20

Alignment-Based Learning versus Data-Oriented Parsing

Menno van Zaanen

20.1 Introduction

As described in chapter 2, the Tree-DOP system needs a corpus of trees to extract fragments. However, if no such corpus of trees is available, no fragments can be extracted and the system cannot parse.[1] If for a moment, the DOP framework is related to human language perception, the absence of the fragments compares to a person who does not know anything about the language.

Normally, fragments are extracted from a corpus of trees. If no such corpus is available, but unstructured sentences (examples) of the language are, one might try to automatically generate a corpus of trees based on the examples. Systems that do this are called *structure bootstrapping systems*.

This chapter describes the Alignment-Based Learning (ABL) structure bootstrapping system. It can generate a corpus of trees using unstructured sentences only and therefore help the DOP framework when no fragments or trees are available. In addition to this, it is related to DOP in many ways.

[1] This problem is inherent not only to the DOP framework, but to parsing techniques in general.

Data-Oriented Parsing.
Rens Bod, Remko Scha and Khalil Sima'an (eds.).

20.2 Alignment-Based Learning

This section will give an overview of the ABL system. Firstly, the main ideas behind ABL will be discussed. Next, the two phases in ABL will be described, beginning with the alignment learning phase and followed by the selection learning phase. Finally, different instantiations for both phases will be given.

20.2.1 Ideas behind ABL

The main goal of ABL is to find structure using as little information as possible.[2] Ideally, the system should be given plain sentences only and it should output the same sentences augmented with structure. The system should use a *minimum of language dependent information*, so dictionaries, oracles, part-of-speech tags, etc. may not be used. Furthermore, the system should not have many settings that greatly influence the resulting structure, since this implies that extensive tuning is needed.

In addition, the system should learn *useful* structure. This will be tested by letting the system learn on plain sentences that are extracted from a given corpus of trees. The resulting structured sentences can be compared to the structured sentences of the original corpus and the completeness (recall) and correctness (precision) of the learned structure can be computed.

The Alignment-Based Learning system tries to meet these goals by first intelligently searching for hypotheses of constituents, followed by a second phase, where the "best" hypotheses are selected. The first phase is called *alignment learning*; *selection learning* is the second phase. Both will be described in more detail next.

20.2.2 Alignment Learning

The first phase in the ABL system builds a search space of possible constituents. These possible constituents are called *hypotheses*. The main goal of this phase is to search for hypotheses and store them, so the next phase finds constituents by choosing the best hypotheses.

Alignment learning is based on Harris's (1951) idea of substitutability, which (informally) states that *constituents of the same type can be replaced by each other*. Consider the sentence in example 6a.[3] The noun phrase *the fares* can be replaced by another noun phrase, for example *flights from Dallas to Atlanta* resulting in the sentence of example 6b.

[2] Since the ABL system tries to generate a corpus of trees for use with the Tree-DOP system, structure means standard phrase-structure as described in chapter 2.

[3] All example sentences in this chapter are taken from the Penn Treebank ATIS corpus.

(6) a. Show me the fares
b. Show me flights from Dallas to Atlanta

(7) a. Show me X[the fares]
b. Show me X[flights from Dallas to Atlanta]

Harris also described a procedure that finds constituents by making use of the notion of substitutability. This procedure is the basis of the alignment-learning phase. However, even though the notion of substitutability may always hold, the procedure to find constituents breaks down on certain cases, as described in (van Zaanen 2000a), introducing incorrect constituents. ABL solves this by storing the found word groups as hypotheses. The selection learning step selects the best hypotheses, thus (hopefully) removing the incorrect hypotheses.

Harris's procedure can be explained by considering the sentences in 6. One notices that when these noun phrases are substituted, *Show me* remains the same in both sentences (indicated by underlining). This means that parts of sentences that are different (i.e. not the same in both sentences) may have been substitutions. Harris's procedure now states that when parts of sentences can be substituted, they are constituents of the same type. In ABL, the unequal parts of sentences are stored as hypotheses.

When the alignment learning phase processes the two sentences of example 6, it finds the parts of the sentences that are equal in both (the underlined parts), then takes the parts of the sentences that are *not* underlined to be hypotheses of the same type (X in this case) as indicated in example 7. When there are multiple unequal parts in both sentences, each pair of unequal parts receives a unique type label. This accounts for the bootstrapping step of alignment learning.

Note that it is possible to find empty hypotheses (i.e. hypotheses that do not group words, corresponding with an empty constituent). For example, when the sentence *Show me the fares for all the flights from Dallas to Denver* is aligned to sentence 6a, the latter receives an empty hypothesis at the end of the sentence.

The main problem with empty hypotheses is that it is unclear on which level in the tree they should be attached. In the previous example, ABL only indicates that the empty hypothesis should be at the end of the sentence. If different hypotheses have ending brackets after the last word, the empty hypothesis can be anywhere between these closing brackets (i.e. it can be attached on different levels).[4]

[4]Since it is unclear what should happen with these underspecified hypotheses, they are normally removed after the selection learning phase.

For each sentence s_1 in the corpus:
    For every other sentence s_2 in the corpus:
        Align s_1 to s_2
        Find the identical and distinct parts between s_1 and s_2
        Assign unique type labels to the hypotheses
            (each pair of hypotheses receive the same label)
        Store the hypotheses together with the sentences

FIGURE 20.1 Alignment learning algorithm

During the alignment learning phase, all sentences in the corpus are aligned to all other sentences in the corpus. An overview of the algorithm can be found in figure 1.

Learned hypotheses (in the form of pairs of brackets) are stored with the sentences, but this information is not used when aligning to other sentences (i.e. it always looks like the bootstrapping step). After aligning, new hypotheses are appended to the list of known hypotheses of the sentence.

When learning and storing hypotheses this way, it may occur that an unstructured sentence is aligned to a partially structured sentence (which was already in the partially structured corpus). Aligning these sentences might yield a hypothesis that has the same begin and end brackets of a hypothesis that was already present in the partially structured sentence. The new hypothesis in the unstructured sentence then receives the same type label as the hypothesis in the partially structured sentence. As an example, consider aligning the sentences in 8. This results in the sentences in example 9.

(8) a. What does x[AP57 restriction] mean
    b. What does aircraft code D8S mean

(9) a. What does x[AP57 restriction] mean
    b. What does x[aircraft code D8S] mean

It may even be the case that two partially structured sentences are aligned. This occurs when a new sentence has been aligned to a sentence (and has received some structure) and is then aligned to another partially structured sentence. If this combination of sentences yields hypotheses with the same beginning and end brackets of existing hypotheses, the two existing type labels are merged. All occurrences of these type labels are updated in the corpus. For example, if the structured sentences in 10 are aligned, this would result in the sentences in example 11. The structure is the same, but the type labels have changed.

(10) a. Explain the $_{X}$[meal code]
b. Explain the $_{Y}$[restriction AP]

(11) a. Explain the $_{Z}$[meal code]
b. Explain the $_{Z}$[restriction AP]

Merging type labels reduces the number of different labels. By merging type labels, hypotheses (and thus constituents) in a certain context can only have one type.[5] This assumption is wrong as shown in (van Zaanen 2000a). As an example, consider the sentences in 12. The type label of *morning* is *NN* (a noun), while *nonstop* has *JJ*, an adverb, as type label according to the Penn Treebank ATIS corpus. However, ABL would assign the same type to both *morning* and *nonstop*.

(12) a. Show me the $_{NN}$[morning] flights
b. Show me the $_{JJ}$[nonstop] flights

Instead of merging type labels unconditionally, a clustering technique should be used. This clustering technique might try to merge type labels when there is enough evidence that two types are similar enough.

Storing hypotheses separately introduces another problem. The system may find overlapping hypotheses.[6] This happens for example with the sentences in 13. Aligning sentence 13a to 13b results in the hypothesis *Give me all flights* in sentence 13b (since *from Dallas to Boston* is the same in both sentences). However, aligning sentence 13b to 13c introduces the hypothesis *all flights from Dallas to Boston* in 13b, which overlaps the existing hypothesis *Give me all flights*. In this case, sentence 13b receives two mutually exclusive hypotheses. A sentence with possibly overlapping hypotheses is called a *fuzzy tree*.

(13) a. $_{X}$[Book Delta 128] from Dallas to Boston
b. $_{X}$[Give me $_{Y}$[all flights] from Dallas to Boston]
c. Give me $_{Y}$[help on classes]

20.2.3 Selection Learning

The problem of overlapping hypotheses can be solved by selecting hypotheses. The *selection learning* phase selects the correct (or at least the better hypotheses) out of the hypotheses generated by the alignment learning phase.

[5] When the sentences are reparsed (as described in section 20.5.2), constituents in a certain context may again receive different type labels.

[6] Since the underlying grammar is assumed to be a context-free grammar (as in Tree-DOP), overlapping constituents are not acceptable in final treebank.

Selection learning can be done in several different ways. van Zaanen (1999) discusses three methods (which will all be described in detail in section 20.2.4). These methods can be divided into a non-probabilistic method and two probabilistic methods. For now, the focus will be on the probabilistic methods.

The main idea behind the probabilistic selection learning methods is that the best hypotheses in a sentence should be chosen based on the combined probabilities of the single hypotheses. This means that using the probabilities of the single hypotheses, the probability of each possible combination of hypotheses can be computed. The combination that has the highest probability is chosen to be correct. (However, non-overlapping hypotheses are always considered correct in the current implementations.)

Different ways to compute the probability of each hypothesis result in different methods, as will be described in the next section.

Using probabilities of the single hypotheses, the probability of a combinations of hypotheses can be computed by taking the product of the probabilities of the separate hypotheses as in SCFGs (cf. (Booth 1969)).

However, taking the product of the simple probabilities has a "trashing" effect as described in (Caraballo and Charniak 1998). Since probabilities are all between 0 and 1, multiplying many probabilities tends to reduce the combined probability towards 0. Consider comparing the probability of the singleton set of the hypothesis $\{c_1\}$ with probability $P_{c_1} = P(c_1)$ to the set of hypotheses $\{c_1, c_2\}$ with probability $P_{c_1,c_2} = P(c_1)P(c_2)$. It will always hold that $P_{c_1,c_2} \leqslant P_{c_1}$. Thus in general, taking the product of probabilities prefers smaller sets of hypotheses. To eliminate this effect, the geometric mean is used to compute the combination probability.

The system should consider all hypotheses for removal. However, the geometric mean, in effect, selects only the hypotheses with the highest probability. If the non-overlapping hypotheses were also used in the computation of the combined probability of the hypotheses, many would not be selected (since their probability is small). For the computation of the combined probability, only the overlapping hypotheses are used.

The probability of each combination of mutually non-overlapping hypotheses (taken from the set of overlapping hypotheses) is computed using a Viterbi style optimisation algorithm (Viterbi 1967). The combination of hypotheses with the highest probability is selected and the overlapping hypotheses not present in this combination are removed. If several combinations yield the same probability, the combination with the most constituents is chosen.

20.2.4 Instantiations

Both phases can have different instantiations. The alignment learning phase can use different types of alignment. Selecting hypotheses can be done using non-probabilistic or probabilistic methods. Furthermore, computing the probabilities of hypotheses can also be done in several ways.

Different alignment learning methods

van Zaanen (2000b) describes three different alignment methods. The first method, which is called *default*, is based on the edit distance algorithm by Wagner and Fischer (1974). This algorithm finds the minimum edit cost to transform one sentence into the other based on a predefined cost function γ. This function assigns a cost to each of the edit operations: insertion, deletion and substitution.

The cost function can be assigned in such a way that the algorithm finds the longest common subsequences in two sentences. Since the groups of words that are *not* in these longest common subsequences are the words that are unequal in both sentences, they should be stored as hypotheses.

Using the longest common subsequence to (indirectly) find the unequal parts of the sentences does not always result in the preferred hypotheses. As can be seen when aligning sentences 14, the default algorithm generates the alignment of sentences 15, because *San Francisco* is the longest common subsequence.[7] However, this results in unwanted syntactical structures. The preferred alignment can be found when linking the word *to* (as in sentences 17).

(14) a. ... from San Francisco to Dallas
b. ... from Dallas to San Francisco

(15) a. ... from $_X$[ ] San Francisco $_Y$[to Dallas]
b. ... from $_X$[Dallas to] San Francisco $_Y$[ ]

(16) a. ... from $_X$[San Francisco to] Dallas $_Y$[ ]
b. ... from $_X$[ ] Dallas $_Y$[to San Francisco]

(17) a. ... from $_X$[San Francisco] to $_Y$[Dallas]
b. ... from $_X$[Dallas] to $_Y$[San Francisco]

To let the system have a preference to link *to* instead of *San Francisco* can be accomplished by biasing the γ cost function towards linking words that have similar relative offsets in the sentence. The *San Francisco*s

[7] Of course *from* is also in the longest common subsequence (and some part of the sentence preceeding *from* might also be in the longest common subsequence).

reside in the beginning and end of the sentences (the same applies to *Dallas* in the alignment of sentences 16), so the difference in offset is large. This is not the case for *to*; both reside roughly in the middle. The system with the biased γ function is called *biased*.

It may be the case that the biased does not find the correct alignment, so a third system is introduced. The third system tries to solve this problem by using all possible alignments to find hypotheses. For sentences 14, all three possibilities are used. This system is called *all*. Note that this method can introduce overlapping hypotheses easily.

Different selection learning methods

Not only the alignment learning phase can have different instantiations. Selection learning can also be done in different ways. Three different methods will be described here. A non-probabilistic method and two probabilistic ones. The probabilistic selection learning methods differ in the way probabilities of hypotheses are computed. (van Zaanen 1999)

incr The underlying assumption in this non-probabilistic method is that a hypothesis that is learned earlier is always correct. This means that newly learned hypotheses that overlap with older ones are incorrect, and thus should be removed. Note that when this method is implemented to occur during alignment learning overlapping hypotheses are never introduced. If a hypothesis is found which overlaps an existing hypothesis, it is simply not stored.

leaf This probabilistic model computes the probability of a hypothesis by counting the number of times the particular words of the hypothesis have occurred in the learned text as a hypothesis, divided by the total number of hypotheses.

$$P_{leaf}(c) = \frac{|c' \in C : yield(c') = yield(c)|}{|C|}$$

where C is the entire set of hypotheses.

branch In addition to the words of the sentence delimited by the hypothesis, this model computes the probability based the words of the hypothesis *and* its type label (i.e. it is a normalised version of P_{leaf}).

$$P_{branch}(c|root(c) = r) = \frac{|c' \in C : yield(c') = yield(c) \wedge root(c') = r|}{|c'' \in C : root(c'') = r|}$$

When using one of the probabilistic selection learning methods, it may be possible that several combinations of hypotheses have the same probability (after computing the combined probability). The system then selects one of these combinations at random.

20.2.5 Results

Several systems were described in the previous section. Alignment learning can be done using default, biased or all, but since the differences in results are very small as described in (van Zaanen 2000b), only the default system will be evaluated here. For the selection learning phase the incr, leaf and branch can be used.

The three systems have been applied to two different treebanks. The first treebank is the Penn Treebank 2 Air Traffic Information System (ATIS) corpus. (Marcus et al. 1993) This treebank consists of 577 sentences. The other corpus is the Openbaar Vervoer Informatie Systeem[8] (OVIS) corpus (Bonnema et al. 1997). The larger OVIS corpus consists of 9,941 sentences.

Plain sentences were extracted from the original treebank (ATIS or OVIS). These sentences were fed to the different ABL systems. The structured sentences generated by ABL were then compared to the structured sentences of the original treebank.

The results have been computed using the commonly used EVALB program, which was also used in (Collins 1997). The only difference is that *unlabelled* metrics are used here instead of *labelled*.[9]

Since the algorithms sometimes select constituents at random, the systems have been applied to the plain sentences ten times and the results of all systems have been computed. The mean and standard deviations (between brackets) of the results of the systems applied to the both corpora can be found in table 1.

		UR		UP		F	
ATIS	incr	31.64	(0.94)	38.94	(1.32)	34.91	(1.10)
	leaf	25.82	(0.19)	54.73	(0.42)	35.09	(0.25)
	branch	20.81	(0.20)	46.57	(0.39)	28.76	(0.26)
OVIS	incr	56.01	(3.45)	54.38	(3.35)	55.18	(3.40)
	leaf	53.63	(0.11)	63.78	(0.10)	58.27	(0.10)
	branch	42.24	(0.14)	51.04	(0.11)	46.23	(0.13)

TABLE 20.1 Results on the ATIS and OVIS corpora: UR=unlabelled recall, UP=unlabelled precision, F=F-score

As can be seen in the results, the incr system performs relatively well, although it has a relatively large standard deviation. This means

[8]Openbaar Vervoer Informatie Systeem translates to Public Transport Information System.

[9]If the labelled metrics were to be used, a mapping had to be found between the type labels learned by ABL and the type labels in the structured corpora.

that the order of the sentences in the corpus is quite important.

The leaf system is much more stable; its results hardly depend on the order of the sentences in the corpus. This system performs best on both corpora, especially when looking at the precision and F-score.

The branch system, however, does not perform well at all. It uses a more precise statistical model, which tries to make very precise predictions, while it is based on partially incorrect data. Not all hypotheses found by the alignment learning phase are correct. The selection learning system, however, does depend on that.

20.3 Similarities between ABL and Tree-DOP

Data-Oriented Parsing and Alignment-Based Learning are two completely different systems with different goals. The DOP framework structures sentences based on known *structured* past experiences. ABL is a language learning system searching for structure using *unstructured* sentences only. However, the global approach both choose to tackle their respective problems is similar.

Both DOP and ABL frameworks consist of two phases. The first phase builds a search space of possible solutions and the second phase searches this space to find the best solution.

In the first phase, DOP considers all possible combinations of subtrees in the treebank that lead to possible derivations of the input sentence. The second phase then consists of finding the best of these possibilities by computing the most probable parse, effectively searching the "derivation-space".

ABL has a similar setup. The first phase (alignment learning) consists of building a search space of hypotheses by aligning sentences to each other. The second phase (selection learning) searches this space (using for example a statistical evaluation function) to find the best set of hypotheses.

ABL and DOP are similar in remembering possible solutions for further processing later on. The advantage of this is that when the search space is built, more precise (statistical) information can be found. Definite choices are made when complete information is available (in the search space), in contrast to when choices have to be made at the time mutually exclusive solutions are found.

Note that ABL and DOP do not actually compute all solutions, but all (or at least many) solutions are present in a compact data structure. Considering the solutions is possible by searching this data structure.

20.4 Incorporating ABL in DOP

This section discusses two ways of extending DOP using ABL. One way of extending DOP is to use ABL as a bootstrapping method generating an initial treebank. The other way is to have ABL running next to DOP to make it more robust.

20.4.1 Bootstrapping a treebank

One of the main reasons for developing ABL was to enhance DOP with a method to bootstrap an initial treebank.

All DOP methods assume an initial treebank containing subtrees. These subtrees are normally extracted from a structured corpus. However, if no such corpus is available, DOP cannot be used directly.

Normally, a structured corpus is build by hand. However, "the costs of annotation are prohibitively time and expertise intensive, and the resulting corpora may be too susceptible to restriction to a particular domain, application, or genre." (Kehler and Stolcke 1999) This means that for each language or domain, a new structured corpus is needed, which may not be feasible due to time and cost restrictions.

Automatically building a structured corpus circumvents these problems. Unstructured corpora are built more easily and applying an unsupervised grammar bootstrapping system such as ABL to an unstructured corpus is relatively cheap and fast.

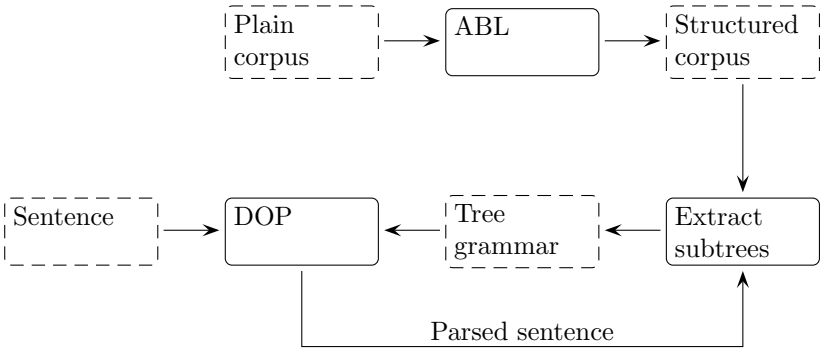

FIGURE 20.2 Using ABL to bootstrap a treebank for DOP

Figure 2 gives an overview of the combined ABL and DOP systems. The upper part describes how ABL is used to build a structured corpus, while the lower part indicates how DOP is used to parse a sentence. Both systems are used in their normal ways, the figure illustrates how both systems can be used in combination.

Starting out with an unstructured corpus, ABL bootstraps a structured version of that corpus. From this, subtrees are extracted in the normal way, which can then be used for parsing. Note that subtrees extracted from the sentences parsed by DOP can again be added to the treebank.

20.4.2 Robustness of the parser

The standard Tree-DOP model breaks down on sentences with unknown words or unknown syntactical structures. One way of solving this problem (in contrast to the DOP models described in (Bod 1998)) is to let ABL take care of these cases. To our knowledge, this is the first extension of the DOP model that really learns new syntactic structures.

Figure 3 shows that when DOP cannot parse a sentence,[10] the unparsable sentence is passed to ABL. Applying ABL to the sentence results in a structured version of the sentence. Since this structured sentence resembles a parsed sentence, it can directly be the output of the system. Another approach may be taken following the ABL phase, where subtrees, extracted from the structured sentence, are added to the tree grammar and DOP will try to parse the sentence a second time.

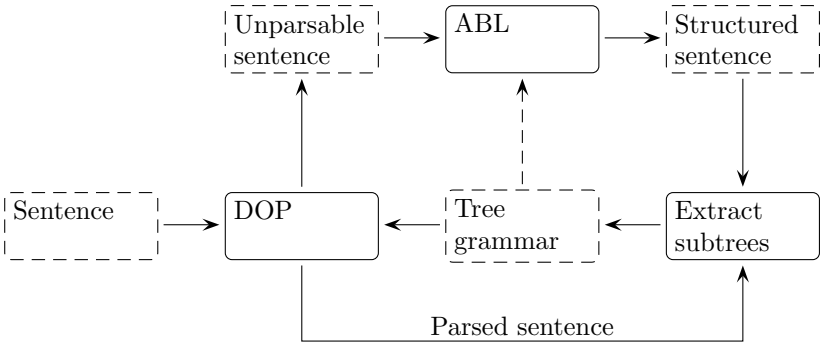

FIGURE 20.3 Using ABL to adjust the tree grammar

The main problem with this approach is that ABL cannot learn structure using the unparsable sentence only; ABL always needs other sentences to align to. Additionally, for any structure to be found, ABL needs at least two sentences with at least one word in common. There are several ways to solve this problem.

One way of finding sentences ABL can use to align is extracting them from the tree grammar DOP uses. If complete tree structures are present

[10] It is assumed that the unparsable sentence is a correct sentence in the language and that the grammar (in the form of DOP subtrees) is not complete.

in the tree grammar, the yield of these trees (i.e. the plain sentences) can be used to align against the unparsable sentence. Furthermore, using the structure present in the trees from the tree grammar, the correct type labels might be inserted in the unparsable sentence (if the sentences are aligned similarly to the sentences in example 8).

If no complete tree structures are present in the tree grammar (for example, because they have been removed by pruning), sentences can still be extracted by generating them from the subtrees. Using a smart generation algorithm can assure that at least some words in the unparsable sentence and the generated sentences are the same.

A completely different method of finding plain sentences to offer ABL is by running ABL parallel to DOP. Each sentence DOP parses (correctly or incorrectly) is also given to ABL. These sentences can then be used to learn structure. By parsing sentences, ABL builds its own structured corpus that is used to align unparsable sentences against.

Additional information about the unparsable sentence can be gathered when ABL initialises the structure of the unparsable sentence with information from the (incomplete) chart DOP has built. The incomplete chart contains structure based on the subtrees in the tree grammar. These subtrees are a good indication of part of the structure of the sentence even though the subtrees cannot be combined into a complete structure.

20.5 Using DOP in ABL

Instead of extending the DOP framework with ABL, as is described in the previous section, it is also possible to incorporate elements of the DOP framework in ABL. The first extension of ABL discussed here, uses the stochastic model of Tree-DOP as the probabilistic function that directs the search in the selection learning phase. In the second extension, an additional phase is appended to the basic ABL method, which reparses the original plain sentences with a learned grammar.

20.5.1 Selection learning based on a DOP approach

As described in section 20.2.4, ABL searches the best hypotheses (out of the overlapping) using an evaluation function. This function is based on chronological order (default) or based on a probabilistic function (leaf and branch).

Even though the best results were found with the less precise leaf statistical model. Both the leaf and branch evaluation functions have a lower standard deviation than the default method.

The branch function is a more fine-grained method than the function in the leaf method, but apparently it depends heavily on the distribu-

tion of the non-terminals of the hypotheses. It may be the case that taking the (possible) internal structure of a hypothesis into account in the computation of the probability will improve over the branch system.

In some ways, finding the best structure using the branch method closely resembles parsing the sentence with a stochastic context-free grammar. If the set of hypotheses is assumed to describe the possible derivations of an SCFG, then the branch evaluation function computes the combined probability of the *inner* probabilities without inner structure of a derivation.

The probability of a hypothesis is independent of its (outer) context. Furthermore, the probability of a hypothesis does not depend on any internal structure. In other words, if the hypothesis encapsulates other hypotheses, this inner structure is *not* taken along in the computation of the probability of the hypothesis.

When the sentence is (re-)parsed with a grammar that is extracted from the set of (overlapping) hypotheses, more precise probabilities can be computed. Computing probabilities of hypothesis by selecting the best parse based on an SCFG that is extracted from the set of (ambiguous) hypotheses will take into account the complete inner and outer probabilities of the hypotheses. Choosing for the Tree-DOP model will take even more structure into account. The Tree-DOP with subtrees of depth larger than one is statistically stronger than an SCFG. (Tree-DOP with subtrees of depth one is equivalent to a SCFG, cf. (Bod 1998, pp. 27–33)).

To be able to parse the sentences, subtrees (or grammar rules) are needed. These can be extracted from the fuzzy trees, however extracting them is not a trivial task. Since fuzzy trees are a compact representation for multiple trees, it might seem logical to extract all possible trees (from the fuzzy trees) first and extract subtrees from these trees.

(18) $_{S}$[$_{X}$[Give me $_{Y}$[all flights$_{X}$] from $_{Z}$[Dallas] to Boston$_{Y}$]]

(19) a. $_{S}$[$_{X}$[Give me all flights] from $_{Z}$[Dallas] to Boston]
b. $_{S}$[Give me $_{Y}$[all flights from $_{Z}$[Dallas] to Boston]]

$S \rightarrow X$ from Z to Boston	(1)	$S \rightarrow$ Give me Y	(1)
$X \rightarrow$ Give me all flights	(1)	$Y \rightarrow$ all flights from Z to Boston	(1)
$Z \rightarrow$ Dallas	(2)		

To illustrate the difficulties with extracting subtrees from fuzzy trees, consider the fuzzy tree in 18.[11] The tree structures in 19 can be extracted from the fuzzy tree. However, if these two sentences are used to extract

[11] Closing brackets are also labelled where needed to show to which opening bracket they belong.

subtrees from (in this case only subtrees of depth one are extracted), the grammar rule "$Z \rightarrow$Dallas" is extracted twice (the counts are in brackets following the grammar rules). Since the hypothesis was present in the original fuzzy tree only once, the preferred count is one. This phenomenon always occurs when hypotheses that do *not* overlap are in a fuzzy tree that *has* overlapping hypotheses. Extracting deeper subtrees further complicates this issue.

20.5.2 Adding a parsing phase to ABL

So far, ABL has been applied to the plain sentences extracted from a structured corpus. The structured sentences generated by ABL were compared to the sentences in the original structured corpus for evaluation purposes. This makes ABL a *structure* induction system.

This section discusses an extension of the original ABL system by adding a grammar extraction and a parsing phase. This extended system (called parseABL) first generates structured sentences using the ABL method and then extracts a grammar from these structured sentences. Finally, the plain sentences are re-parsed using this stochastic grammar. Since parseABL not only outputs structured sentences, but also generates a grammar, it is called a *grammar* induction system.

Note that adding a parsing phase to the ABL system seems similar to adding a bootstrapping step for DOP as described in section 20.4.1. The overview as depicted in figure 2 perfectly matches the overview of parseABL and the entire working is exactly the same. There is, however, a subtle difference. In section 20.4.1 the emphasis is on improvements on the *DOP* framework. This section will show how extending ABL with DOP solves certain problems in *ABL*.

Problems

The main idea behind selection learning is that the most probable combination of hypotheses should be chosen. This means that the probabilities of all possible combinations of (overlapping *and* non-overlapping) hypotheses should be computed. Instead, only probabilities of combinations of *overlapping* hypotheses are computed and from these the best are chosen. This means that non-overlapping hypotheses are *never* removed. However, it may be the case that incorrect non-overlapping hypotheses are found by the alignment learning phase, while these are never considered to be removed.

(20) a. ... from $_{\text{X}}$[Philadelphia] to $_{\text{Y}}$[Boston]
b. ... from $_{\text{X}}$[Boston] to $_{\text{Y}}$[DFW]

Another problem is depicted in example 20. In these sentences *from* and *to* serve as "boundaries". The boundaries serve as a placeholder for

hypotheses. Hypotheses between *from* and *to* are always of type X and hypotheses after *to* are of type Y. However, *Boston* now has both types depending on the context. This may be correct if type X is considered as a "from-noun phrase" and type Y as a "to-noun phrase", but normally *Boston* should have only one type (e.g. a noun phrase).

Solutions

Adding the extraction and parsing phases has two major advantages. First of all, parseABL outputs a grammar, which can be compared to grammars generated by other grammar induction systems. In addition, computing the probabilities of all combinations of hypotheses in selection learning can be simulated. This effectively solves both problems described above.

Comparing grammars is usually done by comparing constituents in sentences *parsed* by the different grammars (Black et al. 1991). Therefore, the evaluation step takes the parsed sentences (instead of the trees generated by the selection learning phase as in ABL) and the original trees from the treebank. Several metrics can now be used to compare the two structured corpora thus comparing the underlying *grammars* indirectly.

During selection learning only overlapping hypotheses are considered for selection, since not all of them can be correct. Non-overlapping hypotheses, however, are considered correct. Unfortunately, even non-overlapping hypotheses may be incorrect, so instead non-overlapping hypotheses should be used in selection learning as well.

As said earlier, taking all hypotheses into account when selection learning is not possible, since the selection learning phase only selects hypotheses with the highest probability. When all hypotheses are considered for deletion, hardly any will be left in the final structure. Instead, we try to simulate this process by re-parsing the plain sentences using a grammar. This grammar is extracted from the structured sentences as found by ABL.

Parsing may find other, more probable parses and thus more probable constituents. This simulates selection learning with *all* hypotheses. Finding more probable hypotheses can happen both on lexical level (when for example *Boston* with type X in example 20 is more probable than with type Y) and on higher levels without lexical items in the grammar rules.

Results

Table 2 shows the results of applying default ABL and parseABL with the leaf selection learning system to the ATIS corpus. The final entry (with depth indicated as -) denotes the results of the standard ABL

system (default alignment learning and leaf+ selection learning). The three other entries are instances of the parseABL system. The depth indicates how deep the subtrees in the extracted grammar are. Depth one comes down to parsing with a SCFG, while depth two means parsing with a Tree-DOP model with subtrees of maximum depth two.

D	UR	UP	F
1	24.87 (0.54)	56.79 (1.00)	34.59 (0.69)
2	25.62 (0.17)	55.38 (0.49)	35.03 (0.25)
3	25.79 (0.19)	54.74 (0.43)	35.06 (0.26)
-	25.82 (0.19)	54.73 (0.42)	35.09 (0.25)

TABLE 20.2 Results on the ATIS corpus including the parsing phase: D=subtree depth, UR=unlabelled recall, UP=unlabelled precision, F=F-score

Each of the reparsed corpora have a lower recall, but a higher precision. When the maximum tree depth is increased, the results grow closer to the unparsed treebank. Since the DOP system has a preference for shorter derivations and thus has a preference for the use of larger subtrees (Bod 2000), the parseABL instances that have a higher maximum tree depth will prefer the larger parts of the structures. This corresponds to the structures that are present in the unparsed treebank generated by the standard system. Increasing the tree depth even more will probably yield results similar to those of depth 3 and to the unparsed treebank.

20.5.3 Recursive definition

If DOP uses ABL (section 20.4) and ABL uses DOP (section 20.5) at the same time, there seems to be an infinite loop between the systems, which is impossible to implement. However, when taking a closer look, DOP is extended with ABL as a bootstrapping method or to improve robustness. On the other hand, ABL is extended with DOP as an improvement to the stochastic evaluation function or to reparse sentences. In the latter case, there is no need for the robuster version of DOP. The DOP system that is used to extend ABL is not the extended DOP system, so effectively there is no recursive use between both systems.

20.6 Summary

Although DOP and ABL are completely different systems with different goals and assumptions, they appear very similar. Both frameworks first build a search space, which is disambiguated afterwards. DOP first parses a sentence using subtrees, building a parse forest. Using for example Monte Carlo sampling, the most probable parse is chosen, effectively

searching the parse forest. By aligning sentences, ABL constructs a set of (possibly overlapping) hypotheses in a sentence. After constructing this set, the 'best' structure is then selected by searching the possible hypotheses for the best structure.

Apart from the similarities between the two systems, they are also closely linked in their use. This chapter discussed several possible interrelations between the ABL and DOP frameworks.

ABL can be used as an addition to the DOP framework. This chapter describes two ways. Using ABL to bootstrap an initial treebank or to enhance the robustness of the grammar of subtrees of DOP.

DOP can also be used as an extension to ABL. The stochastic model of Tree-DOP can replace the simpeler models leaf and branch in the selection learning phase of ABL. Furthermore, the ABL framework can be enhanced with a parsing phase, reparsing the plain sentences with the grammar learned earlier. Unfortunately, this does not improve the performance.

References

Association for Computational Linguistics (ACL). 1997. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL) and the 8th Meeting of the European Chapter of the Association for Computational Linguistics (EACL); Madrid, Spain.* July.

International Conference on Computational Linguistics (COLING). 2000. *Proceedings of the 18th International Conference on Computational Linguistics (COLING); Saarbrücken, Germany.* July 31–August 4.

Black, E., S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In *Proceedings of a Workshop—Speech and Natural Language*, 306–311. February 19–22.

Bod, Rens. 1998. *Beyond Grammar—An Experience-Based Theory of Language.* CSLI Lecture Notes, Vol. 88. Stanford:CA, USA: Center for Study of Language and Information (CSLI) Publications.

Bod, Rens. 2000. Parsing with the Shortest Derivation. In COLING (2000), 69–75.

Bonnema, R., R. Bod, and R. Scha. 1997. A DOP Model for Semantic Interpretation. In ACL (1997), 159–167.

Booth, T. 1969. Probabilistic Representation of Formal Languages. In *Conference Record of 1969 Tenth Annual Symposium on Switching and Automata Theory*, 74–81.

Caraballo, Sharon A., and Eugene Charniak. 1998. New Figures of Merit for Best-First Probabilistic Chart Parsing. *Computational Linguistics* 24(2):275–298.

Collins, Michael. 1997. Three Generative, Lexicalised Models for Statistical Parsing. In ACL (1997), 16–23.

Harris, Zellig S. 1951. *Structural Linguistics*. Chicago:IL, USA and London, UK: University of Chicago Press. 7th (1966) edition. Formerly Entitled: Methods in Structural Linguistics.

Kehler, A., and A. Stolcke. 1999. Preface. In *Proceedings of a Workshop—Unsupervised Learning in Natural Language Processing; Maryland:MD, USA*, ed. A. Kehler and A. Stolcke. June.

Marcus, M., B. Santorini, and M. Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: the Penn Treebank. *Computational Linguistics* 19(2):313–330.

van Zaanen, Menno. 1999. Bootstrapping Structure using Similarity. In *Computational Linguistics in the Netherlands 1999—Selected Papers from the Tenth CLIN Meeting*, ed. Paola Monachesi, 235–245. Utrecht, the Netherlands. Universteit Utrecht.

van Zaanen, Menno. 2000a. ABL: Alignment-Based Learning. In COLING (2000), 961–967.

van Zaanen, Menno. 2000b. Bootstrapping Syntax and Recursion using Alignment-Based Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ed. Pat Langley, 1063–1070. Stanford:CA, USA, June 29–July 2. Stanford University.

Viterbi, A. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Institute of Electrical and Electronics Engineers Transactions on Information Theory* 13:260–269.

Wagner, Robert A., and Michael J. Fischer. 1974. The String-to-String Correction Problem. *Journal of the Association for Computing Machinery* 21(1):168–173.