Robust Spelling Correction*

Manuel Vilares[1], Juan Otero[1], and Jesús Vilares[2]

[1] Department of Computer Science, University of Vigo,
Campus As Lagoas s/n, 32004 Ourense, Spain
{vilares, jop}@uvigo.es
[2] Department of Computer Science, University of A Coruña,
Campus de Elviña s/n, 15071 A Coruña, Spain
jvilares@udc.es

Abstract. The paper introduces a robust spelling correction technique to deal with ill-formed input strings, including unknown parts of unknown length. In contrast to previous works, we derive profit from a finer dynamic programming construction, which takes advantage of the underlying grammatical structure, leading to an improved computational behavior and error repair quality. The formal description applies a deductive approach in order to simplify this task, separating it from the interpretation strategy, and including cut-off facilities.

1 Introduction

Although spelling correction has been a central subject in natural language processing (NLP) for a long time [1], recent years have seen a renewal of interest in it due to the increasing amount of textual information available in electronic format. Here, the state of the art [2] focuses on contextual and non-contextual error correction. In relation to the former, most proposals are based on NLP techniques and/or statistical-language models, integrating linguistic knowledge [3, 4]. For the latter, techniques look for possible editing sequences to reflect the error occurrence phenomenon in spelling. These strategies study correction patterns, most of them taking into account the *edit distance* [5], but also on occasion introducing constraints on the spelling process [6] in order to cut down the computational time needed for the correction.

Even non-contextual strategies can be of interest in a number of practical applications, when no training corpus is available and/or it is not easy to obtain statistics for estimating the linguistic model, these algorithms can be considered as a preliminary phase in a more sophisticated contextual approach such as shallow and partial interpretation. Our proposal extends an original non-contextual regional least-cost spelling correction proposal [7] in order to provide both robustness in noisy conditions and general parameterizable cut-off criteria. In relation to previous works, we provide a formal definition framework and an improved computational behavior.

* Research supported by the Spanish Government under projects TIN2004-07246-C03-01, TIN2004-07246-C03-02, and the Autonomous Government of Galicia under projects PGIDIT03SIN30501PR and PGIDIT02SIN01E.

J. Farré, I. Litovsky, and S. Schmitz (Eds.): CIAA 2005, LNCS 3845, pp. 319–328, 2006.

2 The Operational Model

Our aim is to parse a word $w_{1..n} = w_1 \dots w_n$ according to an RG $\mathcal{G} = (N, \Sigma, P, S)$. We denote by w_0 (resp. w_{n+1}) the position in the string, $w_{1..n}$, previous to w_1 (resp. following w_n). We generate from $\mathcal{G}$ a *numbered minimal acyclic finite automaton* for the language $\mathcal{L}(\mathcal{G})$. In practice, we choose a device [8] generated by GALENA [9]. A *finite automaton* (FA) is a 5-tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_f)$ where: $\mathcal{Q}$ is the set of states, Σ the set of input symbols, δ is a function of $\mathcal{Q} \times \Sigma$ into $2^{\mathcal{Q}}$ defining the transitions of the automaton, q_0 the initial state and $\mathcal{Q}_f$ the set of final states. We denote $\delta(q, a)$ by $q.a$, and we say that $\mathcal{A}$ is *deterministic* iff $\mid q.a \mid \leq 1,\ \forall q \in \mathcal{Q},\ a \in \Sigma$. The notation is transitive, $q.w_{1..n}$ denotes the state $(\overset{n-2}{\dots}\ (q.w_1)\ \overset{n-2}{\dots}).w_n$. As a consequence, w is *accepted* iff $q_0.w \in \mathcal{Q}_f$, that is, the *language accepted by* $\mathcal{A}$ is defined as $\mathcal{L}(\mathcal{A}) = \{w,\ \text{such that}\ q_0.w \in \mathcal{Q}_f\}$. An FA is *acyclic* when the underlying graph is. We define a *path in the* FA as a sequence of states $\rho = \{q_1, \dots, q_n\}$, such that $\forall i \in \{1, \dots, n-1\},\ \exists a_i \in \Sigma,\ q_i.a_i = q_{i+1}$.

We also apply a minimization process [10]. In this sense, we say that two states, p and q, are *equivalent* iff the FA with p as initial state and the one that starts in q recognize the same language. An FA is *minimal* iff no pair in $\mathcal{Q}$ is equivalent. Although the standard recognition is deterministic, the repair one could introduce non-determinism by exploring alternatives associated to possibly more than one recovery strategy. So, in order to get polynomial complexity, we avoid duplicating intermediate computations in the repair of $w_{1..n} \in \Sigma^+$, storing them in a table $\mathcal{I}$ of *items*, $\mathcal{I} = \{[q, i],\ q \in \mathcal{Q},\ i \in [1, n+1]\}$, where $[q, i]$ looks for the suffix $w_{i..n}$ to be analyzed from $q \in \mathcal{Q}$.

Our description uses *parsing schemata* [11], a triple $\langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$, with $\mathcal{H} = \{[a, i],\ a = w_i\}$ an initial set of items called *hypothesis* that encodes the word to be recognized[1], and $\mathcal{D}$ a set of *deduction steps* that allow items to be derived from previous ones. These are of the form $\{\eta_1, \dots, \eta_k \vdash \xi \,/\, conds\}$, meaning that if all antecedents η_i are present and the conditions *conds* are satisfied, then the consequent ξ is generated. In our case, $\mathcal{D} = \mathcal{D}^{\text{Init}} \cup \mathcal{D}^{\text{Shift}}$, where:

$$\mathcal{D}^{\text{Init}} = \{\vdash [q_0, 1]\} \qquad \mathcal{D}^{\text{Shift}} = \{[p, i] \vdash [q, i+1] \,/\, \exists [a, i] \in \mathcal{H},\ q = p.a\}$$

We associate a set of items S_p^w, called *itemset*, to each $p \in \mathcal{Q}$; and apply these deduction steps until no new item is generated. The word is recognized iff a *final item* $[q_f, n+1],\ q_f \in \mathcal{Q}_f$ has been generated. We can assume that $\mathcal{Q}_f = \{q_f\}$, and that there is only one transition from (resp. to) q_0 (resp. q_f). To get this, it is sufficient to augment the original FA with two states which become the new initial and final states, and are linked to the original ones through empty transitions, our only concession to the notion of minimal FA.

3 Spelling Correction

We talk about an *error* in a word to mean the difference between what was intended and what actually appears, and we call *point of error* the point at

[1] A word $w_{1\dots n} \in \Sigma^+$, $n \geq 1$ is represented by $\{[w_1, 1],\ [w_2, 2],\ \dots\ ,\ [w_n, n]\}$.

which that difference occurs. So, a *repair* should be understood as a modification allowing the recognizer both to recover the process and to avoid cascaded errors, that is, errors precipitated by a previous erroneous repair diagnosis. This is the goal of *regional repairs* [7], which we succinctly remember now.

Working on acyclic FAs, we define an order relation $p < q$, with $p, q \in \mathcal{Q}$ iff a path exists in the FA from p to q. A pair of states (p, q) is a *region*, $\mathcal{R}_p^q$, in the FA when it defines a sub-automaton with initial (resp. final) state in p (resp. q). So, we say that a state $r \in \mathcal{R}_p^q$ iff there exists a path ρ in $\mathcal{R}_p^q$, such that $r \in \rho,\ r \neq p, q$. Given $r \in \mathcal{Q}$, it can be proved that there is only one *minimal region*, $\mathcal{M}(r)$, in the FA containing it.

To begin with, we assume that we are dealing with the first error detected. We extend the structure of items, as a pair $[p, i]$, with an error counter e; resulting in a new structure $[p, i, e]$. Given a *point of error* w_j, the associated *point of detection* is the initial state of the minimal region, $\mathcal{M}(w_j) = \mathcal{R}_p^q$, containing w_j. Associated to the point of error (resp. detection) w_j (resp. w_i), we consider the corresponding *error* (resp. *detection*) *item* $[q, j, \_]$ (resp. $[p, i, \_]$). To filter out undesirable repairs, we introduce criteria to select those with minimal cost. For each $a, b \in \Sigma$ we assume insert, $I(a)$; delete, $D(a)$, replace, $R(a, b)$, and transpose, $T(a, b)$, costs. We apply, from the detection item, the deduction steps $\mathcal{D}_{\text{error}} = \mathcal{D}^{\text{Shift}} \cup \mathcal{D}_{\text{error}}^{\text{Insert}} \cup \mathcal{D}_{\text{error}}^{\text{Delete}} \cup \mathcal{D}_{\text{error}}^{\text{Replace}} \cup \mathcal{D}_{\text{error}}^{\text{Transpose}}$, defined as follows:

$$\mathcal{D}^{\text{Shift}} = \{[r, l, e] \vdash [s, l+1, e],\ \exists [a, l] \in \mathcal{H},\ s = r.a\}$$

$$\mathcal{D}_{\text{error}}^{\text{Insert}} = \{[r, l, e] \vdash [r, l+1, e + I(a)]\}$$

$$\mathcal{D}_{\text{error}}^{\text{Delete}} = \{[r, l, e] \vdash [s, l, e + D(w_l)] \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}) = \mathcal{R}_{q_s}^{q_d} \\ r.w_l = s \in \mathcal{R}_{q_s}^{q_d} \textbf{ or } s = q_d \end{array} \}$$

$$\mathcal{D}_{\text{error}}^{\text{Replace}} = \{[r, l, e] \vdash [s, l+1, e + R(w_l, a)], \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}) = \mathcal{R}_{q_s}^{q_d} \\ r.a = s \in \mathcal{R}_{q_s}^{q_d} \textbf{ or } s = q_d \end{array} \}$$

$$\mathcal{D}_{\text{error}}^{\text{Transpose}} = \{[r, l, e] \vdash [s, l+2, e + T(w_l, w_{l+1})] \Big/ \begin{array}{l} \mathcal{M}(q_0.w_{1..j}) = \mathcal{R}_{q_s}^{q_d} \\ r.w_{l+1}.w_l = s \in \mathcal{R}_{q_s}^{q_d} \textbf{ or } s = q_d \end{array} \}$$

where $w_{1..j}$ looks for the current point of error. We also redefine $\mathcal{D}^{\text{Init}}$ as $\{\vdash [q_0, 1, 0]\}$. In any case, the error hypotheses apply on transitions behind the repair region. The process continues until a repair covers that region, accepting a character in the remaining string. When no repair is possible, the process extends to the next region, taking the final state of the previous one as the new point of error. We apply a principle of optimization, saving only those items with minimal counters.

When the current error is not the first one, we can modify a previous repair in order to avoid cascaded errors, by adding the cost of the new error hypotheses to profit from the experience gained from previous ones. This allows us to get a quality close to global methods [7], with a time complexity, in the worst case

$$\mathcal{O}(\frac{n!}{\tau! * (n-\tau)!} * (n+\tau) * 2^{\tau} * \text{fan-out}_{\mu}^{\tau})$$

where τ and *fan-out*$_\mu$ are, respectively, the maximal error counter computed and the maximal fan-out of the automaton in the scope of the repairs considered. The input string is recognized iff a final item $[q_f, n+1, e],\ q_f \in \mathcal{Q}_f$, is generated.

4 Spelling Incomplete Strings

In order to handle incomplete strings, we extend the input alphabet by introducing two new symbols. So, "?" stands for one unknown character, and "$*$" stands for an unknown sequence of input characters. Once the underlying FA detects that the next input symbol to be shifted is one of these two extra symbols, we apply the following set of deduction steps, $\mathcal{D}_{\text{incomplete}}$:

$$\begin{array}{c}\mathcal{D}^{\text{Shift}}_{\text{incomplete}} = \{[p,i,e] \vdash [q,i+1,e+I(a)] \;/\; \exists\, [?,i] \in \mathcal{H},\ q = p.a\} \\ \mathcal{D}^{\text{Loop\_shift}}_{\text{incomplete}} = \{[p,i,e] \vdash [q,i,e+I(a)] \;/\; \exists\, [*,i] \in \mathcal{H},\ q = p.a,\ \nexists\, q.w_{i+1}\} \\ \mathcal{D}^{\text{Loop\_shift\_end}}_{\text{incomplete}} = \{[p,i,e] \vdash [q,i+1,e+I(a)] \;/\; \exists\, [*,i] \in \mathcal{H},\ q = p.a,\ \exists\, q.w_{i+1}\}\end{array}$$

where $I(a)$ is the insertion cost for $a \in \Sigma$. From an intuitive point of view, $\mathcal{D}^{\text{Shift}}_{\text{incomplete}}$ applies any shift transition independently of the current lookahead available, provided that this transition is applicable with respect to the FA configuration and that the next input symbol is an unknown character. In relation to $\mathcal{D}^{\text{Loop\_shift}}_{\text{incomplete}}$, it simulates shift actions on items corresponding to FA configurations for which the next input symbol denotes an unknown sequence of characters, when no standard shift action links up to the right-context. Given that in this latter case new items are created in the same itemset, these transitions may be applied any number of times to the same computation thread, without scanning the input string. These deduction steps are applied until a recognition branch links up to the right-context by using a shift action, resuming the standard recognition mode, as it is described by $\mathcal{D}^{\text{Loop\_shift\_end}}_{\text{incomplete}}$.

In this manner, when we deal with sequences of unknown characters, we can examine different paths in the FA resolving the same "$*$" symbol. Although this could be useful for subsequent syntactic or semantic processing, an uncontrolled over-generation is not of practical interest in most cases. We solve this by tabulating the number of characters used to rebuild the word, using the error counter, and applying the principle of optimization. These steps are applied until new items cannot be generated. The time bound is, also, in the worst case,

$$\mathcal{O}(\frac{n!}{\tau! * (n-\tau)!} * (n+\tau) * 2^{\tau} * \text{fan-out}^{\tau}_{\mu})$$

The correction is defined by a final item $[q_f, n+1, e],\ q_f \in \mathcal{Q}_f$.

5 The Robust Frame

We are now ready to introduce the robust construction. We must now guarantee the capacity to recover the recognizer from any unexpected situation derived either from gaps in the scanner or from errors. To deal with this, it is sufficient to combine the rules previously introduced. More exactly, we have that the new set of deduction steps, $\mathcal{D}_{\text{robust}}$, is given by:

$$\begin{array}{rl}\mathcal{D}_{\text{robust}} = & \mathcal{D}^{\text{Init}} \cup \mathcal{D}^{\text{Shift}} \cup \mathcal{D}^{\text{Insert}}_{\text{error}} \cup \mathcal{D}^{\text{Delete}}_{\text{error}} \cup \mathcal{D}^{\text{Replace}}_{\text{error}} \cup \\ & \mathcal{D}^{\text{Transpose}}_{\text{error}} \cup \mathcal{D}^{\text{Shift}}_{\text{incomplete}} \cup \mathcal{D}^{\text{Loop\_shift}}_{\text{incomplete}} \cup \mathcal{D}^{\text{Loop\_shift\_end}}_{\text{incomplete}}\end{array}$$

where there is no overlapping between the deduction subsets. The final robust recognizer also has a time complexity, in the worst case

$$\mathcal{O}(\frac{n!}{\tau! * (n-\tau)!} * (n+\tau) * 2^{\tau} * \text{fan-out}_{\mu}^{\tau})$$

with respect to the length n of the ill-formed sentence. The input string is recognized iff a final item $[q_f, n+1, e],\ q_f \in \mathcal{Q}_f$, is generated.

6 Pruning Strategies

In dealing with spelling correction, ill-formed expressions can often be resolved in different manners, which forces us to consider a framework involving ambiguities. Although most of these ambiguities will be eliminated in subsequent and more sophisticated analysis tasks, a number of them can already be treated at this stage. Disregarding pure statistical aspects, we focus on the formalization of cut-off schemata in order to limit the repair space and, as a consequence, reduce the computational impact derived from exploring useless repair paths.

Nevertheless, the interpretation of an FA as a sequential transitional formalism imposes an essential guideline on the design of any pruning strategy. If we also take into account that the dynamic frame previously defined updates error counters at each new item generation, it appears that pruning techniques based on threshold error criteria seem to be particularly well adapted. So, we can consider a set of simple cut-off schemata, combining the repair hypotheses in order to allow the user to implement human-like correction strategies.

6.1 Path-Based Pruning

We refer here to a classic technique [5, 12] consisting of pruning repair branches on items with an error below a given threshold. From an operational viewpoint, the consideration of this pruning mechanism does not require any modification in the item structure, and we must only apply a test on the error counter each time a new item is generated. If the counter computed is greater than the defined threshold, ρ, we simply prune the parse process on the corresponding branch by stopping any action on that item. So, we can only take into account what is now in our parse scheme:

$$\forall I \vdash [p, i, e] \in \mathcal{D}_{\text{robust}},\ e < \rho$$

As an example, considering the discrete metric assigning a unitary cost to each repair deduction step in robust mode, we could cut-off all repair branches with an error counter higher than a fixed proportion on the length of the word.

6.2 Sequence-Based Pruning

Another possible approach is to limit the number of consecutive errors included in a path, pruning them on items in these sequences with a quality below a given threshold, σ. In order to implement this pruning strategy, we must first introduce

an additional error counter, e_l, representing the local error count accumulated along a sequence of repair hypotheses in the path we are now exploring. So, items take the new structure $[p, i, e_g, e_l]$, where the error counter e_g is the same as that considered in the original robust algorithm. At this point, we re-define the following deduction steps from the original scheme for the robust mode:

$$\mathcal{D}^{\text{Shift}} = \{[p, i, e_g, e_l] \vdash [q, i+1, e_g, 0],\ \exists [a, i] \in \mathcal{H},\ q = p.a\}$$

which implies that each time a shift action is performed, a sequence of possible repair hypotheses is broken and, as a consequence, no sequence-based pruning can be considered in that case. At this point, all that remains is to test that no sequence of deduction steps in $\mathcal{D}_{\text{robust}}$ exceeds the threshold σ. So, we have that the complete previous deduction step

$$[p, i, e_g] \vdash [q, j, e_g + \triangle] \in \mathcal{D}_{\text{robust}}$$

is now replaced by another one of the form

$$[p, i, e_g, e_l] \vdash [q, j, e_g + \triangle, e_l + \triangle] \in \mathcal{D}_{\text{robust}},\ e_l + \triangle < \sigma$$

So, for example, we could contemplate cutting off any branch including a sequence of repair hypotheses.

6.3 Type-Based Pruning

Sometimes we may be more interested in detecting the presence of some particular hypotheses in a path of the FA or even in a sequence of this path. This translates into applying the previous path and sequence based approaches to a particular kind of deduction hypotheses. Taking, for example, the case of $\mathcal{D}^{\text{Insert}}_{\text{robust}}$ and assuming a threshold τ to locate the pruning action on a path, we have that the new deduction steps are now:

$$\forall I \vdash [p, i, e_g, e_l] \in \mathcal{D}^{\text{Insert}}_{\text{robust}},\ e_g < \tau$$

and, if we deal with a sequence on a path, we have that:

$$[p, i, e_g, e_l] \vdash [q, j, e_g + \triangle, e_l + \triangle] \in \mathcal{D}^{\text{Insert}}_{\text{robust}},\ e_l + \triangle < \tau$$

assuming that standard shift actions re-initialize to zero local counters. However, we need a pair of counters associated to each kind of deduction steps in order to consider type-based pruning for insert, delete, replace or transpose hypotheses. As an example, we could cut-off any branch considering more than two delete hypotheses in the same branch.

7 Experimental Results

We consider a lexicon for Spanish built from GALENA [9], which includes 514,781 different words, to illustrate this aspect. The lexicon is recognized by an FA containing 58,170 states connected by 153,599 transitions, of sufficient size to

allow us to consider it as a representative starting point for our purposes. In order to take the edit distance [5] as the error metric for measuring the quality of a repair, it is sufficient to consider discrete costs $I(a) = D(a) = 1,\ \forall a \in \Sigma$ and $R(a,b) = T(a,b) = 1,\ \forall a,b \in \Sigma,\ a \neq b$. In particular, this choice will allows us to compare our proposal with the original conditions for Savary's one [12].

Our goal is now to illustrate the robustness in a variety of situations. We look for a set of tests that will show both the effects from the topological distribution of errors and unknown sequences in the input string and, whenever possible, the structural influence of the operational kernel in the recognition process. Three different kinds of patterns are considered for modeling ill-formed input strings.

The former, which we call *unknown*, is given by words which do not include spelling errors, but only unknown symbols. This, for example, is the case of the ill-formed word `agu*teis`. Taking a path-threshold 2, the completion is `aguasteis` (*you watered*). The second kind of pattern, which we call *error-correction*, gathers words including only errors. For the error input `augasteis` with path-threshold 2, the correction is `aguasteis`. The third pattern, which we call *overlapping*, groups words combining both unknown symbols and spelling errors. In the case of `aga*teis` with path-threshold 2, the repair is `aguasteis`,

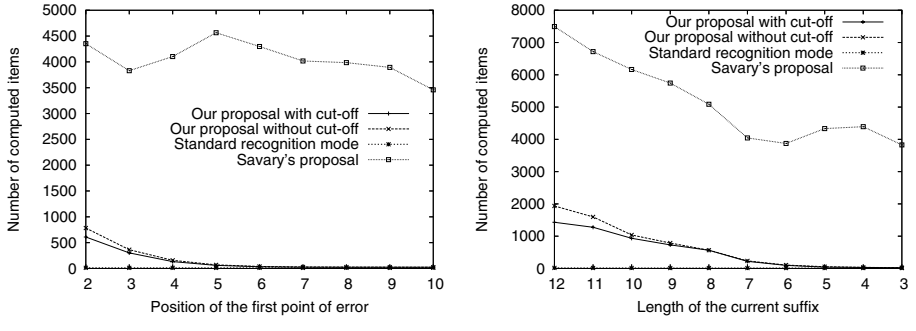

Fig. 1. Items generated for the *unknown* example

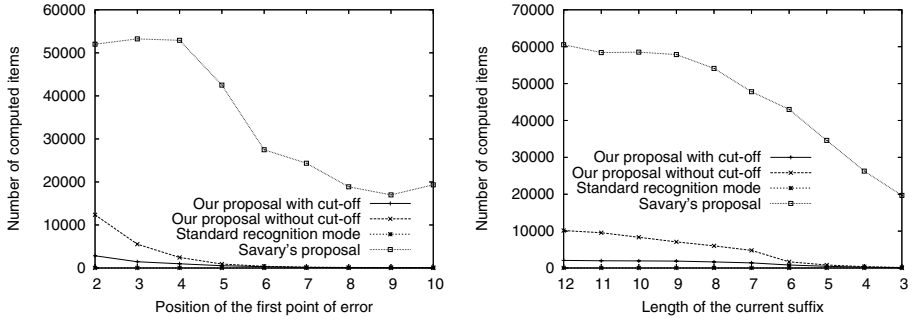

Fig. 2. Items generated for the *error-correction* example

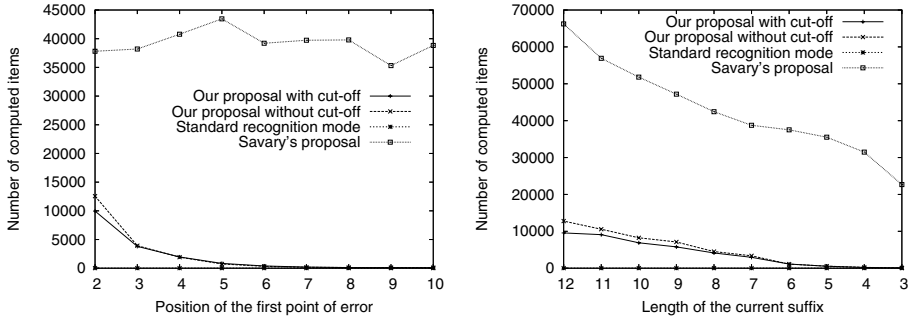

Fig. 3. Items generated for the *overlapping* example

which is generated by rebuilding the unknown sequence "$*$" with ''s'' and, later, re-taking the error mode to insert ''u'' before the second ''a''.

The results are shown, for the *unknown*, *error-correction* and *overlapping* examples in Figs. 1, 2 and 3; respectively. In all cases, we have started from the same sample of words, which has the same distribution observed in the original lexicon in terms of lengths of the strings dealt with. On these words and for each length category, we have randomly generated errors and unknown sequences in a number and position in the input string. This is of some importance since the efficiency of previous proposals depends on these factors [5, 12]. No other morphological dependencies have been detected.

In relation to the pruning criteria chosen, we consider a specific one for each example. So, in the *unknown* case, path and sequence thresholds are 3. Type ones are only considered for delete hypothesis and also fixed to 3. For the *error correction* example, path and sequence thresholds are, respectively, 3 and 2. Here, type ones are considered for all error hypotheses and fixed to 1. In the overlapping case, path and sequence thresholds are 4; and type ones are also fixed for all error hypotheses. In dealing with deletions it has a value of 3, and in the case of insertion, replacement and transposition its value is 1.

The number of items generated by the system during the robust recognition process has been taken as the reference for appreciating the efficiency of our method, rather than purely temporal criteria, which are more dependent on its implementation. These items are measured in relation to both the position of the first point at which a difference which was attended to by the user occurs in the word and the length of the suffix from it. So, we are sure to take into account the degree of penetration in the FA at that point, which determines the effectiveness of the repair strategy since it influences the number of repair schemata to follow. In particular, in our proposal, it determines the number of regions in the FA including the point of error or the first unknown point and, as a consequence, the possibility of considering a non-global resolution.

In each figure, we compare four different graphs corresponding to our pruning proposal, the results without cut-off, the Savary's approach [12] with the

same path-threshold of our pruning schemata and, finally, the number of items that would be generated in standard recognition mode if we had considered the correct input string from which we have obtained the erroneous one analyzed by the previous three graphs corresponding to robust techniques. So, we can estimate the computational behavior of the different robust techniques considered, but we can also to illustrate the computational effort exclusively due to the application of the robust mechanisms in each case. In relation to the Savary's proposal, the original algorithm allows to consider path-based pruning and, in order to introduce unknown symbols, we have simply extended it by simulating insertions.

These results show a noticeable improvement in computational complexity due to the consideration of pruning techniques. Here, it is important to remember that errors and unknown sequences were randomly generated and therefore we have not profited from any linguistic knowledge in order to design efficient pruning criteria. In spite of this apparent lack of performance, the application of these cut-off techniques has augmented the precision [2] by 4'06% for the *unknown* example, 7'76% for the *error correction* one, and 1'95% for the *overlapping* case. In relation to this, although the errors in our tests have been randomly generated, we must remember that we have fixed the original, and correct, corpus. As a consequence, we can easily estimate this parameter.

The graphs corresponding to the standard recognition mode illustrate the complexity of the robust strategy. This is due, essentially, to the number of FA paths to be explored, which also explains the greater amount of items generated when the point of origin for the application of the robust mode is close to the beginning of the word. Finally, comparison with the Savary's method, in the best of our knowledge the most efficient proposal on spelling correction, seems to put into evidence the validity of our approach from the point of view of the efficiency.

8 Conclusions

The gap with human performance in spelling correction, which is mainly due to the mismatch between what was in the text, what actually appears in the input and the set of available dictionary entries, should be covered by a flexible and robust strategy at various levels.

A robust model for non-contextual spelling correction is described, which allows the algorithm to deal with distortions caused by incomplete string acquisition, simulating human performance in non-contextual word recognition. Our goal is to compensate the noise effect resulting from ill-formed word recognition, in order to avoid degradation in the performance of the recognizer. The consideration of cut-off criteria provides the capability to control the correction mechanisms, conducting the process through the nearest neighbors of a given character string in a dictionary.

[2] The rate reflecting when the algorithm provides the repair attended by the user.

References

1. Peterson, J.: Computer Programs For Spelling Correction. Springer-Verlag, Inc., Berlin, Germany / Heidelberg, Germany / London, UK / etc. (1980)
2. Kukich, K.: Techniques for automatically correcting words in text. ACM Computing Surveys **24** (1992) 377–439
3. Agirre, E., Gojenola, K., Sarasola, K., Voutilainen, A.: Towards a single proposal in spelling correction. In Boitet, C., Whitelock, P., eds.: Proc. of the 36th Annual Meeting of the ACL, San Francisco, California, Association for Computational Linguistics, Morgan Kaufmann Publishers (1998) 22–28
4. Elmi, M., Evens, M.: Spelling correction using context. In Boitet, C., Whitelock, P., eds.: Proc. of the 36th Annual Meeting of the ACL, San Francisco, California, Association for Computational Linguistics, Morgan Kaufmann Publishers (1998) 360–364
5. Oflazer, K.: Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. Computational Linguistics **22** (1996) 73–89
6. Du, M., Chang, S.: A model and a fast algorithm for multiple errors spelling correction. Acta Informatica **29** (1992) 281–302
7. Vilares, M., Otero, J., Graña, J.: Regional finite-state error repair. Lecture Notes in Computer Science **3317** (2005) 269–280
8. Lucchesi, C., Kowaltowski, T.: Applications of finite automata representing large vocabularies. Software-Practice and Experience **23** (1993) 15–30
9. Graña, J., Barcala, F., Alonso, M.: Compilation methods of minimal acyclic automata for large dictionaries. Lecture Notes in Computer Science **2494** (2002) 135–148
10. Daciuk, J., Mihov, S., Watson, B., Watson, R.: Incremental construction of minimal acyclic finite-state automata. Computational Linguistics **26** (2000) 3–16
11. Sikkel, K.: Parsing Schemata. PhD thesis, Univ. of Twente, The Netherlands (1993)
12. Savary, A.: Typographical nearest-neighbor search in a finite-state lexicon and its application to spelling correction. Lecture Notes in Computer Science **2494** (2001) 251–260