Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 403 – 411

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

HASCH: High Performance Automatic Spell Checker for Portuguese Texts from the Web

G. Andrade[a], F. Teixeira[a], C. R. Xavier[a,b], R. S.Oliveira[a], L. C. Rocha[a], A. G. Evsukoff[b]

[a]*DCOMP/UFSJ - São João del-Rei, MG , Brasil*
[b]*COPPE/UFRJ- Rio de Janeiro, RJ , Brasil*

Abstract

The rise of the Web 2.0 caused a real democratization in the context of data generation. These data are mostly provided in the form of texts, ranging from the reports provided by news portals, using a formal language, to comments in blog and micro-blogging applications that abuse the use of an informal language. Address this heterogeneity is an essential preprocessing so that these data can be used by tools that aim to infer accurate information based on such data. Thus, this work presents the *HASCH* (**H**igh Performance **A**utomatic **S**pell **CH**Ecker), whose objective is to correct spelling in Portuguese texts collected from the Web. Being a tool that aims to handle a large volume of data, *HASCH* is completely parallelized in shared memory. In our evaluation, we found that the *HASCH* was extremely effective in the correction of very large texts from different Web sources, with a almost superlinear speedup.

Keywords: automatic spell checker, parallel algorithm, openMP, text processing

1. Introduction

With the advent of Web 2.0, we see a real democratization of data generation. Lot of tools have been developed allowing anyone with Internet access to publish and distribute data with a speed never seen before. This democratization, coupled with the lack of control on what is available on the Web, allows the generation of massive data. Most of the data are provided as a texts, range from reports published by several news portals to comments added into blog and micro-blogging applications (e.g. Twitter [1]). While the news are usually written by experts and properly revised, comments added in micro-blogging applications are made by users with different levels of schooling, who use many abbreviations and symbols, an informal language, here called Web Language. These differences mean that the quality of data is very heterogeneous, where quality is related to syntactic and semantic aspects of the data. The quality of data is a critical aspect to be considered by tools that aim to infer accurate information based on these data, such as data mining, recommender systems, etc.

A scenario that demonstrates the importance of this aspect is related to the use of reviews posted on the micro-blogging applications by large corporations in strategic decision making. In these applications, millions of users share opinions about different topics, such as products, personalities, among others. Since it is a very informal space, with no quality control of the content that is published, it is essential to consider the data's quality.

[1] http://www.twitter.com

doi:10.1016/j.procs.2012.04.043

An important set of tools that precedes any other task of treatment of these texts are the spell checkers [1]. The spell check can be divided into two main subproblems: the error verification and the correction of errors found. The first one is simply to determine whether the word belongs to the target language. The process may involve a morphological analysis of the word and statistical approaches, or just check the existence of this word in a dictionary that represents almost completely the target language. The second subproblem, the correction itself, is done by creating a list of similar words to the wrong word, i.e., a list of candidates words. Moreover, the spell checkers can perform the corrections following two approaches: interactive and non interactive. The interactive spell checkers list a set of candidates words and wait for interaction of the users to select the most appropriate for replace the wrong word. By contrast, the non interative spell checkers, which are fully automatic, check and correct the wrong words by applying different heuristics [2].

In this paper we propose a fully automatic spell checker to be used at the end of automatic collectors for Portuguese Web text (tweets, news, etc..), the *HASCH* (**H**igh performance **A**utomatic **S**pell **CH**ecker). The aim of this corrector is to preprocess the collected texts, addressing the issue of quality, so that they can be effectively used by tools that aim to infer information from the collected texts. The *Hasch* is based on correction heuristics well known in more general contexts [1, 2, 3], and other approaches to treat the specific problem of correcting "Web Language" as well to address the peculiarities of the Portuguese language, i.e. the accentuation. Moreover, as a tool aimed at Web, intended to treat a large volume of data, *HASCH* is completely parallelized in shared memory in order to minimize the processing runtime. We evaluated the *HASCH* both from the perspective of effectiveness (quality of the corrections made) and efficiency (runtime) using various texts collected from different sources. The experimental results showed *HASCH* was extremely effective in the correction of texts from different sources of the Web, showing an almost linear speedup in the processing of very large texts.

2. Related Work

The spell checking by automated tools has a long history in computing. Researchers have been writing programs for the detection and correction of spelling errors for over 30 years [3, 4, 5]. The researches on automatic text correction are usually divided between methods for the detection of errors and methods for correction suggestions. The most common method for detecting spelling errors in a given text is to check if the word belongs to a particular dictionary, which should represent well the language in question.

Storing a dictionary in memory can be extremely expensive computationally. Thus, some studies suggest the maintenance of this dictionary in the form of a bitmap [6, 7]. In addition, there are proposals for correctors that do not use complete dictionaries [6], or even none dictionary structure [8, 9]. The work presented in [6], shows a way to keep only the root of the word to save space, and shows strategies for removing prefixes and suffixes for the verification in this simplified dictionary, a technique called "affix- stripping". In [8] the authors present a technique based on tabulation of trigrams (three -letter sequence). The idea is that certain trigrams occur more often than others and some never appear. Thus, the text to be corrected is divided into trigrams which are compared with the created table. If a trigram can not be found in the table, it probably represents a misspelling. A similar idea appears in [9], in which the text is divided in trigrams, but in the creation of the table, the frequency that each trigram appears somewhere in the text is also stored. In our spell checker we use a complete dictionary with 261,945 words in the Portuguese language, including words with prefixes and suffixes, thus comprising most of the vocabulary of the Portuguese language.

A spell checker can fail when it identifies a word as an error and that word, in fact, is correct, or when it fails to point an true error. Leaving pointing an error is the most worrying failure, since the user is not sure if the text is really free from errors. This type of failure can happen when the spelling errors coincide with words in dictionaries and is known as "real-word" error. As shown in [6], a set of about four thousand errors taken from the writing of secondary school students, forty percent were "real-word" errors. Although most spell-checkers do not apply any techniques for the correction of "real-word" errors, there are some works that try to address this problem. The first is a system called CRITIQUE, developed by IBM in [10]. The second is a variant of a system, developed at the University of Lancaster, to tag words in a text with their syllables [11, 12]. Following this line, an interesting study, also presented by IBM researchers [13], which is based on probabilities of words occurrence in very large sets of text, with the aim of correcting the writing of words based on this probability. In our study we used a strategy based on this idea of probabilities. Through two calibration steps (one prior to the correction of the text and another in the course of

correction), we determine the words that most occur and we give to those words the greatest weight in the process of correction, according to the specific style of the collected text, aiming minimize "real-word" errors.

The spell check itself often does not completely satisfy the needs of some applications, requiring the suggestion of possible fixes. Therefore, various techniques and recognition forms have been studied to make possible listing the candidates to replace the word pointed out as wrong. We can calculate, for example, how close a word is from the wrong word, as shown in [14]. Probability is a very common strategy for the calculation of similarity, and one of these methods is described in [15], and is also the basis of an approach developed at Bell Labs for correcting typing errors [16, 2]. Although our spell checker is not interactive, we adapted these probabilistic techniques to replace the misspelled word with one that is as appropriate as possible. To do so, we use the edit distance to calculate of how far the candidates word is from the wrong word.

As a semantic analysis is computationally infeasible, studies to increase the quality of checks and corrections of words remain focused on spelling level, as we saw in the work described above. Unlike these, the spelling checker presented in this paper aims not to propose new techniques but adapt the various studies presented above in order to apply it in a different scenario: Portuguese text collectors from the Web. In this scenario, it is necessary that the spell checker is fully automatic, able of correcting not only spelling, but also to infer words from abbreviations, dealing with the peculiarities associated with the Portuguese language. Moreover, the spell checker needs to be extremely fast and capable of processing a large volume of data in a short space of time.

3. HASCH

The problem of detecting and correcting errors in written texts involves analysis at different levels. The first level, spelling, treats words as isolated objects, regardless of its use in the text. Above, we have the grammatical level, which verifies the agreement, punctuation and use of prepositions. Finally, sentences that are grammatically correct, but for reasons of style need adjustments [17]. This last level, stylistic, tries to make the grammatical construction easier or reduce the size of the sentences in order to make the text more readable.

The great challenge of a fully automatic spell checker is, given a wrong word, find the most probable correction for it. We know that this is not possible just with a spelling-level analysis, i.e. the corrector will not be able to find exactly what is the correct choice for replacement in all cases since, even spelled correctly, a word can insert grammatical or stylistic errors. In addition, as a tool that aims the Web, which goal is treat a large volume of data, another challenge of our application is related to the performance. So in this section we describe our spell checker for Portuguese Web text, the *HASCH* (**H**igh Performance **A**utomatic **S**pell **CH**ecker) that aims to address the challenges proposed above. All implementation was done in the C programming language, using the language standard libraries and completely parallelized using the OpenMP library.

3.1. HASCH Operation

As we mentioned, the focus of our spell checker is work on the tip of Web text collectors. These collected data are full of language habits, here called "Web Language", and disrespect every spelling rule of the Portuguese language. Thus, our spell checker is based on two dictionaries: (1) the principal, containing over 261,945 words used in the brasilian Portuguese, including words with the prefixes and suffixes (inflected words), trying to encompass most of the vocabulary of the Portuguese language; and (2) dictionary of "Web Language", which gives us a list of language vices and their correct form. Both dictionaries are implemented using a hash table so that the cost of search is the lowest possible ($O(1)$).

After the building process of the main dictionary, we propose two strategies to weigh the words according to the specific style of the text to be collected. Words of higher weight will be more likely to be used in the correction process. This weighting aims to minimize "real-word" errors described in Section 2.

The first strategy, called **External Calibration** can be defined as a pre-processing of the automatic spell checker. Different spelling patterns are observed in different types of text. For example, while in texts coming from news sites are used more formal and sophisticated words, in blogs and microblogs we observe the predominance of a simpler and less formal spelling. Thus, in this strategy, the collector reads several training texts of the same style of the texts to be corrected, and counts the occurrence of all words in these training texts, filling the hash table that stores the main dictionary before the correction start. In the second strategy, called **Internal Calibration**, the word count is done

during the execution of the spell checker, as suggested by [1]. As the words are read from the text to be corrected, the occurrence of the word in the main dictionary is checked. When found, its value in the hash table that stores the dictionary is incremented.

Having established the strategies for the dictionaries construction, as well as the weight of the main dictionary, the next step is to effectively describe the operation of our spell checker. *HASCH* is based on four main steps described below:

1. **Query Main Dictionary**: For each word w obtained from the text to be corrected, our application checks whether it is present in the main dictionary at $O(1)$ cost. Finding the word in the dictionary means that the word is correct, and nothing needs to be done. If the word is not found in the main dictionary the step 2 is triggered;
2. **Query "Web Language" Dictionary**: For each word w obtained from the text to be corrected, our application checks whether it is present in the "Web Language" dictionary at $O(1)$ cost. Case the word is found, its replacement is done by the correctly word stored. This strategy ensures that errors from language habits are corrected without the need to pass through other correction strategies, thus making our spell checker faster. If the search foe w fail, step 3 is triggered;
3. **Accentuation Check:** A key issue in the Portuguese language is the use of accents. The lack of accentuation or the inadequate accentuation represent a large number of spelling errors, since it is a small detail that involves many rules. Thus, this spell checker step consists in accentuate the each vowel of w and checks, for every possibility, whether it is in the main dictionary. The cost of this phase is $O(v)$, where v is the number of vowels of w. Considering that the size of w equals to n, we have that , in the worst case, the cost of this step is $O(n)$. Like the previous step, this strategy aims to prevent the execution of the next steps, thus making our checker faster. A failure in this step triggers step 4;
4. **Edit Distance Check**: As previously described, this step is triggered only when the other previous steps failed to correct w. To determine the correct word c to replace w, we first raise the possible options. For this, we propose a strategy based on the proposal in [1]. Given a word w to be corrected, we consider a list of words that are orthographically close to it. This proximity is given by the edit distance, i.e., the number of modifications required to turn w into another word. These edits can be of four types: (1) deletion: when we remove one letter of w, (2) insert: when we insert one letter in w, (3) substitution: when we change one letter of w for some letter of the alphabet, and finally (4) transposition: when we exchange of adjacent letters in w. By the literature 80% to 95% of the errors are in distance 1 and therefore for the automatic spell checker presented in this paper we consider only the words that are on edit distance 1 of w and are in the dictionary. It is also important to mention that this process is computationally expensive, since the complexity of the deletion is n, the transposition $n-1$, the replacing 26^n and the insert 26^{n+1}. Therefore, the full complexity of this heuristic is (26^n), n being the size of w. Among the selected options, we select to the correction the word c of higher rank in the dictionary, at a cost $O(n)$.

Correction of large text, with millions of words, is clearly a time consuming process, since the words are analyzed and corrected one by one over the presented steps. For each of these words the correction time is different. If we consider a correct word, it will have its analysis quite fast, since it is only a dictionary look up. But when we have words that are not correct it requires the application of one of the above steps to correct it. The application of these steps may also present different times, as the word can be corrected in step 2 (Query "Web Language" Dictionary), in step 3 (Accentuation Check) or fail in both, and only be corrected in step 4 (Edit Distance Check) which is clearly the most expensive. We can observe from the descriptions of the execution steps that our spell checker presents a complexity given by $T * max(O(1), O(n_{avg}), O(26^{n_{avg}}))$ where n_{avg} the average number of characters of the words to be corrected and T the total number of processed words. Thus, in the worst case, the complexity of our spell checker is given by an exponential function in relation to the average number of characters of the words to be corrected, which is computationally expensive. This analysis reinforces the argument that an efficient implementation is essential. Thus, we introduce in the next Section, the implementation of the adopted parallel strategy.

3.2. Parallelization

An interesting question to be considered in the correction of texts using the proposed automatic spell checker is the fact that it considers only the word level, i.e., the correction of a word is entirely independent of the correction

of the other words, making this process a embarrassingly parallel task. Given these considerations, it is clear that a data parallelization is easily applicable to the problem, in which the text to be corrected can be divided into different processes. Thus, in the implementation of our spell checker, we adopted the data parallelization using shared memory and the OpenMP library was used. Following, we present the details regarding our parallelization.

3.3. Data division

The division of the original text is made considering the number of words in it. Thus, the first step before the correction is to count how many words are in the text and then, for each thread, define how many words each one will receive. The word count is trivial and the division is made by two following formulas:

$$P_{begin} = id_{thread}(N/P) + min(id_{thread}, r)$$
$$P_{end} = (id_{thread} + 1)(N/P) + min(id_{thread}, r) - 1,$$

where N is the total number of words, P is the number of threads and r is given by $r = NmodP$. Thus, the amount of words that a thread will get given by $P_{end} - P_{begin}$.

Thus, each thread, through its *id*, will know in advance, the total number of words that it should process. These words are arranged in sequential blocks. In other words, the thread with id 0 will receive the first block, the with *id* 1 the second and so on.

3.4. Critical Regions

Each thread has its own section of the text and therefore all actions for correction are independent and one thread does not affect the execution and correction of another. The global structure shared between the threads is the "Web Language" dictionary, used only for reading, and the main dictionary containing the words' weights, which will be used both for reading and for writing (to increase occurrences of words). Therefore, the critical region is the update in the main dictionary weights and we implemented a simple mutual exclusion to solve the problem. The remaining processing and queries are local and other synchronization or mutual exclusion strategies were not necessary

3.5. Correction Synchronization

The data independence between threads is not complete, since we must maintain the order of the text, i.e., the piece of text corrected by thread 0 must come before the piece corrected by thread 1, and so on. To do so, we used a buffer that receive the corrected parts, concatenating the texts according the threads ids. If a thread, for example with $id2$, ends its correction before the thread with $id1$, it waits until the thread 1 finishes and writes the buffer and then, writes its corrected text. At the end of the correction process the buffer will have the same order as before the correction, which assures us that the separate execution of threads does not break the semantics of the text being corrected.

4. HASCH Evaluation

In this section we present and discuss the results for the evaluation of our automatic spell checker, the *HASCH*. Our evaluation was divided into two perspectives: (1) efficacy, related to the quality of the corrections, and (2) efficiency, related to the execution times of the spell checker.

In order to evaluate *HASCH* efficacy we used, basically, two different contexts: formal texts collected from news portals and informal texts collected from microblogs (Twitter). Based on these contexts, we created different evaluation scenarios, varying the application context and the texts used in the calibration. For example, text correction and calibration using the same context and correction and calibration using texts from different contexts. After the tests, we validated the spell checking manually, allowing a detailed analysis of the behavior of our spell checker, observing the treatment or not of each errors in the text, and individually evaluating each implemented step.

To evaluate the efficiency of our spell checker, we performed different tests, where the focus was the response time of the application. Therefore, we execute our spell checker in different scenarios where we varied the size of the input files and the number of threads (from 1 to 4), logging the real-time execution. The tests were run on a machine with Intel(R) Core(TM) i7 2.67Ghz and 8GB RAM. To measure the time, the spell checker was executed 20 times for each scenario (file size + number of threads) and the final time shown is the result of the average of the 20 executions. In the subsequent sections we detail the tests, analyze the results and describe the databases used.

PC	NC or EC	GE	UC	NF
157	64	45	7	5

Table 1: Overview of the Results Obtained for the Test Set Containing Twitter Posts

4.1. Databases

To perform the efficacy evaluation of our spell checker, we used two different test databases, collected from the Web: the first containing a total of 200 posts of different Twitter users and the second containing complete articles from a news portal, all related to "Economy". Each of the bases have approximately 3, 300 words to be evaluated and possibly corrected by *HASCH*. In addition to these two sets, we also collected other two sets, related to the same two subjects presented above, to perform the calibration process, each containing approximately 6, 000 words. Based on these two test bases and two calibration bases, we created several evaluation scenarios. Both test bases were processed using three different versions of the main dictionary, without calibration, calibrated with Twitter posts and calibrated with texts from news portals.

For evaluating the efficiency of the proposed spell checker, we used 10 test sets containing different amounts of posts collected from Twitter. The sets were constructed by varying the number of posts from 100 to 100,000. The total amount of words to be processed ranged from 1,852 to 1,211,881, according to each set.

4.2. Correction Efficacy

In the first evaluation we used a test set containing only Twitter posts and a main dictionary built without external calibration (preprocessing). Table 1 presents an overview of the results. The errors identified were classified into five classes: errors identified and properly corrected (PC); errors identified but not corrected (NC) or erroneously corrected (EC); grammatical errors (GE),that is, these errors should be corrected, but they were not because the words were in the dictionary; unnecessary corrections (UC); and errors whose expected word was not found in the main dictionary (NF). For this analysis, correction of hashtags, usernames, URLs, emoticons, names and expressions in foreign languages were disregarded.

As we can observe in Table 1 that even in a scenario where the frequency of errors is considerably high (microblogging), the spell checker worked well, correcting approximately 56% of the errors found. It is important to observe that a large part of the errors not corrected are in grammar level. This type of error is outside the correction scope of our tool, since the word in question, though inadequate to the context, is correctly spelled. This is the case of words like "esta" and "está", for example.

Although relatively low, the number of unnecessary corrections and errors in which the expected word was not found in the main dictionary shows that increasing the size of main and "Web Language" dictionaries could positively influence the operation of the spell checker. In the first case the word is not in the used dictionaries, and therefore was improperly identified as error and corrected. In the second case, the word is in fact incorrect and was identified as such, but there is no way to correct it properly, since the correct word does not exist in the used dictionaries. The number of uncorrected or erroneously corrected errors corresponds to about 23% of total errors. We can say that this value is relatively low, since we are dealing with a scenario where the concern about the writing is minimal, and the number of abbreviations and deliberate spelling errors is significantly high. Figure 1 presents more details regarding the operation of our spell checker in each execution step.

Regarding the edit distance verification step, the erroneously corrected errors were divided into two types: caused by score and caused by level. In the first case, the correct word could have been selected, but it was not because another word, with a higher score, was also found. In part, this is due to lack of an external calibration of the main dictionary. By using only the internal calibration the words less utilized have the same chance of being selected than those commonly used. In the second case, the so-called errors by level, occur when the edit distance of the word to be corrected in relation to expected one is very large, and therefore the heuristic can not reach it. Errors of this type are common in microblogs scenario because users have the unnecessary habit of replicating the vowels of a word. Thus, we have that the step that checks the edit distance of the words has the worst results when dealing with microblogging texts, correcting approximately 46% of the errors identified by it.

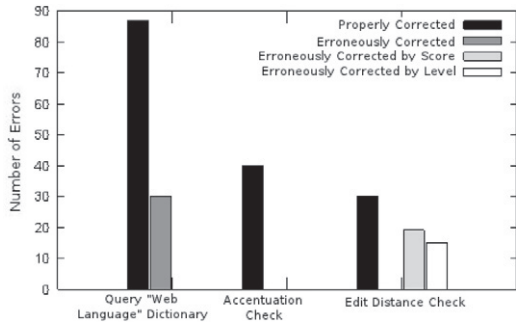

Figure 1: Evaluation of the Correction in Each Step

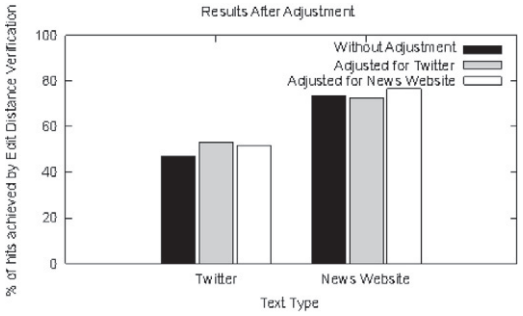

Figure 2: External Calibration Evaluation.

In order to evaluate the importance of the external calibration process, we compared the operation of our spell checker using the two mentioned test bases (Twitter and economy news) in three different scenarios: the first without calibration, the second calibrated with Twitter texts and the third calibrated with texts about economy. The results of this evaluation are presented in Figure 2.

We can observe that whenever the dictionary is calibrated in a manner consistent with the type of text to be treated, there is an increase in efficacy. When calibration is performed using different styles of text, there is a risk of loss of efficacy, even in relation to the dictionary without calibration. This case is shown in the graph in Figure 2, when we use a dictionary calibrated with Twitter texts to correct texts of news portals. This is due to the fact that the text styles are different: while one has a more casual language, the other has a formal language. Thus, inadequate words end up having a higher probability of being selected. Even with a small amount of training text in the external calibration we can note an important improvement in the quality of correction performed when the calibration is done properly. From these results we conclude that the use of a larger amount of text for external calibration of the dictionary can provide even better results.

Therefore, regarding the efficacy evaluation of *HASCH*, we have that the use an extra dictionary dedicated to "Web Language" proved to be an excellent strategy and that an expansion of this dictionary tends to lead to an even more precise spell checker. Another highlight was the strategy of checking the accents, which managed to rescue a fair amount of common errors in Portuguese (the lack of accents). Finally we can highlight the strategy of external calibration also showed very promising. The use of a larger and better training base can further increase the efficacy of our spell checker.

4.3. Parallelization Efficiency

In this section we present the results for the efficiency evaluation of*HASCH* for different input sets, with different sizes, varying the number of threads used. We summarize these results through the graph in Figure 3 in which we present the values of speedup obtained for the different test sets used and the ideal speedup.

Analyzing Figure 3 we can see that the parallelization is inefficient to deal with small sets of words. The speedup values for bases with less than 400,000 words (2,500 tweets) are insignificant (small reduction in execution time as

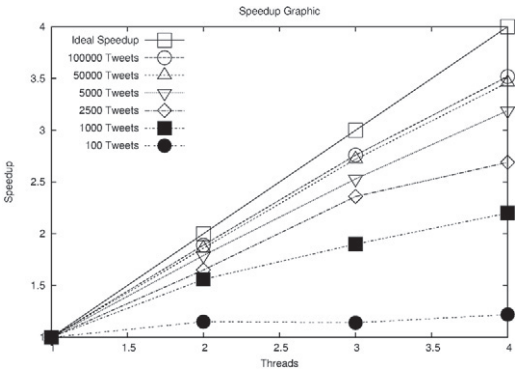

Figure 3: Speedup

we increase the number of threads), not justifying the use of more than one thread. This is due to the fact that the process of dividing the text to be corrected between the threads generates an overhead that ends up canceling the gain of the data parallelization, since, given the small number of words, the execution of algorithm, even in serial, is quite fast. However, when the size of the words set grows, the increase in the number of threads running simultaneously provides a reduction in the correction time. For test sets with more than 400,000 words (5000 tweets), the speedup achieved is close to the ideal speedup. This is because the time spent in the division process of the text keeps the same, but the task of correcting is divided equally between the threads, and executed in parallel. Finally, it is clear that, despite the value of speedup have stabilized over the increase in the test set, the gain is still significant when we increase the number of processors and deal with relatively large test sets. This fact indicates that, in these cases, the increasing running threads can continue providing a linear decrease in correction time.

5. Conclusions and Future Work

In this paper we propose a fully automatic spell checker to be used after the automatic collectors of Portuguese Web text (tweets, news, etc..), the *HASCH* (**H**igh performance **A**utomatic **S**pell **CH**ecker). The aim of this corrector is to preprocess the collected Web texts, so that they can be used by tools that extract useful information of such data. Our spell checker combines known approaches, such as calibration of the internal dictionary, edit distance check [1], as well as own approaches such as an extra dictionary of words based on "Web Language" (informal language with too much abbreviations and symbols), and a accentuation check (peculiar to the Portuguese language). Being a tool aimed at Web, which aims to treat a large volume of data, *HASCH* is completely parallelized on shared memory in order to minimize the processing time. We evaluated *HASCH* both from the efficacy and efficiency perspectives using various texts collected from different sources.

Regarding the efficacy evaluation *HASCH*, we have that the use of an extra dictionary based on "Web Language" showed to be an excellent strategy and an expansion of this dictionary tends to lead to an even more accurate spell checker. Another highlight was the strategy of accentuation checking, which managed to rescue a fair amount of common errors in Portuguese language. Finally, we can highlight the strategy of external calibration (preprocessing) which also showed very promising. The use of a larger and more detailed training base tends to improve the efficacy of our spell checker. Regarding the efficiency evaluation, our tool presented a very good performance, reaching, for texts with over 400,000 words, an almost linear speedup.

As future work, we propose an evaluate of the spell checker using even bigger and complete dictionaries of words and "Web Language" in the most varied contexts. Moreover, we intend to incorporate other known approaches focused on spelling level, as well as, to evaluate other parallelization strategies, based on distributed memory for example.

Acknowledgments

This work was partially funded by CNPq, CAPES, FINEP, Fapemig, and INWEB.

References

[1] P. Norvig, How to Write a Spelling Corrector, http://norvig.com/spell-correct.html.
[2] W. A. G. Kenneth W. Church, Probability scoring for spelling correction, Statistics and Computing 1 (1991) 93–103.
[3] J. L. Peterson, Computer programs for detecting and correcting spelling errors, Communications of the A.C.M. 23 (12) (1980) 676–687.
[4] J. L. Peterson, Computer programs for spelling correction: an experiment in program design, Springer-Verlag.
[5] K. Kukich, Spelling correction for the telecommunications network for the deaf, Communications of the A.C.M. 35 (5) (1992) 80–90.
[6] M. D. McIlroy, Development of a spelling list, IEEE Transactions on Communications COM-30 (1) (1982) 91–99.
[7] R. Nix, Experience with a space efficient way to store a dictionary, Communications of the A.C.M. 24 (5) (1981) 297–298.
[8] A. R. H. Edward M. Riseman, A contextual post-processing system for error correction using binary n-grams, IEEE Trans Computers C-23 (5) (1974) 480–493.
[9] L. L. C. Robert Morris, Computer detection of typographical errors, IEEE Trans Professional Communication PC-18 (1) (1975) 54–64.
[10] G.E. Heidorn, K. Jensen, L.A. Miller, R.J. Byrd, M.S. Chodorow, The epistle text-critiquing system, IBM Systems Journal 21 (3) (1982) 305–326.
[11] I. Marshall, Choice of grammatical word-class without global syntactic analysis: tagging words in the lob corpus, Computers and the Humanities 17 (1983) 139–50.
[12] G. S. Roger Garside, Geoffrey Leech, The computational analysis of english: a corpus-based approach, Longman.
[13] R. L. M. Eric Mays, Fred J Damerau, Context based spelling correction, Information Processing and Management 27 (5) (1991) 517–522.
[14] R. L. W. D.M. Joseph, Correction of misspellings and typographical errors in a free-text medical english information storage and retrieval system, Methods of Information in Medicine 18 (4) (1979) 228–234.
[15] M. J. F. Robert A. Wagner, The string-to-string correction problem, Journal of the A.C.M. 21 (1) (1974) 168–173.
[16] W. A. G. Mark D. Kernighan, Kenneth W. Church, A spelling correction program based on a noisy channel model, COLING-90 13th International Conference on Computational Linguistics 2 (1975) 205–210.
[17] J. C. d. Medeiros, Processamento Morfológico e Correcção Ortográfica do Português, Master's thesis, Instituto Superior Técnico, Lisbos, Portugal (1995).