Application of Explanation-based Learning for Efficient Processing of Constraint-based Grammars

Günter Neumann, DFKI
66123 Saarbrücken, Germany
neumann@dfki.uni-sb.de

In Proc. of 10th IEEE on Artificial Intelligence for Applications, San Antonio, Texas, 1994

Abstract

The paper describes the application of Explanation-based Learning for efficient processing of constraint-based grammars. The idea is to generalize the derivations of training instances created by normal parsing automatically and to use these generalized derivations (called templates) during the run-time mode of the system. In case a template can be instantiated for a new input, no further grammatical analysis is necessary. The approach is not restricted to sentential level but can be applied also to arbitrary phrases. Therefore, the EBL method can be interleaved straightforwardly with normal processing to get back flexibility that otherwise would be lost.

1 Introduction

In the paradigm of feature-based grammar formalisms and theories, one line of research has been oriented towards developing high level languages which are based exclusively on constraint specifications and which allow grammar writers to specify different levels of linguistic abstraction in one homogeneous data structure.

In HPSG [4], currently one of the most prominent member of this class of formalisms the constraint-based view leads to a very schematic representation of universal and language specific linguistic principles. From a computational point of view, a HPSG describes the search space of possible grammatical structures in a highly abstract and implicit characterization, so that phrasal rules have to be instantiated on-line using phrasal schemata and lexical information.

However, on-line processing of such highly specified knowledge demands an enormous amount of computation, especially when declarative specifications should be used in both parsing and generation.

Currently, various lines of research are pursued to compensate for this deficiency and to obtain a better performance. Among these are compilation into lower level formats for which more efficient algorithms exist (e.g., [2], who investigates the compilation of HPSG into Tree Adjoining Grammar), or strategies which allow the attachment of control information determining the order of evaluation to constraints , e.g., [8].

In this paper we describe a different approach which is based on *explanation-based learning.* Rather than modifying the grammar sources or controlling the constraint evaluation process, this approach allows the *automatic extraction of sublanguages* from a competence base and its application within a natural language system. The basic advantages of this approach are:

- Processing of typical occurring grammatical constructions is vastly sped up using *simple* and *efficient* mechanisms,
- *interleaving* with normal processing allows for flexibility that would otherwise be lost.
- The acquired structures are *justified* generalizations with respect to the competence grammar in use.

In the next section we discuss briefly the relationship between explanation-based learning and natural language processing. Our approach is described in detail in sections 3 to 5. In section 6 we relate our approach to other approaches, namely with that of [5, 6]. In section 7 we shortly outline some possible extensions of our approach.

2 Explanation-based Learning and Natural Language Processing

Explanation-based learning (EBL) is a technique by which an intelligent system can learn by observing examples. An EBL system derives justified generalizations from training instances on the basis of declarative background knowledge. As a method, EBL performs four different learning tasks: generalization, chunking, operationalization and justified analogy [1].

In the context of natural language processing, the declarative grammar (and the lexicon) serves as the background knowledge. The parser takes over the task of the problem solver. For a single sentence, the obtained derivation tree is interpreted as a justified explanation for the training instance. Note, that in the case of constraint-based grammars, each node of the derivation tree contains information from phonology, syntax and semantics. Seen as an EBL problem, the derivation tree represents an explanation why the sentence in question is a sentence of the language defined by the grammar.

The obtained derivation tree of the training instance then is processed to obtain a justified generalization. This is achieved by "un-instantiating" values of features of the lexical level (i.e., of the derivation tree's yield). These generalized derivation trees are used for new input in the sense that it is checked whether the same derivations can be drawn by applying the set of rules specified in the template according to its (abstract) derivation. If this is possible, the instantiated derivation tree is considered as the problem solution. If a previously derived derivation can be applied to a new problem, then processing had been restricted by just "replaying" the same derivation, which can be done reasonable more efficiently than by normal parsing (see also 4).

3 Overview of the EBL Approach

The architecture of a natural language system as an EBL system is shown in fig. 1.

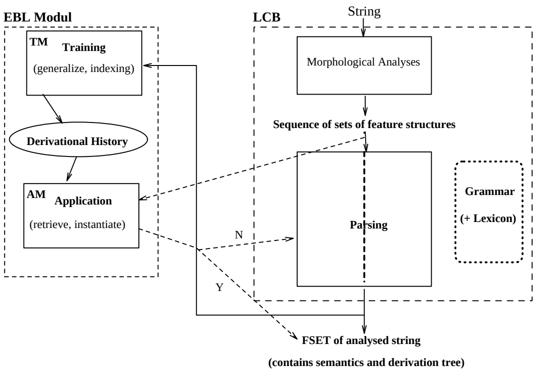

Figure 1: The architecture of a natural language system together with an EBL component.

The right part of the graphic shows the linguistic competence base (LCB) and the left one the EBL module. LCB consists of a morphological component, the parser, the grammar and the lexicon. The EBL module consists of a training module (TM), an application module (AM) and the set of generalized derivation trees (called *derivation history*) determined by TM and applied by AM.

Briefly, the basic flow of control is as follows: During the training phase of the system, a new string α is given as input to LCB. After morphological and grammatical analysis the resulting derivation tree $d(\alpha)$ (i.e., a feature structure that contains among others semantic information and a representation of the derivation tree) is passed to TM. TM computes a template $d'(\alpha)$ of $d(\alpha)$ and stores it in a *discrimination tree*, where indices are computed from the lexical information of $d(\alpha)$.

For a new input β only a morphological analysis is performed. The resulting information is used by AM for traversing the discrimination tree. If a candidate can be found and successfully instantiated, the resulting derivation tree $d(\beta)$ constitutes the result of the analysis of β. If the instantiation fails the parser is activated to compute possible derivations. Note, that the discrimination tree is not only used for efficiency reasons but specifies a necessary condition for selecting possible candidate templates, too.

Consider the sentence *terry writes a program* as an instance for the training phase. The corresponding feature structure after successful parsing is as follows (to save space, only parts are shown):

$$\begin{bmatrix} \texttt{cat} & s \\ \texttt{phon} & \langle terry,\ writes,\ a,\ program\rangle\text{-}\langle\ \rangle \\ \texttt{morph} & \ldots \\ \texttt{sem} & \begin{bmatrix} \texttt{pred} & write \\ \texttt{arity} & 2 \\ \texttt{arg1} & [\texttt{pred}\ terry] \\ \texttt{arg2} & \begin{bmatrix} \texttt{pred} & exists \\ \texttt{scope} & [\texttt{pred}\ program] \end{bmatrix} \end{bmatrix} \\ \texttt{deriv} & \ldots \end{bmatrix}$$

From this parsing result we extract the derivation tree (i.e., the value of the **deriv** feature) and generalize it by "un-instantiating" the yield. This is done by replacing the name of each terminal node with new variables. The below feature structure shows the result of this generalization step, i.e., the template[1].

$$\begin{bmatrix} \texttt{rn}\ s \\ \texttt{dt}\ \left\langle \begin{bmatrix} \texttt{rn}\ np1 \\ \texttt{dt}\ \langle[\texttt{rn}\ T1]\rangle \end{bmatrix}, \begin{bmatrix} \texttt{rn}\ vp1 \\ \texttt{dt}\ \left\langle[\texttt{rn}\ T2], \begin{bmatrix} \texttt{rn}\ np2 \\ \texttt{dt}\ \langle[\texttt{rn}\ T3][\texttt{rn}\ T4]\rangle \end{bmatrix}\right\rangle \end{bmatrix} \right\rangle \end{bmatrix}$$

In the next step a path expression has to be computed to serve as an index for the template in the discrimination tree. We use the sequence of lexical elements of the original derivation tree found under the feature **morph**:

[1] rn stands for rule name and dt for daughters.

$$\left\langle \begin{bmatrix}\texttt{cat} & \mathit{pn}\\ \texttt{stem} & \mathit{terry}\end{bmatrix}, \begin{bmatrix}\texttt{cat} & \mathit{tv}\\ \texttt{stem} & \mathit{write}\end{bmatrix}, \begin{bmatrix}\texttt{cat} & \mathit{det}\\ \texttt{stem} & \mathit{a}\end{bmatrix}, \begin{bmatrix}\texttt{cat} & \mathit{n}\\ \texttt{stem} & \mathit{program}\end{bmatrix} \right\rangle$$

If we would use this information directly, the template would be accessible only for exactly the same input sentence (mimicking a simple caching method). Therefore, a further abstraction step is applied. The basic idea is to generalize away lexical and morphological information present. We do this by "un-instantiating" the values of morphological features by replacing actual values with variables. Which features are to be considered as relevant has to be defined externally. In the ongoing example, we will simply assume that only the stem feature is un-instantiated. The resulting abstract sequence is as follows:

$$\left\langle \begin{bmatrix}\texttt{cat} & \mathit{pn}\\ \texttt{stem} & \mathit{S1}\end{bmatrix}, \begin{bmatrix}\texttt{cat} & \mathit{tv}\\ \texttt{stem} & \mathit{S2}\end{bmatrix}, \begin{bmatrix}\texttt{cat} & \mathit{det}\\ \texttt{stem} & \mathit{S3}\end{bmatrix}, \begin{bmatrix}\texttt{cat} & \mathit{n}\\ \texttt{stem} & \mathit{S4}\end{bmatrix} \right\rangle$$

This sequence is inserted as a path into the discrimination tree, where the end of the path points to the derivation template. Note, that each morphological feature structure corresponds to a node of the discrimination tree, hence discrimination will be performed with respect to the values of morphological features.[2]

The template together with the corresponding index characterize the derivation of all sentences of the language that are equal in length (based on the length of the index), whose morphological information is consistent with that of the index and who share the derivation described by the template.

The template is therefore applicable not only for the sentence *Terry writes a program* but also for *John loves a cat*. This can be recognized in the application phase as follows. Firstly, morphological analysis of *John loves a cat* results in:

$$\left\langle \begin{bmatrix}\texttt{cat} & \mathit{pn}\\ \texttt{stem} & \mathit{john}\end{bmatrix}, \begin{bmatrix}\texttt{cat} & \mathit{tv}\\ \texttt{stem} & \mathit{love}\end{bmatrix}, \begin{bmatrix}\texttt{cat} & \mathit{det}\\ \texttt{stem} & \mathit{a}\end{bmatrix}, \begin{bmatrix}\texttt{cat} & \mathit{n}\\ \texttt{stem} & \mathit{cat}\end{bmatrix} \right\rangle$$

This information now is used to traverse the discrimination net. Because the morphological information is compatible, the template can be used to "replay" the derivations described by the template, which successfully yields the analysis:

$$\begin{bmatrix}\texttt{cat} & \mathit{s}\\ \texttt{phon} & \langle \mathit{john},\ \mathit{loves},\ \mathit{a},\ \mathit{cat}\rangle\text{-}\langle\ \rangle\\ \texttt{morph} & \ldots\\ \texttt{sem} & \begin{bmatrix}\texttt{pred} & \mathit{love}\\ \texttt{arity} & \mathit{2}\\ \texttt{arg1} & [\texttt{pred}\ \mathit{john}]\\ \texttt{arg2} & \begin{bmatrix}\texttt{pred} & \mathit{exists}\\ \texttt{scope} & [\texttt{pred}\ \mathit{cat}]\end{bmatrix}\end{bmatrix}\\ \texttt{deriv} & \ldots\end{bmatrix}$$

[2] If a path already exists in the discrimination tree, we have to check whether a new template is identical to one already found under the path in question.

4 Extending the Approach to Arbitrary Subsentential Phrases

The two most important problems of the approach described so far (which we also call *simple approach*) are

- the restricted flexibility: the whole input length is a necessary condition for successful retrieval of derivation trees
- the amount of redundancy: templates cannot share identical subtrees.

In order to overcome these problems we now extend our approach also to arbitrary subsentential phrases.[3]

Training Phase We start the training phase with a given sentence. The parser is used to obtain an analysis as already known. However, we are now using a more sophisticated generalization step to obtain a derivation template.

The basic idea is to extract from the derivation of the parsing result a *skeleton* of the derivation using a set of predefined *phrasal types*. Phrasal types are defined as constraints. They are used to determine structural properties of candidate phrases like maximal projections (e.g., saturated nominal phrases), structural properties of phrases (e.g., no immediate recursive structures, consider information of mother node etc.). The root node of a derivation will always be regarded as a phrasal type. In the current system, these phrasal types are defined declarative (using logical operations like and, or, not).

A skeleton reflects a *compact structural description* of the original derivation tree. The phrasal types are used to define non-terminal nodes of the skeleton. Non-terminal symbols represent derivations of phrases that will be instantiated for substrings of an input. Terminal nodes are used to determine paths for the discrimination net. A skeleton will be represented using *context free* rules. It is constructed from a derivation tree using the phrasal types as non-terminal elements and the elements of the yield as terminal elements.

For example, suppose we obtain the derivation tree given in fig. 2 for the training sentence *Terry writes a program.*

A skeleton is obtained from the original derivation tree by top/down traversal. Each terminal node will be replaced by a new variable (i.e., T_i) which is used

[3] To facilitate the description of the extended version we will use the well-known tree–like representation of derivations. Note, that each node corresponds to a complex feature structure.

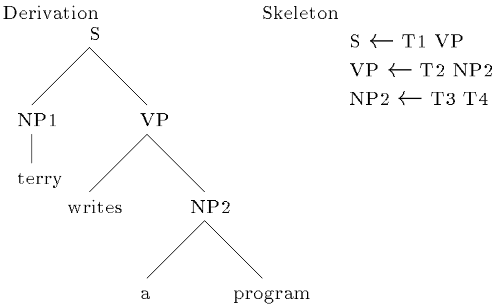

Figure 2: A derivation tree and its skeleton

as a terminal element in the skeleton. Each node that corresponds to a phrasal type will define a skeleton rule, with this node as the left-hand side. For instance, assume that Vp and Np2 are defined as phrasal types (note, that S is considered as a phrasal type, too). The completely extracted skeleton is as given in fig. 2.

After the creation of the skeleton, the template and the generalized morphological sequence are computed as already known. The skeleton and the template are directly associated.

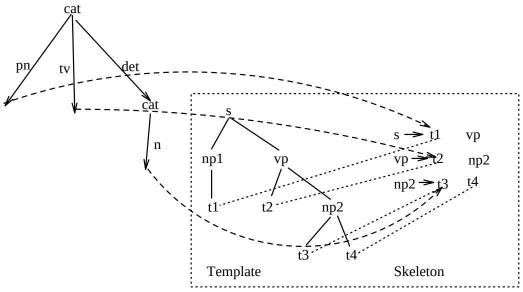

Figure 3: Representation of template, skeleton and indices for example *Terry writes a program.* For simplicity, we only represent the value of the category feature. The dotted box is used to represent that the template and the skeleton are associated together.

Next the skeleton is used to determine indices for the discrimination net from the generalized morphological sequence repeated for convenience:

$$\left\langle \begin{bmatrix} \texttt{cat} & pn \\ \texttt{stem} & \mathit{S1} \end{bmatrix}, \begin{bmatrix} \texttt{cat} & tv \\ \texttt{stem} & \mathit{S2} \end{bmatrix}, \begin{bmatrix} \texttt{cat} & det \\ \texttt{stem} & \mathit{S3} \end{bmatrix}, \begin{bmatrix} \texttt{cat} & n \\ \texttt{stem} & \mathit{S4} \end{bmatrix} \right\rangle$$

If a rule of a skeleton has only terminal elements on the right hand side then the sequence of terminal elements is used as an index otherwise each terminal of a rule is used. For instance, the index for the third rule of the skeleton (i.e., $Np2 \rightarrow T3\ T4$) is

$$\left\langle \begin{bmatrix} \texttt{cat} & det \\ \texttt{stem} & \mathit{S3} \end{bmatrix}, \begin{bmatrix} \texttt{cat} & n \\ \texttt{stem} & \mathit{S4} \end{bmatrix} \right\rangle$$

and for the first rule of the skeleton (i.e., $S \rightarrow T1\ Vp$) the index is

$$\left\langle \begin{bmatrix} \texttt{cat} & pn \\ \texttt{stem} & \mathit{S1} \end{bmatrix} \right\rangle$$

The whole structure of the example after the training phase is shown in fig. 3.

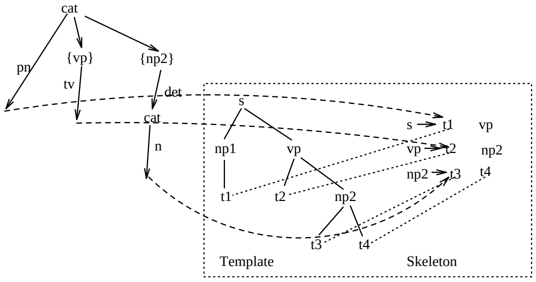

Figure 4: Use of phrasal types for dividing the discrimination tree of fig. 3 in corresponding classes. The curly brackets indicate that these classes represent only a sufficient condition not a necessary one.

Traversal of the Discrimination Tree In the simple approach the discrimination tree must be visited only one time for a new input. In the new approach, however, the discrimination tree describes possible *prefixes* of incrementally examined suffixes of a sentence. Hence, we have to repeat retrieval of the discrimination tree as long as the remaining suffix is not empty. Therefore, each endpoint of an index bears implicitly a pointer to the root of the discrimination tree. However, to be able to restrict possible entry-points into the tree, we can use information from the skeleton. For example, a rule like $Vp \rightarrow T2\ Np2$ does not only tell us that T2 is the left corner of a Vp phrase but also constrains the phrasal type of the following substring to be a member of Np2. In other words: if the current substring α of a new input can be used to access the skeleton rule $Vp \rightarrow T2\ Np2$, the next possible prefix of the suffix (i.e., T2-α) has to be of type $Np2$ otherwise the template cannot be used for α. More generally, this means that an adjacent non-terminal element of a terminal one determines the type of the substring that can follow it. Thus, we can use the phrasal types also for traversing the discrimination net. The extended discrimination tree is shown in figure 4.

Application Phase The skeleton rules will be selected in a strictly bottom-up left-to-right manner. That is, an index which corresponds to a sequence of terminal elements is used for selecting the corresponding skeleton rule which then is used for partial instantiation of the corresponding template. The matching sequence of terminals in a skeleton rule divides the rule into an inactive part and an active part. Two cases must be distinguished: either the active part is empty or it starts with a non-terminal element. Using a dotted notation as known from table driven parsing algorithms, selection of the skeleton rule $Vp \rightarrow T2\ Np2$ can be represented as $Vp \rightarrow T2\ .\ Np2$ and selection of $Np2 \rightarrow T3\ T4$ as $Np2 \rightarrow T3\ T4..$ Now we can use a strictly deterministic Earley type strategy for processing the skeleton rules.

The following example will help to clarify the method. Assume that *Terry loves a cat* is passed as input to the application phase. The morphological analysis yields:

$$\left\langle \begin{bmatrix} \texttt{cat} & pn \\ \texttt{stem} & terry \end{bmatrix}, \begin{bmatrix} \texttt{cat} & tv \\ \texttt{stem} & love \end{bmatrix}, \begin{bmatrix} \texttt{cat} & det \\ \texttt{stem} & a \end{bmatrix}, \begin{bmatrix} \texttt{cat} & n \\ \texttt{stem} & cat \end{bmatrix} \right\rangle$$

Using the discrimination tree of figure 4 *terry* is found as a possible prefix. The selected skeleton rule is partially completed (i.e., $S \rightarrow T1\ .\ Vp$) and used for (partial) instantiation of the corresponding template. In order to complete the current skeleton rule we have to complete the *Vp* rule of the skeleton (indicated by the dot). Therefore, we retrieve the discrimination tree classified under *Vp* which partially completes the rule $Vp \rightarrow T2\ .\ Np2$. The inactive part of the rule is used to instantiate the template partially. In the next step retrieval of the discrimination tree yields the completed rule $Np2 \rightarrow T3\ T4.$ used for instantiating the template. Completion of the *Np2* rule also leads to completion of rules *Vp* and *S*.

Because there is no possible suffix, traversal of the discrimination tree stops and since, $S \rightarrow T1\ Vp.$ has been completed the resulting instantiation of the template constitutes the analysis of the whole input. Note, that the skeleton rules have been completed in a strict left-to-right manner.

Now, suppose that the grammar would also cover the sentence *a cat loves terry*. The first possible prefix that matches the discrimination tree is *a cat*. This activates and partially instantiates the template using the skeleton rule $Np2$. However, because there is no skeleton rule which has $Np2$ as its left corner further processing of the input string is stopped. The completed partial derivation tree with root node $Np2$ is used for instantiating the parser.

Avoiding Redundancies So far, this method suffers from a large amount of redundancy, because templates cannot share identical substructures. For example, if the sentences *Terry writes a program* and *Terry sleeps* would be passed as input to the training phase two separate templates are constructed which have the

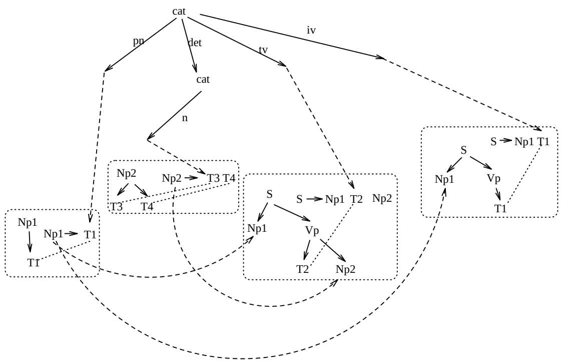

Figure 5: Avoiding redundant substructures. In this case the template rooted with Np1 is shared by both templates rooted with S.

same substructures (the derivation rooted with node $Np1$). However, it is possible to use the skeleton also to avoid this kind of redundancies. The basic idea is to isolate from the template and its associated skeleton those (generalized) substructures which correspond to phrasal types that have been used to build the skeleton. Only the root node of these substructures is kept in the template. The substructures together with its skeleton rules are stored separately in the discrimination tree under the corresponding index (in the case of the $Np2$ substructures, the index is pn). The root node of the skeleton rule of the isolated structure is used as a pointer to the corresponding node of the template it has been isolated from. If a template X is created for a new training instance, the same approach is followed. However, for an extracted substructure it is checked whether the same structure is already in the discrimination tree. If this is the case, the root node of the skeleton of that already existing substructure also serves as a pointer to the appropriate node in X. Figure 5 represents the method graphically.

Interleaving with Normal Processing The fact that the derivation templates are derived from the competence grammar guarantees that they represent correct analysis. If the choice of the training instance is based on the evaluation of corpora of the domain in question, then the templates represent structures that actually occur (see also [6]).

Furthermore, the integration of EBL with normal processing also in the application phase guarantees that if AM is unsuccessful for a new input (as it would be for example in the case of a sentence like *a cat loves terry* or *Terry sleeps*) the normal parser is activated to look for possible analyses. Using a chart based (or table driven) parser, the instantiated derivation trees of subsentential phrases can be used to initialize the chart. This is easily achieved by inserting for each (intermediate) derivation a corresponding inactive edge. If the parser uses an agenda for sorting inactive edges according to a given priority function (see e.g., [7]), and if inactive edges with wider span are given higher priority than those with a smaller span, the parser would try to combine the largest derivations before smaller ones, i.e., it would prefer those structures determined by EBL. For example, if the instantiated derivation for the Vp phrase is passed to the parser, then the chart would be instantiated with inactive edges for Vp, $Np2$ etc. However, by using the priority just mentioned the parser first tries to use the Vp edge in the completion step. If this fails, the parser will be able to check dependent inactive edges. In this sense, using derivation trees for EBL guarantees that the parser is able to "backtrack" into instantiated templates, if for some reasons, the whole instantiated template does not lead to a proper analysis. Interleaving EBL and normal processing in this way does not effect the completeness condition of the parsing algorithm.[4]

Implications of the Degree of Generalization for Performance The degree of generalization directly influences the trade-off between space and time complexity. For example, currently the templates only contain the rule identifiers. Therefore, during the instantiation step of the application phase, all the indicated rules have to be applied explicitly in order to "fill in" the constraints expressed in the rules. However, it is also possible to apply the rules of a template already during TM. The resulting "expanded template" then consists of all constraints but the lexical. Although the storage of such expanded template is larger than its unexpanded counterpart, instantiation during AM reduces to simple pattern matching.[5]

5 Implementation

The simple approach is fully implemented and integrated into the natural language system DISCO [9]. DISCO uses a large HPSG grammar specified in TDL, a high level description language for typed feature structures. The unifier in use handles disjunction, distributed disjunction, full negation as well as functional constraints. The parser (an agenda-based bidirectional bottom-up chart parser) operates directly on the highly specified grammar, i.e., there is no compilation to a lower level format. First experiments have shown that the EBL method leads to a speed up of a factor between 30 to 50.

In the case of the subsentential approach, we have implemented a first prototype. However, this prototype must still be interleaved with the normal processing (although this is fairly straight forward). Based on first experiments we expect that the increased flexibility of the subsentential approach will not affect runtime behaviour very much, because the main overhead is only caused by a slightly more complex retrieval of the discrimination tree and because templates can be stored more economically.

6 Related Work

The basic idea for applying EBL in the context of natural language systems is due to [5] and [6]. The major difference to our approach is that they construct grammar rule chunks instead of keeping track of the original (generalized) derivation trees. These chunks consist of the root node and the yield of a previously analysed derivations, i.e., they abstract away all intermediate derivations. The major problems of using such chunked rules are that a. redundancies can hardly be avoided, b. the training phase is more complex, because subsumption is necessary to check whether a chunked rule has already been acquired and c. it is difficult to get back completeness, because it is not possible to "backtrack" into templates. Therefore normal processing cannot benefit from the templates in that case. Furthermore it seems that they only apply EBL on a sentential level, although they are able to acquire phrasal chunks like nominal phrases or relative clauses, too. However, these are only used during processing of sentences. Therefore, interleaving their approach with normal processing seems to be much harder as it is for our approach (actually, they consider integration of EBL with normal processing as future work).

7 Conclusion and Further Directions

In this paper we showed how Explanation-based Learning can be combined with natural language processing to improve the performance of a natural language system considerably. Because the competence

[4] This is also the case if a complete analysis of a string is required for which the EBL module could successfully instantiate one or more templates. In this case, all intermediate derivations of the instantiated derivation tree are transformed into corresponding inactive edges.

[5] Our system can be parameterized with respect to these two kinds of behaviour. First experiments have shown that the use of expanded templates increases the overall performance reasonable and that the subsentential approach of EBL minimizes storage.

part of the system is not only used during the training phase but also during the application phase, the whole language power is potentially available although the system would prefer those structures it was trained with. We believe that this is a realistic and attractive alternative approach to investigate *competence-based performance models* of natural language where methods from both areas, computational linguistics and artificial intelligence can be combined fruitfully.

Of course, the approach can be and should be extended and improved. Two of many possible ways which we started to investigate are briefly considered now.

An important line of research will be to incorporate statistical methods in order to control the size and retrieval of the discrimination tree. A possible strategy is to attach preference values to the edges of the discrimination tree. These preference values can be determined automatically based on the frequency of successful retrieval and in dependence of the point of time of the last successful retrieval. The discrimination tree then can be seen as a self-organizing datastructure. In a similar way alternative templates can be organized to implement a kind of "preferred template first" strategy.

Another attractive extension is the acquisition and processing of recursive templates. The approach described for avoiding redundancies in templates can be used as a starting point. However, in this extended approach the phrasal types are used to define equivalence classes. Then the derivation tree of a training instance is checked whether it has been constructed by applying recursive rules. If so, also the recursively applied structure will be considered as a phrasal type and isolated from the embedded structure. Thus a recursive NP would be split into the base case and its recursive structure. Next, reference to the templates should only be done by means of the names of phrasal types, i.e. not by means of direct links. Then EBL also would be applicable for processing of input independently of the length of the strings, as long as the string can be processed using the recursive templates. In [3] we give a more detailed description of this extension and its relevance for linguistic theories and natural language systems.

Acknowledgement

Most of the stuff presented in this paper has been implemented by Stefan Haas. Without the many discussions about the 'linguistic realism' together with Klaus Netter, it would have been difficult to "beef up" the approach as it is.The research underlying this paper was supported by a research grant, FKZ ITW 9002 0, from the German Bundesministerium für Forschung und Technolog e to the DFKI project DISCO.

References

[1] T. Ellman. Explanation-based learning: A survey of programs and perspectives. *ACM Computing Surveys*, 21(2):163–221, 1989.

[2] Robert Kasper. On compiling HPSG in lexicalized TAGs. In *Proceedings of the* 2^{nd} *Workshop on TAGs*, Philadelphia, 1992.

[3] K. Netter and G. Neumann. Explanation-Based Learning for Automatic Acquisition of Recursive Sublanguages. Submitted to ACL-94.

[4] C. Pollard and I. A. Sag. *Information Based Syntax and Semantics, Volume 1.* Center for the Study of Language and Information Stanford, 1987.

[5] M. Rayner. Applying explanation-based generalization to natural language processing. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, 1988.

[6] C. Samuelsson and M. Rayner. Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *IJCAI-93*, pages 609–615, Sydney, Australia, 1991.

[7] S. M. Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

[8] H. Uszkoreit. Strategies for adding control information to declarative grammars. In *29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, 1991.

[9] H. Uszkoreit, R. Backofen, S. Busemann, K. Diagne, E. Hinkelman, B. Kiefer, U. Krieger, K. Netter, G. Neumann and S. Oepen. DISCO — An HPSG-based NLP System and its Application for Appointment Scheduling. Submitted to COLING-94.