AN EFFICIENT TOP-DOWN PARSING ALGORITHM FOR UNDERSTANDING SPEECH BY USING STOCHASTIC SYNTACTIC AND SEMANTIC MODELS

Holger Stahl, Johannes Müller, Manfred Lang

Institute for Human-Machine-Communication
Munich University of Technology
Arcisstrasse 21, D-80290 Munich, Germany
email: {sta,mue,lg}@mmk.e-technik.tu-muenchen.de

ABSTRACT

The paper is concerning an approach for understanding speech using a new form of probabilistic models to represent syntactic and semantic knowledge of a restricted domain. One important feature of our grammar is that the parse tree directly represents the semantic content of the utterance. Since we determine that semantic content by an integrated search, we avoid consistency problems at the interface between the recognizer and the language understanding part of the speech understanding system. We succeeded in designing such an incremental algorithm, which integrates semantic, syntactic, and acoustic-phonetic knowledge in a seamless, consistent way. High efficiency is achieved by using a chart-parsing technique with structure-sharing and a strict top-down strategy for opening new word hypotheses in the pronunciation layer.

1. INTRODUCTION

Stochastic methods have proved to be powerful for speech *recognition*, mainly due to the uniformity of knowledge representations on different levels. Further pursuing the stochastic approach, speech *understanding* can be interpreted as maximizing the a-posteriori probability $P(S|O)$ of the semantic content S given the input pattern O (i.e. the sequence of feature frames from the preprocessed speech signal) of an utterance. Applying *Bayes*' inversion formula and taking into account just the most likely word chain W, we obtain the following classification rule [1] for maximum-a-posteriori probability decoding (MAP):

$$S_E = \underset{S}{\operatorname{argmax}} \max_{W} [P(O|W) \cdot P(W|S) \cdot P(S)] \qquad (1)$$

As acoustic-phonetic models for the calculation of $P(O|W)$ we use Hidden-Markov-Models (HMMs). The grammar knowledge is splitted into a *semantic model*, providing the a-priori probability $P(S)$ and a *syntactic model*, providing the conditional probability $P(W|S)$.

2. PROBABILITIES IN THE GRAMMAR

The following overview is a brief description of our semantic representation and our grammar formalism. A more detailed and formal description can be found in [2].

Definition of the Semantic Structure: In our approach, the semantic structure S is a tree consisting of a finite number N of semantic units (we simply call them *semuns*), each containing the semantic contribution of one significant word in W. Fig. 1 shows an example within the domain *'speech-understanding graphic editor'* with $N = 5$ semuns:

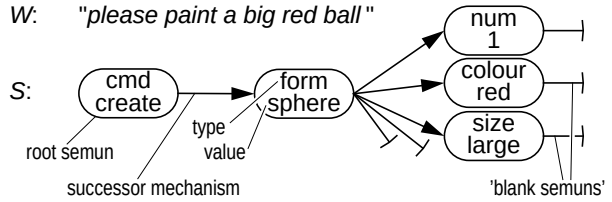

Figure 1: Word chain W and corresponding semantic structure S

Each semun of S consists of a type, a value and $X \geq 1$ *successor*-semuns. The number X is fixed by the semun's type. The leaves of the tree are terminated by 'blank semuns'.

Semantic Model: For each possible type of a semun, there exist two sets of probabilistic rules for determining

(i) the value of the considered semun,

(ii) the types of the combination of successor semuns.

There also exists a set of probabilistic rules for determining

(iii) the type of the root semun.

Assuming statistical independence, $P(S)$ is calculated as product of the probabilities for applying all rules for assembling a semantic structure S. Note that these rules are not context-free in the sense of fixing any chronological order!

Syntactic Model: The syntactic model serves for the production of syntactical correct word chains W corresponding to a given semantic structure S.

For each semun, a phrase is originated into W, consisting of one *significant word*, an optional *insignificant word*, and further phrases for each of the successor semuns. Depending on the semun's type, the syntactic model provides a set of probabilistic context-free rules for determining

(iv) the order of these substrings (time-alignment).

These time-alignment rules are implemented as probabilistic networks, which consist of nodes $A_1 \ldots A_X$, B, C, distinguished start/end nodes 's' and 'e', and a probability matrix containing all network connections. Fig. 2 shows such

a network called *syntactic module* (SM) for the type 'cmd', taken from the example in fig. 1, with $X = 1$ successor:

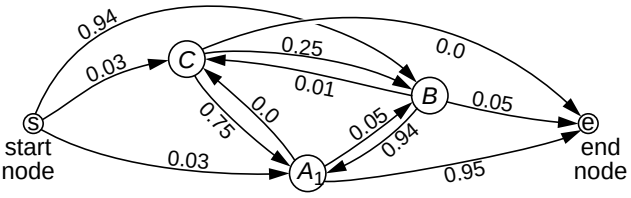

Figure 2: Syntactic module for a type with $X = 1$ successor

From node A_1, that phrase is originated, which is associated with the first (and sole) successor semun of type 'form'. Entering this node will perform the entry of a child SM one level below the current one. In the example, the start node of the type 'form' is encountered. Passing the end node of that syntactic module signifies a return to the parent level.

For the node *B*, there is a set of probabilistic rules for producing

(v) the significant word associated with the semun

depending on the semun's value. For the node *C*, the syntactic model provides a set of rules as well for producing

(vi) the insignificant word associated with the semun.

The appliance of these rules depends on the semun's type.

There are additional constraints for the transitions to be considered inside an SM: Since we defined that for every semun in the semantic structure exactly one significant word, one phrase for each successor and one optional insignificant word has to be originated, it is inhibited to pass one of the nodes $A_1 \ldots A_X$, *B*, *C* twice. Furthermore, the end node cannot be encountered before the nodes $A_1 \ldots A_X$, and *B* have been passed. To illustrate the required bookkeeping inside an SM, we cross out ('✕') the passed nodes and place a dot ('●') at the current position. Using a simplified depiction of the SM, a dot left of a node indicates that it just has been encountered, whereas a dot right of it indicates that the node just has been passed:

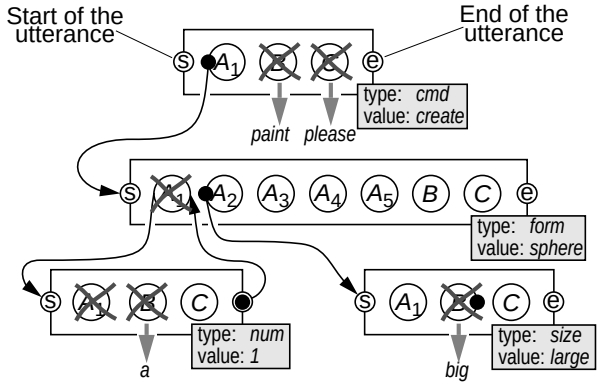

Figure 3: Part of the hierarchic connection of SMs according to the example in fig. 1. Each SM produces a phrase for one semun. The words "*please paint a big ...*" have already been originated.

The probability $P(W|S)$ is approximated by maximizing the product of the probabilities concerning all productions (iv) to (vi) in all possible derivations.

3. THE SEARCH ALGORITHM

For the search, an active chart parser is applied. The parser consists of two layers, the *grammar layer* and the *pronunciation layer*. In the grammar layer, hypotheses for potential word chains are generated by applying the respective grammar rules (i) to (vi). In the pronunciation layer, the acoustic-phonetic emission probabilities are calculated. Since the computations in this layer for $P(O|W)$ in eq. (1) are much more expensive as those for $P(W|S)$ and $P(S)$ in the grammar layer, the parser works strictly top-down. Only those word chains W are then considered, with $P(W,S) \neq 0$. A time-synchronous search allows a direct comparison of different hypotheses and facilitates real-time applications. The grammar layer of the parser is based on *Earley's* algorithm [3], some extensions for processing continuous input with probabilistic knowledge bases could be adopted from [4].

3.1 Representation of Hypotheses in the Search

During the search, the knowledge about open hypotheses is kept in so-called *items* [4], each representing a certain subparse referring to a certain SM. An item contains information about the matching between input frames $o_i \ldots o_{j-1}$ and one single semun with a particular type. In addition to the bookkeeping of the SM (fig. 3), the item maintains properties concerning semantics, time-alignment, and probabilistic information. Fig. 4 shows an example:[1)]

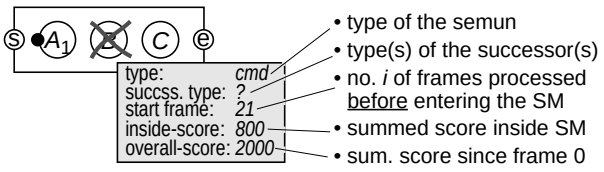

Figure 4: Properties contained in an item

The item only holds the parsing history since the *start-frame* index *i*, when the respective SM was entered, until the currently processed frame. The *inside-score* denotes the score[2)], which has been summed up inside the SM. This score includes all partial probabilities contributing to the MAP estimation of eq. (1), accumulated along this subparse since the start frame. The *overall-score* contains the inside-score as well and additionally the accumulated score for the left context of the item, i.e. the complete history to enter the SM.

Like the 'dotted rules' used in [3], our items contain information only concerning the subparse of a piece of the input pattern, but they show some fundamental differences:

- The probabilistic extension by the two scores (from [4]).
- The bookkeeping for the time-alignment is not only contained in the position of the dot, but also in the list of crossed-out (= passed) nodes.
- The *successor types* include semantic information by fixing the type(s) of the successor semun(s).

1) For simplification, properties required for trace-back are omitted!

2) We transform all probabilities from the models into their negative logarithm to replace multiplications by faster additions.

3.2 Item Lists and Agendas

All items, which have been generated after processing frame o_{j-1}, form the *item list* L_j. Each item list is splitted up into two parts: An *agenda* (the active part) contains items which still have to be processed by the control structure, a passive part contains items which already have been processed. Items can be *checked in* and *retrieved* from the list:

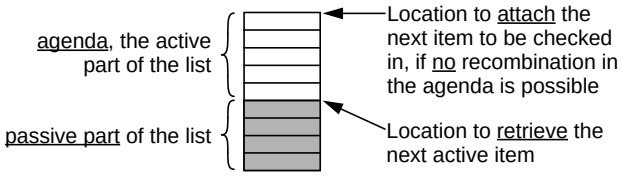

Figure 5: Organization of an item list and operations on it

Whenever an item is checked in, it is tested for recombination with another item in the list. Items are *equivalent*, if they concur in all their properties except the scores. Equivalent items can recombine! That item with the lower overall-score (i.e. the higher probability) survives, the other is discarded. Instead of attaching each item to be checked in, the following strategy is chosen to maximize the efficiency:

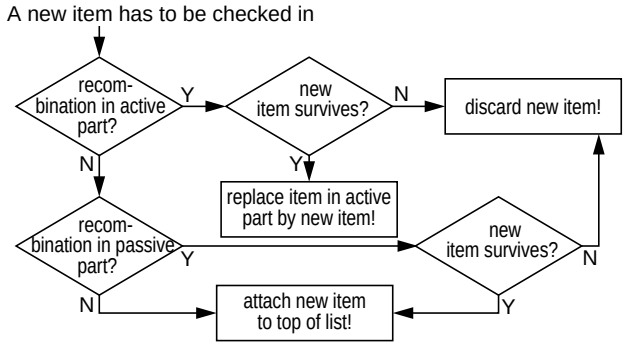

Figure 6: Strategy for recombination of items to be checked in

After retrieving an item, the pointer on the location to retrieve the next one is incremented. Actually, the item lists can be seen as a 'FIFO', that implements a breadth-first search.

3.3 Control Structure for Operations on Item Lists

Fig. 7 illustrates the function of item lists and the operations on them. First, item list L_0 is generated by the *initialization* ①. Subsequently, one item at a time is retrieved from the agenda of list L_0 and is processed by one of the applicable in-list operations ② *transitions*, *entries* or *completions*. The resulting new items are checked in again in the same item list. This process continues until the agenda is empty, i.e. no items remain to be processed by in-list operations.

Afterwards, all items in the list L_0 with one of the nodes *B* or *C* encountered are processed to start up new words in the pronunciation layer. We call this operation *push-words* ③. As a counterpart to this, the *pop-words* operation ④ pops out words which have been completely matched to a sequence of frames by the pronunciation layer. From these words, the next item list L_1 is then initialized.

Now, another round of ②, ③, and ④ is repeated for each of the input frames, generating one item list after another, until all *J* frames are consumed. All inactive item lists $L_0, \ldots, L_j$, which remain after processing frame o_j are kept back in the chart for the completion steps (in ②) and ④.

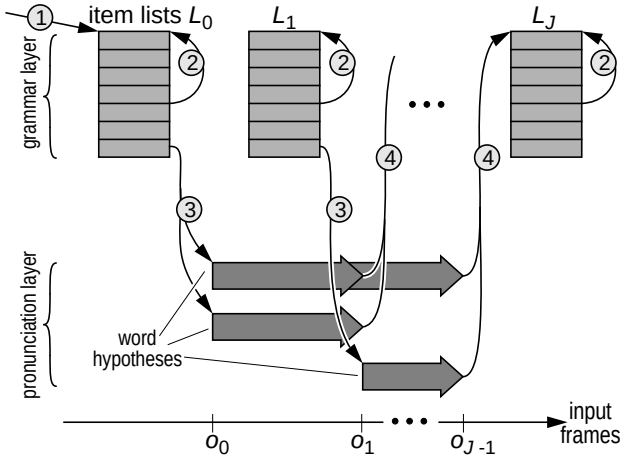

Figure 7: Illustration of the main steps of the algorithm: The initialization step ①, the in-list operations ②, the push-words operation ③, and the pop-words operation ④.

The following lines will give a short description of the operations, which generate a new item from a retrieved item:

Initialization ①: L_0 is initialized by applying rules (iii) creating items of all possible semun types with the start node just encountered.

In-List Operations ②: Transitions are invoked to encounter an SM node from another already processed node by rules (iv). Entries fix the type(s) of the *X* successor semun(s) and enter the start-node of a new SM. Expressed in the terminology of chart-parsing [3][5], the transitions and the entries can be seen as the PREDICTOR step. The completions move the dot over a node, which represents the child SM of a completed subparse, i.e. which has just passed the SM from start to end. In the chart-parsing terminology, the completions correspond to the COMPLETER step.

Note that recursive constructs in the grammar do not cause any problem for the parser, since they will be caught as equivalent subparses, which can recombine to avoid a loop!

Push-words operation ③ and pop-words operation ④: Each item with one of the nodes *A* or *B* just encountered has to produce a word to move the dot over this node. If a push-words operation is invoked for an item (with encountered node *A* or *B*), all words are started, which can be produced by the respective rule (v) or (vi). If these words have been completely matched to a certain number of input frames, they can be used to complete that item, which they were started from by invoking the pop-words operation.

In contrast to the easy task of parsing deterministic input, both the words itself and their boundaries are ambiguous with our task. Without countermeasures, the search space grows unimaginably huge already after a few input frames.

3.4 Improving Efficiency

To reduce both computation and memory effort down to manageable values, we applied a beam search technique. Our incremental algorithm guarantees that all paths ending in the items contained inside one item list L_j have consumed exactly the same number j of input frames. This is an important claim for a direct comparison of the items' overall-scores for pruning them. Pruning is carried out both in the grammar layer and in the pronunciation layer.

The number of word hypotheses to be started up, can be further reduced, if the grammar layer is not activated for each frame index. In practice, after finishing the push-words operations ③, a number of frames is skipped before invoking the next pop-words operations ④. Note, however, that this practice is no longer consistent, since it constrains the words to be forced into a fixed temporal grid.

4. EXPERIMENTAL RESULTS

Our grammar was trained from 1843 utterances out of the domain 'speech-understanding graphic editor'. For testing our decoder, we used 100 utterances, which are a subset of the training set, to avoid out-of-vocabulary errors. The utterances have been collected by a 'Wizard-of-Oz' experiment [6] with 33 speakers. Each utterance is represented by the speech signal, sampled at 16 kHz, and the associated semantic structure, which was used both for training the grammar and as reference for testing the decoder's semantic accuracy.

As acoustic models, we took the phoneme-based HMMs from the SPICOS system [7]. Note that these models were trained from a multi-speaker data base unrelated to the considered domain. They are not the most powerful ones in the meantime, but tried and tested. Due to the independence of our knowledge bases, they can be embedded balanced into the search. In contrast to the reports of others [5][8], we had no need for any normalization of the scores.

In the tests, we determined semantic errors as the percentage of wrongly assigned semantic structures and the computation effort as the average number of check-in operations (see chap. 3.2) on the item lists per utterance.

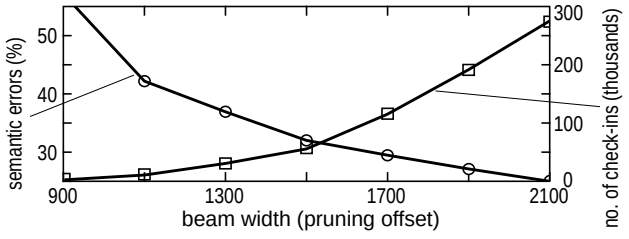

Figure 8: Performance depending on the beam width

Fig. 8 shows the computation effort and understanding errors related to the beam width. It can be seen that pruning is necessary to prevent an uncontrollable increase of the computations. These results were obtained if the grammar is activated every fourth frame index. The performance of the algorithm for other *grammar activity intervals* is listed in fig. 9, determined for the beam width set to 1500. Hence, a grammar activity interval of four reduces the computation effort by about factor 14 without impairing semantic accuracy!

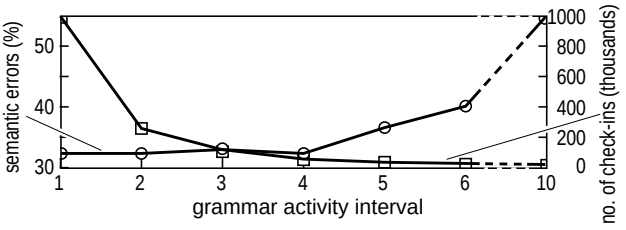

Figure 9: Performance depending on the grammar activity interval

Even though it is a $O(J^3)$ parser in general, we gained a dependence of the number of operations on the number J of input symbols, which seems to be between linear and quadratic. The following diagram was obtained for a beam width of 1500 and a grammar activity interval of four:

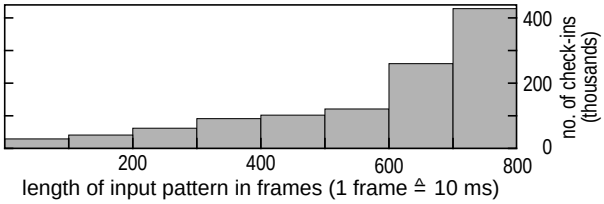

Figure 10: Computation effort depending on the input length. Results have been averaged inside the bar width (= 100 frames).

5. CONCLUSIONS

We designed a MAP-decoder for the outlined new grammar formalism to determine the semantic content of an utterance directly by integrating semantic, syntactic, and acoustic-phonetic knowledge. The huge search space is kept manageable by using a top-down chart-parser combined with beam search and discontinued grammar activity.

REFERENCES

[1] R. Pieraccini, E. Levin, E. Vidal: *Learning how to Understand Language*, Proc. EUROSPEECH 1993 (Berlin, Germany), pp. 1407-1412

[2] H. Stahl, J. Müller: *A Stochastic Grammar for Isolated Representation of Syntactic and Semantic Knowledge*, Proc. EUROSPEECH 1995 (Madrid, Spain), pp. 551-554

[3] J. Earley: *An Efficient Context-Free Parsing Algorithm*, Comm. of the ACM, vol. 13 (1970), no. 2, pp. 94-102

[4] A. Paeseler: *Modification of Earley's Algorithm for Speech Recognition*, Proc. NATO ASI, vol. F 46, Springer, 1988, pp. 465-472

[5] H. Weber: *Time Synchronous Chart Parsing of Speech Integrating Unification Grammars with Statistics*: Proc. 8th Twente Workshop on Language Technology (1994)

[6] J. Müller, H. Stahl: *Collecting and Analysing Spoken Utterances for a Speech Controlled Application*, Proc. EUROSPEECH 95 (Madrid, Spain), pp. 1437-1440

[7] H. Höge: *SPICOS II - a Speech Understanding Dialogue System*, Proc. ICLSP 90 (Kobe, Japan), pp. 1313-1316

[8] S. Senneff: *TINA: A Natural Language System for Spoken Language Applications*, Computational Linguistics, vol. 18 (1992), no. 1, pp. 62-86