Search in a Learnable Spoken Language Parser

Finn Dag Buø and Alex Waibel[1]

Abstract. We describe and experimentally evaluate a system, FeasPar, that learns parsing spontaneous speech. The FeasPar architecture consists of neural networks and a search. The neural networks learns the parsing task, and the search improves performance by finding the most probable and consistent feature structure.
This paper focuses on the search component, and shows how the search improves overall performance considerably. N-best lists of feature structure fragments and agendas are used to speed up the search.
To train and run FeasPar (Feature Structure Parser), only limited handmodeled knowledge is required. FeasPar with the search component performs better than a hand modeled LR-parser in all six comparisons that are made. FeasPar is trained, tested and evaluated in the Time Scheduling Domain, and compared with the LR-parser. The handmodeling effort for FeasPar is 2 weeks. The handmodeling effort for the LR-parser was 4 months.

1 Introduction

In natural language processing, search is becoming a more important role, as parsers get more robust and fault-tolerant. The reason is obvious: parsers containing only (hard) rules often have a limited robustness, and also fail to provide *the best* parse or the N-best parses. Therefore, a mixture of 'hard' and 'soft' rules (scores and penalties, probabilistic rules, and constraints) is applied. In most parsers, the core consists of hand modeled rules. With great success, these rules have been annotated with 'soft' information[3, 10, 8, 6]. In this paper, we present a parser, FeasPar, that learns to parse, instead of having hand modeled rules. The FeasPar architecture consists of neural networks and a search. The search finds the best feature structure based on the neural network outputs, and feature structure constraints.

FeasPar requires only minor hand labeling, and learns the parsing task itself. It generalizes well, and is robust towards spontaneous effects and speech recognition errors. The parser design is based on chunk features, and how they relate to each other.

The parser is trained and evaluated with the Spontaneous Scheduling Task, which is a negotiation situation, in which two subjects have to decide on time and place for a meeting. The subjects' calendars have conflicts, so that a few suggestions have to go back and forth before finding a time slot suitable for both. The data sets are real-world data, containing spontaneous speech effects. The training set consists of 560 sentences, the development test set of 65 sentences, and the unseen evaluation set of 120 sentences. For clarity, the example sentences in this paper are among the simpler in the training set.

The parser produces feature structures, holding semantic information. These feature structures are used as interlingua in the speech-to-speech translation system JANUS[5]. Within our research team, the design of the interlingua ILT was determined by the needs of unification based parser and generator writers. Consequently, the interlingua design was not tuned towards connectionist systems. On the contrary, our parser must learn the form of the output provided by a unification based parser.

This paper is organized as follows: First, a short tutorial on feature structures, and how to build them. Second, we describe the parser architecture and how it works. Then the search algorithm is motivated and explained in detail. Finally, results and conclusion follow.

2 Building a Feature Structure

Feature structures[4, 9] are used as output formalism for FeasPar. Their core syntactic properties and terminology are:

1. A *feature structure* is a set of none, one or several *feature pairs.*
2. A *feature pair*, e.g. `(frame *clarify)` , consists of a *feature*, e.g. `frame` or `topic`, and a *feature value.*
3. A *feature value* is either:

   (a) an *atomic value*, e.g. `*clarify`
   (b) a *complex value*

4. A *complex value* is a *feature structure.*

In contrast to the standard feature structure definition above, an alternative view-point is to look at a feature structure as a tree, where *sets of feature pairs* with *atomic values* make up the branches, and the branches are connected with *relations.* Atomic feature pairs belonging to the same branches, have the same relation to all other branches. Further, when comparing the sentence with its feature structure, it appears that there is a correspondence between parts of the feature structure, and specific chunks of the sentence. In the example feature structure of Figure 1, the following observations about feature pairs and relations apply:

[1] Interactive Systems Laboratories, University of Karlsruhe (Germany), Carnegie Mellon University (USA), finndag@ira.uka.de

ECAI 96. *12th European Conference on Artificial Intelligence*
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd.

```
((speech-act *confirm)
 (sentence-type *state)
 (frame *clarify)
 (topic ((frame *simple-time)
        (day-of-week monday)))
 (adverb perhaps)
 (clarified ((frame *simple-time)
             (day-of-week monday)
             (day 27))))
```

Figure 1. Feature structure with the meaning "by monday i assume you mean monday the twenty seventh"

- **feature pairs:**

feature pairs:	**corresponds to:**
`(day 27)`	"the twenty seventh"
`((frame *simple-time) (day-of-week monday) (day 27))`	"monday the twenty seventh"

- **relations:** the complex value of the feature `topic` corresponds to the chunk "by monday", and the complex value of the feature `clarified` corresponds to "you mean monday the twenty seventh".

Manually aligning the sentence with parts of the feature structure, gives a structure as shown in Figure 2. A few comments apply to this figure:

```
([]((speech-act *confirm)
    (sentence-type *state)
    (frame *clarify))
    ([]
       ([topic]((frame *simple-time))
           ([]                      by)
           ([]((day-of-week monday)) monday))
       ([] ([]                      i))
       ([]((adverb perhaps))
           ([]                      assume)))
    ([clarified]
       ([] ([]                      you))
       ([] ([]                      mean))
       ([]((frame *simple-time))
           ([]((day-of-week monday)) monday)
           ([]                      the)
           ([]((day 27))             ([rego] twenty
                                           seventh)))))
```

Figure 2. Chunk parse: Sentence aligned with its feature structure.

- The sentence is hierarchically split into chunks.
- Feature pairs are listed with their corresponding chunk.
- Relations are shown in square brackets, and express how a chunk relates to its parent chunk. Relations may contain more than one element. This allows several nesting levels.

Once having obtained the information in Figure 2, producing a feature structure is straight forward, using the algorithm of Figure 3. Summing up, we can define this procedure as the *chunk'n'label* principle of parsing:

1. Split the incoming sentence into hierarchical chunks.
2. Label each chuck with feature pairs and feature relations.
3. Convert this into a feature structure, using the algorithm of Figure 3.

```
FUNCTION convert()
VAR
  S: set;
  C: chunk;
BEGIN
  S := empty set;
  assign(S,top_level_chunk);
  return(S);
END;
  PROCEDURE assign(VAR S: set;
                       C: chunk);
    BEGIN
      P := chunk_relation(C);
      FOR each relation element PE in P
        BEGIN
          S' := empty set;
          include (PE,S') in S;
          S := S';
        END;
      FOR each feature pair FP in C
        include FP in S;
      FOR each chunk C' in C
        assign(S,C);
    END;
```

Figure 3. Algorithm for converting a parse to a feature structure

3 Baseline Parser

The chunk'n'label principle is the basis for the design and implementation of the *FeasPar* parser. FeasPar uses neural networks to learn to produce chunk parses. It has two modes: learn mode and run mode. In learn mode, manually modeled chunk parses are split into several separate training sets; one per neural network. In run mode, the input sentence is processed through all networks, giving a chunk parse, which is passed on to the converting algorithm shown in Figure 3. In the following, the three main modules required to produce a chunk parse are described:

The Chunker splits an input sentence into chunks. It consists of three neural networks. In total, there are four levels of chunks: word/numbers, phrases, clauses and sentence.

The Linguistic Feature Labeler attaches features and atomic feature values (if applicable) to these chunks. For each feature, there is a network, which finds one or zero atomic values. Since there are many features, each chunk may get no, one or several pairs of features and atomic values. Since a feature normally only occurs at a certain chunk level, the network is tailored to decide on a particular feature at a particular chunk level. A special atomic feature value is called lexical feature value. It is indicated by '=' and means that the neural network only detects the *occurrence* of a value, whereas the value itself is found by a lexicon lookup. The lexical feature values are a true hybrid mechanism, where symbolic knowledge is included when the neural network signals so.

The Chunk Relation Finder determines how a chunk relates to its parent chunk. It has one network per chunk level and chunk relation element. Further details on the baseline parser can be found in [2, 1].

3.1 Lexicon and Neural Architecture

FeasPar uses a full word form lexicon. The lexicon consists of three parts: one, a syntactic and semantic microfeature vector per word, second, lexical feature values, and three, statistical microfeatures.

Syntactic and semantic microfeatures are represented for each word as a vector of binary values. The number and selection of microfeatures are domain dependent and must be made manually. For the English Spontaneous Scheduling Task (ESST), the lexicon contains domain independent syntactic and domain dependent semantic microfeatures. To manually model a 600 word ESST vocabulary requires 3 full days.

Lexical feature values are stored in look-up tables, which are accessed when the Linguistic Feature Labeler indicates a lexical feature value. These tables are generated automatically from the training data, and can easily be extended by hand for more generality and new words. An automatic ambiguity checker warns if similar words or phrases map to ambiguous lexical feature values. Further information on the lexicon can be found in [1].

All neural networks have one hidden layer, and are conventional feed-forward networks. The learning is done with standard back-propagation, combined with the constructive learning algorithm PCL[7], where learning starts using a small context, which is increased later in the learning process. This causes local dependencies to be learned first. Further techniques for improving performance are described in [1]. For the neural networks, the average test set performance is 95.4 %.

4 Consistency Checking Search

The complete parse depends on many neural networks. Most networks have a certain error rate; only a few networks are perfect. When building complete feature structures, these network errors multiply up, resulting in not only that many feature structures are erroneous, but also inconsistent and making no sense. A search algorithm compensates for this. It is based on two main information sources: First, probabilities that originate from the network output activations; second, a formal feature structure specification, stating what combination of feature pairs is consistent. This specification is already available as an ILT specification document.

4.1 Global Constraints

Additionally, a few other ILT constraints must be considered. that are not modeled in the ILT specification document. They are called *global* constraints, and include two types:

1. **Frame Constraint**: An ILT is a feature structure, where at each branch the feature `frame` has one and only one value.
2. **Compulsory Constraints:** Not only a feature pair F_1 *may* appear with another feature pair F_2, but that F_1 *must* appear with F_2, i.e. in some sense, F_1 triggers F_2.

4.2 Search Task

In combining the network output and the constraints, the search finds the feature structure with the highest probability, under the given constraints being consistent. The outputs of each neural network are normalized[2]to give a probabilistic interpretation. Then they are sorted by probability. They can now be viewed as an N-best list. Hence, the search input is one N-best list per network. To combine these N-best lists hierarchically to build an N-best list of feature structures, forms the search task.

4.3 Search Complexity Precautions

The ESST baseline version of FeasPar had 37 Linguistic Feature Labeler Networks and 4 Chunk Relation networks. Each network has up to 15 different output values. It is crucial to keep complexity and search times low. Therefore, the following principles and constructs are applied:

Hierarchy of Feature Structure Fragments: A feature structure is assembled using partial feature structures. These are called *fragments*. The hierarchy corresponds to the chuck hierarchy and in what sequence the fragments are put together to form a complete feature structure (see the algorithm in Figure 3).

Agendas: Agendas (one per fragment) are used to direct the search, so that always the most probable of the unexamined combination is examined first.

Lazy Evaluation: The Lazy Evaluation delays the expensive calculations of fragments and agenda as long as possible. This is extremely important to reduce search time.

4.4 Search Principles

When building a feature structure, the search uses structures as shown in Figure 4: For the partial feature structure of every chunk, it defines an N-best list of fragments. The fragment parts correspond to chunk relation, chunk features, and subchunks.

Example: The fragment example for chunk$_{type=n}$ in Figure 4 corresponds to the chunk:

```
([when](( frame *special-time))
 ([]                                    in)
 ([]                                    the)
 ([](( time-of-day =morning))           morning))
```

Building a fragment is an expensive operation. In order to build fragments as few times as possible, an agenda is used in parallel to each fragment list. Agenda calculations are much cheaper than fragment calculations. The agenda and fragment interact as follows:

The agenda keeps hold of possible fragment configurations. It is sorted by log probability. Upon a request for a new fragment, the next configuration is fetched from the agenda, and

[2] The outputs are linearly adjusted, so that they sum up to 1.

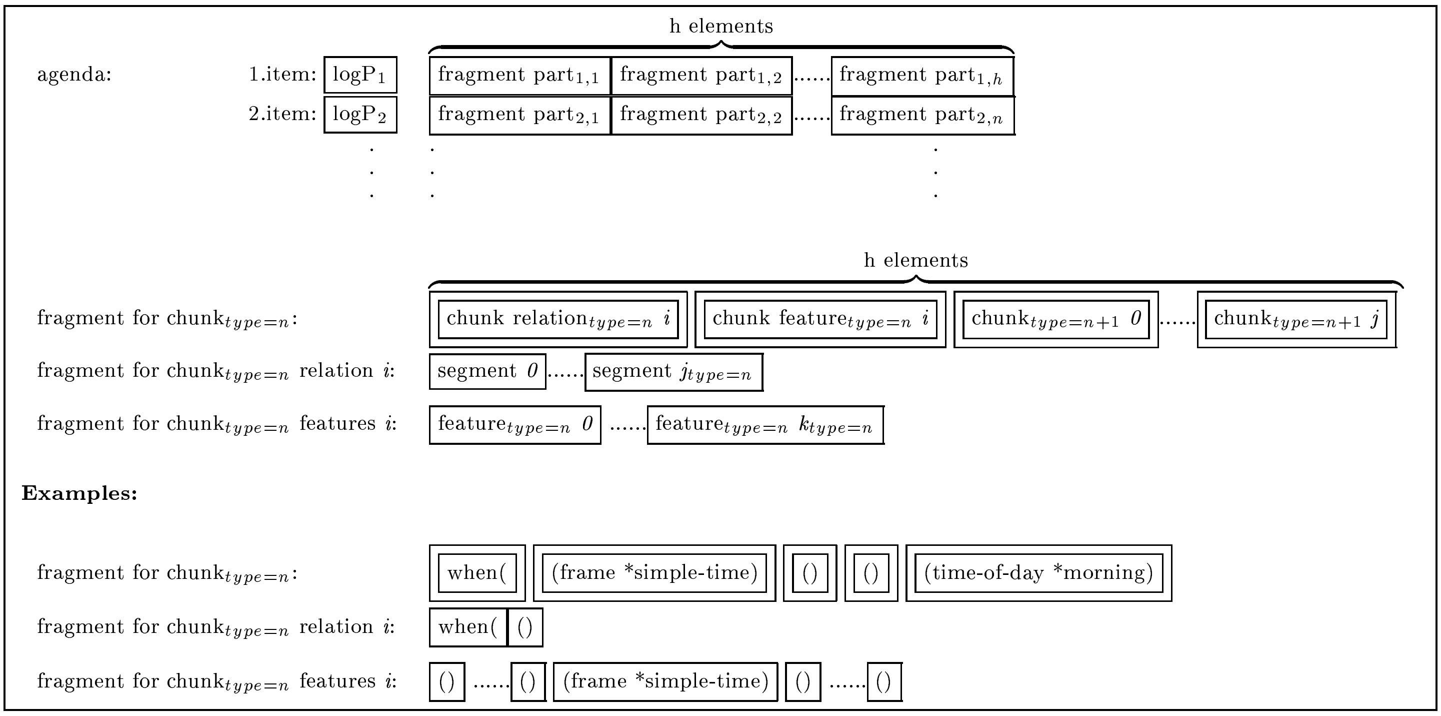

Figure 4. Agenda and fragments. Double framed fragment parts indicates another fragment. Single framed fragment parts indicates neural network output.

the fragment is built. If during fragment building, an inconsistency is detected, the building operation is abandoned, and the next element on the agenda is used as configuration. Then a new fragment is built. This continues until a complete consistent fragment has been built. This fragment is then stored in the N-best list of its chunk, and returned.

During building, fragment parts must be fetched. These are mostly fragments themselves (e.g. in Figure 4 fragment for chunk relation$_{type=n}$ i). If this fragment part has already been calculated during the search, it is already available in the N-best list. If not, a request for a new fragment is made.

The agenda itself is expanded as little as possible: When a new agenda item has been accessed, those candidate agenda items that may follow immediately are inserted in the agenda. This avoids an combinatorial explosion, but ensures that no configuration is left out, or tried too late, with respect to logP.

The consistent constraints mentioned above, are derived as follows: A feature structure formalism contains rules that express in which context what feature pairs may appear. Prior to the parsing process, the program statically calculates for every combination of two feature pairs, if the two feature pairs may occur together or not. This information is consulted during fragment building, as mentioned above.

Global constraints (see Section 4.1) can only be tested on the complete feature structure. When the search returns a complete feature structure for the upper most chunk, the global constraints are tested on the feature structure. If a test fails, the search is continued, until a complete feature structure satisfying all global constraints have been found.

Even if all possible care is taken to speed up the search, the worst-case search is too long. To prevent this, the search is broken off at a certain depth, and the search is repeated, this time allowing *one* inconsistency. If this search gets too deep, two inconsistencies are allowed, and so on.

4.5 Improvements

The following improvements are added to FeasPar in order to gain performance:

Allowing Multiple Equal Feature Pairs: Occasionally, when building a fragment during a search, more than one subfragment contains the same feature pair, i.e. more than one chunk is responsible for adding a particular feature pair at a particular feature structure branch (even if it is not supposed to happen, according to the principles of the hand modeled alignment). The earlier Consistency Checker Search does not accept this, but that one and only one instance of a feature pair is produced for a particular branch. A later version allows multiple feature pair instances.

Sloppy Lexical Feature Value: As described in Section 3 and Section 3.1, FeasPar uses lexical feature values. These are collected from the training data and stored in the lexicon. However, due to incompleteness or speech recognizer errors, a situation may arise, where a natural language chunk is not being stored in the lexical feature value lookup table. In many cases however, a similar chunk may be present, and could be used.

Constraint Relaxation: One important problem with the search algorithm is that sometimes (1 % to 3 % of the analyses), the search takes too long, [3]and therefore has to be

[3] In normal cases (97 % to 99 % of the analyses), the search takes 1 to 3 seconds. In the remaining few cases, the search can run for 10 minutes without completing.

broken off. This is due to the worst case scenario, where all combinations must be searched to find a consistent configuration.
To escape from infinite searches, is the purpose of the following break strategy: If a fragment N-best list exceeds a fixed large number, e.g. 5000, then the search is stopped, and an empty ILT is returned as a parse result. However, since it is better to get a suboptimal analysis than no analysis at all, a constraint relaxation mechanism is added: If a search is broken off, then a new search is made, where one constraint may be relaxed. If this doesn't give any parse result, then two inconsistencies are allowed etc. (A maximum number of inconsistencies, e.g. 7, is there to prevent infiniteness.)

5 Evaluation

	FeasPar (with Search)	FeasPar (without Search)	GLR* parser (4 months)
PM1 - T	71.8 %	33.8 %	51.6 %

	FeasPar (with Search)	GLR* Parser (4 months)
PM1 - T	71.8 %	51.6 %
PM1 - S	52.3 %	30.3 %
PM2E - T	74 %	63 %
PM2E - S	49 %	28 %
PM3G - T	49 %	42 %
PM2G - S	36 %	17 %

Table 1. Comparing FeasPar with a GLR* parser hand modeled for 4 months (Evaluation set (Set 3), S=speech data, T=transcribed data).

FeasPar is compared with a handmodeled LR-parser. The handmodeling effort for FeasPar is 2 weeks. The handmodeling effort for the LR-parser was 4 months.

The evaluation environment is the JANUS speech translation system for the Spontaneous Scheduling Task. The system have one parser and one generator per language. All parsers and generators are written using CMU's GLR/GLR* system[8]. They all share the same interlingua, ILT, which is a special case of LFG or feature structures.

All Performance measures are run with transcribed (T) sentences and with speech (S) sentences containing speech recognition errors. Performance measure 1 is the feature accuracy, where all features of a parser-made feature structure are compared with feature of the correct handmodeled feature structure. Performance measure 2 is the end-to-end translation ratio for acceptable non-trivial sentences achieved when LR-generators are used as back-ends of the parsers. Performance measure 2 uses an English LR-generator (handmodeled for 2 years), providing results for English-to-English translation, whereas performance measure 3 uses a German LR-generator (handmodeled for 6 months), hence providing results for English-to-German translations. Results for an unseen, independent evaluation set are shown in Figure 1.

As we see, FeasPar is better than the LR-parser in all six comparison performance measures made.

6 Conclusion

We described and experimentally evaluated a system, FeasPar, that learns parsing spontaneous speech. To train and run FeasPar, only limited handmodeled knowledge is required (chunk parses and a lexicon).
FeasPar is based on a principle of chunks, their features and relations. The FeasPar architecture consists of two major parts: A neural network collection and a search. The neural networks first spilt the incoming sentence into chunks. Then each chunk is labeled with feature values and chunk relations. Finally, the search uses a formal feature structure specification as constraint, and outputs the most probable and consistent feature structure. N-best lists of fragments and agendas are used to speed up the search.
FeasPar was trained, tested and evaluated in the Time Scheduling Domain, and compared with a handmodeled LR-parser. The handmodeling effort for FeasPar was 2 weeks. The handmodeling effort for the LR-parser was 4 months. FeasPar performed better than the LR-parser in all six comparison benchmarks that were made.

REFERENCES

[1] Finn Dag Buø, *FeasPar - A Feature Structure Parser Learning to Parse Spontaneous Speech*, Ph.D. dissertation, University of Karlsruhe, upcoming 1996.

[2] Finn Dag Buø and Alex Waibel, 'Search in a Learnable Spoken Language Parser', in *Proceedings of the 12th European Conference on Artificial Intelligence*, (August 1996).

[3] J. Dowding, J. M. Gawron, D. Appelt, J. Bear, L. Cherny, R. Moore, and D. Moran, 'Gemini: A Natural Language System for Spoken-Language Understanding', in *Proceedings ARPA Workshop on Human Language Technology*, pp. 43–48, Princeton, New Jersey, (March 1993). Morgan Kaufmann Publisher.

[4] G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag, 'A theory of syntactic features', in *Generalized Phrase Structure Grammar*, chapter 2, Blackwell Publishing, Oxford, England and Harvard University Press, Cambridge, MA, USA, (1985).

[5] P. Geutner, B. Suhm, F. D. Buø, T. Kemp, L. Mayfield, A. E. McNair, I. Rogina, T. Schultz, T. Sloboda, W. Ward, M. Woszczyna, and A. Waibel, 'Integrating Different Learning Approaches into a Multilingual Spoken Language Translation System', in *Workshop on New Approaches to Learning for Natural Language Processing, International Joint Conference on Artificial Intelligence*, Montreal, Canada, (August 1995).

[6] Sunil Issar and Wayne Ward, 'CMU's robust spoken language understanding system', in *Proceedings of Eurospeech*, (1993).

[7] Ajay N. Jain, *A Connectionist Learning Architecture for Parsing Spoken Language*, Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, Dec 1991.

[8] A. Lavie and M. Tomita, 'GLR* - An Efficient Noise-skipping Parsing Algorithm for Context-free Grammars', in *Proceedings of Third International Workshop on Parsing Technologies*, pp. 123–134, (1993).

[9] C. Pollard and I. Sag, 'Formal Foundations', in *An Information-Based Syntax and Semantics*, chapter 2, CSLI Lecture Notes No.13, (1987).

[10] Stephanie Seneff, 'TINA: A Natural Language System for Spoken Language Applications', *Computational linguistics*, **18**(1), (1992).