AI-Based Syntactic Pattern Recognition of Sequences

Inventors : Ghada Badr[*] and B. John Oommen[†]

1 Problem Statement and Potential Applications

This patent concerns the traditional problem encountered in the syntactic Pattern Recognition (PR) of strings or sequences.

The primary investigator[1] involved in this work is a Full Professor at Carleton University in Ottawa, Canada, and is a *Fellow of the IEEE*.

The primary problem solved by the invention involves determining the string or sequence that is most similar to a sequence presented to the system. The search could be initiated by presenting, to the system, a noisy or inexact version of a string contained in memory - for example, at a web-site or in the library or database. The invention will yield the closest string/sequence by searching the dictionary of possible words using a newly invented AI-based strategy. The core of this invention is this search strategy, called the Clustered Beam Search.

Experiments have been done to show the benefits of the CBS over the current state-of-the-art, and the results demonstrate an unbelievably marked improvement (sometimes as high as 90%) for large libraries and databases.

The solution provided by the invention would be applicable in numerous areas including : Inexact or proximity searching on the Internet, keyword-based search in libraries and databases, spelling correction, speech and character recognition (including optical character recognition), and the processing of biological sequences, for example, in human genome projects. These applications are briefly described below.

[*]This author can be addressed at: School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. e-mail address : badrghada@hotmail.com.

[†]*Professor* and *Fellow of the IEEE.* This author can be contacted at: School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. e-mail address : oommen@scs.carleton.ca.

[1]More information about this inventor, who holds a Doctorate from Purdue University, can be found at `www.scs.carleton.ca/~oommen`.

1.1 The Problem Domain

Searching through large alpha and numeric data structures involving nodes such as in the Internet, or large directories or databases of data (each of which are referred to here as a "Dictionary") requires significant computing power due to the large number of possible locations in which potentially relevant information is stored. Additionally, the possibility that the search string or data has been mistyped or recorded erroneously presents augmented difficulties, particularly when considering large data sets such as those involving the Internet, or large directories or databases of data.

The present patent involves an invention which deals with the problem of processing *such symbolic* information. Because the Internet is so pervasive, the market for such Internet-based search strategies is phenomenal. Besides this, the volume of information stored in libraries and databases is astronomical, and searching through such data repositories is by no means trivial. The goal of this invention is to provide the user a method that will enhance such a search especially when the data is erroneous or the data itself is stored inaccurately.

1.2 Applications

The first references to this problem [25] are from the 1960's and 1970's, where the problem appeared in a number of different fields. In those times, the main motivation for this kind of search came from computational biology, signal processing, and text retrieval. These are still the largest application areas that motivate the research in this problem.

The application domains where the methods presented in this invention can be utilized are numerous and include:

1. **The Internet:** When searching the Internet, it is often the case that the user enters the word to be searched incorrectly. Our results can be used to achieve a proximity search on the Internet, and to locate sites and documents that contain words which closely match the one entered. They can thus be used to greatly enhance the power of search engines.

2. **Keyword Search:** When searching libraries and collections, it can occur that the user enters the keyword with spelling or phonetic errors. Our results can be used to search the library or respective collection by first determining the keywords which best match the entered string, and then executing the search.

3. **Spelling Correction:** Our results can be used to achieve the automatic correction of

misspelled strings/substings in a document.

4. **Speech Recognition:** If the waveform associated with a speech signal is processed to yield a string of phonemes, our results can be used to process the phoneme sequence to recognize the speech utterance or speaker.

5. **Optical Character Recognition:** If the digital pixels associated with a sequences of handwritten or printed characters are processed to yield a string of syntactic primitives, our results can be used to process the primitives sequence to recognize the words represented by the handwriting or sequence of printed characters.

6. **Processing of Biological Sequences:** Our results can be used to locate subsequences in sequences when the former are inaccurately represented. Thus, they have potential applications in the human genome project, in the detection of targets for diseases and ultimately in the drug-design process.

7. **Applications in Communication Theory:** Finally, our results can be used for designing and recognizing fast convolutional codes if their noisy versions are processed. They can thus be also used in communication channels for detecting symbols by finding the "most-likely" noiseless sequence.

1.3 The Competing Technology

The competing state-of-the-art search strategies store the data dictionary in a data structure called the *Trie*, which will be described presently. When the location of a desired word is sought for, the current algorithms search the Trie to yield the required information that best matches the inexact or proximity search. This search can go through the Trie level-by-level as described in [18] and [27], or along the branches as described in [35]. The latter is called the Depth First Search mechanism, which is currently recognized as an "industrial benchmark" [32] as seen below.

This state-of-the-art Depth First Search method was incorporated in a patent (K. M. Risvik. "Search system and method for retrieval of data, and the use thereof in a search engine". United States Patent, April 2002), which was purchased by the Norwegian-based company, Fast Search & Transfer^{TM} (FAST^{TM}). From its web-site[2] we observe the following:

- FAST^{TM} began as the company Fast Internet Transfer. The name was later changed

[2]These facts were taken from the information available on FAST^{TM}'s web-site as of June 2005.

to Fast Search & Transfer to reflect the fact that it was involved in developing a wide range of search solutions.

- FAST^{TM}'s powerful enterprise search technology solutions are used by a wide range of global customers and partners including America Online (AOL), AT&T, Cardinal Health, CareerBuilder.com, Chordiant, CIGNA, CNET, Dell, Factiva, Fidelity Investments, Findexa, FirstGov.gov (GSA), IBM, Knight Ridder, LexisNexis, Overture, Rakuten, Reed Elsevier, Reuters, Sensis, Stellent, Tenet Healthcare, Thomas Industrial Networks, T-Online, US Army, Virgilio (Telecom Italia), Vodafone, and Wanadoo.
- FAST^{TM} provides a consultative knowledge-transfer service that enables it to optimize the business and organizational value derived from its search applications.
- One of FAST^{TM}'s products, FAST SBP^{TM}, offers two services that can help the user identify and implement the optimal search solution for an organization. FAST SBP^{TM} is now an industry leader in enterprise search solutions, and provides businesses and government organizations with the ability to intelligently and dynamically, access, retrieve and analyze information in real time, regardless of data format, structure, or location. As a result, organizations make better-informed, more effective decisions that, ultimately, drive their bottom lines.
- With regard to revenues, 2003 was a banner year for FAST, achieving revenue of \$67.6M, which was an increase of 18% from 2002.

1.4 Commercialization Potential

In the area of the Internet, the customers who could use the new technology are all those who develop and use search engines. Our technology can be used expediently to enhance the search and to achieve proximity search. With regard to keyword search, our technology can be used by companies who use large libraries and databases, such as those serviced by the products of FAST. Of course, the applications in the biological domain will involve searching for targets and remedies, and will thus be useful for pharmaceutical companies involved in the design and testing of drugs.

Each of the above mentioned companies in their respective areas could utilize the new technology along their particular vertical domain, and so it is possible that a variety of applications can be built for each of them. This, of course, suggests the development of the core technology for these potential clients.

1.5 Academic Approval of our Technology

Our technology is included a *plenary* talk at an international pattern recognition conference in England in August 2005. Details of the conference and the paper to be published can be provided if necessary.

1.6 Patent Applications

A preliminary patent application to protect our intellectual property rights in the invention has been filed as of June 6, 2005.

1.7 Formal Problem Formulation

Let Y be a misspelled (noisy) string obtained from an unknown word X^*, which is an element of a finite (possibly, large) dictionary H, where Y is assumed to contain Substitution, Insertion and Deletion (SID) errors. Various algorithms have been proposed to obtain an appropriate estimate X^+ of X^*, by processing the information contained in Y, and the literature contains hundreds (if not thousands) of associated papers. We include a *brief* review here.

In what follows, we assume that the dictionary is stored as a trie, whose structure is explained below. Storing the dictionary as a trie yields two distinct phenomena:

- When the dictionary is stored as a trie, solving the approximate string matching problem will be a question of how we can efficiently search the *entire* space representing the dictionary. We propose to utilize results from the general field of AI to permit various graph related searching techniques that can be applied to the underlying structure. The literature already reports two approaches that have been applied, namely, the Breadth First Search scheme introduced in [18] and [27], and the Depth First Search scheme, as shown in [35].

- On the other hand, the trie is a data structure that offers search costs that are independent of the document size. Tries also combine prefixes together, and so, by using tries in approximate string matching, we can utilize the information obtained in the process of evaluating any one $D(X_i, Y)$, to compute any other $D(X_j, Y)$, where X_i and X_j share a common prefix.

Most techniques proposed to prune the search in the trie have applied the so-called "cutoff" strategy to decrease the computational burden. The "cutoff" is based on the assumption that the maximum number of errors permitted is known *a priori*, and is more useful when the inter-symbol costs are of form 0/1. The cutoff based literature is silent for the scenarios when the maximum number of errors is not available, and when the inter-symbol costs are general. We seek a strategy by which we can prune the entire search space by using AI heuristic search strategies.

2 Contribution

This patent describes how we can optimize non-sequential PR computations by incorporating heuristic search schemes used in AI into the approximate string matching problem. First, we present a new technique enhancing the Beam Search (BS), which we call the Clustered Beam Search (CBS), and which can be applied to any tree searching problem[3]. We then apply the new scheme to the approximate string matching when the dictionary is stored as a trie. The trie is implemented as a Linked List of Prefixes (LLP). The latter permits *level-by-level* traversal of the trie (as opposed to traversal along the "branches"). The newly-proposed scheme can be used for Generalized Levenshtein distances (i.e., those which are not of a 0/1 form) and also when the maximum number of errors is not given *a priori*.

It has been rigorously tested on three benchmarks dictionaries by recognizing noisy strings generated using the model discussed in [28], and the results have been compared with the acclaimed standard [32], the Depth-First-Search (DFS) trie-based technique [35]. The new scheme yields a marked improvement (of **up to** 75%) with respect to the number of operations needed, and at the same time maintains almost the same accuracy. The improvement in the number of operations increases with the size of the dictionary. The CBS heuristic is also compared with the performance of the original BS heuristic when applied to the trie structure, and the experiments again show an improvement of **more than** 91%. Furthermore, by marginally sacrificing a small accuracy in the general error model, or by permitting an error model that increases the errors as the length of the word increases (as explained presently), an improvement of **more than** 95% in the number of operations can be obtained.

The details of the experimental results are described presently.

[3]The new scheme can also be applied to a general graph structure, but we apply it to the trie due to the dominance of the latter in our application domain, approximate string matching.

3 A Survey of the Prior-Art

The literature contains hundreds of papers which deal with the Syntactic PR of strings/sequences. Although our review is brief, the bibliography included in this write-up is quite comprehensive. Excellent recent surveys about the field can be found in [12], [25].

3.1 Dictionary-based Approaches

Most of the time-efficient methods currently available require that the maximum number of errors be known *a priori*, and these schemes are optimized for the case when the edit distance costs are of a form 0/1. In [14], Du and Chang proposed an approach to design a very fast algorithm for approximate string matching that divided the dictionary into partitions according to the lengths of the words. They limited their discussion to cases where the error distance between the given string and its nearest neighbors in the dictionary was "small".

Bunke [10] proposed the construction of a finite state automaton for computing the edit distance for *every* string in the dictionary. These automata are combined into one "global" automaton that represents the dictionary, later used to calculate the nearest neighbor for the noisy string when compared against the active dictionary. This algorithm requires time which is linear in the length of the noisy string. However, the number of states of the automaton grows exponentially. Oflazer [26] also considered another method that could easily deal with very large lexicons. To achieve this, he used the notion of a **cut-off** edit distance: this measures the minimum edit distance between an initial substring of the incorrect input string, and the (possibly partial) candidate correct string. The cutoff-edit distance required *a priori* knowledge of the maximum number of errors found in Y and that the inter-symbol distances are of a form 0/1, or when general distances are used, a maximum error value.

Baeza-Yates and Navarro [6] proposed two speed-up techniques for on-line approximate searching in large indexed textual databases when the search is done on the vocabulary of the text. The efficiency of this method depends on the number of allowable error values.

The literature[4] also reports some methods that have proposed a filtering step so as to decrease the number of words in the dictionary that need to be considered for calculations. One such method is "the similarity" keys method [34] that offers a way to select a list of possible correct candidates in the first step. This correction procedure, proposed in [34], can be argued to be a variant of Oflazer's approach [26]. The time required depends merely on the permitted number of edit operations involved in the distance computations.

[4]More details about the state-of-art can be found in [4] and [5].

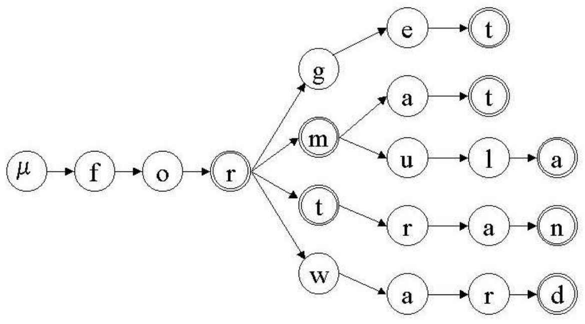


Figure 1: An example of a dictionary stored as a trie with the words {for, form, fort, fortran, formula, format, forward, forget}.

A host of optimizing strategies have also been reported in the literature for methods which model the language probabilistically using N-grams, and for Viterbi-type algorithms [3], [9], [17], [36]. These methods do not explicitly use a "finite-dictionary" (trie or any other) model, and so we believe that it is not necessary to survey them here. The same is also true for methods that apply to error correcting parsing [2], [31] and grammatical inference [24], where the dictionary is represented by the language generated by a grammar whose production probabilities are learnt in the "training" phase of the algorithm.

3.2 Dictionaries represented as tries

The *trie* is a data structure that can be used to store a dictionary when the dictionary[5] is represented as a set of words. Words are searched as a character by character basis.

The data is represented not in the nodes but in the path from the root to the leaf. Thus, all strings sharing a prefix will be represented by paths branching from a common initial path. Figure 1 shows an example of a trie for a simple dictionary of words {for, form, fort, forget, format, formula, fortran, forward}. The figure illustrates the main advantage of the trie as it only maintains the minimal prefix set of characters that is necessary to distinguish all the elements of H. The trie has the following features:

1. The nodes of the trie correspond to the set of all the prefixes of H.

2. If X is a node in the trie, then X_p, the left derivative of order one, will be the parent node of X, and X_g, the left derivative of order two, will be the grandparent of X.

3. The root of the trie will be the node corresponding to μ, the null string.

[5]In terms of notation, A is a finite alphabet, H is a finite (possibly large) dictionary, and μ is the null string, distinct from λ, the null symbol.

4. The leaves of the trie will all be words in H, although the converse is not true.

With regard to traversal, the trie can be considered as a graph and can be searched using any of the possible search strategies applicable to AI problems. The aim, of course, is to minimize the computations, and to utilize the computations performed when computing the distances for the prefixes of the strings, maximally. The literature includes two possible strategies that have been applied to tries, namely the Breadth First Search strategy [18], [27] and the Depth First Search strategy [35], currently recognized as an "industrial benchmark" [32].

Although the methods proposed in [35] are elegant, in order to apply these cutoff principles the user has to know the maximum number of errors, K, *a priori*, and also resort to the use of 0/1 costs for inter-symbol edit distances.

4 Heuristic Search

Heuristics and the design of algorithms to implement heuristic search have long been a core concern of AI research [22], [30]. Game playing and theorem proving are two of the oldest applications in AI: both of these require heuristics to prune spaces of possible solutions; It is not feasible to examine every inference that can be made in a domain of mathematics, or to investigate every possible move that can be made on a chessboard, and thus a heuristic search is often the only practical answer. It is useful to think of heuristic algorithms as consisting of two parts: the heuristic measure and an algorithm that uses it to search the state space[6].

For approximate string matching the problem encountered involves both ambiguities and the excessive time required as the dictionary is large. Observe that there is no exact solution for the noisy string that one is searching for, and at the same time the process is time consuming because one has to search the entire space to find it. We thus seek a heuristic to determine the nearest neighbor to the noisy string, and one which can also be used to prune the space. The ambiguity of the problem can be resolved by several methods including the Depth-First-Search trie-based heuristic that uses the dynamic equations and string edit distance calculations.

[6]When we refer to heuristic search we imply those methods that are used for solving the problems possessing ambiguities or which are inherently time consuming. In both these cases, we seek a heuristic to efficiently prune the space and lead to a good (albeit, suboptimal) solution.

4.1 Beam Search (BS)

The simplest way to implement a heuristic search is through a *Hill-Climbing* (HC) procedure [22], [30]. HC strategies expand the current state in the search space and evaluate its children. The best child is selected for further expansion and neither its siblings nor its parent are retained. The search halts when it reaches a state that is better than any of its children. The algorithm cannot recover from failures because it keeps no history. A major drawback to HC strategies is their tendency to become stuck at local maxima/minima.

To overcome the problems of HC, and to provide better pruning than BFS, researchers have proposed other heuristics such as the *Beam Search*[7] (BS) [33]. In BS, we retain q states, rather than a single state as in HC, and these are stored in a single pool. We then evaluate them using the objective function. At each iteration, all the successors of all the q states are generated, and if one is the goal, we stop. Otherwise, we select the best q successors from the complete list and repeat the process. This BS avoids the combinatorial explosion of the Breadth-First search by expanding only the q most promising nodes at each level, where a heuristic is used to predict which nodes are likely to be closest to the goal and to pick the best q successors.

One potential problem of the BS is that the q states chosen tend to quickly lack diversity. The major advantage of the BS, however, is that it increases both the space and time efficiency dramatically, and the literature includes many applications in which the BS pruning heuristic has been used. These applications are include: handwriting recognition [15], [20], [23], Optical Character Recognition (OCR) [16], word recognition [19], speech recognition [8], [21], information retrieval [37] and error-correcting Viterbi parsing [2].

5 Salient Aspects of the Patent : Clustered Beam Search

We propose a new heuristic search strategy that can be considered as an enhanced scheme for the BS. This scheme, called the Clustered Beam Search (CBS) is like BS in that it only considers some nodes in the search, and discards the others from further calculations.

The details of this scheme are contained in the patent application. However, we mention that this scheme too has a paramater, q, which quantifies the size of the retained set of states. As we increase q, the accuracy increases and the pruning ability decreases. When the evaluation function is informative, we can use small values for q. As q increases, the cost

[7]When we speak about Beam Search, we are referring to *local* beam search, a combination of an AI-based local search and the traditional beam search [33] methodologies.

associated with maintaining the order of the lists may overcome the advantage of pruning. The possibility of including a larger number of nodes per level increases with the new CBS scheme when compared to the BS leading to increased accuracy.

The patent papers describe:

1. A heuristic search for the approximate string matching problem capable of pruning the search space when the inter-symbol distances are general, and the maximum number of errors cannot be known *a priori*.
2. The use of two heuristic functions that be used for this purpose.
3. The use of an efficient data structure, the Linked List of Prefixes (LLP), which can be used to efficiently store and process the Trie.

The salient details of all these issues are contained in the patent papers.

6 Experimental Results

6.1 The Test Bed

To investigate the power of our new method with respect to computation, we conducted various experiments on three benchmark dictionaries. The results obtained were (in our opinion) remarkable with respect to the gain in the number of computations needed to get the best estimate X^+. By computations we mean the number of addition and minimization operations needed. The CBS-LLP-based[8] scheme was compared with the acclaimed DFS-trie-based work for approximate matching [35] when the maximum number of errors was not known *a priori*.

Three benchmark data sets were used in our experiments. Each data set was divided into two parts: a *dictionary* and the corresponding *noisy file*. The dictionary was the words or sequences that had to be stored in the trie. The noisy files consisted of the strings which were searched for in the corresponding dictionary. The three dictionaries we used were as follows:

- *Eng*[9]: This dictionary consisted of 946 words obtained as a subset of the most common English words [13] augmented with words used in computer literature. The average length of a word was approximately 8.3 characters.

[8]This scheme uses the CBS method as the AI-based searching strategy, and incorporates the LLP implementation of the trie. It is refereed to below with the *prefix* CBS-LLP.

[9]This file is available at `www.scs.carleton.ca/~oommen/papers/WordWldn.txt`.

Table 1: Statistics of the data sets used in the experiments.

	Eng	**Dict**	**Webster**
Size of dictionary	8KB	225KB	944KB
number of words in dictionary	964	24,539	90,141
min word length	4	4	4
max word length	15	22	21

- *Dict*[10]: This is a dictionary file used in the experiments done by Bentley and Sedgewick in [7].
- *Webster's Unabridged* Dictionary: This dictionary was used by Clement *et. al.* [1], [11] to study the performance of different trie implementations.

The statistics of these data sets are shown in Table 1. The alphabet is assumed to be the 26 lower case letters of the English language. For all dictionaries we removed words of length smaller than or equal to 4.

Three sets of corresponding noisy files were created using the technique described in [28], and in each case, the files were created for three specific error characteristics, where the latter means the number of errors per word. The three error values tested were for 1, 2 and 3, referred to by the three sets SA, SB, and SC respectively.

For each of the three sets, SA, SB and SC (generated using the noise generator model described in [28]), we assumed that the number of insertions was geometrically distributed with parameter $\beta = 0.7$. The conditional probability of inserting any character $a \in A$ given that an insertion occurred was assigned the value 1/26, and the probability of deletion was 1/20. The table of probabilities for substitution (typically called the confusion matrix) was based on the proximity of character keys on the standard QWERTY keyboard and is given in [28][11].

6.2 Experimental Results : Unconstrained Errors

The two algorithms, the DFS-trie-based and our algorithm, CBS-LLP-based, were tested with the three sets of noisy words for each of the three dictionaries. We report the results obtained in terms of the number of computations (additions and minimizations) and the accuracy for the three sets. The calculations were done on a Pentium V processor, 3.2 GHZ. Figure 2 shows a graphical representation of the results. The figures compares both time and

[10]This file can be downloaded from `www.cs.princeton.edu/~rs/strings/dictwords.`

[11]It can be downloaded from `www.scs.carleton.ca/~oommen/papers/QWERTY.doc.`

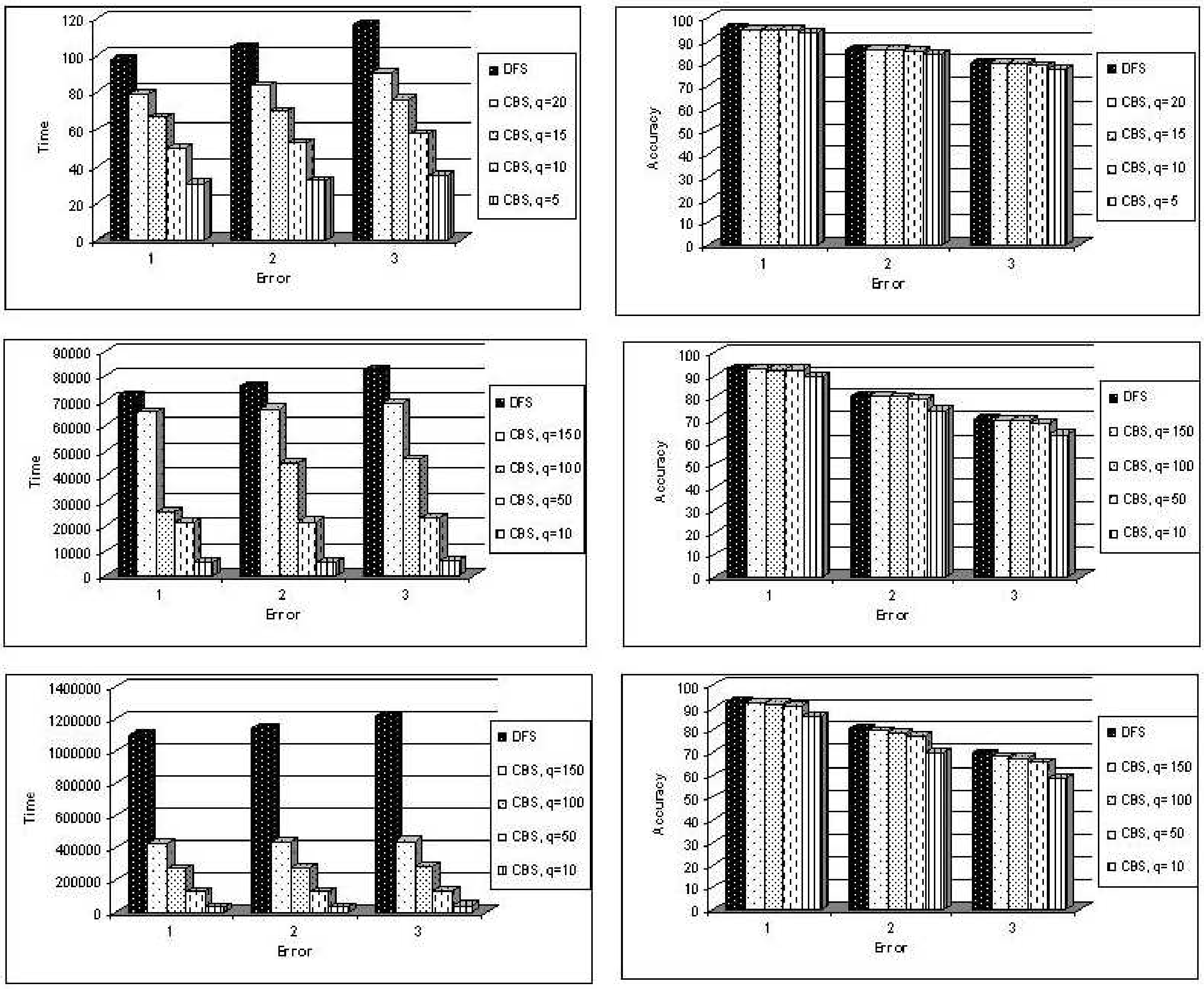


Figure 2: The results for comparing the **CBS-LLP**-based method with the **DFS-trie**-based method for (top) the **Eng** dictionary, (middle) the **Dict** dictionary, and (bottom) the **Webster** dictionary. The time is represented by total number of operations in millions.

accuracy. The numbers are shown in millions. The results show the significant benefit of the CBS-based method with respect to the number of computations, while maintaining excellent accuracy. For example consider the *Webster* dictionary, for the SA set, and $q = 100$: the number of operations for DFS-trie-based is 1,099,279, and for the CBS-LLP-based method is 271,188 representing a savings of 75.3%, and a loss of accuracy of only 0.5%. For the *Dict* dictionary, for the SA set, and $q = 100$, the number of operations for the DFS-trie-based is 72,115, and for the CBS-LLP-based method is 44,254 which represents a savings of 36.6%, and a loss of accuracy of only 0.2%. When $q = 50$, the number of operations for the CBS-LLP-based method is 21,366, representing a savings of 70.4%, with a loss of accuracy of only 0.5%. There is always a trade-off between time and accuracy but the loss of accuracy here is negligible compared to the "phenomenal" savings in time.

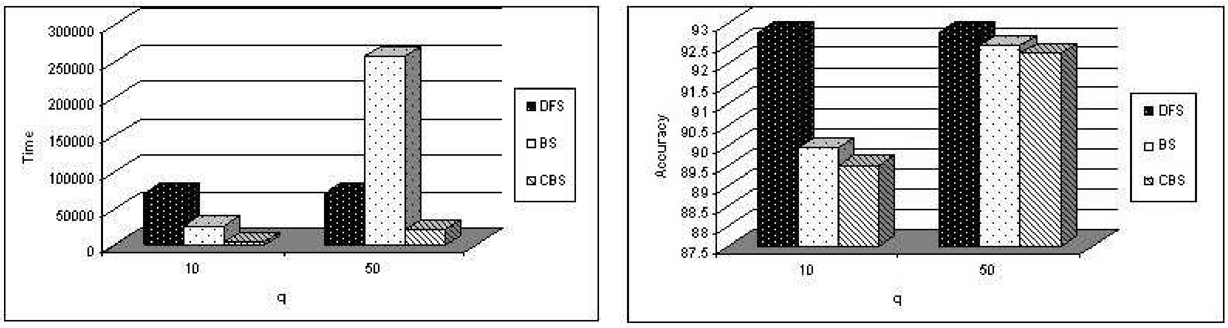


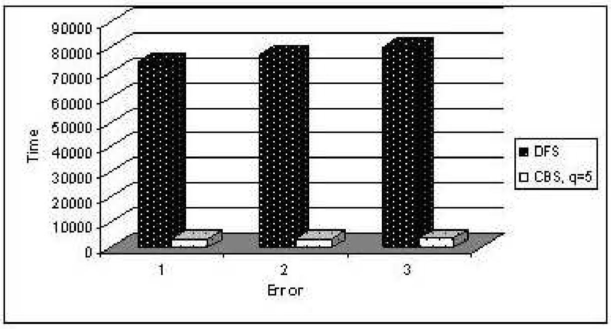
Figure 3: The results for comparing the **CBS-LLP**-based method with the **BS-LLP**-based method for the **Dict** dictionary when applied to set **SA**. The time is represented by total number of operations in millions.

To show the benefits of the CBS over the BS, we show the results when applying the BS for the *dict* dictionary in Figure 4, when the approximately equivalent width (number of nodes taken per level) is considered for the BS. The width is considered approximately equal when we approximately equate the number of addition operations. The figure shows only the result obtained for the set SA; the results for the other sets are analogous and almost identical. From the figures the reader will observe the tremendous gain in the minimization operations needed when the same ordering technique is used for arranging all the priority queues. The results are shown only for $q = 10$ and $q = 50$, because if we increase q it will yield bad results for the BS which is much worse than when $q = 50$. For example, the number of operations for BS-LLP-based method is 256,970, and for the CBS-trie-based method is 21,366, representing a savings of 91.7% in the total number of operations when the accuracy obtained is 92.3%. In this case, the number of operations for the BS method is much more than the 72,115 operations of the DFS-trie-based method. From Figure 2 (middle), we see that we can increase q in CBS-LLP-based method to 100 and get an accuracy of 92.6 with savings of 36.6% in the total number of operations with respect to DFS-trie-based method. This is not feasible by the applying the BS method.

6.3 Experimental Results : Constrained Error Model

A we see from the results in the previous section, by marginally sacrificing a small accuracy value for the general error model (by less than 1%) a noticeable improvement can be obtained with respect to time.

By permitting an error model that increases the errors as the length of the word increases (i.e., the errors do not appear at the very beginning of the word), an improvement of more than 95% in the number of operations can be obtained, which is, in our opinion, absolutely



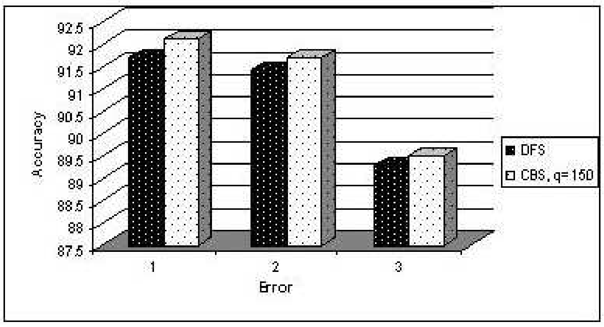


Figure 4: The results for comparing the **CBS-LLP**-based method with the **DFS-trie**-based method for the **Dict** dictionary when the optimized **error model** is used and $q = 5$. The time is represented by total number of operations in millions.

amazing. This is because if errors are less likely to appear at the very beginning of the word, the quality of pruning, with respect to accuracy, will be more efficient at the upper levels of the tree. Thus we can utilize a small value for q, the width of the beam, and as a result achieve more pruning. All our claims have been verified experimentally as shown in Figure 4. The results are shown for $q = 5$, (which is a very small width) demonstrating very high accuracy. For example, for the set SA, the number of operations for DFS-LLP-based method is 74,167, and for the CBS-trie-based method is just 3,374, representing a savings of 95.5%. This has obviously great benefits if the noisy words received are not noisy at the beginning, in which case we still need to apply approximate string matching techniques. Even here we would like to make use of the approximately exact part at the very beginning of the words (which are variant from one word to another) to lead to a superior solution.

7 Conclusions

In this document we have described the functionalities of a patent concerning the traditional problem encountered in the syntactic Pattern Recognition (PR) of strings.

We have explained the primary problem solved by the invention, which involves determining the string or sequence that is most similar to a sequence presented to the system. The search could be initiated by presenting, to the system, a noisy or inexact version of a string contained in memory - for example, at a web-site or in the library or database. The goal of the invention is to yield the closest string/sequence by searching the dictionary of possible words using our newly invented AI-based strategy. The core of this invention is this search strategy, called the Clustered Beam Search.

The experimental results presented on stand benchmark data sets and dictionaries have demonstrated the benefits of the CBS over the current state-of-the-art, and the results display an unbelievably marked improvement (sometimes as high as 90%) for large libraries and databases.

In conclusion, we state that the solution provided by the invention would be applicable in numerous areas including : Inexact or proximity searching on the Internet, keyword-based search in libraries and databases, spelling correction, speech and character recognition (including optical character recognition), and the processing of biological sequences, for example, in genomic projects.

References

[1] A. Acharya, H. Zhu, and K. Shen. Adaptive algorithms for cache-efficient trie search. *ACM and SIAM Workshop on Algorithm Engineering and Experimentation*, January 1999.

[2] J. C. Amengual and E. Vidal. Efficient error-correcting viterbi parsing. *IEEE Transactions on Communications*, 20(10):1109–1116, October 1998.

[3] J. C. Amengual and E. Vidal. The viterbi algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):268–278, October 1998.

[4] G. Badr and B. J. Oommen. Enhancing trie-based syntactic pattern recognition using ai heuristic search strategies. Unabridged version of the present paper.

[5] G. Badr and B. J. Oommen. Search-enhanced trie-based syntactic pattern recognition of sequences. 2005. Patent.

[6] R. Baeza-Yates and G. Navarro. Fast approximate string matching in a dictionary. *in Proceedings of the 5th South American Symposium on String Processing and Information Retrieval (SPIRE'98), IEEE CS Press*, pages 14–22, 1998.

[7] J. Bentley and R. Sedgewick. Fast algorithms for sorting and searching strings. *Eighth Annual ACM-SIAM Symposium on Discrete Algorithms New Orleans*, January 1997.

[8] E. Bocchieri. A study of the beam-search algorithm for large vocabulary continuous speech recognition and methods for improved efficiency. *In Proc. Eurospeech*, 3:1521–1524, 1993.

[9] A. Bouloutas, G. W. Hart, and M. Schwartz. Two extensions of the viterbi algorithm. *IEEE Transactions on Information Theory*, 37(2):430–436, March 1991.

[10] H. Bunke. Fast approximate matching of words against a dictionary. *Computing*, 55(1):75–89, 1995.

[11] J. Clement, P. Flajolet, and B. Vallee. The analysis of hybrid trie structures. *Proc. Annual A CM-SIAM Symp. on Discrete Algorithms, San Francisco, California*, pages 531–539, 1998.

[12] R. Cole, L. Gottieb, and M. Lewenstein. Dictionary matching and indexing with errors and don't cares. *Proceedings of the thirty-sixth annual ACM Symposium on Theory of Computing, Chicago, IL, USA*, pages 91–100, June 2004.

[13] G. Dewey. *Relative Frequency of English Speech Sounds.* Harvard Univ. Press, 1923.

[14] M. Du and S. Chang. An approach to designing very fast approximate string matching algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):620–633, 1994.

[15] J. T. Favata. Offline general handwritten word recognition using an approximate beam matching algorithm. *IEEE Transactions on pattern Analysis and Machine Intelligence*, 23(9):1009–1021, September 2001.

[16] Z. Feng and Q. Huo. Confidence guided progressive search and fast match techniques for high performance chinese/english ocr. *Proceedings of the 16th International Conference on Pattern Recognition*, 3:89–92, August 2002.

[17] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973.

[18] R. L. Kashyap and B. J. Oommen. An effective algorithm for string correction using generalized edit distances -i. description of the algorithm and its optimality. *Inf. Sci.*, 23(2):123–142, 1981.

[19] P. Laface, C. Vair, and L. Fissore. A fast segmental viterbi algorithm for large vocabulary recognition. *in Proceeding ICASSP-95*, 1:560–563, May 1995.

[20] C. Liu, M. Koga, and H. Fujisawa. Lexicon-driven segmentation and recognition of handwritten character strings for japanese address reading. *IEEE Transactions on pattern Analysis and Machine Intelligence*, 24(11):1425–1437, November 2002.

[21] F. Liu, M. Afify, H. Jiang, and O. Siohan. A new verification-based fast-match approach to large vocabulary continuous speech recognition. *Proceedings of European Conference on Speech Communication and Technology*, pages 1425–1437, September 2001.

[22] G. F. Luger and W. A. Stubblefield. *Artificial Intelligence Structure and Strategies for Complex Problem Solving*. Addison-Wesley, 1998.

[23] S. Manke, M. Finke, and A. Waibel. A fast technique for large vocabulary on-line handwriting recognition. *International Workshop on Frontiers in Handwriting Recognition*, September 1996.

[24] L. Miclet. Grammatical inference. *Syntactic and Structural Pattern Recognition and Applications*, pages 237–290, 1990.

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

[26] K. Oflazer. Error-tolerant finite state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1):73–89, March 1996.

[27] B. J. Oommen and G. Badr. Dictionary-based syntactic pattern recognition using tries. *Procedings of the Joint IARR International Workshops SSPR 2004 and SPR 2004*, pages 251–259, August 2004.

[28] B. J. Oommen and R. L. Kashyap. A formal theory for optimal and information theoretic syntactic pattern recognition. *Pattern Recognition*, 31:1159–1177, 1998.

[29] B. J. Oommen and R. K. S. Loke. Syntactic pattern recognition involving traditional and generalized transposition errors: Attaining the information theoretic bound. Submitted for Pubication.

[30] J. Pearl. *Heuristics : intelligent search strategies for computer problem solving*. Addison-Wesley, 1984.

[31] J. C. Perez-Cortes, J. C. Amengual, J. Arlandis, and R. Llobet. Stochastic error correcting parsing for ocr post-processing. *International Conference on Pattern Recognition ICPR-2000*, 2000.

[32] K. M. Risvik. Search system and method for retrieval of data, and the use thereof in a search engine. *United States Patent*, April 2002.

[33] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2003.

[34] K. Schulz and S. Mihov. Fast string correction with levenshtein-automata. *International Journal of Document Analysis and Recognition*, 5(1):67–85, 2002.

[35] H. Shang and T. Merrettal. Tries for approximate string matching. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):540–547, August 1996.

[36] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[37] J. G. Wolff. A scaleable technique for best-match retrieval of sequential information using metrics-guided search. *Journal of Information Science*, 20(1):16–28, 1994.