Algorithm for developing Urdu Probabilistic Parser

Neelam Mukhtar, Mohammad Abid Khan and Fatima Tuz Zuhra
Department of Computer Science, University of Peshawar, Khyber Pakhtoonkhawa, Pakistan
Emails: sameen_gul@yahoo.com; abid_khan1961@yahoo.com; fateeshah@yahoo.com

Abstract- Any decision error in greedy search procedure results in a wrong parse. Best-first strategy is complicated. The novel algorithm developed in this work is based on multi-path shift reduce-strategy. In this algorithm, if the input matches with more than one clause on the right hand side of a rule in Urdu probabilistic context free grammar the items on the old stack are copied to a new processing stack and subsequently reduce operation is performed. A number of stacks can be created whenever required. The probabilities of the rules can be added for each path. Parse tree with the highest probability is selected as the correct solution. Dry run of the main algorithm shows the superiority of this algorithm over its previous counter parts.

1. INTRODUCTION

There is a strong Perso-(Persian/Arabic) influence on Urdu Vocabulary. Urdu is written in a cursive, context-sensitive Perso-Arabic script from right to left. The morphological richness of Urdu is making this language a highly challenging language because Urdu has inherent grammatical forms and borrowed vocabulary from different languages such as Arabic, Persian, Turkish and the native languages of South Asia [1].

In Urdu, research is going on from different point of views such as creating an Urdu corpus [2, 3] and tagging the Urdu corpus [4]. Researchers have proposed different tagsets for Urdu whose number of tags is ranging from 10 [5] to 350 [6]. A Finite-state morphological analyzer for Urdu is already developed [7]. Now, one of the demanding areas is to parse Urdu corpora. Considerable amount of work has not yet been done in this direction. An efficient and accurate parser is needed to parse Urdu corpus of natural text.

A probabilistic parser is needed to parse Urdu sentences for better efficiency and accuracy compared to traditional rule-based parsers. Before developing such a parser, a Probabilistic Context Free Grammar (PCFG) is required. A PCFG is a Context Free Grammar (CFG) with probabilities assigned to grammar rules that can accommodate the ambiguity and the need for robustness in real-world applications more efficiently [8]. A PCFG for Urdu probabilistic parser is already developed [9].

Part of the table showing Urdu PCFG, with 127 rules is given below:

<table>
<thead>
<tr>
<th>S.No</th>
<th>Rules</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>S-&gt;NP VP</td>
<td>0.9637</td>
</tr>
<tr>
<td>2.</td>
<td>S-&gt;PP VP</td>
<td>0.0167</td>
</tr>
<tr>
<td>3.</td>
<td>S-&gt;PP NP</td>
<td>0.0019</td>
</tr>
<tr>
<td>4.</td>
<td>S-&gt;VP NP</td>
<td>0.0109</td>
</tr>
<tr>
<td>5.</td>
<td>S-&gt;VP PP</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

Parsing algorithms are typically grouped according to the direction in which the parse tree is constructed. The two most common types of parsing are top-down and bottom-up, though there are parsing algorithms that are of other types and there are some that are a combination of these two [10].

Top down parsers start with the start symbol S (sentence) and expand the sentence by applying the productions until the desired string is reached. In Bottom up parsing, the sequence of symbols are taken and compared to the right hand side of the rules. So the parser starts with the words (from the bottom) and attempts to build a tree from the words up [11]. The bottom up parsers (commonly known as shift reduce parsers) work by “shifting” input tokens to the top of the stack. This process continues until the top of the stack matches the right hand side of the production rule. At this stage “reduce” operation is then performed. The production’s right hand side is replaced by its left hand side. The process of reduction continues till the string is reduced to the start symbol of the grammar.
Shift-reduce strategy is commonly used in bottom up parsing. Statistical parsers, based on a shift-reduce algorithm, are already developed [12, 13, and 14]. Bottom-up algorithm (essentially shift-reduce parsing) is used by [12]. They allowed multiple passes over the input. Nivre and Scholz’s parser [13] is somehow similar to [12] as both parsers are deterministic parsers with few differences. Deterministic parsers for constituent structures are also developed [14, 15]. Usually, these parsers perform parsing by taking a series of shift reduce decisions. Typically, in such models there is a single stack and a queue (where the stack keeps the parsed fragments and the queue holds the unprocessed tokens). Broadly, two actions are performed, a shift action and a reduce action. Item on the front of the queue is moved to the top of stack when performing shift operation whereas one/several elements on the top of the stack are combined to form a larger fragment when reduce action is performed [16]. Previous experiments indicate that such models give an acceptable level of performance at very high processing speed. But there is a high risk of a serious problem. The transitions may follow a wrong path. As only a single pair of stack and queue is used, there is only one transition path. In a deterministic way, one state moves into another state. Any decision error results in a wrong parse. A best first probabilistic shift reduce parser is proposed [17], that performs the best-first search instead of considering a single analysis path that is done in a deterministic fashion. It uses a prior heap that contains different possible states. The top-most state with the highest probability is selected and expanded. This strategy is complicated. Multi-path shift-reduce parsing is proposed that keeps multiple transition paths during decoding [16]. It allows several best derived states after each expansion. The idea of multi-path shift-reduce parsing is used in developing algorithm for Urdu probabilistic parser. Work in different sections in the paper is organized as follows:

In section 2, the proposed method is discussed. In section 3, four algorithms are discussed. In section 4, dry run of the main algorithm is presented. In section 5, conclusion and limitations are given.

II. PROPOSED METHOD

Multi-path shift-reduce parsing is proposed that keeps multiple transition paths during decoding [16]. It allows several best derived states after each expansion. The idea of multi-path shift-reduce parsing is used in developing algorithm for Urdu probabilistic parser. This idea is modified for the improvements of results. Special features of this work are given:

a. A structured array is used, where at each location there is:
1. Identification whether to take shift action or reduce action.
2. A separate input buffer.
3. A separate processing stack.
4. A separate output stack.
5. A variable for calculating the total probability of the parse in the cell of the array.
b. The output goes to the output stack.
c. The correct parse is obtained by selecting the parse with the highest probability.

While using shift-reduce multi-path strategy, the first input from the buffer is pushed to the processing stack. If the input matches the (whole) right hand side of a single rule then simple reduce action is performed. In case of ambiguity, e.g. if the input matches with the right hand side of more than one rule, then as many stacks are created as are the number of matches and the item(s) on the old stacks are copied to the corresponding newly created processing stacks. Reduce action is performed on each new processing stack, thus allowing multiple transition paths. With each newly created processing stack, there is a corresponding input buffer, an output stack and a variable, at another memory location in the structured array. The input buffer contains the inputs (called input tokens). The output stack has the output (i.e. successive rules that are used by ‘reduce’ operation, along with the associated probabilities). The variable contains the probabilities. After each reduce action, the probability of the rule (i.e. probability assigned to this rule in Urdu probabilistic context free grammar), is added/multiplied to the probability of the previous successive rule. These results are accumulated in the variable mentioned above. This procedure is continued till there is only a $ sign ($ is automatically added by the software to the input buffer in the beginning) left in the input buffer, showing the end of the input. At this stage, the top of each output stack is checked. If it is “successful”, then the corresponding structure item contains a successful parse. Total of the probabilities (accumulated in the variable in each cell of the array) for each transition path is compared with one another. The parse with the highest probability is selected. The output of this parse is popped from the output stack and put in reverse order in the user interface. This is the most suitable parse of the sentence. The probabilistic parser displays the structure of the sentence.
III. THE ALGORITHM

The algorithm developed for this work has four parts, i.e., a, b, c, and d. Part a is the main algorithm whereas parts b, c, and d are sub-algorithms. Algorithm a is for developing Urdu probabilistic parser. Algorithm b shows the procedure for checking/comparing the input on the top of the processing stack with the right hand side of the rules. Algorithm c is used for copying items onto a new processing stack. Algorithm d is finding the highest probability. Parts of the algorithm are discussed in detail below.

a. Algorithm for Urdu probabilistic parser
1. Read POS (Part-of-Speech) tagged text into input buffer (from already tagged data).
2. Create a new parse record.
3. Call algorithm b.
4. Repeat step 1 to 3 for each parse.
5. Calculate total probabilities for all the parses.
6. Call algorithm d i.e. compare the total probability of each parse and select the parsed output with the highest total probability.

This algorithm starts by taking input from the input buffer and taking different steps. It ends when a $ sign is left in the input buffer. At this stage the parser shows the result by selecting the parse with the highest probability from a list of candidate parses in the form of rules thus showing the structure of the input sentence.

b. Checking the top of the stack against the right hand side of the rules
1. Read one rule at a time.
2. Take right hand side of the rule.
3. Pop item from the top of the processing stack and compare it with the right hand side of the rule. If there is no match, then pop the next item and compare the sequence of both against the right hand side. (Check up to the bottom of the processing stack).
4. If matched, call algorithm c and then perform Reduce i.e. Push the left hand side of the given production and write the rule in the output along with its probability.
5. If match found with more than one right hand side, then create additional parses according to the number of matches. Copy the data on the current processing stack to each new processing stack that is created and the current output to each new output stack created for new parses (successive rules with their probabilities are copied to each new output stack).
6. Repeat steps 4 and 5 for all the productions.

7. If no match found up to the bottom of the processing stack, perform shift operation.

This algorithm performs shift-reduce operation by checking the right hand side of the rules in Urdu probabilistic context free grammar. The successive rules (each along with its probability) are copied to the corresponding output stack after each reduce action.

c. Copying from a stack to a new processing stack
During the execution of algorithm b, both in step 4 and 5, the contents of a processing stack needs to be copied into another processing stack before the reduce operation is performed. The program segment for this copying operation in C# is given:

```
max = 0; // initialization
for (i= 0; i < arrsize; i++) // repeat till loop counter is less than size of array of parses.
```

The above segment copies all the processing stack items in the array. The next step is that the items from the array are inserted in reverse order into the new processing stack.

d. Highest probability
As mentioned earlier, a structured array is used to store the results of multiple parses of the same string. Here one has to select the most suitable parse. It is already discussed that a variable is maintained in each cell of the array containing the total probability of that parse. The value of the variable is extracted from each cell and compared. The maximum of them is selected. The associated output with this probability is the true parse. Thus disambiguation is performed on the basis of knowledge of probability. The program segment in C# for getting maximum probability is given below:

```
max = 0; // initialization
for (i= 0; i < arrsize; i++) // repeat till loop counter is less than size of array of parses.
```
{  
    if \( p[i] > \text{max} \); // comparison for getting  
    \text{max} = \text{p}[i]; // assignment of maximum value  
}

Output = \( i \); // assignment of output

The block diagram showing the relationship of these four algorithms is presented in Fig 1, while Fig 2, Fig 3, Fig 4 and Fig 5 show the flow charts for algorithm a, b, c and d respectively.

---

**Fig.1: Block diagram of the main algorithm and three sub-algorithms**
Fig 2: Flowchart for the main algorithm (Algorithm a)
Fig 3: Checking the top of the processing stacks (Algorithm b)
Fig 4: Copying from an old stack to a new stack (Algorithm c)
Fig 5: Calculating the highest probability (Algorithm d)
IV. DRY RUN OF THE MAIN ALGORITHM

Before actually implementing the main algorithm, the process of dry running the algorithm was carried out. In this phase, the accuracy of the algorithm was checked by performing all the steps according to the algorithm manually (by taking 50 sentences). The results were carefully observed. Results guaranteed the success of the algorithm before implementation. A section of the results is presented in Table 1.

Table 1: Manual testing of the algorithm before implementation

<table>
<thead>
<tr>
<th>Input buffer</th>
<th>Model</th>
<th>Output file</th>
<th>Rules used</th>
<th>Stack 1</th>
<th>Output file</th>
<th>Rules used</th>
</tr>
</thead>
<tbody>
<tr>
<td>First input</td>
<td>$NP</td>
<td>$NP (reduced)</td>
<td>NNP</td>
<td>$NP</td>
<td>$NP (reduced)</td>
<td>NNP</td>
</tr>
<tr>
<td>Second input</td>
<td>$NP</td>
<td>$NP (reduced)</td>
<td>NNP</td>
<td>$NP</td>
<td>$NP (reduced)</td>
<td>NNP</td>
</tr>
</tbody>
</table>

In the above table, there are two input tokens in the sentence i.e. این and پشتہ with tags NNP and NNP (where NNP stands for proper noun). The following rules are used:

\[ S \rightarrow NP \]
\[ NP \rightarrow NNP \]
\[ NP \rightarrow NP \]

S stands for sentence and NP stands for noun phrase. The first input to the stack is این (NNP). It is reduced by using the rule, NP->NNP. Now we have $NP on the top of the stack. We have another rule, S->NP. So another stack is created. First all the old items are copied and then reduce operation is performed on the second stack by using this rule. On the top of one stack, there is $NP and on the top of second stack the value is $S. Now the second input is popped which is again NNP. On the first stack, there is presently $NP NNP and on the second stack there is $S NNP. On both stacks reduce operation is performed using the rule, NP->NNP. So the status of the first stack is now $NP NNP while the status of the second stack is $S NP. Again reduce operation is performed on the first stack using the rule, NP->NP NNP. As there is no rule for the result ($S NP) on the second stack, it gives result as “unsuccessful parse”. We have $NP at the top of the first stack. Reduce operation is performed again using the rule $NP. On the first stack we have $S and the input array is empty. The result on this stack is “successful parse”.

In case of two elements on the top of the stack (i.e. NP VP, where NP is used for noun phrase and VP is used for verb phrase), if a single element (e.g. VP) on the top of the stack matches the right hand side of any rule (i.e. S->VP), it is reduced. At the same time, there is a chance that both elements also match any other rule (i.e. S->NP VP). In such a case, if reduce action is performed on any one of the two elements (one of the two rules mentioned above is used), then the possibility for the reduction of the second one is lost (if the rule, S->VP is used, then on the top of the stack there is NP S left and now it is not possible to use the second rule, S->NP VP). To overcome this problem, first the elements on the top of the stack are copied, then reduce operation is performed, so that there is a chance of reduction for the second element.

V. CONCLUSION, LIMITATIONS AND FUTURE WORK

The idea of multi-path shift-reduce strategy is used with a number of modifications for improvements. The algorithm (with four parts) is developed to provide a set of phrases based on successive rules (with highest probability) for a sentence, showing the structure of the sentence. At present the main limitation is that as fixed number of stacks (both processing and output) is used in a structured array so the memory requirement is very large. One solution to this problem is to use linked lists that will decrease the memory requirement, but at the cost of making the program more complex. As memory is not very expensive, a compromise is made on consuming large memory instead of making the program more complex by using linked lists. In future the algorithm will be implemented by using appropriate tool to get a probabilistic parser that will show the structure of the sentence. Presently the grammar (Urdu PCFG) developed, is ambiguous. By disambiguating the grammar, the performance can be improved a lot.

REFERENCES


