EXTENSION OF HVS SEMANTIC PARSER BY ALLOWING LEFT-RIGHT BRANCHING

Filip Jurčíček, Jan Švec, and Luděk Müller

Department of Cybernetics, Faculty of Applied Sciences, University of West Bohemia,
Pilsen, Czech Republic

{filip,honzas,muller}@kky.zcu.cz

ABSTRACT

The hidden vector state (HVS) parser is a popular method for semantic parsing. It is used in the language understanding module of the statistical based spoken dialog system. This paper presents an extension of the HVS semantic parser. It enables the parser to generate broader class of semantic trees. This modification can be used to improve the performance of the parser by generating not only the right-branching trees (like original HVS parser) but also limited left-branching trees and their combinations. The extension retains simplicity and properties of the original HVS parser. We tested the method on Czech human-human train timetable corpus. The modified HVS parser yields statistically significant improvement. The accuracy of the system increased from 50.4% to 58.3% absolutely.

Index Terms— semantic analysis, spoken dialog systems, spoken language understanding

1. INTRODUCTION

Communicating with a user in a form of spoken language seems to be an effective and comfortable tool to obtain exact travel, tourist, or cultural information in modern computer information retrieval dialog systems. One of the key technologies in such conversational systems is spoken language understanding. The understanding involves identifying the semantics from the query, and subsequently retrieving the relevant information to produce an suitable response. The research is motivated by the increasing demand of complex user initiated spoken dialogs with multiple intentions and attributes per dialog turn.

In this article, we focus on the statistical semantic parsing which we understand as a search process of the sequence of concepts $S = c_1, c_2, \ldots, c_T$ that has the maximum aposteriori probability $P(S|W)$ for the word sequence observation $W = w_1, w_2, \ldots, w_T$. A semantic concept is considered to be a basic unit of a particular meaning (for example DEPARTURE, STATION, TIME). The search can be described as

$$S^* = \underset{S}{\operatorname{argmax}}\, P(S|W) = \underset{S}{\operatorname{argmax}}\, P(W|S)P(S) \qquad (1)$$

where $P(S)$ is called the semantic model and $P(W|S)$ is called the lexical model. The semantic model $P(S)$ provides the probability of the semantics S. The lexical model $P(W|S)$ assigns a probability to the underlying utterance (word/lexical sequence) W given the semantics S.

He and Young proposed a hidden vector state (HVS) parser [1] which performs very well on the ATIS task [2] even though the HVS parser allows to generate right-branching semantic trees only. We present an extension of the HVS parser which enables it to generate not only right-branching semantic trees but also limited left-branching semantic trees and their combinations. In right-branching languages, for example Spanish, adjectives usually follow nouns, direct objects follow verbs and prepositions. On the opposite, in left-branching languages, for example Japanese, adjectives precede nouns, direct objects come before verbs, and there are postpositions. Nevertheless, for most languages, the main branching tendency is just a tendency with a number of exceptions. For instance, English shows left branching at the level of noun phrases but it is mostly right-branching at the sentence level. In case of Czech, we found that a substantial portion of parsing errors of the original HVS parser was caused by the inability to produce left-branching semantic trees. As a result, the possibility of at least limited left-branching parsing is convenient.

2. THE HIDDEN VECTOR STATE PARSER

The HVS parser is an approximation of a pushdown automaton. A *vector state* in the HVS parser represents a stack of a pushdown automaton which keeps semantic concepts assigned to several words (see Figure 1).

The transitions between vector states are modeled by stack operations: popping from zero to four concepts from the stack, pushing a new concept onto the stack, and generating a word. The first two operations belong to the semantic model which is given by:

$$P(S) = \prod_{t=1}^{T} P(pop_t|c_{t-1}[1,\ldots 4])P(c_t[1]|c_t[2,\ldots 4]) \qquad (2)$$

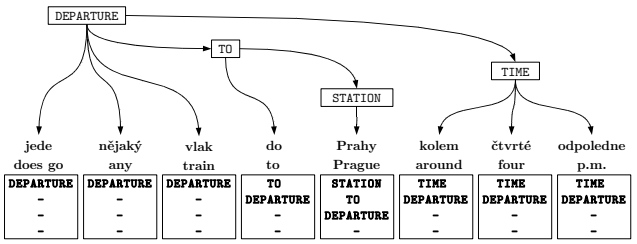

Fig. 1. An example of a full semantic parse tree with the corresponding stack sequence. Corresponding abstract semantic tree: DEPARTURE(TO(STATION), TIME).

where pop_t is the vector stack shift operation and takes values in the range $0, 1, \ldots, 4$, and c_t at a word position t is a vector state consisting of 4 concepts, i.e. $c_t = [c_t[1], c_t[2], c_t[3], c_t[4]]$ (shortly $c_t[1, \ldots 4]$), where $c_t[1]$ is a preterminal concept dominating the word w_t and $c_t[4]$ is a root concept. Note the concept $c_t[1]$ in the stack is always the most recent concept pushed onto the stack (the concept $c_t[1]$ is the topmost one in the depiction of the stack in Figure 1).

The probability $P(pop_t|c_{t-1}[1, \ldots 4])$ represents a model for popping 0 to 4 concepts from the stack. The variable pop_t defines the number of concepts which will be popped off the stack.

The lexical model performs the last operation - generation of a word. The lexical model is defined as

$$P(W|S) = \prod_{t=1}^{T} P(w_t|c_t[1, \ldots 4]) \tag{3}$$

where $P(w_t|c_t[1, \ldots 4])$ is a conditional probability of generation of a word w_t given $c_t[1, \ldots 4]$. For more details about the HVS parser see [1].

2.1. Training data

It is possible to train the HVS parser using so called abstract semantic annotations, which do not explicitly provide the alignment between the semantic tree and the words of the utterance. For the utterance "*Jede nějaký vlak do Prahy kolem čtvrté odpoledne*" (Lit.: "*Does any train go to Prague around four p.m.?*") (see Figure 1), the corresponding semantics is DEPARTURE(TO(STATION),TIME). For more details see [3].

The acquisition of the abstract semantic annotation does not need to be too expensive because it is not necessary to obtain fully annotated parse trees. The full parse tree defines not only the tree structure but also the alignment of all words to the leafs of the tree. Dialogue annotators have to define only the semantic tree that represents the meaning of each training utterance but they do not have to provide a full parse tree. An example of the full parse tree of the previous utterance is illustrated in Figure 1.

3. LEFT-RIGHT-BRANCHING PARSING

In Section 2, we described the original HVS parser. It pops out from zero to four concepts from and pushes *exactly* one concept onto the stack for every input word. These pop and push operations determine that the class of the generated semantic trees have to be right-branching.

To extend the class of the generated semantic trees we propose the following modifications. Firstly, in Section 3.1, we will replace the implicit (deterministic) pushing one concept onto the stack for every input word with an explicit pushing operation of zero or one concept onto the stack so that the parser will be prepared to be easily extended for limited left-branching parsing. Secondly, in Section 3.2, we will extend the pushing operation to pushing zero, one, or two concepts. The pushing zero, one, or two concepts will allow the parser to generate not only the right-branching parse trees but also the limited left-branching parse trees or their combinations. Left-branching parts of parse trees have the limited depth which cannot exceed the maximum number of concepts pushed at once.

3.1. Push operation

This section describes a modification of the HVS parser which we call the HVS parser with probabilistic pushing (HVS-PP).

To enable either pushing no concept or one concept onto the stack, we add a new hidden variable $push$ into the HVS parser (Equation 2). As a result, the semantic model is as follows:

$$P(S) = \prod_{t=1}^{T} P(pop_t|c_{t-1}[1, \ldots 4]) P(push_t|c_{t-1}[1, \ldots 4]) \cdot \begin{cases} 1 & \text{if } push_t = 0 \\ P(c_t[1]|c_t[2, \ldots 4]) & \text{if } push_t = 1 \end{cases} \tag{4}$$

where $push_t$ is a new hidden variable representing the pushing operation which takes values 0 for pushing no concept and 1 for pushing one concept onto the stack. If there is no pushed concept onto the stack ($push_t = 0$), then the transition between two neighboring chunks is modeled by the product $P(pop_t|c_{t-1}[1, \ldots 4])P(push_t = 0|c_{t-1}[1, \ldots 4])$. If there is one concept pushed onto the stack ($push_t = 1$), then the transition between two neighboring chunks is modeled by the product $P(pop_t|c_{t-1}[1, \ldots 4])P(push_t = 1|c_{t-1}[1, \ldots 4])P(c_t[1]|c_t[2, \ldots 4])$.

3.2. Implementation of left-right-branching

In the previous section, we introduced the $push$ hidden variable of the HVS parser; however, its value was limited to be either 0 or 1. The previous modification in the structure of the HVS parser was the first step in enabling the HVS parser

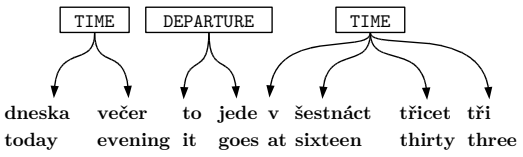

Fig. 2. An incorrect parse tree from a right-branching parser.

to insert two concepts at the same time. In the second step we enable the parser to push more than one concept onto the stack so the parser can now generate both the right- and left-branching parse trees and furthermore also their combinations. As a result, we call this parser the left-right-branching HVS (LRB-HVS) parser.

However, because our error analysis of the baseline HVS model on HHTT corpus [3] did not find errors caused by the inability to insert more than two concepts, we limited the number of concepts inserted onto the stack at the same time to two, but in general, the LRB modification is not limited to two inserted concepts. It can be straightforward to extend the number of inserted concepts to more than two.

To illustrate the difference between right-branching and left-right-branching, we can use the utterance "*Dneska večer to jede v šestnáct třicet tři*" (Lit.: "*Today, in the evening, it goes at four thirty p.m.*") .

The incorrect parse tree (Figure 2) is represented by the semantic annotation TIME, DEPARTURE, TIME. Such parse tree would be an output of the HVS-PP parser, which allows to generate right-branching parse trees only; the parser is not able to push more than one concept onto the stack. However, the correct parse tree is represented by the semantic annotation DEPARTURE(TIME, . . . , TIME). The correct parse tree (Figure 3) would be an output of LRB-HVS parser because it is able to push two concepts DEPARTURE and TIME onto the stack at the same time so that the first word *dneska* (Lit.: *today*) can be labeled with the hidden vector state [TIME, DEPARTURE].

Because the $push$ variable takes values in range $0, 1, 2$, the semantic model is as follows:

$$P(S) = \prod_{t=1}^{T} P(pop_t|c_{t-1}[1,\dots 4])P(push_t|c_{t-1}[1,\dots 4])\cdot \cdot \begin{cases} 1 & \text{if } push_t = 0 \\ P(c_t[1]|c_t[2,\dots 4]) & \text{if } push_t = 1 \\ P(c_t[1]|c_t[2,\dots 4])P(c_t[2]|c_t[3,4]) & \text{if } push_t = 2 \end{cases} \quad (5)$$

In the case of inserting two concepts onto the stack ($push_t = 2$), we approximate the probability $P(c_t[1,2]|c_t[3,4])$ by $P(c_t[1]|c_t[2,\dots 4])P(c_t[2]|c_t[3,4])$ in order to obtain more robust semantic model $P(S)$.

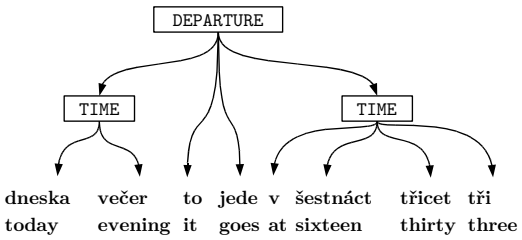

Fig. 3. A correct parse tree from a left-right-branching parser.

3.3. Model complexity

He and Young examined the complexity of the HVS model. The design of the HVS model ensures that the number of model parameters is linear in the stack depth, number of concept labels, and vocabulary size [1]. The additional complexity of the probability tables $P(push_t|c_{t-1}[1,\dots 4])$, $P(c_t[1]|c_t[2,\dots 4])$ and $P(c_t[2]|c_t[3,4])$ are still linear in these parameters. Therefore the LRB-HVS preserves the complexity of the baseline HVS model.

4. EXPERIMENTS

The semantic parsers evaluated in this article were trained and tested on the Czech human-human train timetable (HHTT) dialog corpus, which was described in [3]. The HHTT corpus consists of 1,109 dialogs completely annotated with semantic annotations. Both operators and users have been annotated. It has 17,900 utterances in total. The vocabulary size is 2,872 words. There are 35 semantic concepts in the HHTT corpus. The dialogs were divided into training data (798 dialogs - 12972 segments, 72%), development data (88 dialogs - 1,418 segments, 8%), and test data (223 dialogs - 3,510 segments, 20%). Each segment has assigned exactly one abstract semantic annotation. The development data were used for finding the optimal concept insertion penalties and the optimal semantic model weights [4].

The training of the semantic and the lexical models of HVS parser is divided into three parts: (1) initialization, (2) estimation, (3) smoothing. All probabilities are initialized uniformly. To estimate the parameters of the models, it is necessary to use the expectation-maximization (EM) algorithm because the abstract semantic annotations do not provide full parse trees. We smooth all the probability tables using a back-off model. To build the semantic parser, we used the Graphical modeling toolkit (GMTK) [5].

Both the baseline and the proposed modification were trained and evaluated using the corresponding transcriptions. We evaluated our experiments using two measures: semantic accuracy and concept accuracy. These measures compare the reference tree, which annotates the transcription, with the hypothesis tree, which is found using the search process.

4.1. Semantic accuracy

The reference and the hypothesis annotations are considered equal only if they exactly match each other. The semantic accuracy of a hypothesis is defined as $SAcc = \frac{E}{N} \cdot 100\%$, where N is the number of evaluated semantics, E is the number of hypothesis semantic annotations which exactly match the reference. The exact match is very tough standard because for example under the exact match the difference between semantics ARRIVAL(TIME, FROM(STATION)) and ARRIVAL(TIME, TO(STATION)) is equal to the difference between semantics ARRIVAL(TIME, FROM(STATION)) and DEPARTURE(TRAIN_TYPE). Therefore we introduce the concept accuracy.

4.2. Concept accuracy

Similarity scores between the reference and the hypothesis semantics can be computed by the tree edit distance algorithm [6]. The tree edit distance measures the similarity between two trees by comparing subtrees of both the reference and the hypothesis annotations.

The tree edit distance algorithm uses the dynamic programing to find the minimum number of substitutions, deletions, and insertions required to transform one tree into another one. The operations act on nodes and modify the tree by changing the parent/child relationships of given trees.

The concept accuracy of a hypothesis is defined as $CAcc = \frac{N-S-D-I}{N} \cdot 100\%$, where N is the number of concepts in the reference semantics, S is the number of substitutions, D is the number of deletions, and I is the number of insertions.

4.3. Results

In Table 1, there is shown the performance of the HVS-PP parser and the LRB-HVS parser on development and test data. We found that both parsers perform significantly better than the baseline system which is the original HVS parser presented in Section 2. We used the paired t-test to measure the statistical significance. The p-value < 0.01 indicates significant difference.

Table 1. The performance of the HVS-PP and LRB-HVS parsers.

	Test data			Development data		
	SAcc	CAcc	p-value	SAcc	CAcc	p-value
baseline	50.4	64.9		52.8	67.0	
HVS-PP	54.1	67.2	< 0.01	56.6	68.4	< 0.01
LRB-HVS	58.3	69.3	< 0.01	60.1	70.6	< 0.01

5. CONCLUSIONS

In this work, we have presented a technique which factors the combined pop and push operation of the original HVS parser into two independent operations pop and push. The deterministic push operation (one inserted concept for every input word) became a probabilistic push operation of pushing zero or one concept onto the stack given the previous stack. This approach significantly increased the performance of the parser. The improvement comes from the ability to not insert a new concept onto the stack if it is not desirable.

In addition, we extended the parser to generate not only right-branching parse trees but also left-branching parse trees or their combinations. We found that insertion of two concepts into the hidden vector state at the same time is sufficient. We analyzed the errors of the original HVS parser and we did not find any example in which the insertion of more then two concepts into the hidden vector state would have helped. We found that the ability to generate left-right-branching parse trees significantly increases the performance of the parser. In total, SAcc was significantly increased from 50.4% to 58.3% and CAcc from 64.9% to 69.3% measured on the test data.

6. ACKNOWLEDGMENTS

This work was supported by the Ministry of Education of the Czech Republic under project No. 1M0567 (CAK), No. ME909, and No. LC536.

7. REFERENCES

[1] Yulan He and Steve Young, "Semantic processing using the hidden vector state model," *Computer Speech and Language*, vol. 19:1, 2005.

[2] Charles T. Hemphill, John J. Godfrey, and George R. Doddington, "The ATIS spoken language systems pilot corpus," in *Proceedings of DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, USA, 1990.

[3] Filip Jurčíček, Jiří Zahradil, and Libor Jelinek, "A Human-Human Train Timetable Dialogue Corpus," in *Proceedings of EUROSPEECH*, Lisboa, Portugal, 2005.

[4] Filip Jurčíček, Jiří Zahradil, and Luboš Šmídl, "Prior of the Lexical model in the Hidden Vector State Parser," in *Proceedings of SPECOM*, St.Petersburg, Russia, 2006.

[5] Jeff Bilmes and Geoff Zweig, "The graphical models toolkit: An open source software system for speech and time-series processing," in *Proc. IEEE ICASSP*, 2002.

[6] Philip Klein, "Computing the edit-distance between unrooted ordered trees," in *Proceedings of the 6th Annual European Symposium*. 1998, Springer-Verlag, Berlin.