

# Learning Balls of Strings with Correction Queries<sup>\*</sup>

Leonor Becerra Bonache<sup>1</sup>, Colin de la Higuera<sup>2</sup>, Jean-Christophe Janodet<sup>2</sup>,  
and Frédéric Tantini<sup>2</sup>

<sup>1</sup> Research Group on Mathematical Linguistics, Rovira i Virgili University  
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain

`leonor.becerra@urv.cat`

<sup>2</sup> Laboratoire Hubert Curien, Université Jean Monnet  
18 rue du Professeur Benoît Lauras, 42000 Saint-Étienne, France  
{`cdlh,janodet,frédéric.tantini`}@univ-st-etienne.fr

**Abstract.** During the 80's, Angluin introduced an active learning paradigm, using an Oracle, capable of answering both membership and equivalence queries. However, practical evidence tends to show that if the former are often available, this is usually not the case of the latter. We propose new queries, called correction queries, which we study in the framework of Grammatical Inference. When a string is submitted to the Oracle, either she validates it if it belongs to the target language, or she proposes a correction, *i.e.*, a string of the language close to the query with respect to the edit distance. We also introduce a non-standard class of languages: The topological balls of strings. We show that this class is not learnable in Angluin's MAT model, but is with a linear number of correction queries. We conduct several experiments with an Oracle simulating a human Expert, and show that our 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 Shwartzenegger* was born? A few years ago, just 2 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 organised 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 your favourite search engine,

---

<sup>\*</sup> This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

type 2 keywords, follow 3 links and note down the answers. In this particular case, you discover...that no king of Babylon was called *Nabcodonosaur* but 2 *Nabuchodonosor*'s reigned there many centuries ago. Again, the day *Arnold Shwartzenegeger* was born is not clear, but it is easy to check that *Arnold Schwarzenegger* was born in 1947, July 30<sup>th</sup>.

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 correction queries allow to get new challenging results in the field of Active Learning. In this framework developed by Angluin in the 80's [1], 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. Assuming that the Oracle is a human expert, however, introduces new constraints. On the 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 minimising 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 distance used by the search engine to find a closest string is a variant of the *edit distance* which measures the minimum number of deletion, insertion or substitution operations needed to transform one string into another [2, 3]. This distance and variants where each elementary operation may have a different weight have been used in many fields including Computational Biology [4], Language Modelling [5] and Pattern Recognition [6]. Edit distance appears in specific Grammatical Inference problems, in particular when one wants to learn languages from noisy data [7]. The classes of languages studied there 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 the parity functions (which can be defined as regular languages) are not learnable in the presence of noise [8]. In this paper also, in order to avoid this difficulty, we shall consider only special finite languages, that seem elementary

to formal language theoreticians, but are relevant for topologists and complex for combinatorialists: the *balls of strings*.

Hence, we study the problem of identifying balls of strings from correction queries. After some preliminaries in Section 2, we prove that balls are not learnable with Angluin's membership and equivalence queries (Section 3). Then, we show in Section 4 that balls are learnable with a linear number of correction queries. In Section 5, we study the effectiveness of our algorithm from an experimental standpoint, showing that it is *robust* when the answers of the Oracle are approximate. We conclude in Section 6. Due to the lack of space, we have skipped most formal proofs. The interested reader may find them at <http://labh-curien.univ-st-etienne.fr/~tantini/pub/bhjt07Long.pdf>.

## 2 On Balls of Strings as Languages

An *alphabet*  $\Sigma$  is a finite nonempty set of symbols called *letters*. A *string*  $w = a_1 \dots a_n$  is any finite sequence of letters. We write  $\Sigma^*$  for the set of all strings over  $\Sigma$ , and  $\lambda$  for the empty string. Let  $|w|$  be the length of  $w$  and  $|w|_a$  the number of occurrences of  $a$  in  $w$ .

The *edit distance*  $d(w, w')$  is the minimum number of *edit operations* needed to transform  $w$  into  $w'$  [2]. The edit operations are either (1) *deletion*:  $w = uav$  and  $w' = uv$ , or (2) *insertion*:  $w = uv$  and  $w' = uav$ , or (3) *substitution*:  $w = uav$  and  $w' = ubv$ , where  $u, v \in \Sigma^*$ ,  $a, b \in \Sigma$  and  $a \neq b$ . *E.g.*,  $d(abaa, aab) = 2$  since  $\underline{a}baa \rightarrow aa\underline{a} \rightarrow aab$  and the rewriting of  $abaa$  into  $aab$  cannot be achieved with less than 2 steps. Notice that  $d(w, w')$  can be computed in  $\mathcal{O}(|w| \cdot |w'|)$  time by dynamic programming [3].

It is well-known that the edit distance is a *metric* [9], so it conveys to  $\Sigma^*$  the structure of a *metric space*. The *ball of centre*  $o \in \Sigma^*$  and *radius*  $r \in \mathbb{N}$ , denoted  $B_r(o)$ , is the set of all strings whose distance is at most  $r$  from  $o$ :  $B_r(o) = \{w \in \Sigma^* : d(o, w) \leq r\}$ . *E.g.*, if  $\Sigma = \{a, b\}$ , then  $B_1(ba) = \{a, b, aa, ba, bb, aba, baa, bab, bba\}$  and  $B_r(\lambda) = \Sigma^{\leq r}$  for all  $r \in \mathbb{N}$ .

The latter example illustrates the fact that the number of strings in a ball grows exponentially with the radius. This remark raises the problem of the representation scheme that we should use to learn the balls. Basically, we need representations whose size is reasonable, which is not the case of an exhaustive enumeration, nor of the *deterministic finite automata* (DFA) since experiments show that the corresponding minimum DFA is often exponential with  $r$  (but linear with  $|o|$ ) [10], even if a formal proof of this property remains a challenging combinatorial problem.

On the other hand, why not represent the ball  $B_r(o)$  by the pair  $(o, r)$  itself? Indeed, its size is  $|o| + \log r$ . Moreover, 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  $\mathcal{O}(|o| \cdot |w| + \log r)$ . Finally, when the alphabet has at least 2 letters,  $(o, r)$  is a unique thus *canonical* representation of  $B_r(o)$ :

**Theorem 1.** *If  $|\Sigma| \geq 2$  and  $B_{r_1}(o_1) = B_{r_2}(o_2)$ , then  $o_1 = o_2$  and  $r_1 = r_2$ .*

Notice that if  $\Sigma = \{a\}$ , then  $B_2(a) = B_3(\lambda) = \{\lambda, a, aa, aaa\}$  for instance.

Hence, representing the ball  $B_r(o)$  by the pair  $(o, r)$  is reasonable. However, it is worth noticing that *huge* balls, whose radius is not polynomially related to the length of the centre (e.g.,  $r > 2^{|o|}$ ), will pose tricky problems of complexity. For instance, to learn the ball  $B_r(\lambda) = \Sigma^{\leq r}$ , one needs to manipulate at least one string of length  $r + 1$ . Therefore, in the following, we will always consider *good* balls only:

**Definition 1.** *Given any fixed polynomial  $q()$ , we say that a ball  $B_r(o)$  is  $q$ -good if  $r \leq q(|o|)$ .*

### 3 Learning Balls from Queries

Query learning is a paradigm introduced by Angluin [1]. Her model brings a Learner (he) and an Oracle (she) 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 (i.e., she does not lie). Moreover, the Learner is bound by efficiency constraints: (1) He can only submit a polynomial number of queries (in the size of the target representation) and (2) the available overall time must be polynomial in the size of the target representation<sup>3</sup>.

Between the different combinations of queries, one, called MAT (Minimally Adequate Teacher), is sufficient to learn DFA [11]. Two kinds of queries are used:

**Definition 2.** *Let  $\Lambda$  be a class of languages on  $\Sigma^*$  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  $\text{MQ}(w)$ , 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  $\text{EQ}(K)$ , is either YES if  $K = L$ , or a string belonging to the symmetric difference  $((K \setminus L) \cup (L \setminus K))$  if  $K \neq L$ .*

Although MQ and EQ have established themselves as a standard combination, there are real grounds to believe that EQ are too powerful to exist or even be simulated. As suggested in [11] we may be able to substitute them with a random draw of strings that are then submitted as MQ (*sampling*), but there are many cases where sampling is not possible as the relevant distribution is unknown and/or inaccessible [12]. Besides, we will not consider MQ and EQ because they do not help to learn balls:

**Theorem 2.** *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 EQ and MQ necessarily uses  $\Omega(|\Sigma|^n)$  queries in the worst case.*

<sup>3</sup> The time complexity usually concerns the time spent after receiving each new example, and takes the length of the information returned by the Oracle into account; Thus, our constraint is stronger but not restrictive, if we focus on good balls only.

*Proof.* Following [13], we describe an Adversary who maintains a set  $S$  of all possible balls. At the beginning,  $S = \mathcal{B}_{\leq m, n}$ . Her answer to the equivalence query  $L = B_r(o)$  is the counterexample  $o$ . Her answer to the membership query  $o$  is No. At each step, the Adversary eliminates many balls of  $S$  but only one of centre  $o$  and radius 0. As there are  $\Omega(|\Sigma|^n)$  such balls in  $\mathcal{B}_{\leq m, n}$ , identifying them requires  $\Omega(|\Sigma|^n)$  queries.  $\square$

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

**Definition 3.** *Let  $L$  be a fixed language 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  $w$  w.r.t.  $L$  if  $w \notin L$ , that is a string  $w' \in L$  at minimum edit distance from  $w$ :  $\text{CQ}(w) = \text{one string of } \{w' \in L : d(w, w') \text{ is minimum}\}$ .*

Notice that the CQ 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, CQ are relevant from a cognitive point of view: There is growing evidence that corrective input for grammatical errors is widely available to children [14].

## 4 Identifying Balls using Corrections

In this section, we propose an algorithm that learns balls using a *linear* number of CQ. First, when one 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. *E.g.*, the possible corrections for the string  $aaaa$  w.r.t. the ball  $B_2(bb)$  are  $\{aa, aab, aba, baa, aabb, abab, abba, baab, baba, bbaa\}$ . By definition of the CQ, the Oracle will choose one of them arbitrarily, potentially the worst one w.r.t. the Learner’s point of view. Nevertheless, the Oracle’s potential malevolence is limited by the following result, that characterises the set of *all* the possible corrections for a string:

**Theorem 3.** *Let  $B_r(o)$  be a ball and  $m \notin B_r(o)$ . Then the set of possible corrections of  $m$  is exactly  $\{z \in \Sigma^* : d(o, z) = r \text{ and } d(z, m) = d(o, m) - r\}$ .*

Here is a geometric interpretation of the result above. Let us define the *segment*  $[o, m] = \{w \in \Sigma^* : d(o, w) + d(w, m) = d(o, m)\}$  and the *circle*  $C_r(o) = \{w \in \Sigma^* : d(o, w) = r\}$ . Theorem 3 states that a string  $z$  is a possible correction of  $m$  iff  $z \in [o, m] \cap C_r(o)$ . The fact that  $m$  has several possible corrections shows that the geometry of  $\Sigma^*$  is very different from that of  $\mathbb{R}^2$ .

Now, building the centre of a ball from strings on its periphery is difficult for at least 2 reasons. On the one hand,  $(\Sigma^*, d)$  is a metric space with no vector space as an underlying structure. This is the same story as if we were trying to learn the disks of the plane with just a compass but no ruler. . . On the other hand, the

---

**Require:** a string  $w = x_1 \dots x_n \in B_r^{max}(o)$ , the radius  $r$   
**Ensure:** the centre  $o$  of the ball  $B_r(o)$

- 1:  $c \leftarrow 0; i \leftarrow 1; a \leftarrow x_n$
- 2: **while**  $c < r$  **do**
- 3:   Assume  $w = x_1 \dots x_n$  and let  $w' = ax_1 \dots x_{i-1}x_{i+1} \dots x_n$
- 4:   **if**  $CQ(w') = \text{YES}$  **then**  $w \leftarrow w'; c \leftarrow c + 1$  **end if**
- 5:    $i \leftarrow i + 1$
- 6: **end while**
- 7: Assume  $w = x_1 \dots x_n$  and **return**  $x_{r+1} \dots x_n$

---

**Fig. 1.** Algorithm EXTRACT\_CENTRE

problem is formally *hard*: Given a finite set of strings  $W = \{w_1, \dots, w_n\}$  and a constant  $K$ , deciding whether a string  $z \in \Sigma^*$  exists such that  $\sum_{w \in W} d(z, w) < K$  (*resp.*  $\max_{w \in W} d(z, w) < K$ ) is  $\mathcal{NP}$ -hard [15].

Therefore, we must study the balls in more detail and make the best possible use of the CQ so as not to build the centres from scratch. We begin by distinguishing the longest strings of any ball:

**Definition 4.** *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:  $B_r^{max}(o) = \{z \in B_r(o) : \forall w \in B_r(o), |w| \leq |z|\}$ .*

*E.g.*, let  $\Sigma = \{a, b\}$ , then  $B_1^{max}(ba) = \{aba, baa, bab, bba\}$ . 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. 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 CQ allow us to get hold of  $a^r o$  from any string  $w \in B_r^{max}(o)$  by swapping the letters (see Algorithm EXTRACT\_CENTRE in Figure 1).

Consider  $B_2^{max}(bb) = \{aabb, abab, abba, abbb, baab, baba, babb, bbaa, bbab, bbba, bbbb\}$ . Running EXTRACT\_CENTRE on the string  $w = baab$  and radius  $r = 2$  transforms, at each loop, the  $i^{th}$  letter of  $w$  to an  $a$  that is put at the beginning and then submits it to the Oracle.  $c$  counts the number of times this transformation is accepted. We get:

$i$	$w$	$w'$	$CQ(w')$	$w$ changes	$c$
1	<u>b</u> aab	aaab	baab	no	0
2	ba <u>a</u> b	abab	YES	yes	1
3	ba <u>b</u> a	aabb	YES	yes	2

When  $c = 2 = r$ , EXTRACT\_CENTRE stops with  $w = aabb$  and returns  $o = bb$ .

**Lemma 1.** *Given  $w \in B_r^{max}(o)$  and  $r$ , Algorithm EXTRACT\_CENTRE returns  $o$  using  $\mathcal{O}(|o| + r)$  CQ and a polynomial amount of time.*

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 lemma is a step towards finding this string (although we have no information about  $r$  and  $|o|$  yet):

**Lemma 2.** *Suppose  $\Sigma = \{a_1, \dots, a_n\}$ . Then every correction  $w$  of the string  $m = (a_1 \dots a_n)^k$  where  $k \geq |o| + r$  belongs to  $B_r^{max}(o)$ .*

Submitting  $(a_1 \dots a_n)^k$  with a sufficiently large  $k$  is sure to be corrected by a string from  $B_r^{max}(o)$ . So all that remains is to find such an interesting  $k$ . The following lemma states that if one asks the Oracle to correct a string made of a lot of  $a$ 's, then the correction contains precious informations on the radius and the number of occurrences of  $a$ 's in the centre:

**Lemma 3.** *Consider the ball  $B_r(o)$  and let  $a \in \Sigma$  and an integer  $j \in \mathbb{N}$  such that  $a^j \notin B_r(o)$ . Let  $w = \text{CQ}(a^j)$ . If  $|w| < j$ , then  $|w|_a = |o|_a + r$ .*

Now, let us assume that the alphabet is  $\Sigma = \{a_1, \dots, a_n\}$  and let  $j_1, \dots, j_n \in \mathbb{N}$  be large integers. Define  $k = \sum_{i=1}^n |\text{CQ}(a_i^{j_i})|_{a_i}$ . Then, Lemma 3 brings  $k = \sum_{i=1}^n (|o|_{a_i} + r) = |o| + |\Sigma| \cdot r \geq |o| + r$ . Thus we can plug  $k$  into Lemma 2 to get a string  $w = \text{CQ}((a_1 \dots a_n)^k) \in B_r^{max}(o)$ . Moreover, we have  $|w| = |o| + r$  and  $k = |o| + |\Sigma| \cdot r$ . So, we deduce that the radius is  $r = (k - |w|)/(|\Sigma| - 1)$ .

Let us summarise, by assuming that  $\Sigma = \{a_1, \dots, a_n\}$  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) We define  $k = \sum_{i=1}^n |\text{CQ}(a_i^j)|_{a_i}$  and suppose the Learner gets the correction  $w$  for the string  $m = (a_1 \dots a_n)^k$ ; (3) From  $k$  and  $|w|$ , we deduce  $r$ ; (4) The Learner uses `EXTRACT_CENTRE` on  $w$  and  $r$  in order to find  $o$ . In other words, we are able to learn the balls with `CQ` (see Algorithm `IDF BALL` in Figure 2).

---

**Require:** The alphabet  $\Sigma = \{a_1, \dots, a_n\}$   
**Ensure:** The representation  $(o, r)$  of the ball  $B_r(o)$

- 1:  $j \leftarrow 1; k \leftarrow 0$
- 2: **for**  $i = 1$  **to**  $n$  **do**
- 3:   **while**  $|\text{CQ}(a_i^j)| \geq j$  **do**  $j \leftarrow 2 \cdot j$  **end while**
- 4:    $k \leftarrow k + |\text{CQ}(a_i^j)|_{a_i}$
- 5: **end for**
- 6:  $w \leftarrow \text{CQ}((a_1 a_2 \dots a_n)^k)$
- 7:  $r \leftarrow (k - |w|)/(|\Sigma| - 1)$
- 8:  $o \leftarrow \text{EXTRACT\_CENTRE}(w, r)$
- 9: **return**  $(o, r)$

---

**Fig. 2.** Algorithm `IDF BALL`

For instance, consider the ball  $B_2(bb)$  defined over  $\Sigma = \{a, b\}$ . `IDF BALL` begins by looking for the corrections of  $a^j$  and  $b^j$  with a sufficiently large  $j$ . We might observe:  $\text{CQ}(a) = \text{YES}$ ,  $\text{CQ}(a^2) = \text{YES}$ ,  $\text{CQ}(a^4) = aabb$ ,  $\text{CQ}(a^8) = abba$ ,

$CQ(b^8) = bbbb$ . So  $k = |abba|_a + |bbbb|_b = 2 + 4 = 6$ . Then  $CQ((ab)^6) = CQ(ababababab) = baab$ , for instance, so  $r = (6 - 4)/(2 - 1) = 2$ . Finally,  $EXTRACT\_CENTRE(baab, 2)$  returns  $bb$ . So the algorithm returns  $(bb, 2)$ .

**Theorem 4.** *Given any fixed polynomial  $q()$ , the set of all  $q$ -good balls  $B_r(o)$  is identifiable with an algorithm using  $\mathcal{O}(|\Sigma| + |o| + r)$  CQ and a polynomial amount of time.*

*Proof.* The identifiability is clear. Concerning the complexity, the corrections of the strings  $a_i^j$  requires  $\mathcal{O}(|\Sigma| + \log(|o| + r))$  CQ (lines 2-5).  $EXTRACT\_CENTRE$  needs  $\mathcal{O}(|o| + r)$  CQ (line 8).  $\square$

Notice that the set of all the balls, that contains good balls but also *huge* ones such that  $r > 2^{|o|}$  for instance, is not polynomially identifiable with  $IDF\_BALL$  since  $\mathcal{O}(|\Sigma| + |o| + r) > \mathcal{O}(2^{|o|})$  for some of them.

## 5 Experiments with a Human-like Oracle

In this section, we would like to show the advantages of our approach. Therefore, we have made several experiments that aim at studying the responses of  $IDF\_BALL$  faced with an Oracle that could be human. As we said in introduction, our algorithm should not believe unwisely the answers he gets since they can be approximate. We would like to show here that  $IDF\_BALL$  withstands such approximate (*i.e.*, inaccurate, noisy) answers.

### *Designing the Approximate Oracle*

We begin by modelling a human expert by an Approximate Oracle. Firstly, we assume that an expert can easily determine whether an example fulfils a concept or not, thus here, whether  $w$  belongs to  $B_r(o)$  or not. Secondly, what is really hard for the expert is to *compute* the 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*.

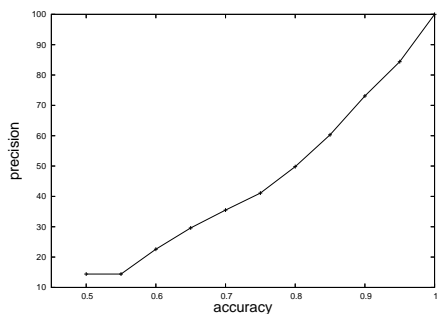
Let  $X = d(w, CQ_h(w)) - d(w, CQ(w))$  measure how far an approximate correction is from a perfect one. Intuitively, an Approximate Oracle will often provide corrections such that  $X = 0$ , sometimes  $X = 1$  and rarely  $X \geq 2$ . . . To formalise this idea, we introduce a confidence parameter  $0 < p \leq 1$ , called the *accuracy level of the Oracle*, that translates the quality of her answers, and use a geometric distribution: We assume that  $Pr(X = k) = (1 - p)^k p$ , for all  $k \in \mathbb{N}$ . Therefore, with probability  $(1 - p)^k p$ , the correction  $CQ_h(w)$  of a string  $w$  will be in the target ball, at distance  $k$  of  $CQ(w)$ . 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  $\simeq 1.5$ .

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!* However, we want here to study the resistance of  $IDF\_BALL$  to approximate answers, not to design the best possible algorithm, so our model is sufficient.

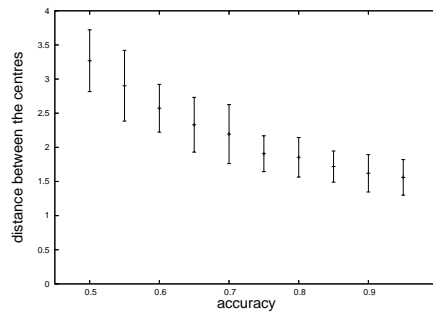


### Behaviour of the Algorithm faced with an Approximate Oracle

Following Theorem 4, 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. For every accuracy level  $0.5 \leq p \leq 1$ , we randomly choose a set of 100 balls  $B_r(o)$  such that  $|o| + r = 200$ . More precisely, the radius  $r$  varies between 20 and 180 by step of 20, and we randomly choose 10 centres  $o$  of length  $200 - r$  for each radius. Then we ask IDF\_BALL to learn them and compute the percentage of balls he is able to retrieve, which we call the *precision of the algorithm*. We show the result in Figure 3. 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  $p = 0.5$ ). We also show, in Figure 4, the average distances between the centres of the target balls and the centres of the learnt balls when he fails to retrieve them. We observe that these distances are not that important: Even with an accuracy level of  $p = 0.5$ , this distance is less than 4.



**Fig. 3.** Precision of IDF\_BALL faced with an Approximate Oracle in function of the accuracy level  $p$ . Each point is assessed on 100 balls.



**Fig. 4.** 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.

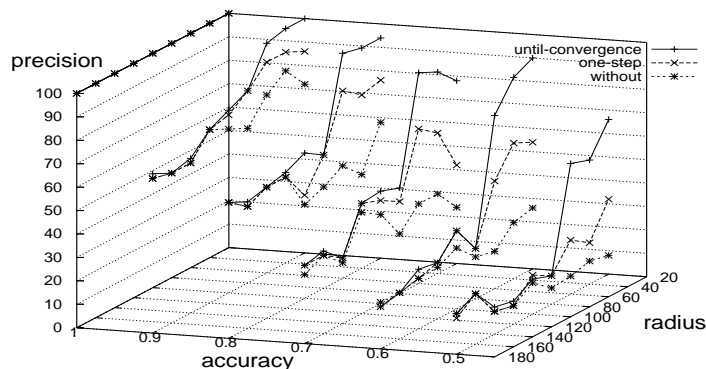
### Improving the Precision with a posteriori Heuristics

We have seen that IDF\_BALL was able to assimilate the approximations of the Oracle up to a certain level of accuracy. Moreover, the centre returned by the algorithm is generally not far from the target one. Thus, it is reasonable to think that we could improve the precision by mining the neighbourhood of the learnt centre, using local edit modifications. This kind of approaches has been pioneered by Kohonen in [16] and is surveyed in [17].

Suppose that the learnt ball is  $B_k(u)$ . We test each neighbour (at distance 1) of  $u$  and examine if it is better (*i.e.* if  $k$  can be reduced) in such a way as to contain all the corrections given up to now. This heuristics will be very good each

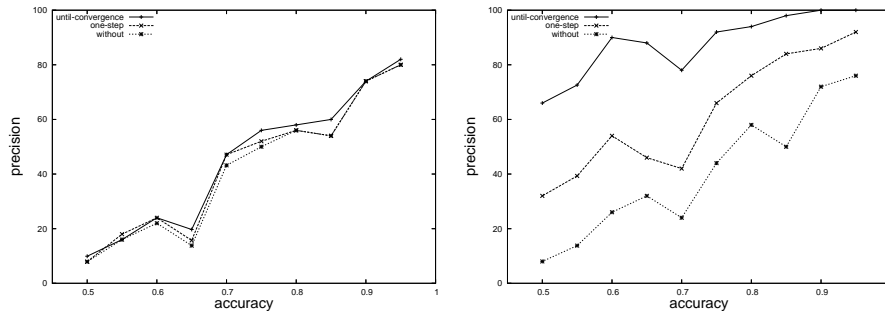
time  $u$  is at distance 1 from the target centre. But as soon as this distance grows, IDF\_BALL will fail again. In order to enhance the one-step heuristics, we can iterate the process and design a second until-convergence heuristics by repeating the local search until the size of the ball cannot be diminished 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  $|o| + r = 200$ . The accuracy level varies from 0.5 to 1 and the radius, from 20 to 180. For each pair (accuracy, radius), we randomly draw 50 centres of length  $200 - r$ , and ask IDF\_BALL to retrieve them. We plot the resulting precisions in Figure 5.



**Fig. 5.** 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 precision over 50 balls.

Firstly, we must explain the bumpiness of the graph. Actually, each point assesses the precision of IDF\_BALL using a sample of 50 balls. However, the number of balls of radius 100, for instance, is greater than  $10^{30}$ ! So, whatever the sample, we will definitively not get any relevant estimate of the true precision, that explains the variance. On the other hand, this is not limiting since our experiments still allow to compare the heuristics themselves: 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. But it is also clear that 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 2 transverse sections, shown in Figure 6, where we fix the radius.



**Fig. 6.** Precision of IDF\_BALL when  $|o| + r = 200$  for  $r = 120$  (left) and  $r = 20$  (right). For each accuracy, we compute the average over 50 balls.

The left curves of Figure 6 describe the precision of IDF\_BALL for target balls such that  $r = 120$  and  $|o| = 80$ . In this case, we gain little using the heuristics. Notice that these balls are not *good* for the identity polynomial. On the other hand, the right curves of Figure 6 describe the precision for target balls such that  $r = 20$  and  $|o| = 180$ . Basically, our heuristics outperform the precision *w.r.t.* 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 26% to 90%. 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.

## 6 Discussion and Conclusion

In this work, we have used correction queries to learn languages from an Oracle. The intended setting is that of an inexact Oracle, and experiments show that the algorithm we propose can learn a language sufficiently close to the target for simple local modifications (with no extra queries) to be possible. In order to do this, the languages we consider are 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. For instance, with the substitutions costing less than the insertions and the deletions, a much faster algorithm exists, requiring only a number of queries in  $\mathcal{O}(\log(|o| + r))$  [18].

A second question concerns the inaccuracy model we are using: As noticed in Section 5, with the current model it would be possible to repeat the same

CQ 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  $0 < k < 1$ ).

*Acknowledgement* The authors wish to thank the anonymous reviewers as well as Dana Angluin, Jose Oncina and Rémi Eyraud for their helpful comments.

## References

1. Angluin, D.: Queries and concept learning. *Machine Learning Journal* **2** (1987) 319–342
2. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR* **163**(4) (1965) 845–848
3. Wagner, R., Fischer, M.: The string-to-string correction problem. *Journal of the ACM* **21** (1974) 168–178
4. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: *Biological sequence analysis*. Cambridge University Press (1998)
5. Amengual, J.C., Sanchis, A., Vidal, E., Benedí, J.M.: Language simplification through error-correcting and grammatical inference techniques. *Machine Learning Journal* **44**(1) (2001) 143–159
6. Oncina, J., Sebban, M.: Learning stochastic edit distance: Application in handwritten character recognition. *Pattern Recognition* **39**(9) (2006) 1575–1587
7. Tantini, F., de la Higuera, C., Janodet, J.C.: Identification in the limit of systematic-noisy languages. In: Proc. of the 8th Int. Coll. in Grammatical Inference (ICGI'06), LNCS 4201 (2006) 19–31
8. Kearns, M.J., Li, M.: Learning in the presence of malicious errors. *SIAM Journal of Computing* **22**(4) (1993) 807–837
9. Crochemore, M., Hancart, C., Lecroq, T.: *Algorithmique du texte*. Vuibert (2001)
10. Schulz, K.U., Mihov, S.: Fast string correction with levenshtein automata. *Int. Journal on Document Analysis and Recognition* **5**(1) (2002) 67–85
11. Angluin, D.: Learning regular sets from queries and counterexamples. *Information and Computation* **75** (1987) 87–106
12. de la Higuera, C.: Data complexity issues in grammatical inference. In: *Data Complexity in Pattern Recognition*. Springer-Verlag (2006) 153–172
13. Angluin, D.: Queries and concept learning. *Machine Learning* **2** (1988) 319–342
14. Becerra-Bonache, L., Yokomori, T.: Learning mild context-sensitiveness: Towards understanding children's language learning. In: Proc. of the 7th Int. Coll. in Grammatical Inference (ICGI'04), LNCS 3264 (2004) 53–64
15. de la Higuera, C., Casacuberta, F.: Topology of strings: median string is NP-complete. *Theoretical Computer Science* **230** (2000) 39–48
16. Kohonen, T.: Median strings. *Information Processing Letters* **3** (1985) 309–313
17. Martínez-Hinarejos, C.D., Juan, A., Casacuberta, F.: Use of median string for classification. In: Proc. of the 15th International Conference on Pattern Recognition. Volume 2. (2000) 2903–2906
18. Becerra-Bonache, L., de la Higuera, C., Janodet, J.C., Tantini, F.: Apprentissage des boules de mots avec des requêtes de correction (*in french*). In: Proc. of 9th Conférence en Apprentissage, Presses Universitaires de Grenoble (2007)