IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. I, pp. I-284- I-287, IEEE, Hong Kong, China, 2003

CONCEPT ACQUISITION IN EXAMPLE-BASED GRAMMAR AUTHORING

Ye-Yi Wang and Alex Acero
Microsoft Research
Redmond, Washington 98052, USA

ABSTRACT

To facilitate the development of speech enabled applications and services, we have been working on an example-based semantic grammar authoring tool. Previous studies have shown that the tool has not only significantly reduced the grammar development effort but also yielded grammars of better qualities. However, the tool requires extra human involvement when ambiguities exist in the process of grammar rule induction. In this paper we present an algorithm that is able to automatically resolve the segmentation ambiguities, hence acquire the language expressions for the concepts involved. Preliminary experiment results show that the Expectation-Maximization algorithm we investigated has not only eliminated the human involvement in ambiguity resolution but also improved the overall spoken language understanding accuracy.

1. INTRODUCTION

Semantic-based robust understanding is a successful technology that has been widely used in research conversational systems [1-4]. However, it is not very practical for regular developers to implement a conversational system with the technology, to a large extent due to the fact that such implementations have relied on manual development of domain-specific grammars, a task that is time-consuming, error-prone and requires a significant amount of expertise.

To facilitate the development of speech enabled applications and services, we introduced SGStudio, an example-based grammar authoring tool that could greatly ease grammar development by taking advantage of many different sources of prior information [5, 6]. It allows a regular developer with little linguistic knowledge to build a semantic grammar for spoken language understanding. Experiment results have shown that good quality semantic grammars can be derived semi-automatically with a small amount of data --- the technique not only significantly reduces the effort in grammar development, but also improves the understanding accuracy across different domains.

While the tool has some basic learning capabilities, it often resorts to users when ambiguities need to be resolved to induct grammar rules. This makes it very intrusive and greatly slows down the grammar development. This paper introduces an automatic learning algorithm that eliminates the intrusive questions raised by SGStudio.

1.1 Example Based Grammar Authoring

To build a grammar for a specific domain, SGStudio requires a semantic schema that defines the semantics of the domain. Semantic class is the basic element in a semantic schema. For example, in the Personal Information Management domain, semantic class "NewAppt" defines the semantics for meeting scheduling:

```
<command name="NewAppt">
    <slot type="Person" name="Attendee"/>
    <slot type="Time" name="StartTime"/>
</command>
```

This simplified example of semantic class states that to schedule a new meeting, a user can (optionally) specify the attendee and the start time of the meeting (slots). The slots can be specified with different ways to refer to a Person or Time (types of the slots).

The semantics of a domain is language independent. A user can successfully schedule a meeting by specifying the slots wit hout using speech or natural language. A semantic grammar relates natural languages to the domain semantics. SGStudio semi-automatically learns to produce such a grammar, such that a user can fulfill the task with natural language or speech. It does so by automatically incorporating the domain semantics defined in the schema into an initial template grammar. Following is an example of the template grammar for "NewAppt", where symbols inside braces are optional:

```
<C_NewAppt> ➔ <NewApptCmd> {<NewApptProperties>}                (1)
<NewApptProperties> ➔ <NewApptProperty>
                        {<NewApptProperties> }                  (2)
<NewApptProperty> ➔ <NewApptAttendeeProperty> |
                    <NewApptStartTimeProperty>                  (3)
<NewApptAttendeeProperty> ➔
        {<PreAttendee>} <T_Person> {<PostAttendee>}             (4)
<NewApptStartTimeProperty> ➔
        {<PreStartTime>} <T_Time> {<PostStartTime>}             (5)
```

The template grammar models the language for appointment scheduling with a command part and a properties part (1). The properties part incorporates the slots in the schema (2, 3). It brackets each slot with a preamble and a post-amble that serve as hints for the slot. Initially the commands, preambles and post-ambles (in general, concepts) are not defined. The language expressions for these concepts will be learned from the semantically annotated examples. (6) is an example annotation for the sentence "New meeting with Peter at five":

```
<NewAppt>                                                       (6)
    <Attendee type="Person">Peter</Attendee>
    <StartTime type="Time">five</StartTime>
</NewAppt>
```

In addition to the template grammar and training samples, the tool also takes advantage of domain-independent library grammar (for examples, rules for date and time expressions that can be used for "<T_Time>") and domain-dependent data like contact names that can be used for "<T_Person>."

1.2 Segmentation Ambiguities

A one-to-one mapping exists between the slots in a schema and the non-terminals in the template grammar. Therefore, given a semantic annotation, the annotated slots become the anchor points in the semantic grammar parse tree, and the rest words in the input can be aligned to the pre-terminals in the parse tree according to their positions relative to the slots. For example, given annotation (6), the word "at" has to align to the pre-terminals <PostAttendee> or <PreStartTime>, the only two pre-terminals that can appear between the two slots (<Attendee> "Peter" and <StartTime> "five"). Since "at" is a preposition and has to go with the phrase after it, the tool can induct the rule <PreStartTime> → at. Similarly, the words "New meeting with" have to align to the pre-terminals in front of "Peter (T_Person)", i.e., <NewApptCmd> and <PreAttendee>. However, it is not straightforward to decide which words should align to each pre-terminal. Basically every position in the word sequence is a potential candidate for the segmentation that separates it into <NewApptCmd> and <PreAttendee> parts. The previous version of the tool reported in [5, 6] would pop up a dialog window (Figure 1) to solicit from the user for proper segmentation.

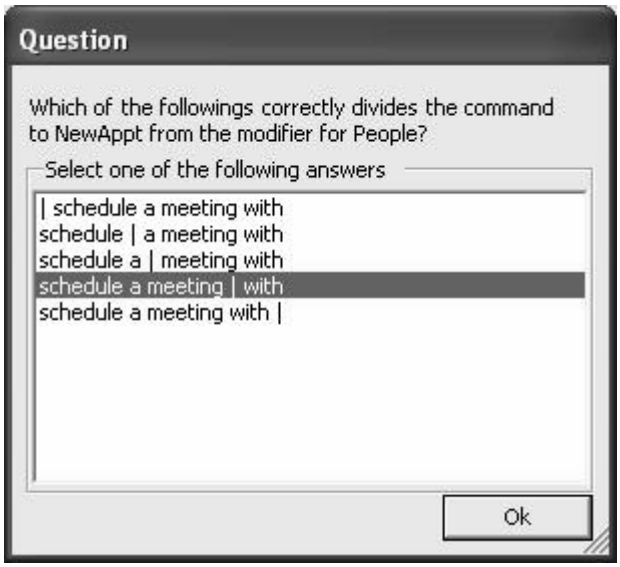


Figure 1. Dialog box of SGStudio that asks the user to select the correct segmentation for "NewApptCmd" and "PreAttendee".

This disambiguation solution has several disadvantages:

1. The interaction with users is intrusive and time-demanding.
2. It violates the design principle that users should only be asked to annotate sentences semantically. While it is easy to do semantic annotation, it is not as easy to segment the expressions effectively without knowledge about the inner working mechanism of the tool.
3. When there are more pre-terminals (this is possible with hierarchical slot structures) and terminal words involved in segmentation ambiguity, the question list will be huge, and it is very hard (if not impossible) to effectively display all the candidate segmentations and let the user pick the correct one. In the tool reported in [5, 6], when the question is too complicated to display, the tool simply skipped the question and did not learning from the example. This is a waste of the precious annotated data.
4. Since the template grammar is not linguistically motivated, user's segmentation according to their linguistic intuition may not work in the best way with the template grammar in a SLU system.

In the following sections, we will introduce an automatic segmentation algorithm and present some experiment results.

2. EM ALGORITHM FOR CONCEPT ACQUISITION

Segmentation ambiguity resolution can be formalized as the problem of finding an m block partition $\pi = \alpha_1, \alpha_2, ..., \alpha_m$ for the word sequence $w = w_1, w_2, ..., w_n$, such that each block aligns to a pre-terminal in the sequence $\mathbb{N} = NT_1, NT_2, ..., NT_m$. A block may contain 0 or more words from w.

If we model the joint probability of $\pi, \mathbb{N}$ and w with

$$P(\pi, \mathbb{N}, w) = \prod_{i=1}^{m} P(NT_i \rightarrow \alpha_i) \quad (7)$$

Then given $\mathbb{N}$ and w, the most likely segmentation can be obtained as:

$$\hat{\pi} = \underset{\pi}{\operatorname{argmax}}\, P(\pi, \mathbb{N}, w) = \underset{\pi=\alpha_1,...,\alpha_m}{\operatorname{argmax}} \prod_{i=1}^{m} P(NT_i \rightarrow \alpha_i) \quad (8)$$

Such a partition can be found with Viterbi search. Thus the only problem left is to estimate the model parameter $P(NT \rightarrow \alpha)$ for every pre-terminal (concept) NT and word sequence α. This could be straightforward with maximum likelihood (ML) estimation if the training data is a list of pre-terminals paired with a word sequence for each pre-terminal. However, the training examples we obtained from the user via the authoring tool are pairs of pre-terminal sequences and terminal sequences. The partition is a hidden variable and unknown to the tool.

Expectation-Maximization (EM) algorithm [7] is often used for ML estimation of model parameters when hidden variables exist and only partial observations are available. It initially sets the parameters P_ϕ for the model, and then iteratively modifies the parameters to $P_{\phi'}$ such that the likelihood of the observation D increases.

To find such $P_{\phi'}$, we define the auxiliary function Q in (9):

$$Q(P_{\phi'} \mid P_\phi) = \sum_{\mathbb{N},w} c(\mathbb{N}, w) \sum_{\pi} P_\phi(\pi \mid \mathbb{N}, w) \log \frac{P_{\phi'}(\pi, \mathbb{N}, w)}{P_\phi(\pi, \mathbb{N}, w)} \quad (9)$$

It is a lower bound of $L(D \mid P_{\phi'}) - L(D \mid P_\phi)$, the log-likelihood difference of the training data between the two model parameterizations [7]. The EM algorithm resets the parameters

$P_{\phi'}$ greedily by maximizing Q, subject to the constraints that the probabilities of all possible rewriting rules for a pre-terminal must sum to 1. Therefore, for each rule $NT \to \alpha$, its new probability can be obtained by solving (10):

$$\frac{\partial(Q(P_{\phi'} \mid P_{\phi}) + \lambda(\sum_{\alpha} P_{\phi'}(NT \to \alpha) - 1))}{\partial P_{\phi'}(NT \to \alpha)} = 0 \tag{10}$$

Since $P_{\phi'}(\pi, \mathbb{N}, w) = \prod_{NT,\alpha} P_{\phi'}(NT \to \alpha)^{c(NT \to \alpha; \pi, \mathbb{N}, w)}$,

$$\frac{\partial Q(P_{\phi'} \mid P_{\phi})}{\partial P_{\phi'}(NT \to \alpha)} = \sum_{\mathbb{N},w} c(\mathbb{N}, w) \sum_{\pi} \frac{P_{\phi}(\pi \mid \mathbb{N}, w) c(NT \to \alpha; \pi, \mathbb{N}, w)}{P_{\phi'}(NT \to \alpha)} = -\lambda. \tag{11}$$

Therefore, the probability should be reset to the expected count times the normalization factor $-1/\lambda$:

$$P_{\phi'}(NT \to \alpha) = -1/\lambda \sum_{\mathbb{N},w} c(\mathbb{N}, w) \sum_{s} P_{\phi}(\pi \mid \mathbb{N}, w) c(NT \to \alpha; \pi, \mathbb{N}, w). \tag{12}$$

To calculate the expected counts, note that

$$\frac{\partial P_{\phi}(\mathbb{N}, w)}{\partial P_{\phi}(NT \to \alpha)} = \frac{\partial \sum_{\pi} P_{\phi}(\pi, \mathbb{N}, w)}{\partial P_{\phi}(NT \to \alpha)} = \sum_{\pi} \frac{c(NT \to \alpha; \pi, \mathbb{N}, w) \prod_{NT,\alpha} P_{\phi}(NT \to \alpha)^{c(NT \to \alpha; \pi, \mathbb{N}, w)}}{P_{\phi}(NT \to \alpha)} = \sum_{\pi} \frac{P_{\phi}(\pi, \mathbb{N}, w) c(NT \to \alpha; \pi, \mathbb{N}, w)}{P_{\phi}(NT \to \alpha)} = P_{\phi}(\mathbb{N}, w) \frac{\sum_{\pi} P_{\phi}(\pi \mid \mathbb{N}, w) c(NT \to \alpha; \pi, \mathbb{N}, w)}{P_{\phi}(NT \to \alpha)}. \tag{13}$$

hence

$$\sum_{\pi} P_{\phi}(\pi \mid \mathbb{N}, w) c(NT \to \alpha; \pi, \mathbb{N}, w) = \frac{P_{\phi}(NT \to \alpha)}{P_{\phi}(\mathbb{N}, w)} \frac{\partial P_{\phi}(\mathbb{N}, w)}{\partial P_{\phi}(NT \to \alpha)}. \tag{14}$$

Let $E_{ij}^{k} = (\mathbb{N} \Rightarrow w_1, ..., w_{i-1}, NT^{k}, w_{j+1}, ..., w_n)$ be the event that in the process of rewriting the pre-terminal sequence $\mathbb{N}$ to the word sequence w, the rule $NT \to \alpha$ is used for the kth pre-terminal in $\mathbb{N}$ to generate the sub-sequence $\alpha = w_i, ..., w_j$, and let $\lambda_s^t(p,q)$ be the probability that the pre-terminals from position s to t in the sequence $\mathbb{N}$ cover the terminal words $w_p, ..., w_{q-1}$. Then

$$P_{\phi}(\mathbb{N}, w) = \sum_{ij} E_{ij}^{k} = \sum_{ij} \lambda_1^{k-1}(1,i) \lambda_{k+1}^{m}(j+1, n+1) P_{\phi}(NT \to w_i, ..., w_j) \tag{15}$$

$$\frac{\partial P_{\phi}(\mathbb{N}, w)}{\partial P_{\phi}(NT_k \to \alpha)} = \sum_{ij: \alpha = w_i, ..., w_j} \lambda_1^{k-1}(1,i) \lambda_{k+1}^{m}(j+1, n+1) \tag{16}$$

Therefore if we can compute $\lambda_s^t(p,q)$, we can combine (12), (14) and (16) to obtain the expected counts and reset the model parameters. In fact $\lambda_s^t(p,q)$ can be computed with dynamic programming according to (17), where ε is the null string:

$$\lambda_s^t(p,q) = \sum_{p \le r \le q} \lambda_s^{t-1}(p,r) \lambda_t^t(r,q); \qquad \lambda_s^s(p,q) = \begin{cases} P_{\phi}(NT_s \to w_p, ..., w_{q-1}) & \text{if } p < q \\ P_{\phi}(NT_s \to \varepsilon) & \text{if } p = q \end{cases}; \tag{17}$$

Note that $P_{\phi}(\mathbb{N}, w) = \lambda_1^m(1, n+1)$ can be used in (14).

3. EXPERIMENTAL SETTING

We conducted experiments with the ATIS3 set A (sentences that can be interpreted without context) data. We used the ATIS3 1993 set A test data for testing. Since there are some tasks in the test data do not have any training data, we followed the practice in [6] to augment the training set with nine sentences that we collected for the experiments with the authoring tool.

As described in [6], we constructed the semantic schema for ATIS by abstracting the CMU Phoenix grammar for ATIS. Training and test sentences were annotated against the schema like the one in (6). This was done by taking the CMU Phoenix/ATIS parsing results and automatically converting them into the semantic annotations with an XSL transformation. The annotations were manually checked and corrected. The resulting canonical meaning representations served as the targets for SLU in our experiments. In doing so, we avoided building the module that translates the meaning of sentences into SQL queries in the original DARPA-sponsored end-to-end ATIS evaluation.

We used our robust parser [3, 8] with the learned grammar to obtain the semantic representations for input sentences and compared them with the manually corrected annotations. The baseline grammar was the one produced with the tool reported in [6] that asked users questions like the one in Figure 1.

4. EXPERIMENTAL RESULTS

We studied the topic classification (henceforth Topic ID) and slot identification (henceforth Slot ID) performance of the two grammars. Topic ID performance was measured by comparing the parser-found frame/semantic class name of a sentence with the manually labeled one. In slot ID evaluation, slots were extracted by listing all the paths from the root to the pre-terminals in the semantic parse tree, and the resulting list was compared with that of the manual annotation. Hence a topic ID

error will cause all the slots in a parse tree to be incorrect in the slot ID evaluation. The total insertion-deletion-substitution error rates are reported for slot ID.

Table 1 compares the error rates between the grammar learned with manual segmentation and the one learned with automatic segmentation. The EM segmentation improves the Topic ID accuracy by 60%. However, there is no change in Slot ID accuracy.

	Manual Segmentation	EM Segmentation
Topic ID	5.06	2.03
Slot ID	7.67	7.67

Table 1. Error rates of the grammars learned with manual segmentation and automatic segmentation. Full paths from root to a leaf slot were used in the calculation of the slot error rate.

As in [5, 6], we also investigated the effect of training data on the performance. Figure 2 shows the Topic ID error rates of the learned grammars relative to the amount of annotated training data. Figure 3 illustrates the Slot ID performance of the same grammars. As a reference, we also included the performance of the Phoenix/ATIS system that used a manually authored semantic grammar.

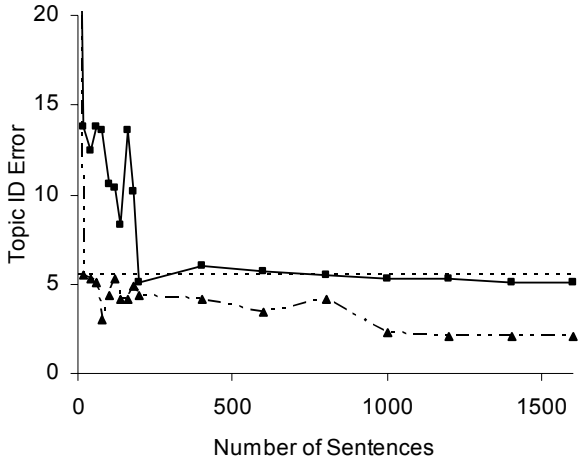
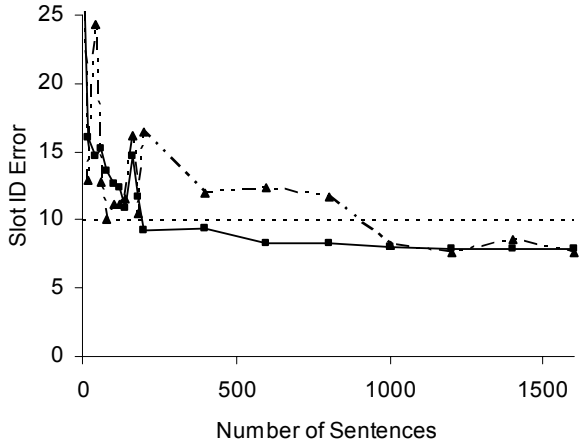


Figure 2. Topic ID error rate vs. amount of annotated training data. The solid curve represents the grammar trained with manual segmentation; the dashed curve represents the grammar trained with automatic segmentation. The dashed horizontal line represents Phoenix/ATIS.

For the simpler problem of topic classification, with very few training sentence, the error rate drop significantly. This is due to the fact that the topic classification task mostly depends on the command pre-terminals in the template grammar. Given that there are only seven tasks, EM algorithm can fast learn the command concepts, and robust parser can use them to identify the correct classes. However, for the more complicated task of slot ID, the accuracy from the EM-trained grammar is worse than the one trained with manual segmentation with fewer examples. This may due to the fact that much more non-terminals exist for slot preambles and post-ambles, which require more training data to achieve a decent performance.



F**igure 3.** Slot error rate (ins-del-sub) vs. amount of annotated training data. The solid curve represents the grammar trained with manual segmentation; the dashed curve represents the grammar trained with automatic segmentation. The dashed horizontal line represents Phoenix/ATIS.

5. SUMMARY

In this paper we presented the EM algorithm for segmentation disambiguation for the example-based grammar authoring tool. Experiment results show that the automatic leaning procedure not only eliminates the human involvement in ambiguity resolution but also improves the overall understanding accuracy.

6. REFERENCES

1. Ward, W. *Understanding Spontaneous Speech: the Phoenix System*. Proceedings of *ICASSP*. 1991. Toronto, Canada.
2. V. Zue, e.a., *JUPITER: A Telephone-Based Conversational Interface for Weather Information.* IEEE Transactions on Speech and Audio Processing, 2000. **8**(1).
3. Wang, Y.-Y. *Robust Spoken Language Understanding in MiPad*. Proceedings of *Eurospeech*. 2001. Aalborg, Denmark.
4. Waibel, A., *Interactive Translation of Conversational Speech.* Computer, 1996. **29**(7).
5. Wang, Y.-Y. and A. Acero. *Grammar Learning for Spoken Language Understanding*. *IEEE workshop on Automatic Speech Recognition and Understanding*. 2001. Madonna di Campiglio, Italy: IEEE.
6. Wang, Y.-Y. and A. Acero. *Evaluation of Spoken Language Grammar Learning in ATIS Domain*. Proceedings of *ICASSP*. 2002. Orlando, Florida.
7. Dempster, A.P., N.M. Laird, and D.B. Rubin, *Maximum likelihood from incomplete data via the EM algorithm.* Journal of the Royal Statistical Society B, 1977. **39**: p. 1-38.
8. Wang, Y.-Y. *A Robust Parser for Spoken Language Understanding*. Proceedings of *Eurospeech*. 1999. Budapest, Hungary: ESCA.