. *On the Learnability of Mildly Context-Sensitive Languages using Positive Data and Correction Queries*. PhD thesis, Rovira i Virgili University, Tarragona, 2006.

L. Becerra-Bonache and T. Yokomori. Learning mild context-sensitiveness: Towards understanding children's language learning. In *Proc. of the 7th International Colloquium in Grammatical Inference (ICGI'04)*, pages 53–64. LNAI 3264, 2004.

L. Becerra-Bonache, A. H. Dediu, and C. Tirnauca. Learning DFA from correction and equivalence queries. In *Proc. of the 8th International Colloquium in Grammatical Inference (ICGI'06)*, pages 281–292. LNAI 4201, 2006.

L. Bréhélin, O. Gascuel, and G. Caraux. Hidden Markov models with patterns to learn boolean vector sequences and application to the built-in self-test for integrated circuits. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):997–1008, 2001.

J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1):33–67, 2007.

E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

N. Chomsky. *Syntactic Structure*. Mouton, 1957.

A. Clark, C. Costa Florêncio, and C. Watkins. Languages as hyperplanes: Grammatical inference with string kernels. In *Proc. of the 17th European Conference on Machine Learning (ECML'06)*, pages 90–101. LNCS 4212, 2006.

F. Coste, K. Lang, and B. A. Pearlmutter. The Gowachin automata learning competition, 1998. URL `http://www.irisa.fr/Gowachin/`.

M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.

C. de la Higuera. Data complexity issues in grammatical inference. In M. Basu and T. K. Ho, editors, *Data Complexity in Pattern Recognition*, pages 153–172. Springer-Verlag, 2006.

C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27: 125–138, 1997.

C. de la Higuera and F. Casacuberta. Topology of strings: Median string is NP-complete. *Theoretical Computer Science*, 230:39–48, 2000.

C. de la Higuera, J.-C. Janodet, and F. Tantini. Learning languages from bounded resources: The case of the DFA and the balls of strings. In *Proc. of the 9th International Colloquium in Grammatical Inference (ICGI'08)*, page ? (to appear). LNAI, 2008.

R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

C. L. Giles, S. Lawrence, and A.C. Tsoi. Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine Learning Journal*, 44(1):161–183, 2001.

D. Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

A. Hagerer, H. Hungar, O. Niese, and B. Steffen. Model generation by moderated regular extrapolation. In *Proc. of the 5th International Conference on Fundamental Approaches to Software Engineering (FASE'02)*, pages 80–95. LNCS 2306, 2002.

M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

M. J. Kearns and M. Li. Learning in the presence of malicious errors. *SIAM Journal of Computing*, 22(4):807–837, 1993.

E. B. Kinber. On learning regular expressions and patterns via membership and correction queries. In *Proc. of the 9th International Colloquium in Grammatical Inference (ICGI'08)*, page ? (to appear). LNAI, 2008.

T. Kohonen. Median strings. *Pattern Recognition Letters*, 3:309–313, 1985.

K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Proc. of the 4th International Colloquium in Grammatical Inference (ICGI'98)*, pages 1–12. LNAI 1433, 1998.

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965.

S. Lucas. Learning deterministic finite automata from noisy data competition, 2004. URL `http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html`.

C. D. Martínez-Hinarejos, A. Juan, and F. Casacuberta. Use of median string for classification. In *Proc. of the 15th International Conference on Pattern Recognition (ICPR'00)*, volume 2, pages 2903–2906, 2000.

L. Mascheroni. Geometria del compasso. Pavia, 1797.

B. Melichar. Approximate string matching by finite automata. In *Proc. 6th International Conference on Computer Analysis of Images and Patterns (CAIP'95)*, pages 342–349. LNCS 970, 1995.

G. Mohr. Euclides danicus. Amsterdam, 1672.

G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

M.-F. Sagot and Y. Wakabayashi. Pattern inference under many Guises. In *Recent Advances in Algorithms and Combinatorics*, pages 245–287. Springer-Verlag, 2003.

J. Sakarovich. *Eléments de Théorie des Automates*. Vuibert, 2004.

A. Salomaa. On languages defined by numerical parameters. In *Formal Models, Languages and Applications*, volume 66 of *Machine Perception and Artificial Intelligence*, chapter 8. World Scientific Publishing Company, 2006.

K. U. Schulz and S. Mihov. Fast string correction with Levenshtein automata. *International Journal on Document Analysis and Recognition*, 5(1):67–85, 2002.

F. Tantini, C. de la Higuera, and J. C. Janodet. Identification in the limit of systematic-noisy languages. In *Proc. of the 8th International Colloquium in Grammatical Inference (ICGI'06)*, pages 19–31. LNCS 4201, 2006.

B. Trakhtenbrot and Y. Barzdin. *Finite Automata: Behavior and Synthesis*. North Holland Pub. Comp., Amsterdam, 1973.

E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, 1985.

R. Wagner and M. Fisher. The string-to-string correction problem. *Journal of the ACM*, 21:168–178, 1974.