Key to Crossword Solving : NLP

A R Jobin
Computer Science & Engineering
National Institute of Technology
Calicut, Kerala 673601
Email: jobinar007@gmail.com

Anand G Menon
Computer Science & Engineering
National Institute of Technology
Calicut, Kerala 673601
Email: anandgmenon02@gmail.com

Ashwin Sekhar
Computer Science & Engineering
National Institute of Technology
Calicut, Kerala 673601
Email: sekhar.ashwin@gmail.com

Vinay Damodaran
Computer Science & Engineering
National Institute of Technology
Calicut, Kerala 673601
Email: vinaydamodaran95@gmail.com

***Abstract*—The struggle of technology to understand natural language has been one of the greatest hurdles of humankind. Various techniques and algorithms have contributed to significant advancements in the field of NLP. One of the most primary challenges that NLP faces is being able to determine the meaning and essence of a sentence which may have multiple variations of syntax and semantics. Crossword solving involves utilizing the knowledge one has over the language and applying it with the various constraints that a clue will provide. As opposed to generic brute force crossword solvers, this work aims to implement automated solving of a crossword using knowledge-based concepts of NLP. The solver provides the user with a potential solution set which contains a collection of words out of which the user may choose the answer which he or she feels is most appropriate. Thus, the entire crossword can be solved by using the knowledge obtained from natural language processing, and this method stays true to the essence of crossword solving.**

I. Introduction

Natural Language Processing deals with the interaction between humans and machines using natural language and explores concepts used in computer science, computational linguistics, and artificial intelligence. Besides learning the words in a language, machines will also have to learn the syntax and semantics to meaningfully process it. The ambiguity of language makes NLP a tough problem for computers to master, even though it is one of the easiest things for a human to learn. From providing relief to readers during WWII blackouts, to serving as a pastime for enthusiasts of all ages, the role of the modern-day crossword has evolved to be one of the most beloved hobbies of the digital age. The reason crosswords have withstood the test of time, is its simple yet challenging way to engage and improve the mind's grasp of natural language. Here we attempt to create a crossword solving assistant using the principles of NLP.

Most automated crossword solvers attempt to solve the crosswords through random brute force method - inserting a word of the correct length at the first clue and proceeding with the next clue and backtracking at each instance in the case of an error or conflict. The generic solvers fail to grasp the essence of how a crossword was meant to be solved - to understand the clue and then solve it by applying our knowledge of the language. The solver designed and implemented as part of this work, albeit incapable of solving an entire crossword on its own, relies solely on the principles of NLP for determining a solution, primarily word sense disambiguation (WSD).

In Section II, the problem statement for the work has been elaborated. The research materials which have been referred are discussed in Section III. The design of the algorithm used (Section IV) and its implementation (Section V) have been explained in this paper. The aforementioned algorithm was tested on fifty random crosswords (quick category) from the Guardian newspaper and the results (refer Section VI) have been tabulated at the end of the paper.

II. Problem Statement

The work detailed here aims to implement a crossword solver, which aids the user by providing a solution set to a given clue, by applying knowledge-based concepts of NLP. The Clue Solving Algorithm (CSA for short) devised by us returns a list of the first fifty words that are most likely to be the answer. The user may choose the one that he feels is right and all answers to subsequent clues are generated based on the letters confirmed by the answer chosen. Based on this procedure, an entire crossword can be solved.

III. Related Works

The first crossword was published on December 21, 1913, by Arthur Wynne, a journalist from Liverpool. Crosswords, which were initially meant to be a fun pastime for readers who subscribed to national dailies, soon turned into a nationwide craze in the US. Up until the 1940s, the New York Times scathingly disapproved of the game claiming that "The craze evidently is dying out fast and in a few months it will be forgotten"[1]. The Times was one of the most prominent

978-1-5090-6367-3/17/$31.00 ©2017 IEEE

national papers that did not succumb to the pressure of including crosswords but later added their first crossword in 1942 following the Pearl Harbour bombings as the editors decided that "readers might need something to occupy themselves during blackouts"[2]. Today, the Times Crossword is one of the most prominent puzzles available. The Guardian published its first crossword in 1929 and over the years has expanded its assortment of puzzles into different categories like quick, speedy, cryptic and so on. Our work was tested on crosswords set by The Guardian (quick category) and the results have been recorded.

The primary premise of this work is that clues in a crossword can be solved by assuming that an answer will have a certain degree of influence based on the value of its path similarity. The relatedness of concepts that can be measured has been described in [3] and throws light on the various types of similarity that WordNet provides. The similarity measures are primarily divided into thesaurus-based and corpus-based. The thesaurus-based ones are further divided into 3 types: path, LCH, and Wu-Palmer. The path similarity for two concepts is the inverse of the distance between them, and this length is calculated by traversing through a semantic hierarchy tree which is provided by WordNet. The LCH similarity is merely the negative logarithm of the result obtained by path similarity while Wu-Palmer follows a metric of assigning edges with weighted distance. We have chosen to measure the relatedness using 'path' similarity (as opposed to the Wu-Palmer and LCH similarities). The corpus-based similarities, namely, Jiang-Conrath, Lin, and Resnik all give a higher priority to semantic relatedness rather than similarity and hence are avoided for this work.[4]

The Lesk algorithm[5] was one of the earliest WSD algorithms developed and provided a much needed breakthrough in the field of NLP. Michael Lesk succeeded in providing a method to differentiate between two senses of the same word used in different contexts based on their overlap count and this later proved to be the cornerstone of many other NLP techniques. Since the algorithm in itself was insufficient in providing accurate answers, we have proposed a modification of the same which, when combined with the aforementioned path similarity, is capable of determining satisfactory solutions. The modified Lesk algorithm has been described in the Design section.

IV. Design

The work here tries to complete a crossword by emulating the way the human mind works. Each clue is solved individually and subsequent clues are solved by taking into consideration the already existing letters from previously solved clues. An output list is generated for each individual clue and the user may choose which answer he or she feels is most suitable. The list generated has the words sorted in descending order of their relevance score. The relevance score of a word is used to determine its likelihood for it to be a solution to the clue sentence and this score is calculated using our unique Clue Solving Algorithm.

A. The Clue Solving Algorithm

The Clue Solving Algorithm (CSA for short) was designed so that the clues of a crossword can be solved by using the principles of NLP rather than relying on brute force techniques. Fig. 1. shows the method of assigning such a score for a word from the WordNet. The algorithm makes use of modified versions of the Lesk algorithm (given in Algorithm 1) and path similarity scores (given in Algorithm 2) to create a ranked list of words from which the user could choose the right answer. The detailed design for the algorithm is explained below:

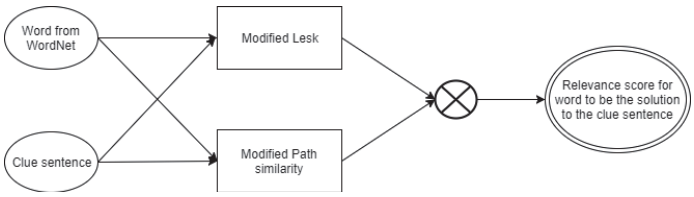

Fig. 1. Clue Solving Algorithm acting on a word in WordNet

1) Modified Lesk: The Lesk algorithm relies on the principle that if a word can have multiple senses in a sentence, then the definition of the sense that has the most number of words common with the sentence, is the most probable sense of the word in that sentence. Applying this principle, the modified Lesk algorithm calculates a Modified Lesk score for every word in WordNet.

First, it takes the context words of the definition of every synset of the word, and finds the intersection it has with the context words of the original sentence (*twords*) as well as the context words of the definition of every synset of the original context words (*cwords*). Each score is normalized by dividing it by the total number of matches possible (*twords* and *cwords* respectively), and these scores are then weighted appropriately in a 3:1 ratio with higher priority given to matches with the original sentence. This resultant value will be the modified Lesk score for a word.

2) Modified Path Similarity: The path similarity score of two senses gives us the smallest number of edges connected between them. Among the various similarity measures present, the thesaurus and corpus-based ones (Resnik, Lin, Jiang-Cornath) are rejected as we do not want the probabilistic distributions from other corpora to affect the factors in determining the clue. Path similarity is preferred over Wu-Palmer as weighted edges may affect the calculations adversely[3]. The similarity measure that was obtained between nouns and verbs are marginally different, so the scores have been normalized appropriately to minimize this disparity.

The inbuilt path similarity function in NLTK is used and one of the parameters taken to calculate this score is $\text{synset}_{\text{wn}}$, which is the collection of all the synsets in WordNet with the same PoS tag as the answer. Each member of $\text{synset}_{\text{wn}}$ is compared with the synsets (with same PoS tag) of the context words of the clue description ($\text{synset}_{\text{pos}}$) and the sum of these values is taken. The synset in $\text{synset}_{\text{wn}}$ with highest value is

Algorithm 1 Modified Lesk

Input: sent, k
$sent \leftarrow$ the clue which has to be solved
$k \leftarrow$ length of the word required

Output: List of words with Modified Lesk score

procedure MODIFIED LESK(sent, k)
  $output \leftarrow$ empty array
  $t_w \leftarrow$ words in *sent*
  $syn \leftarrow$ empty list
  for all w_i in t_w **do**
    for all synset s_i of w_i **do**
      $syn \leftarrow syn.append(s_i)$
    end for
  end for
  $c_w \leftarrow$ empty list
  for all s_i in *syn* **do**
    $d_i \leftarrow$ definition of synset s_i
    $c_i \leftarrow$ words of d_i
    $c_w \leftarrow c_w.append(c_i)$
  end for
  for all $word$ in $WordNet$ **do**
    if $length$ of $word = k$ **then**
      $lesk \leftarrow 0$
      for all $synset$ of $word$ **do**
        $d_s \leftarrow$ definition of $synset$
        $d_w \leftarrow$ words of d_s
        $l_1 \leftarrow count(d_w \cap t_w)/count(t_w)$
        $l_2 \leftarrow count(d_w \cap c_w)/count(c_w)$
        $lesk \leftarrow max(lesk, l_1 * 0.75 + l_2 * 0.25)$
      end for
      $output[word] \leftarrow lesk$
    end if
  end for
  return $output$
end procedure

Algorithm 2 Modified Path Similarity

Input: sent, k, pos
$sent \leftarrow$ the clue which has to be solved
$k \leftarrow$ length of the word required
$pos \leftarrow$ the PoS category to which the word belongs
#The user can select $noun$, $verb$, $adjective$ and $adverb$.
In case PoS is $unknown$, all of the above tags are checked.

Output: List of words with Modified Path Similarity score

procedure MODIFIED PATH SIMILARITY(sent, k, pos)
  $output \leftarrow$ empty array
  $twords \leftarrow$ words in *sent*
  $syn_{pos} \leftarrow$ empty list
  for all w_i in $twords$ **do**
    for all synset s_i of w_i **do**
      if $PartOfSpeech(s_i) = pos$ **then**
        $syn_{pos} \leftarrow syn_{pos}.append(s_i)$
      end if
    end for
  end for
  for all $word$ in $WordNet$ **do**
    if $length$ of $word = k$ **then**
      $sim_{pos} \leftarrow 0$
      for all $synset$ of $word$ **do**
        if $PartOfSpeech(synset) = pos$ **then**
          $tmp \leftarrow 0$
          for all p_i in syn_{pos} **do**
            $tmp \leftarrow tmp + sim_{path}(synset, p_i)$
          end for
          $sim_{pos} \leftarrow max(sim_{pos}, tmp)$
        end if
      end for
      $output[word] \leftarrow sim_{pos}$
    end if
  end for
  $score_{max} \leftarrow max\_value(output)$
  for all $word$ in $output$ **do**
    $output[word] \leftarrow output[word]/score_{max}$
  end for
  return $output$
end procedure

selected for each clue. But since the score varies substantially for different PoS tags, all the values are divided by the max score for their corresponding PoS tag. This normalized value will be the modified path similarity score for a word.

The sum of both these scores for a particular word is calculated and the resulting list is sorted in descending order and returned. Results obtained with this algorithm are given in Section VI.

V. IMPLEMENTATION

The different phases in the implementation are captured in Fig. 2 given in the next page.

The first part that the user interacts with is the UI in which he/she is required to create a crossword, clue by clue. The following inputs are collected from the user: clue details (which include clue number, clue description, clue direction), answer length, clue type (single or compound), answer type (noun, verb, adjective, adverb and unknown). After the entire crossword has been entered thus, we move to the solving part.

All these clues have been stored in a database from where a Python implementation of our CSA (refer subsection IV-A) can retrieve and solve them. On selecting each clue, you will have the option to either directly type in the answer or solve it. On choosing solve, a list of the most likely fifty words are generated and the user can choose his or her answer from that list. If any of the letters in the clue have already been revealed, then the list will contain words that match the required regex. The output received *i.e.*, the answer list and answer selected

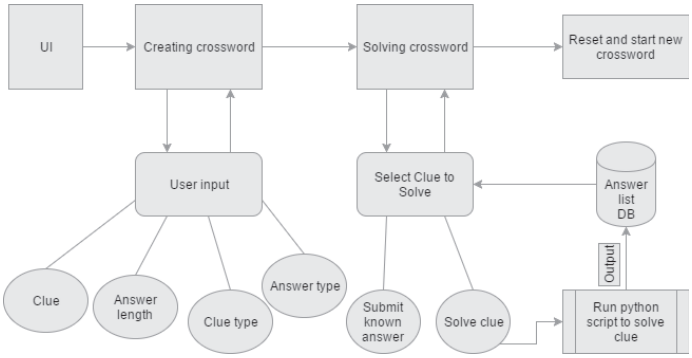

Fig. 2. Working of the crossword solving assistant

is stored in the database which can be accessed and modified at a later stage (in the case of any wrong answers). After completing the crossword, the user has the option to reset it and begin a new one.

The Clue Solving Algorithm (refer Section IV) was implemented in Python and makes use of the NLTK library which is designed especially for computational NLP. The UI of the project was designed using HTML, JS, and CSS. The web framework for this work is a python-based application known as Django and this facilitated easy migration of the python modules used for the CSA. The user inputs and other clue details are stored in and retrieved from MySQL databases with the help of the aforementioned framework. The complete project is available on GitHub[6].

VI. Results

The preceding algorithm was tested on 50 different random crosswords (quick category) from the Guardian website, dating from 28 October 2016 to 12 May 2017[7]. The effectiveness of the solver is based on the position of the answer in the list that is generated as output. Out of the total 1221 clues that were tested, the following results were obtained:

Rank of correct solution in output list			
1^{st}	2^{nd} - 5^{th}	6^{th} - 50^{th}	Does not appear
658	168	56	339

27.76% of the total clues could not be solved by this project. The clues that could not be solved have been categorized into two types: WordNet limitations and Implementation limitations. These categories have been further classified into 4 and 2 groups respectively.

Could Not Solve	
WordNet	Implementation
321	18

A. WordNet limitations

There are certain limitations that a person faces while using WordNet and this accounts for 94.69% of the total clues that could not be solved.

1) Lack of thesaurus: The lack of a better thesaurus to include words which may be in their past tense or which may be direct synonyms of one another have resulted in 16.9% of the clues to remain unsolved. This contributes to around 61% of the total clues which could not be solved.

2) Pop Culture References: WordNet is not equipped to handle modern popular culture references as clues and this contributes for 4.75% of the total clues and 17.10% of the clues unsolved.

3) Not popular phrases: Most clues in the guardian crossword make use of modern-day phrases which WordNet can only identify as single words and not as a single phrase. This, however, only accounts for 2.54% of the total clues and is responsible for 9.14% of the clues unsolved.

4) Colloquial clues: Some clues are given in day-to-day slang and WordNet is unable to register their intended meaning. This roughly estimates to 2.04% of the total set and 7.37% of the unsolved.

WordNet Limitations			
Thesaurus	Pop Culture	Unpopular	Colloquial
207	58	31	25

B. Implementation limitations

5.31% of the clues unsolved are due to limitations created by the way the project was designed. The categories are listed as follows:

1) Compound words: The work is not equipped to handle clues which have more than 2 words as an answer. This amounts to 3.54% of the clues unsolved and nearly 1% of the total clues.

2) Anagrams: Some answers to the clues are anagrams to the clues itself and they are not handled by the solver. However, they only account for a small minority, representing 1.77% of the clues unsolved and 0.5% of the total clues.

Implementation Limitations	
Compound Words	Anagrams
12	6

For a full list of the crosswords that were used, please refer the citations[7].

VII. Conclusion

In this work, we have attempted to develop a solver for simple crosswords using NLP techniques. Existing NLP algorithms are modified and combined to generate a relevance score based on which, each clue sentence given to the solver is provided with a set of possible solutions. With minimal user input, our solver achieves an impressive level of 72% accuracy, where the clue's correct answer appears in the output list provided by the solver. The assistance of a better thesaurus and inclusion of extra dictionaries could improve the accuracy in our tests upto around 90%. This work points to the effectiveness of current NLP techniques in solving a majority of clues in simple crosswords.

VIII. Acknowledgment

We acknowledge the support extended by Dr. Vinod Pathari and Dr. Vasudevan N of the Computer Science and Engineering Department, NIT Calicut during the different stages of this work.

References

[1] Topics of the Times, *"Sees Harm, Not Education"*, The New York Times, March 10, 1925, p. 20

[2] Topics of the Times, *"Bambi is a Stag and Tubas Don't Go 'Pah-Pah': The Ins and Outs of Across and Down"*, The New York Times Magazine, 1992-02-16. Retrieved on 2009-03-13.

[3] T. Pedersen, S. Patwardhan, J. Michelizzi, *WordNet::Similarity - Measuring the Relatedness of Concepts*, University of Minnesota, Duluth. 2004. http://dl.acm.org/citation.cfm?id=1614025.1614037

[4] Dan Jurafsky, James H. Martin, *Computational Lexical Semantics*, Speech and Language Processing 2nd Edition p. 653

[5] M. Lesk, *Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone*, Toronto, Ontario, Canada. 1986. http://doi.acm.org/10.1145/318723.318728

[6] https://github.com/srjobinar/Major

[7] The following crosswords from the Guardian (quick category) were used for result analysis: *14500;14507;14512;14515-16;14518-21;14523;14526-28;14530;14534;14539;14540;14544;14546;14563; 14567;14570;14572;14576;14580;14585;14593;14598-99;14621;14624-25;14627;14630;14634-35;14637;14641;14644;14648-50;14652;14654-57;14659-60;14668.*