International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869 (O) 2454-4698 (P), Volume-7, Issue-2, February 2017

Application of Search & Sorting Techniques – in Natural Language Processing

TVVV Prasad, Raghu B Korrapati

***Abstract*— The most prevalent technique for Natural Language parsing is done by using pattern matching through references to a database with the aid of grammatical structures models. But the huge variety of linguistical syntax and semantics mean that accurate real time analysis is very difficult. We have analyzed several optimization techniques to reduce the search time, space for finding an accurate parse of a sentence where individual words can have multiple possible syntactic categories, and categories and phrases can combine together in different ways. This paper is a comprehensive study of algorithms that we have considered, includes mechanisms for ordering that reduce the search time & cost without loss of completeness or accuracy as well as mechanisms that prune the space & time and may result in eliminating valid parses or returning a suboptimal as the best parse.**

***Index Terms*— Search, Sorting Technique, data, natural language processing, syntactic and Symantec analysis, algorithms, parsing techniques.**

I. INTRODUCTION

The convolution and the sizes of the lexical databases and associated grammatical rules contribute to most of the behavior of Natural Language parsers. By increasing the sizes of the database or including a more complex set of grammatical rules (i.e. 'chandassu'), the parser is able to handle the parsing of more complex sentences (i.e. 'poems') or is able to include more accurate information to the parsed sentences, but the introduction of these results in a more complex parsing procedure and the capability to compute for more cases is necessary for the parser. Even without the extended database or associated business rules ('chandassu'), parsing of long sentences or poems is often avoided due to the extremely large amount of different possibilities in parsing the sentence. To counteract the increase in the parse time form the application of complex grammatical rules, we explore the effects of applying search algorithms to a parser to reduce the search space and hence enhance the parsing speed/time. To measure the accuracy of the parse, we use a simple scoring system derived from the probability that a particular structure would exist. This scoring system does not always parse the sentence correctly, but it provides a good indication of the likeliness of the structure from a statistical point of view based on its complexity. The purpose of the project is to provide a faster way of parsing Telugu language poetry without losing the effect of grammatical structures i.e. chandassu, or the semantic and syntactic information that have been applied to or extracted from the parser. These areas being the key focus of most research done in NLP and will continue to increase in complexity in the future. One such example is "Application of Searching & Sorting techniques for Telugu Language Poetry"

TVVV Prasad, Rayalaseema University Kurnool – 518 002, A.P., India
Raghu B Korrapati, Rayalaseema University Kurnool – 518 002, A.P., India

II. PARSING

The parser we have used is the rule based probabilistic, lexicalized combinatory, categorical grammar embedded parser that incorporates both top-down and bottom-up search strategy. In the pilot stage, the parser builds up a statistical model of the grammatical structure by learning from a manually parsed corpus, which is used to assign the possible categories ('chandassu'), weightage and the probabilities of the particular chandassu for a word, and also the probabilities associated with the actual combination of two structures. The CGE (Categorical grammar embedded i.e. 'chandassu') incorporated in the parser defines the rules and methods used in the combination stage of the parser, and implements an extended set of the standard CGE combinatory 'chandassu' that makes the grammar more flexible. The nature in which a combination occurs is very much like using the link grammar rules to combine between the different states. Intelligent CGE parser is used to find grammar ('Chandassu') of a given poem and enables right prediction of words, while building a new poem in Telugu literature. The key steps in this algorithm include – Parsing, building a lexicon, syntactic analysis, with the help of predefined rule base, determine 'chandassu (grammar) of a given poem. It also, builds a lexicon of all the words derived from a poem. The parsing techniques include leveraging statistical techniques to help the poet in finding an appropriate grammar associated with the poem, while composing a new poem. An intelligent hashing function is used for faster searching. These techniques and algorithms will enable linguists to analyze or study the ancient Telugu language structure or any natural language processing.

III. OPTIMAL SEARCH

The major goal of this project was to explore alternative standard and novel algorithms that were appropriate to the NLP task and could relatively easily be slotted into the existing lexical & CGE parser framework. The kind of algorithms and optimizations that are reasonable is tightly constrained by the nature of the CGE 'Chandassu' model for Telugu poetry and the associated intelligent NLP parser implementation. Another major constraint of the algorithm is one that is often ignored, which is the overhead in the execution of the algorithms. This factor plays an equally important role in the search problem, but has often been ignored due to the increase in the hardware performance rate. The algorithmic design was modularized, so that an easy switching of the algorithm could be done with a uniform interface to the rest of the original parser. This meant that the algorithm relied on some of the existing structure of the parser, which was the cause of some limitations in the

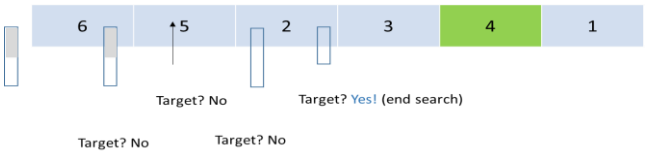
algorithms and is an area that could be modified in the future to further increase the efficiency of the parser by adding the grammar associated with any natural language for processing.

A. *Design Approach*

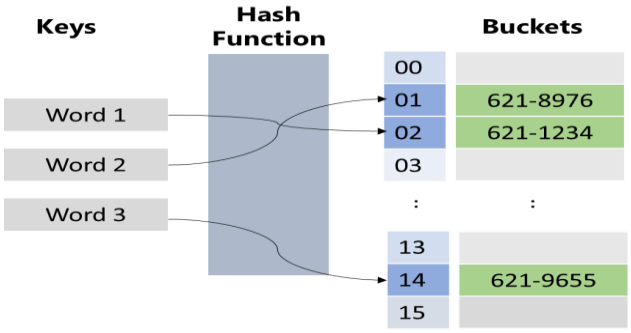
To apply the ranked weightage ordering, the list was maintained in a sorted manner by their probability scores and the pointer simply moved along the list, as more words were used to combine with other words. The state being pointed to by the pointer, which was the state being used to combine with other words of higher scores, was called the pivot state. By combining the pivot state with states of higher scores, the algorithm guaranteed that resulting state of the combination would be equal or lower scored than the pivot state. This allowed for a simple algorithm for maintaining the ordered list. In the initial phase, we determine the 'Chandassu (grammar)' for a given input (i.e. given Telugu poem). We will parse the inbound data feed, calculate the complexity of the word and store it in the Lexicon. A letter code is assigned based on complexity - simple with 'S', medium with 'M', and complex with 'C'. And the syntactical analysis will be conducted simultaneously for the same feed based on the predefined set of business rules, whereas the syntax for the parsed string can be generated from the rules database. The rules database has been designed by using the set of rules based out of 'Telugu' grammar. The notation is similar to the one used to generate the grammar (i.e. chandassu) for a given poem in Telugu literature and the underlying data structure example model is defined below.

B. *Hashing & Predictive Analytic Techniques*

Upon analyzing and implementing the above mentioned design approach & following algorithms for parsing and improving search time, we determine the predictability of words to construct a poem with an appropriate *Chandassu*. Predictability of the words is achieved with the help of predictive techniques that encompass a variety of statistical techniques to analyze and to pick up the appropriate word to construct a poem. By using the Sequential & Binary Search technique, we search the words with appropriate *chandassu* from the lexicon and will be made available for the user to select the suitable word. Following is the sequential search structure that was embedded in the application design.



Here, we have used a hash map (i.e. a data structure) that uses a hash function to map identifying values, known as keys, (e.g., a poem pattern) to their associated values (e.g., their respective *VRUTTA*). The hash function is used to transform the key into the index (the *hash*) of an array element (the *slot* or *bucket*) where the corresponding value is to be sought. Here, we will derive the *Chandassu* of each word and will store the same in the Lexicon by using a predefined data structure i.e. data structure will contain the word, *Chandassu*, Unicode, *VRUTTA*'s and word complexity. As and when a given input matches to any of the predefined *VRUTTAS* i.e. 'U', 'C', 'S' & 'M', we'll retrieve the same from the Lexicon and will populate it on the screen to help the user. And the same has been represented in the below data structure with an example as well. Below is the Hash map function structure, data structure that interprets the word complexity and predictive word data structures.



Word Complexity Data Structure

Complexity	Simple	Medium	Complex
Word – U I I	S	M	C

Predictive Word Data Structure

Word	Chandassu
evvani	U I I

Unicode
ఎవ్వని

U	C	S	M	Complexity
1	0	0	0	M

Pre-Optimization

Once the rules database is formed, we need to update the generated syntax into the words database formed using the lexicon. The next step would be to generate the Unicode for the grammar. Here, in this step if an error occurs in the syntax, then the error will be sent to the error log. The ranking weightage algorithm is essentially embodied by the following pseudo-code:

1. Populate the list with grammar for every letter in the word.
2. Sort the list by their probability scores.
3. Set pointer at the first word in the list.
4. While the list contains un-combined words
5. Set pivot as the next most probable word.
6. Return if pivot state is a terminal state.
7. Combine pivot with all adjacent words with higher probability.
8. Insertion sort all newly created words or states in to the list.
9. Return failure

With the application of this ordering, the algorithm allowed for early termination of the search, since the newly created words (being of equal or lesser probability) must be inserted below the pivot state due to the cascading effect of the product of the probability. Any terminal state found later would have a lower probability than the first one that was found, so the algorithm guarantees the retrieval of the most probable state

without having to exhaustively search all possible combinations.

Post Optimization

By only using a single list to maintain all possible derivation of the words, traversals and maintenance of the ordering of the list used up a lot of valuable time. To counteract this, we re-introduce a charting behavior as the second improvement to the algorithm. We implemented a table, called the indexed table, in which all the words or lines of the poem that were in the used section were placed, rather than keeping them in the same list. The table also grouped together the words that occupied the same starting and ending positions, to simplify the decision process in determining which words were adjacent to the pivot state. The ranked weightage list was replaced by a table, which we called the sorted table that handled the push and pop manipulations to simplify and to modularize the algorithm for future use. The third major step involved the use of a critical score, which is the score of the currently most probable terminal state in the sorted table. By not operating on states that are going to produce a lower probability than the critical score, it allowed for a large pruning of the search tree, weeding out states with very low probability that would not contribute to the most probable terminal state. The algorithm also provides a pre-processing stage before a combination between states took place, which contributed to a little overhead, but managed to cut down the amount of unnecessary combinations and avoided the lengthy combination stage of two words. The experimental tree-climb algorithm used here shows an impressive parse time and huge reduction in the search space & time, but has slight inaccuracies parse compared to the other algorithms, which can be seen in Table 1

	Exhaustive	Optimal	Sub-Optimal
Pre-Implementation			%
Parse Time	100	15.2	1.7
Search Time	100	4.9	0.3
Most probable	100	100	84
Post Implementation			%
Parse Time	100	10.4	0.7
Search Time	100	2.1	0.1
Most probable	100	100	66.7

Table 1: Statistics of parsing of the optimal and suboptimal algorithms for both Pre and Post Optimization.

The parse time and the search space are represented as the proportionality compared to the exhaustive algorithm and the percentage that the algorithm retrieved the most probable parse is indicated in the last row. The optimal algorithm is the combined algorithm of all the algorithms that provided benefits to the parsing speed without the loss of accuracy and the suboptimal algorithm is the tree-climb algorithm, which provided the fastest and also a reasonably accurate result from all tested suboptimal algorithms. The optimal search algorithm returns the most probable parse tree, but sometimes varies in the tagging and bracketing of the parse due to the cases when multiple parses have the same probability. The tree-climb algorithm's performance in the accuracy domain is relatively poor, but some of the loss in the accuracy can be recovered by altering the amount of states used in the seeding stage. However, because the algorithm loses track of the ranking of the words, the algorithm must exhaustively combine all states to determine the most probable parse.

On Pre & Post optimization comparison, it is fairly easy to see the improvements of the developed algorithms, but for the task of NLP, it is probably more important to look at a per sentence comparison, especially if it is in an environment where human interaction is required. Figure 1 indicates the relationship between the parsing time and the number of words in the sentence for the exhaustive, optimal and the suboptimal search algorithms. There is a huge reduction in the parse time from the algorithm with the optimal algorithm, and an even greater reduction from the suboptimal algorithm

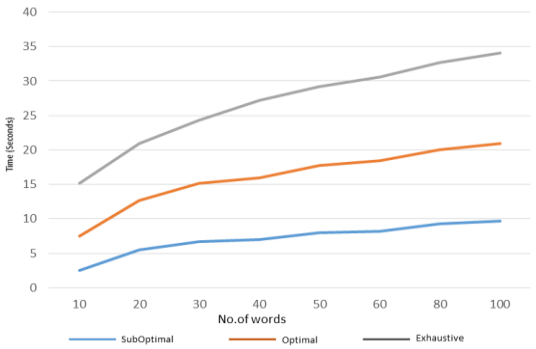


Figure 1: Number of words in the sentence versus parsing time on the post optimization for the exhaustive, optimal and the suboptimal algorithm.

Context Based Search

The goal of a context-based search implementation (or disambiguation) process is to find the most relevant search result(s), T, given a main source query term, S, with the help of L/R contexts. Intuitively, S and T tend to be a relevant query-answer pair if many contexts are "matched". The target object, T*, with highest matching score (or probability) will be the most possible target that S is referring to in the contexts of <Ls, Rs>.

The degree of matching can be measured in terms of different "matching strength" or "matching score" contributed by the contexts. Normally, *exact string match between* two terms in S/T or L/R contexts, such as "the Big Apple" *vs.* "the Big Apple", has the strongest match. But it is least robust since S/T/L/R might be described in terms of other synonymous form. *Partial or fuzzy match*, like "Big Apple" *vs.* "the Big Apple", provides some flexibility for matching. But it may also introduce noise such as matching "the Big Apple" against "Big Apple Pie". The most robust and flexible way for matching S/T and L/R contexts might be to assign a higher matching score to a term pair if they are known to be synonyms or highly related terms in the ontology. The idea behind the current work is an extension of the thoughts explained in the above research work areas. In the work presented here, instead of using just a dictionary based search, the search is conducted using the Meta data, a comprehensive rule base, which will generate the result.

SYSTEM OUTPUTS

The following snapshots interpret the various steps of the process involved in this application.

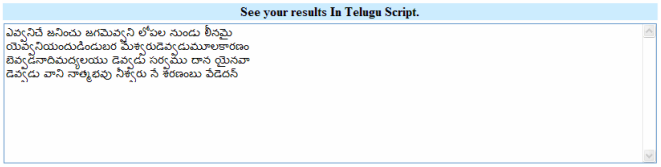


Fig 1: Poem in Native Language (‘Telugu’)

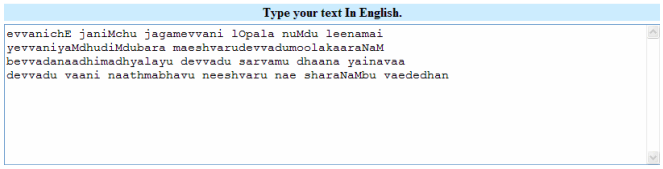


Fig 2: Transliterated form of the Poem

Fig 3: Poetic Pattern

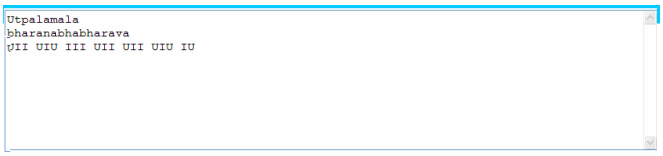


Fig 4: Poetic Analysis – Grammar Display

In the age of smart devices, the need for systems to be accessible everywhere as well as at all the times is considered and the application is delivered to meet these high-availability and high-accessibility needs.

IV. System Implementation

The system is implemented using Natural Language Processing techniques. A knowledge base is created and is dynamically updated based on the input database. A heuristic searching and sorting technique is applied using hashing technique to implement system performance.
This system can run on industry standard cloud based platforms like Microsoft Azure, Amazon cloud (AWS) etc.

V. Conclusion and Summary

This paper describes the sorting and search techniques for an application in natural language processing. Unlike most modern search algorithms that take advantage of the continuously increasing processing power of the modern day computers and hence lose elegance in the search technique, the developed search algorithm allows for the retrieval of the best possible solution in a very efficient manner while also taking into account of the overheads involved in execution of the algorithm. The implementation of the algorithm as the searching mechanism to find the most probable parse for the target parser has dramatically reduced the parsing time required to retrieve the same result as an exhaustive search mechanism. The characteristics of the algorithm has the potential to be converted into a simple chunk parser, which is sometimes enough to extract the relevant grammar ‘chandassu’ from the sentences. The proposed algorithm encourages the quick build-up of sub-parses, rather than the linear build-up algorithm of the exhaustive algorithms, hence the order in which the combination occurs allows for the splitting of the sentence into sections or chunks by early termination of the algorithm.
An example along with sample screen shots is presented for reference. Further study includes algorithm analysis in terms of complexity and system performance aspects, applicability to other forms of Telugu poetry like prose, sonnets etc., and applicability to other languages in the world such as ‘Sanskrit’, to analyze morphological and linguistic aspects. Similar concepts can be extended to business applications such as “Compliance and Regulatory” needs of an enterprise, legal processing systems, product behavior, consumer behavior, etc.

References

[1] Krishna, N. S., & Murthy, H. A. (2004, October). A new prosodic phrasing model for Indian language Telugu. In *INTERSPEECH*.
[2] Madhavi Ganapathiraju and Lori Levin, 2006, TelMore: Morphological Generator for Telugu Nouns and Verbs. Proc. Second International Conference on Universal Digital Library, Vol Alexandria, Egypt, Nov 17-19, 2006.
[3] Takeshi Matsumoto, David M.W.Powers & Geoff Jarad - Application of Search algorithms to NLP
[4] S. Rajendran, Ph.D. August 2006, PARSING IN TAMIL –PRESENT STATE OF ART –LANGUAGE IN INDIA, Strength for Today and Bright Hope for Tomorrow, Volume 6 : 8
[5] ‘Andhra Mahabhagavatamu’ – Bammera Pothana, Potti Sreeramulu Telugu University Press.

Author Profile

TVVV Prasad has received the M.Tech degree in Computer Science and Technology from Andhra University in Visakhapatnam. He has done Executive Leadership Program at Ross Business School, University of Michigan, USA. He has worked for Multinational Companies such as Tata Consultancy Services, Motorola and has more than 25 years of experience in implementing computer based solutions. He is now a Research Scholar at Rayalaseema University. He has participated several international conferences in knowledge based computing and business excellence.