Instance-based Natural Language Generation

Sebastian Varges

Doctor of Philosophy

Institute for Communicating and Collaborative Systems

School of Informatics

University of Edinburgh

2003

Abstract

In recent years, ranking approaches to Natural Language Generation have become increasingly popular. They abandon the idea of generation as a deterministic decision-making process in favour of approaches that combine overgeneration with ranking at some stage in processing.

In this thesis, we investigate the use of instance-based ranking methods for surface realization in Natural Language Generation. Our approach to instance-based Natural Language Generation employs two basic components: a rule system that generates a number of realization candidates from a meaning representation and an instance-based ranker that scores the candidates according to their similarity to examples taken from a training corpus. The instance-based ranker uses information retrieval methods to rank output candidates.

Our approach is corpus-based in that it uses a treebank (a subset of the Penn Treebank II containing management succession texts) in combination with manual semantic markup to automatically produce a generation grammar. Furthermore, the corpus is also used by the instance-based ranker. The semantic annotation of a test portion of the compiled subcorpus serves as input to the generator.

In this thesis, we develop an efficient search technique for identifying the optimal candidate based on the A^*-algorithm, detail the annotation scheme and grammar construction algorithm and show how a Rete-based production system can be used for efficient candidate generation. Furthermore, we examine the output of the generator and discuss issues like input coverage (completeness), fluency and faithfulness that are relevant to surface generation in general.

Acknowledgements

Although it is a common experience, it is still true that writing a Ph.D. is more work than one anticipates. A lot more, in fact. Before moving to the influences at the University of Edinburgh that enabled me to carry out this work, I would like to thank my parents and Jen-Yi Lin for their moral support during this long period of time. Furthermore, the financial support of the British EPSRC and the German Academic Exchange Service (DAAD) allowed me to concentrate on my studies without worrying about how to pay my rent.

Chris Mellish and Jon Oberlander have been great thesis supervisors from whom I learnt a lot about how to conduct scientific research. They were at the same time patient, encouraging, critical and inspiring, and always left it up to me how I wanted this work to continue. A crucial factor that makes Edinburgh such a great place to carry out research is the wealth of ideas you can tap into to form your opinion about current developments. This was especially true at the beginning of this Ph.D. project. It is impossible to name all the influences – I would simply like to mention Chris Brew, from whom I learnt a lot about statistical NLP, and Steve Finch, who interested me in information retrieval methods.

Finally, I would like to thank all the people that made my time in Edinburgh such a memorable experience, starting from the MSc class of 1998/99 to the Ph.D. students and researchers I met in the years after. I know it is unfair to the others but let me just mention Mark Core, who waded through endless lists of output sentences, and Katja Markert, who tolerated my habits and at some points permanent presence in the office.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Sebastian Varges*)

Table of Contents

Chapter 1

Introduction

Ambiguity is one of the most salient characteristics of natural language. This has led the field of computational linguistics to develop techniques for exploring the potentially large space of possible analyses for natural language phenomena and for disambiguating between alternative analyses. The ranking of alternatives has been common for a long time in research fields such as speech recognition (SR; e.g. Huang et al., 2001), machine translation (MT; e.g. Berger et al., 1994) and information retrieval (IR; e.g. Baeza-Yates and Ribeiro-Neto, 1999). Since the beginning of the 1990s, corpus-based methods have been extensively used in parsing but much less so in natural language generation (NLG). However, the seminal work of (Knight and Hatzivassiloglou, 1995; Langkilde and Knight, 1998a) has sparked interest in ranking approaches to NLG and has been followed by work by Ratnaparkhi (2000), Bangalore and Rambow (2000a) and others. This thesis should be seen as being part of a larger research program that explores ranking approaches in natural language generation.

The task of generating natural language can broadly be separated into determining the content ("what to say") and determining its realization ("how to say it"). In this thesis, we are concerned with the task of realizing a given meaning input. We view realization in natural language generation as a one-to-many mapping from meaning representation to surface strings, combined with a ranking scheme. The task is to make a choice between alternative ways of realizing some meaning input. This is in contrast to the more traditional view of 'generation as explicit choice', i.e. as a deterministic, decision-making process (see Appelt, 1985, for example). The new approach can be regarded as replacing some of the explicit choices of more traditional generators by decisions based on machine learning.

Assuming that we are employing explicit grammar rules – in contrast to some approaches to statistical machine translation, for example – realization in NLG entails two tasks that do not necessarily need to be separated in practice: First, we need to explore the space of possible *candidates* that realize the meaning input by employing the available grammatical resources. Second, a method for choosing between alternative output candidates needs to be defined. This is essentially a ranking task. In many cases, we will be most interested in the single best candidate but a generation output containing several ranked candidates might be useful as well, for example for re-ranking in a wider textual or extra-linguistic context.

A basic assumption of the approach to natural language generation explored in this thesis therefore is that of a hybrid system containing a grammar that maps meaning inputs to strings and a ranker that chooses between alternative output candidates. In order to make such an overgeneration-and-ranking approach to NLG feasible, we propose to interleave grammar-based construction of candidates with ranking so that the search space can be restricted early on during processing. However, this does not compromise our view that natural language processing (NLP) can in principle be separated into two parts: opening up the space of possible solutions and disambiguating between solutions within this space.

The particular ranking method employed in this thesis is based on instance-based learning (IBL; see Aha et al., 1991, for example). IBL systems store previously encountered instances in memory and use them directly to process new input, rather than abstracting them into some statistical distribution. Instance-based learning is known to be able to learn exceptions in data well and adapt to subregularities. Since these are highly desirable properties for natural language processing in general, instance-based methods have been used in NLP under various names (*example*-based, *memory*-based, *case*-based; see (Daelemans, 1999) for an overview). Furthermore, they have been used in many real-world applications as well (e.g. Aha, 1998). In this thesis, we explore the application of instance-based learning to candidate ranking in natural language generation. In particular, we draw an analogy between instance-based learning and information retrieval by noting that IR methods are essentially performing a form of nearest-neighbour search when computing the similarity between text documents.

The field of natural language generation is haunted by two notorious problems: first, the question where the input to the generator should actually come from and how it can be constructed (McDonald, 1993); second, the problem of acquiring the knowledge to perform the task. The latter is often referred to as the *knowledge acquisition bottleneck* and is a problem for NLP in general. Any exploration in the field of natural language generation has to address these issues. The traditional answer is to hand-build a sophisticated grammar and assume as input to the generator an equally sophisticated structure to match the grammar formalism.

In this thesis, we employ an input format similar to the templates often used in information extraction (IE). These often make use of semantic roles (e.g. Collins and Miller, 1998). The task of an IE system is to associate substrings of the analysed text with these roles. In some respect, the task of a realizer is the reversal of an information extraction process. The advantage of such a relatively simple input format is its non-commitment to any of the highly specialized grammar formalisms that have been developed in the field of computational linguistics. This is intended to ease the problem of providing the realizer with the input it requires. On the other hand, such a simplified generation input is unlikely to provide the realizer with enough knowledge to make fine-grained linguistic choices. It is the main idea of an overgeneration approach to NLG to fill these "knowledge gaps" (Knight and Hatzivassiloglou, 1995) by corpus-based ranking methods.

In order to alleviate the knowledge acquisition problem on the grammar side, we use a corpus-based approach also for constructing the grammar. We aim at taking advantage of the advances in statistical parsing by using the (in most cases syntactic) output of statistical parsers to form the basis of a generation grammar. To this end, we need to augment the syntactic information produced by the parser with semantic information that corresponds to the input representations of the generator. This is the place in our approach where manual effort is required. However, once a semantic markup of a generation corpus has been obtained, we can automatically construct grammar rules that map meaning input to surface strings. This effectively shifts the focus of manual effort from grammar development to semantic annotation. Since in our approach the instance-based ranker requires a set of semantically annotated examples in any case, using the annotation for the grammar component as well reduces the overall development effort.

1.1 Problem statement

In this thesis, we address the problem of how to make an instance-based approach to surface realization in natural language generation work and investigate the properties of such a system. We explore how to take advantage of corpus-based methods to address the knowledge acquisition problem and limit the requirements on the generator input.

1.2 The main ideas

In this section, we present in greater detail the ideas underlying this thesis. Our goal is to provide the reader with an understanding of how the different parts of this thesis – and the implemented surface generation system – fit together and also what motivates them.

1.2.1 Corpus-based methodology for NLG

A corpus-based approach to natural language generation quite obviously involves the collection of a human-authored corpus. In the context of natural language generation, we will at least require the corpus to contain surface strings and meaning representations. It should be noted that such a set-up implies that content determination has already taken place. The corpus can only tell us about the information the human authors chose to express, not the information they chose to ignore or the real-world context of their decisions. On the other hand, for the purpose of this thesis, a corpus of meaning-string pairs is what is required since we are interested in realizing a given content.

As in other corpus-based approaches to NLP, the corpus needs to be separated into training and test sets. The training set can be used to obtain knowledge about the task at hand. To test our system, we need to take the meaning representations of the test set and present them to the generator (in the context of this work, 'generator' refers to a surface realization system). The separation of test set from training set is valuable as it clearly identifies the resources available to the generator and provides a way of testing the system on real-world data.

Corpus-based approaches aim at acquiring knowledge automatically by means of machine learning techniques, as opposed to having knowledge provided manually (in the form of knowledge bases, grammars etc.). However, as is widely known, creating the training material for machine learning approaches can require considerable manual effort. In this thesis, we aim to take advantage of the output of statistical parsers which have been developed independently of any NLG application. In other words, the idea is to reuse existing resources. In practice, the corpus employed in this work is a subcorpus of the Penn Treebank II (Marcus et al., 1993). This implies that a syntactic analysis is already available so that we do not need to employ a statistical parser. The Penn Treebank is frequently used for training parsers (e.g. Collins, 1999) so that it should be possible to obtain syntactic structures for newly compiled generation corpora as well. Taking advantage of the Penn Treebank in the way proposed in this work is a novel form of reuse because the treebank has not been constructed with natural language generation in mind.

In addition to syntactic structures, a generation corpus also needs to contain semantic information of the kind that is expected to be given to the generator. In the work reported here, we provided this annotation manually. On the other hand, named entity recognition and other techniques might fruitfully be employed to aid semantic annotation with the goal of developing natural language generation systems.

1.2.1.1 Domain specific corpus

The corpus chosen for this work is domain-specific. We would characterize domain specificity by a certain semantic overlap between the texts in the corpus. The alternative is a corpus of entirely unrelated texts which in turn would make it difficult to apply knowledge obtained from a training portion of the corpus to a test portion. In fact, other corpus-based NLP tasks that involve a level of semantic representation are also domain-specific, for example many information extraction tasks.

In this work, we aim at generating sentences in the management succession domain, examples of which can be found in the *Who's News* section of the *Wall Street Journal*. This domain was used in the information extraction task of the Sixth Message Understanding Conference (MUC-6, 1995). Consider the following example:

(1) Carl E. Pfeiffer, chief executive officer, was named to the additional post of chairman of this specialty-metals manufacturing concern. Robert C. Snyder, a director and chief operating officer of the company, succeeds Mr. Pfeiffer as president. `(wsj_0368)`

In the *Wall Street Journal*, these texts are usually preceded by a headline containing name and location of the company in question. We identified 144 of such texts in the Penn Treebank II (as in the example above, the headline has often not been preserved). A collection of 144 texts is obviously not large. However, it is important to investigate how corpus-based NLG can deal with the problem of limited training data. Furthermore, the generation corpus – for which we assume a standard syntactic analysis can be obtained automatically – still needs to be semantically annotated. Since this involves manual effort, it is advantageous to be able to work with corpora of limited size.

1.2.1.2 Flat input semantics

In a corpus-based approach to surface realization, the semantic annotation of the generation corpus corresponds to the assumed input to the generator. The choice of annotation therefore is influenced by considerations of the generation task on the one hand, and the ease of annotation on the other. In the context of an overgeneration-and-ranking approach, we are interested in an input format that facilitates the generation of paraphrases. As has been pointed out by Nicolov et al. (1996), 'flat' (or 'non-hierarchical') input structures can increase paraphrasing power. In contrast, input in the form of more traditional predicate-argument structures tends to bias realization in favour of specific lexical entries. For example, the input may only match verbal entries. Flat input structures have also been used in approaches to chart generation (Kay, 1996; Shemtov, 1998a) and are in accordance with some developments in theoretical computational linguistics (e.g. Copestake et al., 1999). The annotation scheme developed here adheres to the idea of a flat generation input. For example, the first sentence of (1) is annotated as follows:

(2) [INPERSON_FULLNAME Carl E. Pfeiffer], [INPERSON_OTHERPOST_NODET chief executive officer], was named to the [POST_DESCR_ADJ additional] post of [POST_NODET chairman] of this [COMP_DESCR* specialty-metals manufacturing concern].

In this thesis, we limit ourselves to the generation of the first sentences of the articles found in the *Who's News* section of the *Wall Street Journal*. (A brief discussion of issues concerning discourse generation within our approach is provided in section 8.4.1.) The input to the generator is an (unordered) set of attribute-value pairs.[1] For example, from the annotated sentence (2) we extract generation input (3) and expect the generation system to produce a template like (4). Into such a template, the values of (3) can be inserted:

(3)

INPERSON_OTHERPOST_NODET	chief executive officer
POST_DESCR_ADJ	additional
COMP_DESCR*	specialty-metals manufacturing concern
INPERSON_FULLNAME	Carl E. Pfeiffer
POST_NODET	chairman

(4) INPERSON_FULLNAME , INPERSON_OTHERPOST_NODET , was named to the POST_DESCR_ADJ post of POST_NODET of this COMP_DESCR* .

Since the tags also occur in the generated surface string, we regard surface generation as the problem of ordering tags and filling the gaps between them with natural language words. The task is to find most *fluent* completion of the underspecified input where 'fluency', or 'naturalness', is interpreted as being likely to belong to a corpus of human-authored texts.

[1]For content determination, a task outside the scope of this work, this implies not only the selection of attributes but also the generation of the appropriate slot fillers.

1.2.1.3 Automatic grammar construction

The syntactic analyses provided by the Penn Treebank can be combined with the annotation to yield a grammar that maps meaning inputs to surface strings. To this end, we first merge markup and treebank and then recursively construct context-free rules that form the backbone of the generation grammar. As we will see, automatic grammar construction produces grammars that contain a potentially large number of rules.

In chapters 4 and 5, we describe the principles of the semantic annotation of the domain corpus and develop the grammar construction algorithm.

1.2.2 Overgeneration-and-ranking architecture

In this work, we explore the use of an overgeneration-and-ranking architecture for the realization of meaning inputs like (3) above. Such an architecture is hybrid in nature as it combines a rule-based component with a ranking component that makes use of machine learning techniques. Ranking approaches to NLG are a relatively recent innovation, pioneered by the seminal work of (Knight and Hatzivassiloglou, 1995; Langkilde and Knight, 1998a) who present a sentence realizer that uses ngram models trained on large corpora to rank candidate sentences.

In the area of 'traditional' NLG (which we would characterize as purely rule-based, i.e. not involving any ranking of alternatives), architectural issues have received significant attention. As a result of an analysis of several applied NLG systems, Reiter (1994) proposed a general 'pipeline' architecture consisting of content determination, sentence planning and linguistic realization. In many applied systems, the sentence planning stage mediates between content determination and linguistic realization. The sentence planning stage is described as mapping conceptual structures onto linguistic ones, including lexical choice, aggregation of propositions into clausal units and referring expression generation. The RAGs project (see Cahill et al., 1999, 2000, for example) provided a set of declarative representations of the various levels of linguistic representation in generation.

Realization from a set of attribute-value pairs such as (3) involves linguistic realization. It also encompasses lexical choice and may include aggregation and other tasks as well. In chapter 7, we examine the choices made by the generator and discuss the issue of input that is not necessarily sentence-sized.

1.2.2.1 Motivation for overgeneration approach

Discussions about generation architectures are often motivated by *interdependencies between choices* at various levels of linguistic description (e.g. Ward, 1994). Decisions at one point in the generation process can have consequences for the availability of choices at other points. While this problem may not be so apparent when dealing with small-scale grammars and applications, it becomes more urgent when attempts are made to scale systems up.

An example of interdependencies is the 'French pronoun case'. Citing (Danlos and Namer, 1988), Reiter (1994) discusses the problem of references to specific objects becoming ambiguous when the pronouns *le* and *la* are abbreviated to *l'* in certain phonological contexts. The interaction between decisions on the levels of referring expression generation and phonology seems to require a corresponding interaction between the modules that are responsible for these tasks. Cases like this one have motivated generation architectures that enable feedback between modules or are 'integrated' in such a way that an interaction between different parts of the system is facilitated. For example, Mellish et al. (2000) propose an architecture in which modules communicate via a central cumulative blackboard, allowing the flow of communication to be flexible.

An overgeneration-and-ranking approach changes the perspective on problems that seem to require complex communication between modules. An overgeneration-based system would propose a large number of possible output candidates. In the French pronoun example, some would contain abbreviated determiners and some would not. It is then left to the ranker to decide which candidate it prefers. There is no need for a phonological component to ask the referring expression generator whether a different choice of words is possible, for example. This can result in a simplified grammar component because there is less need to handle complex interactions between constraints.

In the approach presented in this thesis, a single (and simple) grammar is used to perform tasks that in purely rule-based approaches are often carried out by different modules. On the other hand, an overgeneration-and-ranking approach requires the definition of a ranking model that is able to make decisions left open by the grammar component. Yet, the approach promises several advantages that will be investigated in this thesis:

- **Robustness.** The overall system will generally be more robust in the sense that it can produce a result even if no perfect candidate can be constructed.
- **Trade-off between goals.** The generator can deal with conflicting goals by choosing candidates that are not optimal with respect to any individual goal but represent the best compromise.
- **Separation of candidate construction and ranking.** A hybrid overgeneration-and-ranking architecture does not require the ranker to work with the kinds of representations used by the grammar. In other words, the ranker can work with representations that cross the boundaries of the grammatical units used to build up the candidates. This contrasts with approaches that augment existing grammar formalisms with probability scores, like Probabilistic Context-Free Grammars (PCFGs) or Stochastic Lexicalized Tree Adjoining Grammars (Schabes, 1992; Resnik, 1992), for example.

  Furthermore, the separation of the generation system into two components does not hold the ranker responsible for proposing structure. It can concentrate on choosing between candidates that are made available by the rule-based component. As a result, an overgeneration-and-ranking architecture can consist of two relatively simple components that contribute different kinds of knowledge.

The most pure instantiation of an overgeneration-and-ranking architecture is one in which the ranker evaluates entire candidates. The alternative, to mix ranking and candidate construction such that partial solutions are ranked (and, crucially, pruned) runs the danger of missing globally best solutions because of local pruning decisions. It therefore seems that we need a new variant of a pipeline architecture (a 'fat' pipeline) in which candidate construction has finished before ranking starts.

In this work, however, we develop an alternative approach. We propose an *interleaved architecture* that eases the burden of candidate construction while still maintaining the ability to find globally best candidates. This architecture allows grammar interpreter and ranker to mutually constrain each other. Such an interleaved generation architecture is orthogonal to the dimensions of linguistic description (morphology, syntax etc.) and to the tasks of standard generation modules (sentence realization, content planning etc.).

1.2.3 Instance-based ranking

There are a large number of machine learning techniques that could potentially be applied to candidate ranking in natural language generation. Taking advantage of a training corpus means employing a supervised learning method. Instance-based learning is a *lazy*, supervised learning method that simply stores training set examples ('instances') until a new input is processed. At run-time, new inputs are compared to each instance in the training set (the 'instance base'). In other words, the instance-based learner computes the *nearest-neighbours* of the new inputs (Cover and Hart, 1967). Instance-based learning has become known under a variety of names, such as 'case-based reasoning', 'analogical reasoning' and 'exemplar models' in the fields of Artificial Intelligence and Cognitive Science. It has a number of properties that can be considered the strengths of this learning method (see Wilson, 1997; Aha, 1998, for example):

- Learning is fast. In general, storing instances in the instance base (or converting them into a suitable representation) is linear to the size of the training set.
- "It is intuitive and easy to understand, which facilitates implementation and modification." (Wilson, 1997)
- It can explain its decisions by pointing to the nearest-neighbour(s) that motivated them.
- It can learn complex 'target functions' by keeping a collection of relatively simple local functions. It therefore tends to learn exceptions in data well.

The last point merits further explanation. In contrast to IBL, most other learning methods are *eager* because they generalize beyond the training data. IBL differs from statistical methods in that the training set is not abstracted into a single representation for all instances (a statistical distribution). The ability of lazy learners to learn exceptions is due to the fact that individual training items are not outweighed by a majority vote based on other examples of the training set.

Concerning the task addressed in this thesis, there are a number of expected benefits from using an instance-based approach to candidate ranking:

- **Flexibility.** The instance base can be changed dynamically depending on the input. For example, the ranker can use different subsets of the instance base, or change its response to the same input over time.

- **Learning exceptions.** We expect the instance-based generator to learn language data that exhibits exceptions, subregularities and cooccurrence patterns that are difficult to capture by the representations of conventional grammar formalisms.

- **Dealing with sparseness of data.** Even very few examples may enable the generator to decide what is and what is not a good output sentence. This is particularly important when the amount of training data is limited.

On the other hand, there are a number of typical disadvantages of instance-based learning that mirror some of its general advantages mentioned above:

- **Runtime efficiency.** Although an instance-based learner can typically be trained quickly, processing new inputs can be inefficient because, in principle, these have to be compared to every instance in the instance base.

- **Sensitivity to noise.** A disadvantage of learning exceptions in the training data is the ability to also learn noisy examples.

- When carrying out similarity computations, nearest neighbour approaches typically take into account all features that are used to represent instances and inputs. This can be a disadvantage if the solution depends on only very few features (see Mitchell, 1997).

An instance-based approach to candidate ranking in NLG has to address these issues. The issue of efficiency is of particular importance in an overgeneration approach because of the expected high number of candidates. It will be dealt with by an 'expectation-based' search algorithm (described in chapter 3) within the interleaved generation architecture. In chapter 7, we investigate the implemented generation system with respect to the expected benefits.

1.2.3.1 Information retrieval methods for surface-based ranking

Like any other IBL-approach, instance-based generation has to answer the question how to represent and retrieve its stored instances. Our proposal is to use standard information retrieval techniques for representation and ranking because nearest-neighbour and IR approaches are essentially based on the same concept of computing the similarity between pairs of individual items. Furthermore, the representation of candidates and instances is *surface-based*. Our goal is to find the best sequence of words. This surface-orientation is motivated by the success of (surface-based) ngram language models in speech recognition and statistical machine translation.

1.2.4 Rete-based "bottom-up" chart generation

The grammar construction algorithm yields a potentially large number of grammar rules. These require a suitable grammar interpreter to form the rule-based component of the hybrid generation system. Again, an overgeneration architecture requires one to place particular emphasis on efficiency issues.

Chart-based techniques are an established method for efficient generation since they allow the generator to share subcomputations (Neumann, 1994; Kay, 1996; Erbach, 1997). However, they do not explicitly address the problem of handling large numbers of rules. The approach developed in this thesis is to address the matching problem for large sets of automatically generated grammar rules by employing techniques developed for production systems (Rete networks; Forgy, 1982) which we show are closely related to chart-based techniques. The proposed Rete-based chart algorithm is effectively a purely input-driven 'bottom-up' generator.

Bottom-up algorithms have the general advantage of being robust by simply ignoring pieces of the input that are ill-formed and cannot be combined into larger structures. Furthermore, a flat input semantics also increases robustness by separating the input into different parts. Even if some parts of the input cannot be expressed at all, the generator will still be able to explore combinations of the remaining parts.

1.2.4.1 Integration of chart generation with surface-based ranking

In the Rete-based chart algorithm, the grammar rules generate short phrases (edges) that are recursively combined into larger ones. These edges contain continuous surface strings. A bottom-up (forward-chaining) rule interpreter enables the ranker to score phrases early during processing because the ranker uses a surface-based representation of the edge contents. Thus, the ranking function and the tree-traversal strategy of the grammar interpreter are a good fit.

1.2.5 Overview of implemented system

The implemented system consists of a preprocessing phase and the generation system proper. The preprocessor takes the syntactic treebank and the semantic annotation, merges them, and produces two resources that the generation system employs at run-time: the grammar which is used in the rule-based part of the generation system, and the instance base which is used by the ranker. These two resources represent different aspects of the same training material.

Figure 1.1 depicts the preprocessing phase. Constructing the grammar rules makes crucial use of both semantic annotation and treebank. From the combined structure, the context-free backbone of the generation grammar can be extracted which in turn is transformed into productions in a standard production language. The implementation language we use for the rule-based component of the generation system is JESS (Java Expert System Shell; Friedman-Hill, 2000). It is closely related to the CLIPS expert system language (Riley, 1999).

The instance representations can in principle also be constructed by using features taken from both treebank and annotation. However, in practice we only need to extract terms from the semantically annotated corpus because of our surface-based approach

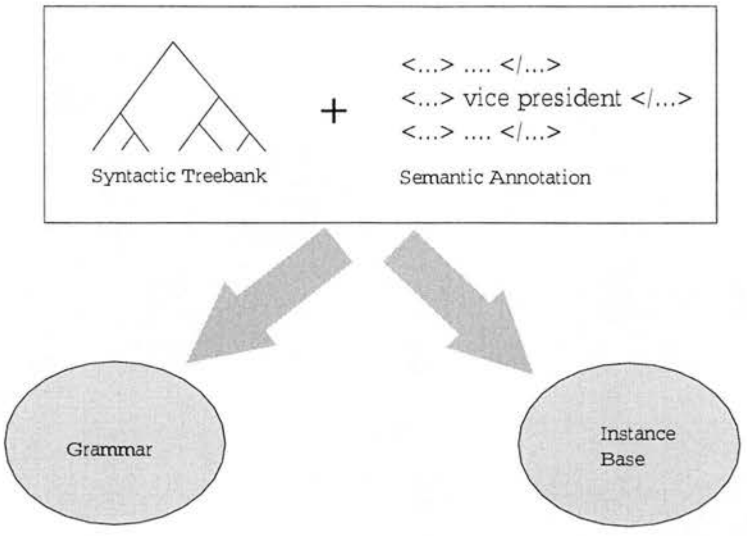

Figure 1.1: Preprocessing in the instance-based generation system

to ranking. The instance representations are stored as serialized Java objects that can be loaded when the generation system is started.

The interleaved architecture of the generation system is shown in figure 1.2. The chart generator passes newly generated edges to the ranker which evaluates them. It then adds the edges to the agenda (unless it decides to block them). The results of the ranker can be used as preference scores on the agenda. To start a new 'cycle', the top-most agenda item is taken from the agenda and added to the chart. Edges that are considered candidates (for example, all edges of syntactic category `S`), are also added to an incrementally increasing ranked candidate list.

1.3 An overview of this thesis

Before we present our approach in detail, we describe relevant previous work on natural language generation and instance-based language processing in chapter 2, including a general comparison between our work and these previous approaches.

Chapter 3 presents a heuristic search algorithm for instance-based generation that

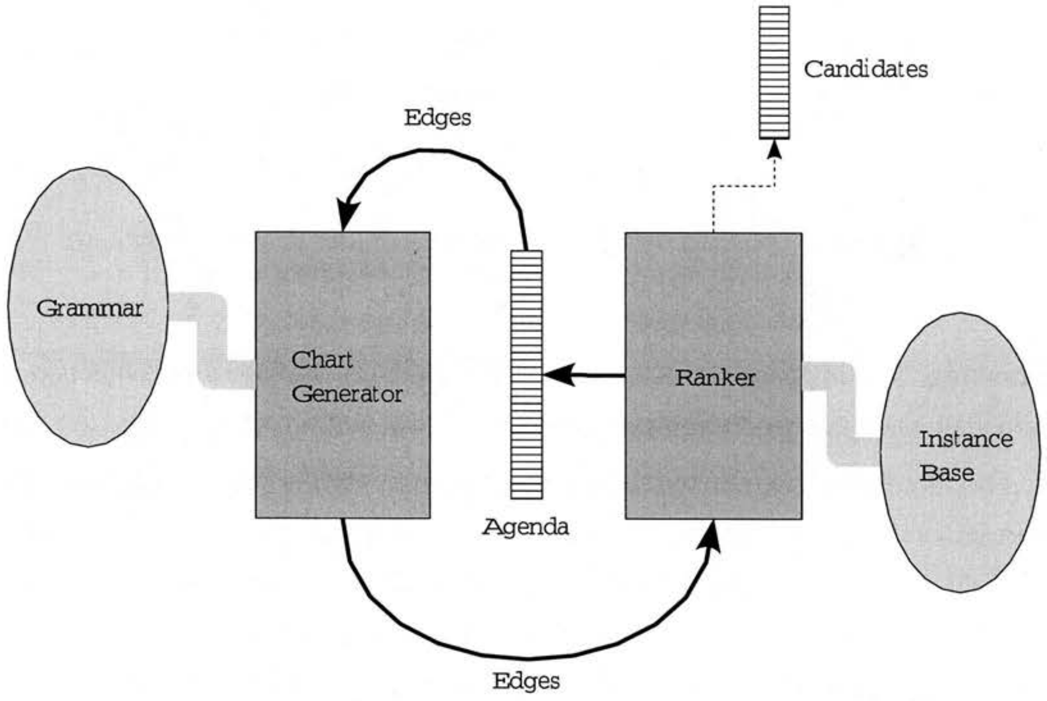

Figure 1.2: Architecture of the instance-based generator

can interact with standard chart algorithms. It shows how decisions about search motivate the overall architecture and how, in practice, the ranking algorithm constrains the grammar interpreter. Furthermore, chapter 3 outlines the similarity metric used for the definition of a nearest-neighbour, investigates an alternative search algorithm and demonstrates the feasibility of the proposed algorithms by means of an empirical study of their efficiency.

Chapter 4 details the semantic annotation scheme for the collected domain corpus.

Chapter 5 shows how a standard syntactic treebank - in combination with the semantic annotation - can be employed to automatically produce a generation grammar.

Chapter 6 presents a bottom-up chart generator that uses a Rete network to share partial matches on grammar rules. This is particularly useful when large numbers of rules are derived automatically. Furthermore, chapter 6 presents empirical results concerning the amount of structure sharing for the grammars used in this thesis.

Chapter 7 describes our findings when running the implemented instance-based generator. It describes several extensions of the basic system based on these findings. For example, it describes the tension between the goals of fluency and completeness in natural language generation and presents two techniques to deal with it.

Chapter 8 presents discussions of several aspects of the approach explored in this thesis: the issue of test set reproduction, the relation of our approach to statistical NLG in general, possible extensions and practical applications of the work presented.

Chapter 9 draws conclusions and summarizes the results of this thesis.

The appendix describes the practical encoding of grammar rules as productions and provides an exhaustive list of the semantic tags used to annotate the domain corpus.

Chapter 2

Previous work

This chapter gives an overview of previous ranking approaches to natural language generation and also discusses a number of related areas of NLP. We review literature in the following areas:

- **Related approaches to NLG.** This section introduces the main reference points for our approach which subscribe to the basic concept of overgeneration-and-ranking for surface realization. The section also describes corpus-based approaches to other NLG subtasks.
- **Instance-based Natural Language Processing.** This section reviews approaches that use 'example-based' or 'memory-based' techniques for NLP tasks other than natural language generation.

In section 2.3, we provide a detailed comparison between our work as described in chapter 1 and the other work described in sections 2.1 and 2.2.

2.1 Related approaches to NLG

In this section, we first briefly examine the 'traditional' paradigm of natural language generation to contrast it with the relatively recent overgeneration approaches. However, we do not attempt to give a comprehensive overview of the history of NLG. We will concentrate on sentence realization since this is the main task addressed in this work. (For a brief discussion of the pipeline architecture in rule-based models of NLG see also section 1.2.2.)

2.1.1 Generation as explicit choice

In the knowledge-based view of NLG, generation is seen as a series of explicit linguistic decisions that ultimately lead to a single output. A number of approaches based on various linguistic theories have been developed for realizing a given meaning input:

- unification-based approaches like Functional Unification Grammar (FUG) (Kay, 1985), unification combined with head-driven search (Shieber et al., 1990) and head-driven search for HPSG (Wilcock and Matsumoto, 1998). Both the `TEXT` system (McKeown, 1985) and `FUF` (Elhadad, 1991, 1993) use formalisms based on Functional Unification Grammar;
- systemic network traversal, embodied in the `PENMAN` system (Mann and Matthiessen, 1985), the `WAG` system (O'Donnell, 1996) and the `ILEX` system (Oberlander et al., 1998);
- realization with Tree Adjoining Grammars (Joshi, 1987; Nicolov et al., 1996; Stone and Doran, 1996);
- realizers based on Meaning-Text Theory (Mel'cuk, 1988) like `RealPro` (Lavoie and Rambow, 1997);
- realization with Categorial Grammar (Calder et al., 1989);
- classification-based approaches (Reiter and Mellish, 1992);
- realization using production systems (Kukich, 1983, 1988).

Systemic approaches may be the most obvious representatives of the explicit-choice model because they generate text by a number of deterministic choices at each point during network traversal. Classification systems are also deterministic. In contrast, unification can be combined with backtracking and other search techniques to explore alternative search paths.

The approaches listed above perform 'deep' generation and tend to expect a large number of explicit features to guide the generation process. In contrast, template-based realization is less demanding concerning the input specifications but offers considerably less flexibility. The difference between deep NLG and templates has sparked some debate in the field (KI-99, 1999). For the purpose of this work, it is important to realize that the demands on the input increase with the expressibility of the generator. If input specifications are missing, default specifications are needed (see McDonald and Meteer, 1988, for example). In the following, we sometimes generically refer to explicit-choice generators as 'rule-based' although they in fact might use 'systems' or 'schemata' rather than 'rules'.

The traditional model of deep NLG follows a knowledge engineering approach. It aims at explicitly representing our understanding of the task and problem domain. Therefore, it inherits a number of typical problems of the knowledge engineering paradigm. Ward (1994) gave a characterization of the problems of rule-based NLG. Apart from the problem of specifying the fine-grained input that is required to choose explicitly between the available options, the knowledge acquisition bottleneck makes it difficult to maintain and extend hand-crafted rule systems. We often find an interaction between decisions at various levels and need to deal with potentially conflicting constraints (see also section 1.2.2). Providing defaults for missing input or grammar specifications is generally haunted by similar problems. In addition, the use of defaults raises the question how they are justified (if not empirically).

These problems have led to the application of machine learning techniques to NLG. The fundamental assumption is that implicit, corpus-based choice – often performed by a statistical model trained on the corpus – can replace part of the knowledge engineering effort required in explicit-choice approaches and keep the input requirements simple. Furthermore, it is assumed to be able to deal with interdependencies and reconcile conflicting constraints in ways that are difficult to emulate by 'hard' constraints.

2.1.2 Statistical surface realization

In this section, we describe four statistical approaches to natural language generation. The first three employ an overgeneration-and-ranking architecture and are therefore relevant reference points for our work. The fourth, a purely statistical realization system, also generates a number of output candidates and provides a ranking function for these. We detail input and output of the described systems, their task and architecture, the corpus used and the kind of evaluation that was carried out (if any).

2.1.2.1 Hybrid surface realization with ngram language model

Probably the most influential statistical approach to NLG is work carried out at the Information Sciences Institute/USC which is why we discuss it here in somewhat greater detail. In (Knight and Hatzivassiloglou, 1995; Langkilde and Knight, 1998a,b), a hybrid sentence realizer called `NITROGEN` is described which combines rule-based overgeneration with ranking based on a bigram language model. The goal is to limit the requirements on input specifications and grammar engineering, in short, to address the "knowledge gaps" (Knight and Hatzivassiloglou, 1995) that limit the usefulness of the generator. The `NITROGEN` system was originally developed in the context of a large Japanese-English newspaper translation system. Machine translation makes it particularly important to be able to deal with missing input specifications because the source language parser may not be able to elicit all features that are required.

`Nitrogen` produces a vast number of alternative realizations in a first step using a rule-based grammar. In a separated second step, these alternatives are filtered by a statistical extractor which uses bigram statistics to find the most fluent (i.e. most likely) realization. Thus, it does not provide missing input specifications directly (by using defaults), but rather chooses them indirectly by selecting one of the sentences already produced. The two-stage pipeline of `Nitrogen` is shown in figure 2.1.

The input to the system is a labelled directed graph similar to the attribute-value matrices used in unification-based formalisms. The input is sentence-sized and generally semantic in nature. The main tasks of the system are lexical choice (mapping concepts to words) and syntactic realization.

The rule-based part of `Nitrogen` consists of grammar rules, lexicon and a morphological component. These are deliberately knowledge-poor in the sense that many linguistically relevant distinctions are not made. The lexicon does not capture syntactic subcategorization frames of lexical items, gradability of adjectives, countability

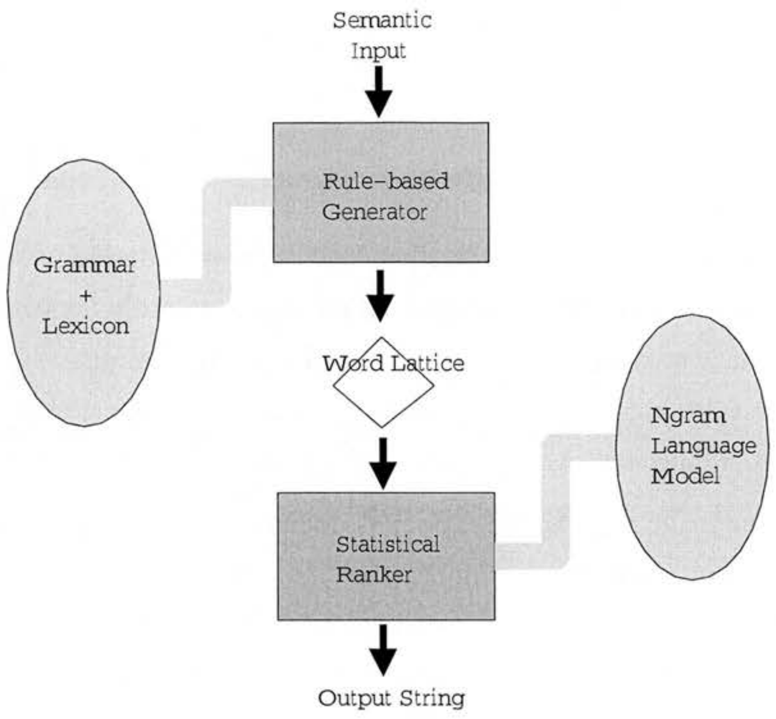

Figure 2.1: Two-stage architecture of `NITROGEN` (adapted from Langkilde and Knight, 1998a)

of nouns etc. On the other hand, the grammar does cover linguistic phenomena like active and passive voice, negation, modality and tense of verbs and accepts input on a number of semantic and syntactic levels. This is necessary to be able to propose a large number of alternatives to the ranker. Furthermore, a recasting mechanism that defines equivalences between input representations is used to increase the paraphrasing power of the generator.

Generation starts by matching the input recursively against grammar rules until lexical entries are reached. Rule expansion proceeds top-down by matching the input feature structure against the left-hand sides of the grammar rules and processing their right-hand sides in turn. The recursion stops when the lexical base case is reached, allowing a bottom-up construction of a word lattice beginning with the most nested levels of the recursion. This word lattice is the output of the rule-based component. Using a word lattice avoids the massive redundancies that would result from simply enumerating alternative realizations sentence by sentence. The statistical ranker extracts the path with the highest probability according to the language model from the lattice.

Recent development efforts on the `Nitrogen` system involve the construction of a forest data structure by the grammar component (Langkilde, 2000). A forest (a non-recursive context-free grammar) has the advantage of greater sharing of substructures than a lattice. Most importantly, it facilitates the use of stochastic lexicalized models of syntax which have proved successful in statistical parsing (Collins, 1999; Charniak, 2000). Recent results presented in (Daume III et al., 2002) indicate that lexicalized models of syntax should improve candidate ranking for surface realization in NLG.

The bigram language model of `Nitrogen` is domain independent as it is trained on a large, general corpus. However, being based on the `SENSUS` knowledge base (Knight and Luk, 1994), the semantic input representation is the result of significant manual effort. Furthermore, the grammar needs to be able to match the wide range of concepts defined by such a knowledge base. Therefore, with any extension of the semantic input language, the grammar component needs to be extended as well.

Langkilde (2002) shows the results of an automatic evaluation of an improved version of the system. In a number of experiments, the ability of the system to reproduce a test portion of the Penn treebank was evaluated (2377 inputs). The input in these experiments was largely syntactic. The task of the generator was mainly to order constituents, to inflect word forms and to insert function words. The experiments showed that generally the quality of the output increases with the degree to which the input is specified. With maximal input specifications, 58% of the output sentences were exact reproductions of the original sentence, dropping to 5% for minimally specified inputs. The evaluation used a number of automatic evaluation metrics. No human judgement of the output was performed. The coverage of the system was measured as the percentage of inputs for which the generator was able to produce an output (about 80%).

The basic philosophy of `Nitrogen` shows some analogies to developments in statistical parsing: output which is sometimes incorrect is accepted for the sake of robustness, broader coverage and speed. In many applications, like machine translation for example, robustness and coverage can be more important than quality. Recent (manual) improvements of the grammar component described in (Langkilde, 2002) aim at achieving greater flexibility concerning the level of input specification so that the system is usable for a wider variety of tasks.

2.1.2.2 Surface realization with supertags and ngram model

Bangalore and Rambow (2000b) present a surface realizer that follows the `Nitrogen` model in that it is a hybrid system organized in a two-stage architecture. The first phase constructs a lattice of alternative realizations from which the path with the highest probability according to a trigram language model is extracted in the second phase. The input to the system, called `FERGUS`, is a syntactic dependency tree labelled with lexemes. The main task of the system as described in (Bangalore and Rambow, 2000b) is to linearize the input tree, excluding lexical choice and morphology.

In the first stage, a stochastic tree model is used to determine supertags (Bangalore and Joshi, 1999) for the words in the input tree. Then, the hand-crafted, wide-coverage XTAG grammar (The XTAG Research Group, 1998) is used to produce a word lattice of all possible linearizations that are compatible with the grammar. Next, the ngram model is used to determine the word sequence with the highest probability. Thus, two different statistical models are used, one at the beginning of processing and one at the end.

In (Bangalore and Rambow, 2000a) an extension of the system is described that addresses the lexical choice problem. The input to the system is now a dependency tree labelled with extended synonym sets rather than lexemes. (Bangalore and Rambow, 2000a) discuss different options of where to fit the lexeme chooser into the generation pipeline.

The tree-based stochastic model of `FERGUS` was trained on 1,000,000 words of a version of the Penn treebank that was converted to XTAG derivations. For evaluation, 100 sentences of this corpus were chosen at random. The experiments used automatic evaluation methods based on the string edit distance between generator output and original corpus sentence. Bangalore and Rambow (2000b) show that the use of the supertag-based stochastic model in combination with the XTAG grammar proper in the first stage improves the performance of the system over a baseline tree model. Results in (Bangalore and Rambow, 2000b) show that taking into account the syntactic context in the dependency tree improves lexical choice from extended sets of synonyms over a baseline model that always chooses the most frequent lexeme.

2.1.2.3 Sentence generation from head words

Uchimoto et al. (2002) describe a two-step approach to generating sentences in Japanese from a list of head words. Possible applications include machine translation (where an additional translation model would be required to generate the head words) and support systems for people with aphasia.

Candidate construction uses rules that generate 'bunsetsus' (phrasal units) when the head word is given. The rules are automatically acquired from a Japanese corpus annotated with the appropriate information (head words and bunsetsus). The proposed bunsetsus for a new input of head words are combined and dependency relationships between the candidate bunsetsus are stipulated. The approach assumes that the input head words are given to the generator in the appropriate order. Uchimoto et al. (2002) do not address any efficiency issues of this overgeneration approach.

The ranking phase makes use of a number of statistical models that incorporate information about dependencies, morphology and word ngrams. These statistical models were trained on corpora of various sizes between about 1000 and 9000 sentences.

The system was evaluated by two human judges using 30 sets of three head words each. The best combination of statistical models resulted in 27 semantically and grammatically correct dependency trees.

2.1.2.4 Surface realization with Maximum Entropy model

Ratnaparkhi (2000) describes a domain-specific generation system which uses maximum entropy probability models to rank generation candidates. The input to the system is a set of attribute-value pairs describing flights in the air travel domain. The output of the system are noun phrases (which can contain relative clauses) in the form of surface words mixed with attributes. The attributes are then replaced by their values. The kind and level of semantic input is similar to our task as described in chapter 1.

Maximum entropy modelling is a general machine learning technique that allows one to incorporate arbitrary features. It has been used for purely statistical machine translation systems such as (Berger et al., 1996). Ratnaparkhi (2000) trains two different models for ranking output sentences. The first uses ngram features; the second uses syntactic dependency features. Furthermore, a baseline model is devised that simply reuses the templates of training sentences without any adaption, i.e. the baseline model cannot handle unseen input.

The approach does not employ a rule-based grammar to construct output candidates. Rather, a search procedure determines the output features by "matching the patterns over the training data" (Ratnaparkhi, 2000). The statistical model learns how to map the input semantics to surface forms directly, including word choice and word order of phrases. The search procedure ensures that the input is completely consumed and that no part of the input is expressed more than once.

For training and testing the system, a corpus of texts in the air travel domain was semantically annotated. There are 26 different attributes to represent flights. 6000 annotated noun phrases were used for training and 1946 for testing. The training corpus for the dependency model also needed to contain the required syntactic dependencies. To evaluate the system, the semantic representations of the test set were given to the generator and two human judges ranked the output according to four criteria ('correct', 'ok', 'bad' and 'no output'). The results showed that the dependency model improves upon the ngram model which in turn was an improvement over the baseline model. The latter achieved over 80% correct (='perfectly acceptable') outputs because of the degree of repetition between test and training data. The best statistical model achieved close to 90%.

2.1.3 Learning methods for other generation tasks

In this section, we describe a number of approaches that apply statistical or other machine learning techniques to NLG tasks. Given the recent rapid development of the field (see the proceedings of INLG, 2002, for example), we do not claim to be exhaustive in this review. Our goal is to show the range of NLG tasks to which learning methods have been applied.

2.1.3.1 Information novelty in a dialogue system

(Ratnaparkhi, 2001) describes a hybrid approach aimed at expressing new information differently to old information in the context of a conversational system in the air travel domain. There are approximately 50 domain-specific, hand-crafted rules specifying template fragments which can be composed by the grammar. The point of the paper is the treatment of information novelty depending on the dialogue state. This is handled by the grammar rules of which there are two kinds for every semantic attribute to be expressed, one rule for new and one rule for old information. The use of the statistical model (an ngram language model) is restricted to cases where there is more than one old and one new attribute so that word order is not fully determined by the grammar rules. The ngram model was trained on about 8000 utterances in the air travel domain. No evaluation was carried out. It was planned to evaluate the system in the context of the overall dialogue system.

2.1.3.2 Determining adjective ordering

Malouf (2000) describes experiments that use a number of statistical and machine learning techniques to determine the order of pre-nominal adjectives. It is suggested that the learned models be used in a distinct adjective ordering module within a larger NLG system (which was not implemented). The starting point of the paper is the observation that simple ngram models do not perform well on the task of adjective ordering: when an ngram model was presented with the permutations of about 5000 adjective sequences taken from the British National Corpus (BNC), only 75.57% were predicted correctly. Malouf (2000) goes on to compare a number of methods for improving adjective ordering. Their correctness was measured on 10% of about 127,000 different adjective pairs taken from the BNC. Because of sparse data problems (most pairs of adjectives only occur once), a method called 'direct evidence' which was adapted from (Shaw and Hatzivassiloglou, 1999) did not perform particularly well. There are two methods, memory-based learning (using a character-based distance metric) and positional probabilities (treating adjectives independently of one another) that achieve close to 90% correctness. The two methods can be combined to yield about 92% correctness. The interesting conclusion of this paper is that a domain-independent model (which is specific to this particular task) performs better than a general purpose ngram language model.

2.1.3.3 Generating nominal expressions

Cheng et al. (2001) investigate corpus-based modifier generation for non-referring nominal phrases. The approach uses a decision tree learning algorithm which was trained on an annotated corpus of 1863 modifiers. The annotation scheme consisted of pragmatic, semantic and syntactic features. The correctness of the realization decisions made by the decision tree was evaluated using 10-fold cross-validation on the available data, achieving a global success rate of 67.5%. The best predictions were made on appositive and possessive modifiers (for which the model achieved close to 89% correctness), and adjectives. Prepositional phrases and posthead NPs were predicted "reasonably well" (Cheng et al., 2001), prehead nouns and posthead participles "rather badly" and relative clauses were not predicted correctly at all.

The learned decision tree was then used within the NP module of the `ILEX` system (Oberlander et al., 1998). The input to the NP module was provided by the aggregation module of `ILEX`. It specified the information to be conveyed by the NP. The rule-based realization decisions were guided by the preferences of the decision tree. The approach assumed a fixed maximal number of pre- and post-modifiers.

Jordan and Walker (2000) address the problem of referring expression generation excluding pronominals. In particular, the (domain-specific) task is to learn what subset of four different attributes describing furniture to include in a specific position in the discourse. There were 393 nominal descriptions in the domain corpus. These were encoded in terms of 58 features which were motivated by theoretical work on discourse processing. The task was framed as a classification problem with 16 classes to choose from. A rule learning algorithm (`RIPPER`; Cohen, 1995) learned an ordered set of if-then rules that can act as a classification model. The best performing rule set achieved 50% correctness as measured against a corpus gold standard. In contrast, a baseline model that just chooses the most frequent class achieved 16% correctness. The experiments showed that some features proposed in the literature were much more effective than others. The paper does not report whether the learned model has been deployed in a working NLG system.

Furthermore, there is research on generating articles, often in the context of machine translation. Recent work on article generation includes (Minnen et al., 2000) who use a memory-based approach (see section 2.2.2) to predict whether to generate *the*, *a/an* or no article.

2.1.3.4 Sentence and text planning

(Walker et al., 2001) present a hybrid sentence planner (called `SPoT`) that follows the overgeneration approach to generation. The specific task is 'sentence scoping', i.e. the choice of syntactic structures for elementary speech acts and their distribution over one or more sentences. The paper focuses on learning a ranking function for candidate sentence plans. These are generated by a simple sentence-plan generator using weighted randomization. For a corpus of 100 text plans generated by a dialogue system, 1868 sentence plan trees were generated offline (with an upper bound of 20 candidate sentence plans per input text plan). These were realized by the `RealPro` realizer (Lavoie and Rambow, 1997) and evaluated by two human judges on a qualitative scale from 1 to 5. A machine learning algorithm based on *boosting* (Freund et al., 1998) was then used to learn conditional rules that change the score of some candidate structure if certain conditions hold. (While the conditions are easily readable, the change of the score is a relative value which is more difficult to interpret.) On average, the learned ranking function performed only 5% worse than the human judges and 36% better than taking a candidate sentence plan at random. In an independent evaluation with 60 judges (which is described in Rambow et al., 2001), it was shown that the choices of the sentence plan ranker were not statistically distinguishable from the output of hand-crafted templates.

(Duboue and McKeown, 2001) use an algorithm based on combinatorial pattern discovery (used in genomics, for example) to identify patterns and ordering constraints between them in a corpus of semantically annotated patient status briefings. The goal was to group and order elementary semantic tags for text planning.[1] The corpus contained 24 transcripts of patient status briefings. The tag set consisted of about 200 tags. The learning task was unsupervised since the training data contained only the semantic tags but not the correct tag groupings (which are difficult to determine manually). The algorithm learned 29 ordering constraints between 23 clusters of tags. Demonstrating the validity of the approach, these constraints were largely replicating the rules of an existing, hand-crafted text planner, and also added some additional information. The learned rules were not yet used in a generation system. The intention was to use them in a top-down planner for selecting and ordering content.

2.1.4 Connectionist language generation

(Ward, 1994) describes a connectionist sentence generator called `FIG` ("Flexible Incremental Generator"). The book also serves as a general exposition of fundamental problems concerning natural language generation. A main motivation is to be able to deal with multiple conflicting goals and interdependent choices. Lexical choice is the main practical problem addressed in this work. The application is Japanese to English machine translation. `FIG` is a spreading activation system (in contrast to distributed connectionist systems) in which concepts, words and syntactic constructions are represented as nodes (or sets of nodes in the case of syntactic constructions). The network computes its activations incrementally for each input concept in turn and the general activation flow is from concepts to words.

The book presents a wealth of ideas. It also proposes a feature representation for semantic information and discusses human language production, for example. No empirical evaluation of the implemented system is reported in the book.

2.1.5 Chart generation

The term 'chart generation' refers to certain algorithmic techniques rather than to particular implemented systems or approaches to NLG in general. We give an overview

[1]The paper talks about "content planning" which is a task similar to "text planning" in other work (see Mellish et al., 1998, for example).

of the literature on chart generation here because candidate generation in our approach is a variation of the chart generation theme (see chapter 6).

Chart generation has often been discussed in the literature under the assumption of a unification-based approach to grammar (Shieber, 1988; Neumann, 1994, 1998). Erbach (1997) proposes a pure bottom-up Earley deduction approach to parsing and generation. This has advantages for processing with grammar formalisms like HPSG which locate most linguistic information in the lexicon. Erbach emphasizes robustness and easy integration with 'preferences' as advantages of bottom-up processing but concedes that efficiency is a problem for bottom-up generation (Erbach, 1997, p108). We contend that this is mainly due to the lack of actual integration with a ranking function that directs search and restricts the search space.

Recent research to a large extent makes use of flat semantic representations for generation. Kay (1996) introduces the concept of internal indices. Internal indices allow a chart generator to prevent an edge from being created if it has not consumed all parts of the input that share some index, assuming that this index is internal to the corresponding grammar rule. An index is defined as internal if it only instantiates variables on the right-hand side of a grammar rule and does not proliferate them to the rule left-hand side. In other words, access to an internal index for incorporating rules is sealed off. A typical example involves adjectival modifiers that are not yet incorporated into an NP. If this NP is combined into a VP, for example, further modification by other modifiers might not be possible anymore. This issue has been taken up by Trujillo (1997) and Carroll et al. (1999). However, we do not aim to adopt the concept of internal indices since we cannot require complete consumption of the input in the general case (see section 7.4).

A number of generation algorithms have been proposed for Shake and Bake Machine Translation where the input to the generator is a multi-set of richly structured signs containing orthographic, syntactic and semantic information. Brew (1992) proposes a combination of shift-reduce parser and constraint propagation. Popowich (1996) presents a chart-based generator. In contrast to our approach, these techniques assume that lexical choice has already taken place. On the other hand, the algorithmic task is similar: generation from a multi-set of input units.

Shemtov (1998b) views generation in the context of machine translation as a many-to-many mapping and introduces packed generation charts as a means of representing alternative renderings without having to enumerate them explicitly. The intention is to handle the problem of ambiguity in machine translation by avoiding disambiguation if possible, i.e. to preserve ambiguities of the source language in the target language. However, this does not seem possible in our approach since the input is unambiguous and we do want to disambiguate the output candidates.

2.2 Instance-based Natural Language Processing

The term 'instance-based' refers to approaches that make decisions based on a nearest neighbour search for new inputs. A number of alternative terms like 'example-based', 'memory-based' or 'case-based' has been used in the literature. In the following, we review two strands of research in NLP other than NLG that use methods we call 'instance-based' in this work.

2.2.1 Example-based Machine Translation

There is an extensive body of research in Example-based Machine Translation (EBMT). A relatively recent review can be found in (Somers, 1999). EBMT systems typically perform the following three steps:

1. (they) match fragments resulting from a source language analysis against a database of examples,
2. identify the corresponding target language fragments and
3. recombine these fragments to produce the target language output.

An important issue in EBMT is the level of representation of the fragments since this affects the required degree of source language analysis, the similarity metrics that are available and the grammaticality of the combined target language fragments. At one end of the spectrum, there are fully annotated tree structures for both source and target language with explicit links between them. For example, Sato and Nagao (1990) populate the example base with linked pairs of word-dependency trees. At the other end of the spectrum, there are approaches that store examples literally or as 'generalized' examples. For example, Brown (1999) generalizes examples by replacing dates and city names etc. with appropriate tokens. The EBMT-component of the `Pangloss` machine translation system (Brown, 1996) performs subsentential alignment at run-time.

The first step performed by an EBMT system is typically called the 'matching' phase. The task is to identify the example or set of examples that most closely matches the source language string. This is the phase which makes use of a similarity metric. Often, matching is string/character-based or uses a thesaurus to determine the degree of semantic similarity between source language input and source language examples. For example, Murata et al. (1999) search for the longest matching substring in the example base to translate tense, aspect and modality from Japanese to English. The stored choices of the English translation of the retrieved example are used for the translation of the input sentence. Other approaches that perform similarity matching on parse trees require appropriate rule-based analysis components (e.g. Sato and Nagao, 1990). These can be regarded as 'hybrid' EBMT systems.

Having identified matching target language examples, the task is to extract the appropriate fragments, possibly adapt them and then recombine them. As Somers (1999) points out, these steps have received less attention in the literature than other aspects of EBMT. In some approaches, substrings retrieved from the example base are simply pasted together. An example is the EBMT component within the `Pangloss` system (Brown, 1996). The approach works best for languages with little or no inflection like Japanese or English. On the other hand, hybrid approaches rely on linguistically motivated grammars to guide analysis and generation. For example, Sato (1995) uses word-dependency analyses of both source and target language in combination with a "non-destructive transfer process" using unification.

Translation Memories are often seen as a starting point for EBMT (e.g. Somers, 1999). They make use of a database of human-authored source and target language pairs. For a new sentence to be translated, they retrieve those sentence pairs that contain similar source language sentences. This method is not very flexible as it does not adapt the examples to the source language text in question. However, it is useful as an aid for human translators, underlined by the fact that Translation Memory systems have proved commercially successful.

Despite the differences between the various EBMT approaches, a common element is the assumption that example retrieval is done before adaption and recombination. In section 2.3.2 we contrast this with the approach adopted in this work.

2.2.2 Memory-based NLP by classification

A number of NLP tasks have been addressed by 'memory-based' techniques using classification.[2] For example, Daelemans et al. (1999b) report on experiments in NP and VP chunking as well as subject/object detection. Training examples are represented as feature-value vectors of fixed dimensionality labelled with a class chosen from a fixed set of labels. Following this methodology, more demanding tasks like full-scale parsing need to be decomposed into a series of classifiers, for example into subtasks like segmentation, disambiguation and attachment resolution.

The memory-based approach to article generation mentioned above (Minnen et al., 2000) also addresses the problem as one of choosing from a fixed set of class labels. However, to the best of our knowledge, full natural language generation has not yet been dealt with in a classification-based framework.

2.3 Relating instance-based realization to previous work

In the following, we compare our instance-based generator as proposed in chapter 1 to the work described in the previous sections. It should be noted that brief descriptions of previous work on automatic grammar construction and the use of production systems for NLG can be found in sections 5.7 and 6.6, respectively.

[2]Note that here the term 'classification' is used in the sense of the machine learning literature. This is different from the 'classification-based' approach to NLG presented in (Reiter and Mellish, 1992).

2.3.1 Comparison to work on statistical surface realization

In contrast to statistical approaches to surface realization, an instance-based realizer stores previously encountered instances in memory and uses them directly to process new inputs, rather than abstracting them into some statistical distribution. Thus, our ranking model is different from the statistical surface realization systems described in section 2.1.2. However, it is similar to ngram models in that it is purely word-based and even uses ngrams extracted from the corpus sentences (which are semantically annotated in our case). On the other hand, unlike some of the statistical realizers, we do not score tree structures.

In contrast to our approach, the `Nitrogen` sentence realizer (see section 2.1.2.1) is a two-stage pipeline without frequent interactions between candidate generator and ranker. On the other hand, both are hybrid systems that use a rule-based grammar and a corpus-based ranker. Furthermore, the `Nitrogen` candidate generator also exhibits a bottom-up element (in the way its lattice is constructed). However, the grammar organization is quite different. The `Nitrogen` generator is geared toward efficient lattice construction, requiring appropriate special-purpose mechanisms. In contrast, we are using a standard chart algorithm with an agenda, albeit one that is implemented in a production system. In our system, the grammar is automatically constructed from a semantically annotated treebank. The `Nitrogen` system uses a hand-built grammar and relies on the semantic formalism provided by a large, manually crafted knowledge base. The semantic input to `Nitrogen` is hierarchically structured and can be defined on various levels of linguistic description. In contrast, the input to our system is a set of attribute-value pairs. Because our grammar is automatically constructed from a corpus, we know how much training material is used by the system. This is much more difficult to estimate for a hand-crafted grammar.

Similar to our approach, the realizer based on maximum entropy modelling described in (Ratnaparkhi, 2000) is domain-specific and uses attribute-value pairs as its input. However, the system is not hybrid since it does not use any grammar rules. Furthermore, it only generates NPs rather than sentences and it is trained on a significantly larger corpus.

The `FERGUS` system (Bangalore and Rambow, 2000a,b) employs a large, hand-crafted grammar which required several person-years to develop. In contrast to our approach, it requires complex input structures (syntactic dependency trees). Like `Nitrogen`, the system architecture is based on a pipeline without frequent interaction between grammar interpreter and ranker. It also constructs a lattice for ngram scoring but in addition uses a tree-based stochastic model at the beginning of processing. In contrast, in our approach and in `Nitrogen`, only the result of grammar construction is evaluated by the ranker.

2.3.2 Comparison to Example-based Machine Translation

A major difference between our approach and EBMT – apart from the task which is not explicitly MT in our case – is the overall architecture. At least conceptually, we first open up a space of possibilities for the ranker to choose from and then rank among the alternatives. In practice, candidate generation and ranking are interleaved to increase efficiency. However, the admissibility of the A^*-algorithm (see section 3.2.1.2.1) guarantees that this does not change the generation output. In contrast, EBMT systems first search for the most similar examples and then try to adapt them. The EBMT approach seems to take a direct route for arriving at a translation. The overgeneration approach takes a more indirect route: we are doing the adaption before/during the computation of the nearest neighbour. 'Initial semantic ranking', a technique which is introduced in section 7.6, comes closest to doing the similarity computation first. However, we only do this to keep the number of rules at a manageable size. The different architecture allows us to use rules taken from all examples as long as they match some portion of the input, opening up a wider range of possibilities than could be achieved by reusing only very few examples at a time.

Like the overgeneration approaches of statistical MT and the statistical realization systems described in section 2.1.2, our architecture allows us to model target language fluency (see also section 8.2). In contrast, target language fluency is typically not taken into account by the similarity metric in EBMT. Rather, EBMT systems perform similarity computations on source language strings in the first processing stage. An interesting exception is the proposal of Sato (1995) to define the score of translation candidates as the combination of the scores of source and target language trees which are composed of dependency-tree fragments. (However, this does not seem to have been applied to real-world data. (Sato, 1995) concentrates on algorithms for tree-to-tree transformations.) The issue of machine translation is taken up again in section 8.6.1.

2.3.3 Comparison to memory-based NLP by classification

In contrast to our approach, the memory-based NLP approach developed by Daelemans and colleagues in Antwerp and Tilburg does not use any rule-based knowledge, i.e. the approach is not a hybrid one. Furthermore, we do not frame our task as a classification problem. Therefore, we do not need to decompose generation into a number of separate decisions as is done in memory-based approaches for the task of shallow parsing (Daelemans et al., 1999b), for example. Because there is no classification, there is no 'carrying over' of a class label from nearest-neighbour(s) to new inputs. As will become clear in the next chapter, we use the raw similarity value to score candidates.

2.4 Conclusion

Our approach subscribes to the basic overgeneration-and-ranking approach to NLG that has been developed for the purpose of statistical generation. In contrast to the `Nitrogen` model and the approach presented in (Ratnaparkhi, 2000), we explicitly aim at dealing with very limited training data. In both cases, it is an open question how well the systems perform when only about 150 examples are available. Furthermore, the system described in (Ratnaparkhi, 2000) only addresses the task of generating nominal phrases. Moreover, in contrast to `Nitrogen` and `FERGUS`, our approach assumes simple, non-hierarchical input structures.

To our knowledge, this is the first work that explores the use of an instance-based ranking function for NLG. In contrast to Example-based Machine Translation, which does similarity testing on the *input* of the system, we do similarity testing on the *output*. In the next chapter, we describe the instance-based ranker.

Chapter 3

Instance-based ranking

In this chapter, we develop a heuristic search algorithm for instance-based generation that can interact with standard chart algorithms. We discuss the similarity metric used for the definition of a nearest-neighbour, investigate an alternative search algorithm and demonstrate the feasibility of the proposed algorithms by means of an empirical study of their efficiency.

Efficiency is of particular importance in an overgeneration-and-ranking approach. Without efficient search, experiments are not possible and there would be no prospect of real-world applications employing the techniques developed in this work. We assume that it is generally not possible to enumerate all output candidates that can be generated by the grammar for a given meaning input. In practical terms, exhaustively enumerating all candidates often takes 30 minutes or runs into memory problems.[1] As outlined in chapter 1, we employ an *interleaved generation architecture* in which the chart generator and the ranker constrain each other. Although slightly non-standard, the Rete-based chart algorithm and the shallow grammar formalism we developed can be seen as place holders for other, more commonly used mechanisms.

In general, we believe that efficiency issues should be addressed even though advances in hardware provide ever faster machines. A more efficient algorithm is generally preferable over a less efficient one. Moreover, computing resources can be limited in certain applications, or an instance-based NLG system might just be part of a larger

[1]The implementation language used is Java; experiments are usually run on a Sun Ultra 10.

software system that claims most of the resources.

3.1 Similarity metric

A crucial ingredient in any nearest-neighbour approach is the distance metric. Here, we need to identify a metric that is suitable for the task of natural language generation. As described in chapter 1, our approach is *surface-based* in the sense that we are interested in finding the best sequence of words, including semantic tags instead of the fillers where they occur. For example, instead of '*was elected a vice president*', we want to use '*was elected a* POST_INDEF'. However, for the purpose of ranking, we can treat POST_INDEF just like any other word. The question therefore is how to compute the similarity between such sequences. Viewed from this angle, there does not seem to be anything very specific about the NLG task. We can thus employ a distance-metric for strings that has proved successful on other tasks.

A prominent application that requires similarity computations between strings is information retrieval (IR). There are strong parallels between instance-based approaches and IR as both involve distance computations between pairs of concrete examples. Instances in a nearest-neighbour approach correspond to stored documents in IR; edges (including candidates) in instance-based NLG can be regarded as queries in IR.

A well-known distance metric for information retrieval is the cosine of the angle between vector representations of the strings in question (Salton, 1989). The vector model produces a ranking that reflects the degree of similarity between documents and query. Despite its simplicity, the cosine similarity metric has proved successful in many IR applications:

> "A large variety of alternative ranking methods have been compared to the vector model but the consensus seems to be that, in general, the vector model is either superior or almost as good as the known alternatives." (Baeza-Yates and Ribeiro-Neto, 1999, p.30)

In the following, we look at three aspects of similarity computations involving the cosine metric: the *term representations*, the *weighting scheme* for these terms and the *distance metric* proper.

3.1.1 Term representation

In the so-called *bag-of-words* models of information retrieval, texts are represented as bags of terms ('features' in machine learning terminology), implying that there is no ordering between the terms. These models enable partial matches and ultimately the computation of degrees of similarity between strings regardless of the position of the terms in the original string.

A theoretical disadvantage of bag-of-words models is the underlying assumption that all terms are mutually independent. This is especially true when the bag-of-words indeed consists of individual words. However, we can capture local word cooccurrences by using ngram terms (with $n > 1$) instead of individual words, thereby representing a limited amount of context. A simple example:

(5) `'was elected a POST_INDEF':`

```
unigrams:   {'was', 'elected', 'a', 'POST_INDEF'}

bigrams:    {'was elected', 'elected a', 'a POST_INDEF'}

trigrams:   {'was elected a', 'elected a POST_INDEF'}
```

In addition to ngram representations where n is fixed to a certain value, there is the possibility of mixing ngrams of different orders. Furthermore, in our approach punctuation is treated just like any other word. No stop list is employed so that terms are not excluded per se. As the example above shows, we do not use stemming or reduce words to their part-of-speech. The reason for this is the loss of information resulting from any such transformation. For example, if all prepositions were reduced to the symbol `P`, the ranker would not be able to make any distinctions between different prepositions.

3.1.2 Weighting scheme

In information retrieval, extensive research has been devoted to term weighting schemes for the vector model (see Salton, 1988, for example). Probably the most successful of these is the well-known $tf.idf$ term weighting scheme. The term frequency (tf) component measures the frequency of a term *within* a document. The inverse document frequency (idf) component assigns highest weights to terms that occur in few documents and little weight to terms that occur in many documents. It therefore quantifies the discriminatory power of the terms *across* documents: how much does knowledge about the presence of a term tell us about the document which it is from? In practice, this means that determiners like *the*, for example, have little weight even without using a stop list. In contrast, rare terms, for example those involving a tag that occurs in only one instance, are assigned a high weight.

The $tf.idf$ weight of a term i in document j is calculated as follows:

(6) $$w_{i,j} = f_{i,j} \times log_2 \frac{N}{n_i}$$

The term frequency is represented by the raw frequency f of i in j. The inverse document frequency is calculated as the logarithm of the total number N of documents in the collection (i.e. of instances in the instance base) divided by the number of documents in which term i actually occurs.

For sentence generation, idf is more important than tf which in many cases will be 1. Furthermore, we will introduce an upper bound on the tf-count for edges requiring that the edge tf does not exceed the instance-tf. This will be discussed in section 3.2.1.1.

3.1.3 Distance metric

In instance-based NLG, the ranker compares edge contents to instances. Following the representation and weighting scheme described above, training set instances are generally represented as bags of terms of the form $I_i = \{w_{i,1}, ..., w_{i,n}\}$ where $w_{i,k}$ is the weight of term k in instance i. These representations can be computed off-line. Like queries in information retrieval, edges need to be converted into 'pseudo-document vectors' at runtime, following the same representation. (In the vector model, terms are generally assigned fixed dimensions. Document terms are mapped to the vector model by representing, for each term, the weight of that term in the corresponding dimension. However, this does not imply that there is an ordering in the basic 'bag-of-words' model.) The definition of the cosine distance between an edge *e* and an instance *i* is

(7) $$cos(\overrightarrow{e}, \overrightarrow{i}) = \frac{\sum_{j=1}^{m} w_{ej} \times w_{ij}}{\sqrt{\sum_{j=1}^{m}(w_{ej})^2 \times \sum_{j=1}^{m}(w_{ij})^2}}$$

where w_{ej} is the weight of term j for edge e and w_{ij} the weight of term j for instance i.

Essentially, the cosine formula adds matches between instance and edge in the numerator and divides them by the overall weight of the terms in both instance and edge. There is obviously no explicit count of mismatches. Rather, mismatches are implicitly expressed since mismatching terms only contribute to the denominator but not the numerator, keeping the cosine below the maximal value of 1. We need the denominator since otherwise ever more non-matching words could be added to an edge without penalties. The sum of the squared instance weights (to the right of the denominator in (7)) can be computed off-line like the instance representation since it does not depend on the matches with edge *e*.

The instance-based generation system will need to compute the cosine between different edge vectors with respect to different instance vectors and compare the results. We therefore need to make the assumption that cosine scores of different origin are actually *comparable*. This does not seem to be an unreasonable assumption given that the cosine is often used in document clustering (see Salton and McGill, 1983, for example).

In contrast to IR which mostly deals with larger texts, the ranking task for sentence generation has to consider relatively short sequences of words. However, there does not seem to be any intrinsic reason why the cosine metric should not work on shorter strings. In fact, the cosine is also used in applications such as automatic call routing in which the caller's request – which can just consist of a single sentence – and possible routing destinations are represented as vectors (Chu-Carroll and Carpenter, 1999). Another example is the FAQ Finder system (Burke et al., 1997) which uses the cosine to compute the similarity between a user question and individual faq-files in order to identify those faqs that should be searched further for an answer to the question. One potential drawback of using term-vector comparisons on short sequences of words is the requirement that exactly the same terms have to be matched (i.e. synonyms do not match). However, in addition to surface words, terms in our system also contain the more abstract semantic tags. As a result, classes of proper names like, for example, the names of the incoming person, can be treated as identical. In this sense, the semantic annotation can be seen as a form of application-specific 'clustering' of words that are similar in meaning.

3.1.3.1 Computing the cosine

In what follows, we indicate how we represent vectors for efficient cosine computations. Each term is represented by its weight and a unique integer index, according to which the terms are sorted in a list or array. The integer indices correspond to the different dimensions of the vector model. Imagine we want to compare two such vectors (left columns: indices, right columns: weights):

```
Vector 1:            Vector 2:


1:  0.37             4:  1.94
8:  1.81             8:  1.81
14: 2.12             65: 0.55
65: 0.55             72: 2.55
                     78: 1.94
```

The basic idea is to traverse both lists in parallel by moving pointers from the head of the lists to the end. If we find equal indices, we calculate the match by multiplying their weights. If we find an integer index in one vector but not in the other, we remember its weight if the vector represents an edge. As mentioned above, if the vector stands for an instance, we can calculate the sum of their squared weights off-line. If the vector represents an edge, we only need to actually compute the square root of the 'sum-squared' weights if the numerator is not 0. (Note that in general this has to be determined for each pair of edge vector and instance vector separately.)

The algorithm proceeds by always moving forward the pointer of the lower index. If the next index found is higher than the other vector's index, we obviously 'jumped too far', resulting in a mismatch. In the example above, we start by noting a mismatch between indices 1 and 4 and move the left index forward to 8. Since this is higher than the current right index, we have to move the right index forward as well. Since both indices are now 8, we record a match (we add 1.81^2 to the numerator) and move both pointers forward. This results in a mismatch between indices 14 and 65, requiring the left pointer to be moved forward which results in a match etc. When the end of one list is reached, we know that the remaining terms of the other list must be mismatches. (In the example above, the cosine is 0.3.)

3.1.4 Unknown terms

Concerning the term representations of the ranker in an NLG system, the question arises whether there is a fixed number of terms. Many approaches to instance-based learning assume a fixed number of features (see Daelemans et al., 1999a, for example). In principle, knowing the lexicon means knowing the number of different lexical items that can be generated, at least as long as the 'filler words' of the generation input are represented by tags rather than surface words. However, the use of higher order ngrams would lead to an explosion of the number of possible terms, only a small fraction of which could be generated by the grammar. On the other hand, new word ngrams can arise because the grammar can generate word sequences that have not been seen in the training corpus. The vector representation chosen here lists only the terms that are present but does not require one to state the absence of terms, thereby leaving the overall number of terms open-ended.

The generation of terms that do not occur in the training set also raises the issue of the idf-weighting of unknown terms. A weight of 0 would allow the system to use non-matching terms without penalty. A weight that is too large would stifle 'innovation'. Generally, we assign unknown terms a document frequency of 1, i.e. we pretend that it has occurred in a single instance.

3.2 The search problem

The search problem we are facing has the following characteristics: Our goal is to find the candidate sentence S that has the highest similarity score w.r.t. some instance I in the instance base:

$$\arg\max_{S} \; cosine(S,I). \tag{3.1}$$

The cosine score of an edge is the distance from its nearest neighbour. The candidate sentences that are produced by the grammar form a subset of the set of edges. In principle, we need to iterate over the entire instance base for any given edge. Since the grammar produces a potentially large number of candidate sentences, both S and I can be chosen from large sets to maximize (3.1). Thus, both S and I are variable and we are interested in finding the pair of S and I whose cosine distance is minimal. (Also, we may want to be able to find the n best sentences. This will be discussed in section 3.2.1.5.)

As pointed out above, we cannot just enumerate all possible sentences for some meaning input. On the other hand, we do not have to wait for the grammar interpreter to produce entire sentences. Due to the interleaved architecture, we have access to all new edges being produced, including shorter ones that do not yet constitute a sentence. The question is how to use this possibility of interaction.

Our goal is to use knowledge about the instances and the distance function to restrict the number of similarity computations between edges and instances. Thus, our search algorithm belongs to the class of *heuristic* search strategies, in contrast to *uninformed* search strategies like breadth-first and depth-first that do not possess any problem-specific knowledge. One obvious way of employing the ranker at the 'point of interaction' is to score all new edges E_{new} and use the *highest* score of E_{new} to determine the ordering of edges on the agenda. (As pointed out in section 3.1.3, this requires that the similarity scores of an edge w.r.t different instances are comparable. This is true since we use the same distance metric, instance representations and term weights for all instances.) Using the cosine score on the agenda should give priority to longer edges that closely resemble some instance.

Unfortunately, it turns out that this is not a very efficient way of ordering edges on the agenda (see section 3.3 for efficiency evaluations). What is more, there is a cost involved in scoring every edge. In contrast, uninformed breadth-first or depth-first search strategies only need to rank edges once they have become full sentences. There is a danger that the cost involved in scoring every edge offsets the gains (if there actually are any). On the other hand, we do need to find a way of directing search since uninformed search strategies ultimately amount to exhaustive search if we want to guarantee that the best solution (i.e.. the best candidate sentence) has been found in the end.

3.2.1 Expectation-based search

Thinking about search in an instance-based approach is made difficult by the fact that there are many nearest-neighbours (instances) towards which search from a partial edge can proceed. We want the system to consider all those instances simultaneously. Instance-based ranking seems to encourage a 'parallel view' on the search problem. At the same time, the grammar interpreter is continuously producing new edges that have to be integrated into the nearest-neighbour search. We aim at addressing this point by taking advantage of our knowledge about what edges are combined. Results (of some form) that apply to subedges should be reused when scoring the combined edge.[2]

We can combine these two observations by trying incrementally to restrict the number of instances considered for scoring edges. The idea is to start by using the entire instance base for initial edges and restrict the number of instances as edges are combined. Eventually, we will be able to drop an edge altogether (i.e. remove it from the agenda) when there are no instances left to consider for this edge.

The key problem is to identify a property that allows us to restrict the instance base incrementally. To identify this property, we have to look more closely at the distance function (this knowledge will make the resulting search procedure a heuristic one). We observe that the cosine score of an edge can increase as well as decrease when it is combined with other edges, depending on whether or not the newly added terms match

[2]This point is also taken up in section 3.6 which discusses the applicability of probabilistic chart algorithms.

the instance terms. Thus, the cosine score does not exhibit a monotonic behaviour which could be exploited.

We can, however, define the notion of the *expectation* of an edge w.r.t an instance which is monotonically decreasing. This is the potentially best cosine score an edge would have with respect to a specific instance if all missing matching words were added to it. With regard to information retrieval techniques, the expectation can be seen as a special kind of 'query expansion'. For example, if an edge is represented by the terms a and d and the instance by a,b and c, one match and one mismatch result for the edge. The best possible similarity score for the edge would be achieved by extending the edge by b and c to yield a,b,c,d. (Of course, the cosine similarity metric is more complicated than the simple overlap considered in this example, but matches and mismatches form its basic ingredients.) Actual further combinations of the edge could never improve upon this potentially best score as the added edge can only contribute matches (which have been assumed already) or mismatches (in which case the new expectation is lower than the previous one). It should be noted that the expectation, in contrast to the cosine, is asymmetric since the expectation of a candidate bag is always computed with respect to a reference bag. If we use the notation $exp(bag_{cand}, bag_{ref})$ for the expectation in general, then $exp(bag_1, bag_2) \neq exp(bag_2, bag_1)$.

Assuming that the ranker is able to compute an expectation for newly generated edges, the question arises how to exploit its monotonic behaviour. The key idea is to look at the similarity score of the best candidate sentence that has already been produced, and prune those similarity computations which involve instance-edge combinations that have an expectation that is lower than this best candidate's score. This is justified since the definition of the expectation guarantees that the edge will never be able to become a better candidate than the one obtained already. When edges are combined, we can use information about the instances considered for the subedges to avoid unnecessary similarity computations for the combined edge. Again, this is a result of the monotonicity of the expectation since the expectation of the combined edge can only be equal to or lower than the lower expectation of the subedges.

Effectively, the similarity score of the current best candidate serves as a threshold for the remaining edges. However, it should be noted that an edge can have many expectations (w.r.t. different instances). Edges can only be dropped if there is no expectation w.r.t. any instance which is higher than the current threshold.

In the following, we will describe the properties of the algorithm in more detail. The reader may have noticed already that it shares important characteristics with the well-known A^*-search algorithm. In fact, it can be seen as an extension of A^* to the case of multiple heuristic functions. This will be discussed in section 3.2.2.

3.2.1.1 Monotonicity of expectation

The expectation w.r.t. an instance i is defined as the best cosine score that a given edge e can possibly obtain. From the perspective of the similarity function, generation means incrementally adding to an initially empty edge. We can therefore show that the expectation is monotonically decreasing:

- **Initialization** The expectation of an initially empty edge e is 1 since it contains no non-matching terms: $exp(e_{empty}, i) = 1$.
- **Induction** For each term t that is added to e, two cases can arise:
  1. t does not occur in i. This will decrease the score as well as the expectation since it only increases the denominator of the cosine formula. The score but not the expectation can recover from this if further matching words are added later (case 2).
  2. t occurs in i. If t has not been present in e before, the expectation does not change since it already assumes that all correct words have been added. On the other hand, the cosine score increases.

     If t occurs in e already, we have to consider the case of the tf-component in the edge term weights. (We assume that the same weighting scheme applies to both candidate and instance terms.) If the number of occurrences of t in e is lower or equal than its number in i ($tf_{t,e} \leqslant tf_{t,i}$), the expectation still remains unchanged. However, if $tf_{t,e} > tf_{t,i}$, e is able to neutralize the effect of its non-matching terms by repeating correct terms. Therefore, we impose an upper bound on $tf_{t,e}$ in the numerator (but not the denominator) of the cosine and restrict it to $tf_{t,i}$. As a result, exceeding correct terms in e can only lead to an increase of the denominator and will decrease the cosine score.

A monotonically decreasing expectation requires two assumptions: First, edges are always extended monotonically. Terms are not allowed to be removed. Since monotonic extension of structures is a standard assumption of many grammar formalisms, the algorithm can be used with a variety of such formalisms. Second, it requires an upper bound on the tf-component of edge term weights. An example might be in order. Assuming that we have the instance terms *a*,*b* and an edge *a*,*x*, the impact of the mismatch caused by term *x* could be largely offset by repeating *a*s if we did not restrict the tf-component. For example, an edge *a*,*a*,*a*,*x* would have a much higher similarity score although the four *a*s of the edge match the same instance term.

The upper bound does not seem to be unreasonable if we consider that only the instances in the instance base but not the candidates are justified by human authorship. In this sense, there is indeed an asymmetric relationship between instances and candidate outputs. In practice, we do not find any noticeable difference between using the normal cosine or the bounded one. We explain this partially by the effect of a constraint in the candidate generator which prevents repeated realizations of the input semantics (see section 6.4.1.1).

3.2.1.1.1 Computing the expectation The expectation can be efficiently computed by largely reusing the precomputed sum-squared instance-weights. To see this, we need to look at the actual definition of the expectation for some edge e and instance i which is a modification of the cosine formula (7):

$$(8)\quad exp(\overrightarrow{e}, \overrightarrow{i}) = \frac{\sum_{j=1}^{m}(w_{ij})^2}{\sqrt{(\sum_{j=1}^{m}(w_{ij})^2+\sum_{j=1}^{m}(w_{ej_{extra}})^2)\times\sum_{j=1}^{m}(w_{ij})^2}}$$

where $w_{ej_{extra}}$ is the weight of those terms in edge e that only occur in e but not in i. The expectation simulates a maximally perfect match between instance and edge terms. This is reflected in the numerator: Since the tf-component of the edge term weights is bounded by the corresponding instance-tf, instance and edge weights are the same. The numerator therefore is just the precomputed sum-squared instance weights which are needed in the denominator part of the instance in any case (to the right in the denominator of (8)).

This leaves as the only component of the expectation to be computed at generation time the denominator part of the edge (to the left in the denominator of (8)). Here again the sum-squared instance weights can be reused since the matching terms assumed in the numerator need to be counted in the denominator as well. However, we also need to add all those terms that only occur in the edge but not in the instance. Furthermore, we need to consider the tf-boundary of the matching terms: although the numerator is bounded by the instance-tf, the denominator needs to use the actual frequency of these terms.

These ‘error terms’ or ‘mismatches’, if present, keep the expectation below its maximal value of 1. The computation of the expectation can be combined with the computation of the cosine with little extra cost.

3.2.1.2 Ranking algorithm

We can now modify the ranker to take advantage of the expectation. Again, the idea is that as soon as we have a first candidate and its *score*, we can discard all similarity computations that involve an *expectation* that is lower than this score. The score of already available candidates serves as a threshold th that can be dynamically adjusted to the current best candidate as more candidates are constructed.

For each chart edge e we define the notion of *relevant instances* ri_e. This is a table of all instances i for which the edge has an expectation higher (or equal) than the score of the current best candidate. Each i in ri_e is associated with its current expectation w.r.t. edge e: $ri_e = \{i_1 : exp(e,i_1), ..., i_n : exp(e,i_n)\}$. As long as no candidate is available, we assume a threshold of 0 to allow all instances to be considered. For each e_{new} built by a grammar rule, the ranker performs the following steps:

1. Determine $ri_{e_{new}}$:

    - In principle, for chart edges built from scratch, $ri_{e_{new}}$ contains references to the entire instance base with an expectation of 1: $ri_{e_{new}} = \{i_1 : 1, ..., i_n : 1\}$.

However, this is an idealized case. An expectation of 1 on all instances implies that the edge is empty – it does not yet contain any non-matching terms to any instance. Since many input grammar rules (see section 5.3.1) introduce new terms, the expectation will be lower than 1 in many cases.

- For chart edges built by combining existing edges, say e_1 and e_2, a cheap initial approximation of $ri_{e_{new}}$ is obtained by intersecting ri_{e_1} and ri_{e_2} and taking the *lower* expectation of the remaining i. This is justified because the resulting edge will contain terms of both e_1 and e_2 and because of the downward monotonicity of their respective expectations.

  Thus: $ri_{e_{new}} = ri_{e_1} \cap ri_{e_2}$ where for each $i \in ri_{e_{new}}$: $exp(e_{new}, i) = exp(e_1, i)$ if $exp(e_1, i) \leq exp(e_2, i)$ and $exp(e_{new}, i) = exp(e_2, i)$ otherwise.

2. Check the (preliminary) expectations in $ri_{e_{new}}$ against the current threshold th. For all $i \in ri_{e_{new}}$: if $exp(e_{new}, i) > th$ keep i in $ri_{e_{new}}$, else remove i. (We can use $exp(e_{new}, i) \geq th$ to find all best solutions.)

   Removing instances from $ri_{e_{new}}$ is possible because of the monotonicity of the expectation: the expectation will never increase beyond the threshold, and it forms an (optimistic) upper bound on the actual similarity score.

   This step incrementally constrains the number of relevant instances that need to be considered for the computation of the actual expectation in the next step.

3. Compute the actual $exp(e_{new}, i)$ for all $i \in ri_{e_{new}}$.

   We use the highest expectation of $ri_{e_{new}}$ to determine the rank of e_{new} on the agenda. Thus, the expectation serves two roles in this algorithm: it helps reducing the number of similarity computations and it determines the agenda ordering. (The expectation can always be used for ordering the agenda, regardless of whether or not a candidate has already been found.)

   In our grammar formalism, most grammar rules introduce new surface words. Therefore, we need to actually compute a new expectation from scratch rather than trying to reuse the expectations of the subedges in some form.

It should be noted that we do not check the new, accurate expectation against the threshold at this point. Rather, the check is delayed until the edge is added to the chart. This allows the ranker to always use the most recent threshold since the threshold is dynamically increasing (see section 3.2.1.3 below).

4. Add e_{new} with its updated $ri_{e_{new}}$ to agenda unless $\mid ri \mid = 0$ in which case we can drop e_{new}.

   This is a very desirable case as it leads to less edge combinations in the chart.

5. If e_{new} also qualifies as a candidate, compute $cos(e_{new}, i)$ for all $i \in ri_{e_{new}}$ and add it to the current list of candidates. If $cos_{max}(e_{new}, i) > th$ adjust threshold: $th = cos_{max}(e_{new}, i)$.

   The actual cosine score only needs to be computed when the edge is considered a candidate. This can be any edge of category S or only edges that satisfy an arbitrary number of constraints. (We generally treat all generated sentences as candidates. Section 7.4 (especially 7.4.3.1) discusses the use of additional constraints on candidates.) In practice, there is no big difference between calculating the cosine similarity and calculating the expectation since both involve cycling over the term representations of instance and edge. They can therefore be computed together.

This ranking algorithm incrementally constrains the set of relevant instances *ri* of the chart edges. It is called for each new edge produced by the grammar. The chart algorithm (see section 6.4) stops when the agenda is empty. However, the ranking algorithm can influence the agenda by dropping edges deemed not promising (step 4). It can deliver an output as soon as the first candidate is available, and further computation can be used to find potentially better ones. Furthermore, it has the advantage of being independent of the order in which chart edges are combined.

3.2.1.2.1 Admissibility of search The algorithm finds a single (or all) globally best solutions (if any exist). It can therefore be called an *admissible* algorithm. To see why this is consider as a starting point exhaustive search which obviously finds the optimal solution. The algorithm presented above restricts search by pruning those edges that will never be able to build candidates that are better than the best candidate already available. (We know that the score of the best candidate already obtained can only improve or stay the same but never decrease.) Pruning edges is only possible if none of the various expectations of the edge is larger than the score of the best candidate and the ranking algorithm does not stop before the remaining expectations have been explored.

The crucial point therefore is the monotonicity of the heuristic function that estimates the best possible similarity score. We have seen in section 3.2.1.1 that the expectation never underestimates the potentially best score. As long as this is guaranteed, the algorithm is not able to prune any edge that might eventually obtain a better score than the available best candidate.

The performance of the expectation-based algorithm depends on how quickly a good candidate can be constructed since this sets the threshold for the remaining edges. If no candidate is available at all, no pruning can take place and the algorithm reverts to exhaustive search. However, these seem to be rather rare cases in practice (see section 3.3).

3.2.1.3 Blocking of agenda edges

In addition to the steps carried out for new edges about to be added to the agenda (see 3.2.1.2), the ranker performs the following check for all agenda edges about to be added to the chart:

1. If $exp_{max}(e_{agenda}) > th$, add agenda edge to chart. Else drop edge. (Again, we can use a non-strict threshold to find all best candidates: $exp_{max}(e_{agenda}) \geq th$.)

This check is performed because there is a possibility that the threshold has been increased while the edge was waiting on the agenda. It is a means for improving efficiency but does not alter the admissibility of the algorithm: since the expectation is non-increasing, it can be checked against the non-decreasing threshold at any point.

As a result of the additional step described here, we check the newly computed expectation of edges about to be added to the agenda twice: the first check in the 'life cycle' of an expectation is in fact the one now described. It can be carried out cheaply since it only compares the single best expectation of the edge against the threshold. The second check occurs when two edges are combined into a new one. It is more detailed since it looks at the different expectations in the relevant instance table (see step 2 in the main ranking algorithm in section 3.2.1.2).

There are parallels – and differences – between checking the expectation of edges against the threshold and redundancy checking in chart parsing. If redundancy checking at the point of a new edge being added to the agenda takes into account chart *and* agenda items, it is not required when edges are added from the agenda to the chart (Shieber et al., 1995). If the agenda were allowed to contain redundant items, these could lead to the computation of spurious new edges (for agenda items added to the chart) which are deemed redundant when about to be added to the agenda.

In contrast, in expectation-based search, the definition of what is regarded a 'useless' edge can change while edges are stored on the agenda because the threshold may increase. If agenda edges are not checked against the current threshold before being added to the chart, they can lead to spurious computations of new edges that are deemed unpromising when about to be added to the agenda.

3.2.1.4 Example of expectation-based search

Figure 3.1 shows an example of the concatenation of two edges $e1$ and $e2$. For both edges and the combined edge $e12$, cosine scores and expectation with respect to two instances are given. For example, the cosine w.r.t. instance $i1$ increases from 0.35 to 0.75 as $e2$ is combined with $e1$ into $e12$. This is because $e1$ contributes two matching terms. On the other hand, the cosine decreases from 0.5 to 0.35 w.r.t. instance $i2$ from $e2$ to $e12$ because $e1$ adds two non-matching terms. These changes of the cosine similarity value show that it is not monotonically changing in one direction.

In contrast, the expectation is monotonically decreasing (w.r.t. the same instances). For example, it decreases from 1.0 for $e1$ w.r.t. $i1$ to 0.89 for $e12$ since $e2$ adds a non-matching term. On the other hand, viewed from $e2$, which has an expectation of 0.89 with regard to $i1$, the expectation stays the same when moving to $e12$. This is because the expectation has assumed the presence of the matching terms of $e1$ already.

Strictly speaking, the ranking algorithm does not need to compute cosines for the

```
Assumptions:
============

all term weights are 2
current best candidate cos: 0.75

Instance i1: { a, b, c, d }
Instance i2: { c, z }


Edge combinations:
==================

 edge e1: { a, b }   edge e2: { c, y }

  i1:                 i1:
    cos: 0.7            cos: 0.35
    exp: 1.0            exp: 0.89

  i2:                 i2:
    cos: 0.0            cos: 0.5
    exp: 0.7            exp: 0.82

  ri: { i1:1.0 }      ri: { i1:0.89,
                            i2:0.82 }

        \                     /
         \                   /
          \                 /
           \               /
            \             /
             \           /

         edge e12: { a, b, c, y }

           i1:
              cos: 0.75
              exp: 0.89

           i2:
              cos: 0.35
              exp: 0.63

           ri: { i1:0.89 }
```

Figure 3.1: Example of expectation-based search

edges unless they qualify as candidates. Under the assumption of a current best candidate cosine of 0.75, we can furthermore identify the relevant instance table *ri* for the edge $e12$. This table indicates which expectations actually need to be computed by the ranker. Knowing that $e1$ has an expectation of only 0.7 w.r.t. *i2* we can immediately conclude that any combination of this edge will never produce a better candidate than the one obtained already. We can therefore exclude *i2* from the relevant instance table of the combined edge. (The actual expectation of 0.63 of $e12$ confirms this.) Thus, we only need to compute the expectation of $e12$ w.r.t. $i1$. Although the resulting expectation of 0.89 remains unchanged compared to the one for *e2*, it should be noted that the expectation of a combined edge can be lower than any one of the component expectations, as $exp(e12, i2)$=0.63 shows. Furthermore, in practice most grammar rules introduce additional words, making further mismatches that might lower the resulting expectation possible.

3.2.1.5 n-best search

The definition of the threshold in the ranking algorithm can be made non-strict in the sense that similarity computations are also not pruned if their expectation equals the similarity score of the current best candidate. This ensures that all best candidates are found if there are more than one. However, it does not ensure that the algorithm finds a fixed number n of best candidates.

In order to find the n best candidates, the search algorithm can be modified so that it keeps the threshold as low as the score of the nth candidate in the ranked list, or at 0 as long as there are less than n candidates. This ensures that we never prune edges that may end up above the current nth candidate in the ranked candidate list. Like the normal threshold defined by the top-most candidate on the list, the n-best threshold increases dynamically as more candidates are generated.

3.2.2 Comparison to A^*-search

Like the instance-based search algorithm presented above, the well-known A^*-search algorithm is a best-first search algorithm which always considers the estimated best partial solution next. In this section, we discuss the differences and similarities between the algorithms.

A^*-search is often employed for problems like finding the shortest path to some goal state in a graph. A variation known as 'stack decoding' is also used in speech recognition (Huang et al., 2001). Discussions of A^* can be found in many AI textbooks (Nilsson, 1980; Rich and Knight, 1991; Norvig, 1995) and studies of search algorithms (Pearl, 1984). Here, we will concentrate on the basic path finding problem.

A^* estimates the overall cost f from a starting state to a goal state via a current state by adding the known cost from the start state to the current state (g) to an estimate of the remaining distance (h):

(9) $f = g + h$

As in our algorithm, the heuristic function h which estimates the remaining path to the goal needs to be optimistic. When dealing with costs like these in path finding problems, the algorithm needs to make sure never to overestimate the minimal cost to the goal which would lead to ignoring potentially good paths. This situation is the opposite of our problem where we are trying to maximize the similarity score rather than minimizing a cost (we need to ensure that the potentially highest similarity score has not been underestimated). However, maximizing the similarity is just the inverse of minimizing the distance between candidates and instances. The problem can be formulated in different ways; the underlying idea is exactly the same.

However, there is a difference in the way the overall estimate is calculated. Our algorithm does not make a distinction between the path from the start to the current state, i.e. from the empty edge to the current edge, and the estimated 'cost' of reaching the goal from the current edge. In other words, the expectation corresponds to f rather than h. In principle, the actual similarity score of an edge can be equated to function g, and correspondingly $h = expectation - cosine$. However, such a distinction is not necessary in our algorithm since the overall expectation can be straightforwardly computed from the bag-of-words representations (see section 3.2.1.1.1).

Furthermore and probably crucially, in contrast to the standard A^*-algorithm, the instance-based ranking algorithm needs to handle several scores and expectations which are derived with respect to different instances. This can be regarded as there being several heuristic functions rather than one. As described in section 3.1.3, a requirement for the instance-based search algorithm to work is the comparability of similarity scores and expectations from different instances. By always considering only the best similarity score and expectation regardless of the instance with respect to which it has been computed, we basically treat the different heuristic functions as one.

A path finding problem in the simplest case has only a single known goal state and the A^*-algorithm stops as soon as the goal state has been reached. In contrast, in the NLG system developed here, the search algorithm can suddenly 'stumble' across a solution from which it cannot proceed (i.e. any edge that qualifies as a candidate). Obviously, there are an a priori unknown number of 'goal states' generated by the grammar. However, in the proof of optimality of A^* in (Norvig, 1995, p.99/100) the case of several goal states is considered. It is assumed that non-optimal solutions are not returned as final solutions since these would not be selected for further best-first expansion. This consideration indicates that A^* works in a similar way to our search algorithm.

Both A^* and the instance-based ranking algorithm depend on the monotonicity of the estimation function and are guaranteed to find an optimal solution if one exists. Because of the similarities between the algorithms, our instance-based ranking algorithm can be seen as an extension of the standard A^*-algorithm to the case of several estimation functions. Devising a suitable heuristic function is crucial to a successful application of A^*-search in any variant. If h underestimates the potentially best score, the algorithm is not admissible; if it is too optimistic, search becomes inefficient. In this work, we defined a heuristic function that takes into account the peculiarities of the distance function and the problem domain.

3.2.2.1 Related literature

Woods (1982) introduces an "A^* shortfall method" for speech understanding that is similar in spirit to our expectation-driven ranking algorithm in that search is guided by optimistic assumptions about the best possible score of parts of the input not yet covered by a partial solution. The speech understanding task involves parsing competing word sequences generated by a lexical component and finding the optimal word sequence that is consistent with the grammar.

Woods (1982) requires the allocation of maximal scores to parts of the input before the beginning of processing. The scores of these segments are additive so that the speech recognition problem can be treated as an A^* path finding problem with a known goal state, separate scores for segments already covered and possible scores for the remaining segments. The shortfall is defined as the difference between the actual score of a partial solution and the best possible score of that partial solution. The priority p of a partial solution on the agenda is

(10) $p = T - (m - q)$

where T is the maximal total score allocated to all segments, m the best potential score for the partial solution and q its actual score (i.e. function g in (9)). The expression $m - q$ therefore represents the shortfall. The resulting p is characterized by Woods (1982, p298) as the "estimated best possible future score" and corresponds to our notion of an expectation (i.e. function f in (9)).

There are still several differences between the shortfall scoring method and our algorithm. As the discussion of A^* in section 3.2.2 has made clear, we do not separate current and future scores. It would not be possible to assign maximal scores to parts of the input (the semantic tags in our case) and assume that the overall score is simply composed of the component scores. Furthermore, the approach of Woods assumes that a lexical component enumerates words in decreasing order of score and that the grammar is merely used as a constraint on the possible word sequences. Moreover, it does not deal with the problem of having several heuristic functions. Finally, the speech understanding problem addressed by (Woods, 1982) assumes a single well-defined goal state which is defined as complete coverage of the input signal. In contrast, we do not make the assumption that the grammar is always able to express the entire input semantics (this will be discussed in section 7.4).

3.3 Efficiency evaluation

In order to investigate the general behaviour of the proposed search algorithm and evaluate its efficiency, we tested our system with different settings using the semantic tags of a test set as generation inputs. To this end, we divided the annotated corpus of *Wall Street Journal* articles into a random selection of 100 first sentences for the training set and 40 for the test set. This should give a realistic indication of the performance of the system since the evaluation is carried out on application data. As mentioned in section 1.2.4, we employ an agenda-based bottom-up chart generator that makes use of a Rete network. The chart generator is described in greater detail in chapter 6. For the purpose of the efficiency evaluation, it can be assumed that the generator behaves like a standard bottom-up chart generator (see also 2.1.5). Furthermore, the exact nature of the input tags and the output candidates is not relevant for the efficiency evaluation.

In the following, we will set out by evaluating the basic system and comparing it with a number of variations and competitors. We then add additional refinements that come closer to the system that we are going to use in the following chapters. Still, the final system of this thesis will contain extensions that will be introduced when the output of the generator is investigated. In particular, we will make it harder for the generator to find a 'good solution'. This in turn will influence search which will then be re-evaluated (see chapter 7). However, the basic behaviour of the proposed hybrid generation system is best investigated using relatively simple initial system settings.

3.3.1 Examining individual generator runs

We first evaluate the algorithm with respect to an individual input in order to give a detailed account of its behaviour. We then evaluate the system on the entire test set to avoid the particularities of individual inputs. For the moment, we assume bigram terms and a $tf.idf$ weighting scheme. Furthermore, the generator runs until the agenda is empty. Also, only instances with expectations that are strictly larger than the threshold are considered. (An alternative is to also keep instances that have an expectation equally large as the current best candidate score; see section 3.2.1.5 above.)

3.3.1.1 Expectation-based agenda ordering

For a random semantic input consisting of 6 tags (close to the average number in the corpus), the expectation-based system produces 92 edges. It takes about 1.5/1.8 seconds to exhaust the agenda (on a Sun Ultra 10/Sun Enterprise E220R). Figure 3.2 shows the rank of newly produced edges on the agenda in relation to the size of the agenda. Generation proceeds from left to right on the x-axis.

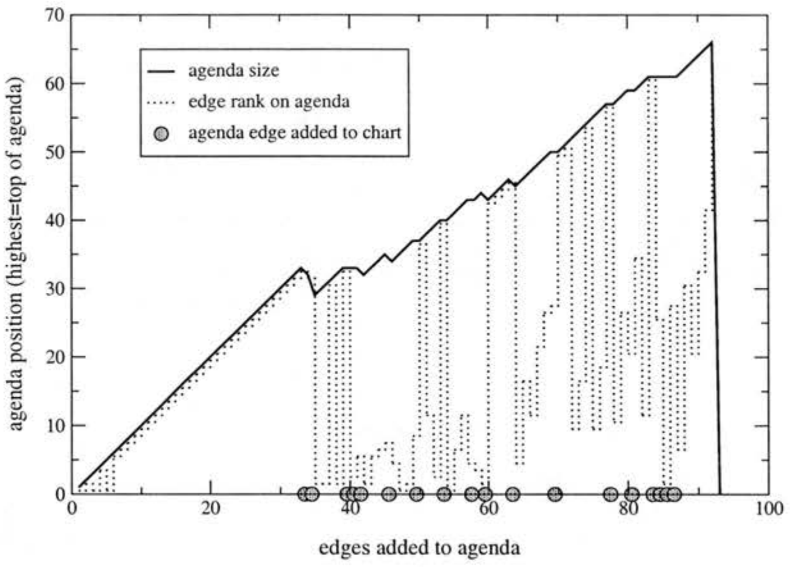

Figure 3.2: Edge ranks on expectation-based agenda (input: wsj_0030)

At the beginning of processing, newly produced edges are placed right on top of the agenda. These edges have been derived from initial edges presenting the input to the system. They can be found to the left of the first point at which an edge is taken from the agenda. (Several assertions can take place at one point if no new edge can be produced.) The first new edges all have expectations of 1.0. They are always moved to the top of the agenda ahead of edges that have the same expectation.

From about edge 35, the ordering is more mixed. We explain this by the fact that longer edges containing more error terms are produced. However, edge 92 is a candidate with a cosine of 1.0. From this point, the ranker blocks all edges that are taken from the top of the agenda and that would usually be added to the chart to produce more edges. This is because the edge expectation is checked against the current threshold when agenda items are added to the chart (see section 3.2.1.3). Blocking edges before they are added to the chart is quite obviously more efficient than computing new edge combinations and blocking these later. (In fact, leaving out the agenda-to-chart threshold check results in 301 edges and takes about twice the time.) As a result of blocking all remaining agenda edges, the agenda size shrinks without any edges being produced. (Note that figure 3.2 only provides a snapshot of the agenda size at the time a new edge is created – with the exception of the final size.)

3.3.1.2 Cosine-based agenda ordering

The expectation serves a two-fold purpose in the ranking algorithm. It determines the agenda priority and it helps pruning the number of instances used for similarity computations. However, we can also employ the cosine to determine the agenda ordering while continuing to prune similarity computations by means of the expectation. This variation of the algorithm generates 355 edges and takes approximately 3.5/7.0 seconds to run. At least for this specific input, this is less efficient than an expectation-based ordering.

Figure 3.3 shows agenda size and rank of new edges for the cosine-based agenda. The agenda seems to shrink and expand periodically. Many new edges end up close to the top of the agenda so that search seems to be predominately depth-first. However, the ranker does not find a candidate until edge 199 and a candidate with a cosine of 1.0 is not found until edge 250 (in contrast to edge 92 for the expectation-based agenda). This indicates that the cosine is less efficient as an agenda ordering. The grammar still generates new edges beyond edge 250 (because the chart generator needs to explore all combinations of the remaining agenda edges). However, these are all blocked before they are added to the agenda (see the ranking algorithm in section 3.2.1.2). Only when the grammar has stopped does the agenda try to add its edges to the chart. Again, these are all blocked.

3.3.1.3 Comparing agenda priorities

Figures 3.2 and 3.3 indicate that both expectation and cosine-based agenda ordering mix aspects of depth-first and breadth-first search. Since the priority policies are differ-

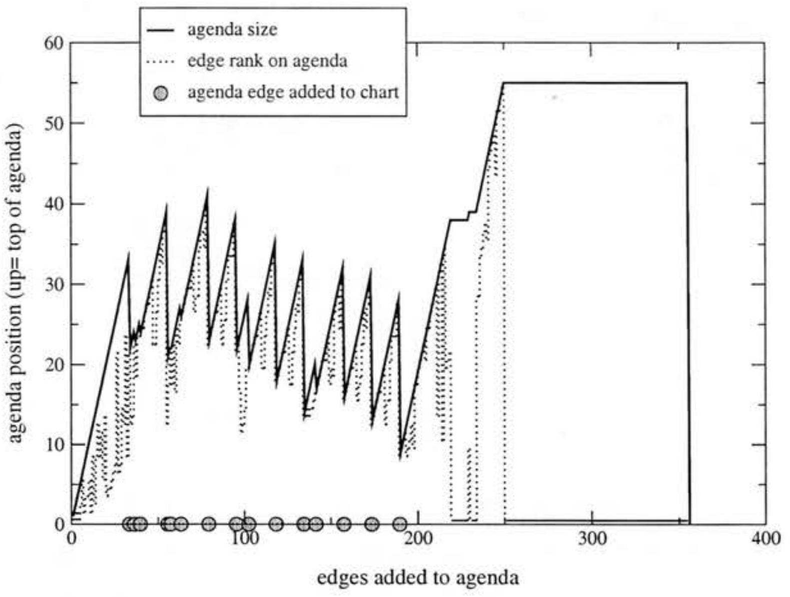

Figure 3.3: Edge ranks on cosine-based agenda (input: wsj_0030)

ent, the results are not directly comparable: a new edge that is assigned rank 1 on the agenda means that the system has been successful in generating an edge continuation that has a high priority *according to its priority criterion*. The question arises how we can measure the ranking of edges in a more objective way in order to compare different priority policies.

One way of doing this is to measure how 'old' edges taken from the agenda actually are. To this end, each edge is assigned an integer for the current cycle of the Rete interpreter (starting from 0) in which it was created. A new cycle is started each time an edge is added from the agenda to the chart. We can then compare the cycle of the point when the edge is taken from the agenda and relate it to the cycle of the edge creation. A small difference means that an edge stayed only briefly on the agenda, resulting in depth-first search. On the other hand, breadth-first search should be reflected by long stays on the agenda. A natural upper bound on the number of cycles an edge can possibly wait is formed by the overall number of cycles at any given point.

Figure 3.4 shows the results for the same generation input used before. The upper bound is a diagonal starting at the origin of the axis. In contrast to figures 3.2 and 3.3 which show all edges that are produced by the grammar, we can only sensibly

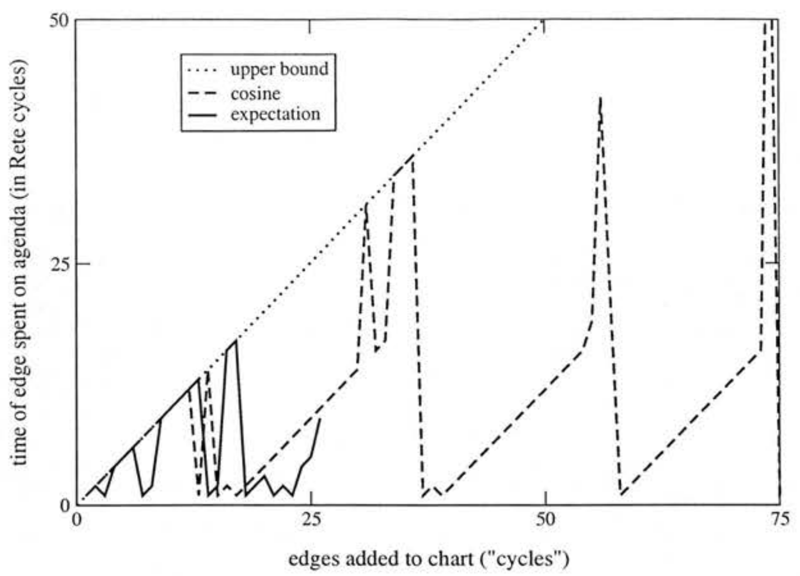

Figure 3.4: Time in Rete cycles of edges waiting on agenda (input: wsj_0030)

measure the waiting time for those edges that are also added from the agenda to the chart. Therefore, expectation and cosine seem to have less edges than in figure 3.2 and 3.3.

Figure 3.4 confirms the previous finding that both expectation and cosine-based priorities mix depth-first and breadth-first search. Cosine-based search continues for considerably more cycles. There seems to be a pattern that edges that waited only briefly on the agenda are followed by increasingly 'older' edges. As long as these are located on a diagonal, they must have been produced at the same cycle, getting older as their predecessors are added to the chart (which increments the cycle counter).

Although agenda policies are easier to compare by counting agenda waiting cycles, it should be noted that cosine and expectation-based system still rank different edges over time. Relating the two ways of measuring the agenda ordering we explored here is not entirely straightforward. For example, all edges produced to the left of the first agenda-to-chart assertion point in figure 3.2 belong to cycle 0. In figure 3.4, these additions to the agenda are not visible because only edges taken *from* the agenda are counted. Another example are edges of expectation-based search that are located at the upper bound in figure 3.4 (cycles 4-6, 9-13 and 16,17). They obviously have had to wait a maximal time on the agenda. However, they might have been ranked on top of the agenda initially but newly produced edges could have been placed ahead of them.

3.3.1.4 Edge length and priority scores

Short edges should generally have high expectations and should therefore be highly ranked on the agenda (which considers the highest of the different expectations). Conversely, long edges will in many cases have only low expectations although a single exception would be enough to give it a high agenda rank. Above, we suggested that a cosine-based priority might lead the system into buying more mismatches since the overall cosine score of long edges containing mismatches can still be higher than the score of a short edge without mismatches.

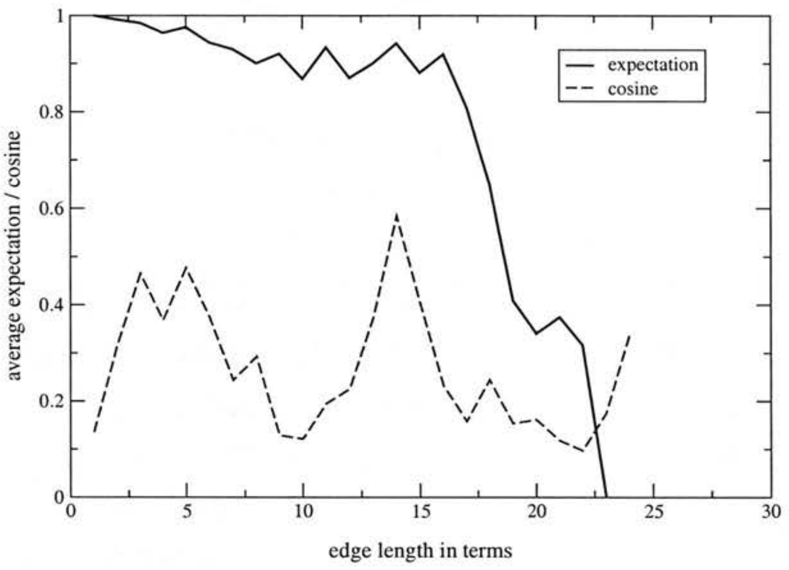

Figure 3.5: Average priority values (input: wsj_0030)

Figure 3.5 relates the average agenda priority values (defined by the expectation or the cosine) of the produced edges to the length of the edges.[3] In the context of ranking, the length of an edge is measured by the magnitude of its pseudo-document vector, i.e. by the number of terms used for presenting the edge to the ranker. Figure 3.5 confirms

[3]To obtain more data, we used a slightly different system setting requiring full consumption of the input in order to force both variations of the system to produce more edges (see section 7.4).

the intuition that the expectation tends to decrease on average as edges become longer. In contrast, the cosine has its peak value at an edge length of 14 terms.

3.3.2 Results for test set

To obtain a better picture of the performance of the expectation-driven algorithm, we tested it on 40 test set inputs. Furthermore, we added two competitors, pure depth-first and breadth-first search. All systems can be simulated by changing the agenda policy. Pure depth-first and breadth-first search have the advantage of being able to determine the agenda ordering cheaply without any similarity computation. The cosine only needs to be computed for edges that have been grown into candidates. However, both approaches will lead to exhaustive search without any pruning of the search space if there is no pre-defined stopping criterion (exhaustive search can take 30 minutes or more). Therefore, we generally assume that the system halts when a first candidate that has a cosine of 1.0 is found. This effectively gives all rankers the information that no better candidate can be produced. However, in case no candidate of 1.0 is found, pure depth-first and breadth-first perform exhaustive search. Therefore, we need to introduce a limit on the total generation time which we set at 1 minute.

agenda ordering	$\overline{E}$	σE	$\sum E$	$\sum C$	$\overline{sim}$	$\sum sim$	a_{empty}	1.0	*t-out*	$\bar{t}$
expectation	316	1305	12 637	4 272	29.0	353 201	17.5%	82.5%	0%	2.3
cosine	1268	2034	50 734	18 565	32.6	1 571 326	15%	72.5%	12.5%	13.3
breadth-first	957	2027	38 277	13 016	34.0	1 301 600	15%	80.0%	5.0%	4.9
depth-first	471	1133	18 826	6 666	35.4	666 600	15%	77.5%	7.5%	6.5

Table 3.1: Efficiency results for test set of 40 inputs: bigram terms, $tf.idf$ weights

Table 3.1 shows the results for the test inputs. For different agenda orderings, we detail the average number of edges ($\overline{E}$), the standard deviation of the edge counts (σE) and the overall number of edges for all 40 inputs ($\sum E$); the overall number of candidates ($\sum C$); the average number of similarity computations, i.e. cosine or expectation, per edge ($\overline{sim}$) and the overall number of similarity computations ($\sum sim$); the reasons for halting the system (a_{empty}= agenda empty, 1.0= candidate of 1.0 found, *t-out*= time limit exceeded); finally, the average time the system requires per input in seconds ($\bar{t}$).

The results show that the expectation-based ranker is the fastest and performs least overall and average similarity computations. The average number of similarity computations per edge (29) is significantly lower than the size of the instance base (100).[4] The high standard deviation in the number of edges reflects the fact that the performance on different inputs varies significantly. This is true for all tested algorithms. One major factor is runs that exceed the time limit (which does not affect the expectation-based ranker).

The cosine-based agenda seems to fare particularly badly, producing the largest number of edges and candidates and requiring the most similarity computations. This is a somewhat surprising result since the cosine does not seem to be an unreasonable candidate for determining the agenda ordering as it gives preference to the current 'best' edge. However, it seems that this leads to prioritizing edges that also contain mismatches. In contrast, the expectation basically avoids mismatches since these are the only way to decrease it.

Table 3.1 also shows that depth-first and breadth-first search perform better than the cosine-based agenda. The average number of similarity computations per edge is fairly low (although still larger than for expectation-based search) because depth-first and breadth-first search only have to compute the cosine for candidates but not for other edges. Since computing the cosine for candidates needs to use the entire instance base, the overall number similarity computations is equal to the size of the instance base (100) times the number of candidates.

The time measurements show that depth-first search, despite producing less edges and candidates than breadth-first search, takes longer on average. This can be explained by the fact that depth-first search runs the danger of exploring the search space in areas where it cannot even successfully combine edges into new ones. We can measure the number of attempted edge combinations by counting the number of checks for non-overlapping semantics (see section 6.4.1.1). For the entire test set, depth-first search performs 980 010 checks whereas breadth-first search performs only 135 327 checks.

[4]This average number includes the empty instance tables of the dropped edges. Furthermore, some edges that were produced by the grammar (and which are therefore included in the overall edge count $\sum E$) are blocked by the redundancy check (see section 6.4.1.2). This is why the average number of similarity computations $\overline{sim}$ is not exactly the same as $\sum sim$ divided by $\sum E$.

However, about twice as many checks are successful for breadth-first search than for depth-first search. The larger number of successful checks for breadth-first search translates into its larger number of edges.

In summary, the results in table 3.1 show that instance-based NLG is feasible in reasonable time, in particular if an expectation-based search algorithm is used.

3.3.2.1 Edge length and similarity computations

Similarity computations are more expensive for longer edges since the size of the pseudo-document vector is larger. The expectation-based ranker attempts to incrementally reduce the number of instances used for similarity computations as edges are combined into larger ones. This is possible because it has access to the chart edges in the interleaved generation architecture. We therefore expect the ranker to perform more similarity computations on shorter edges than on longer ones.

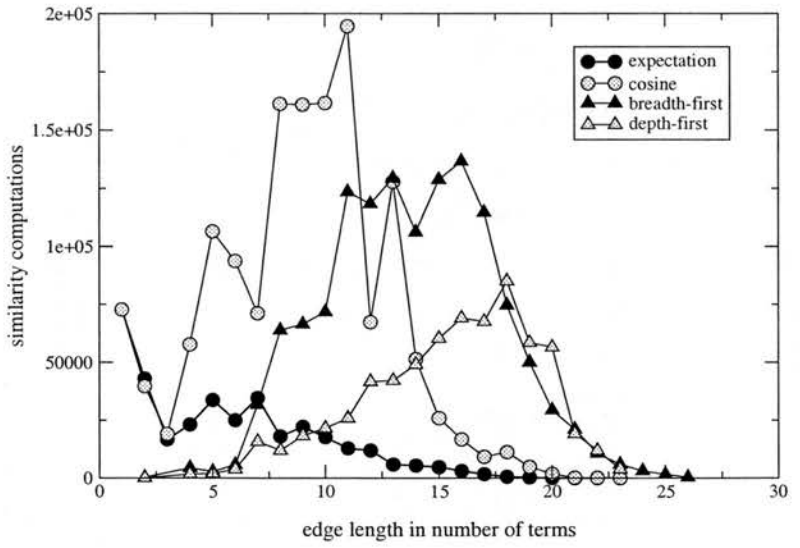

Figure 3.6: Overall number of similarity computations (bigrams, $tf.idf$ weights)

Figure 3.6 shows the number of overall similarity computations for the 40 test inputs w.r.t. edge length as measured by the number of terms. In general, there seems to be a tendency for the overall number of similarity computations to increase over time because more longer edges are generated by combining shorter ones. On the other hand, the maximum edge length is constrained by the size of the semantic input.

Figure 3.6 confirms that expectation-driven search manages to reduce the number of similarity computations for longer edges. Breadth-first and depth-first search naturally score larger edges because they only rank sentences. As before, cosine-driven search fares worst which is due to the large overall number of edges it generates.

3.3.3 Alternative term representations and weights

There are a number of variations of the basic ranker. Instead of bigrams, one can use ngrams of different orders or mixtures of ngrams. Furthermore, a large number of term weighting schemes have been developed in information retrieval that could be applied to ranking for NLG. Here, we will investigate binary term weighting[5] and terms represented as ngrams with $n \leq 3$ including mixtures. To the ranker, only the term weights matter (and their 'identity', obviously), not the fact that the terms are unigrams or bigrams per se. Furthermore, the size of the pseudo-document vectors will have an effect on efficiency.

In general, we can expect a stronger influence of the term weights on cosine and expectation-based agenda priorities than on breath-first and depth-first search. The latter should only be affected by an increase in the absolute number of terms which are used to compute the cosine score of the final candidates. In contrast, the weighting scheme affects the ordering of edges on the agenda for cosine and expectation-based priorities.

Table 3.2 shows efficiency results for $tf.idf$ term weights along the same dimensions as used in table 3.1 (which is included in table 3.2). Confirming the considerations above, there is little difference between unigram and bigram terms for breadth-first and depth-first search. However, there is a slowdown when moving to a combination of unigram, bigram and trigram terms since for n unigram terms, there are $n+(n-1)+(n-2)$ overall terms.

Both cosine and expectation-based search are slower when using unigram terms than when using bigram terms. An explanation may be found in the term weights

[5]When using binary weights in the vector representation, a 1 in a vector component indicates the presence of a term. As for $tf.idf$ weighting, the absence of a term in the vector implies a weight of 0 in the corresponding dimension.

agenda	$\overline{E}$	σE	$\sum E$	$\sum C$	$\overline{sim}$	$\sum sim$	a_{empty}	1.0	*t-out*	$\bar{t}$
unigrams:										
expectation	506	1827	20 227	3018	26.8	514 556	15%	80.0%	5.0%	4.8
cosine	1188	1760	47 538	12690	36.9	1 717 151	15%	75.0%	10.0%	15.4
breadth-first	1003	2220	40 101	13962	34.8	1 396 200	15%	80.0%	5.0%	4.8
depth-first	487	1214	19 480	7187	36.9	718 700	15%	77.5%	7.5%	6.4
bigrams:										
expectation	316	1305	12 637	4272	29.0	353 201	17.5%	82.5%	0%	2.3
cosine	1268	2034	50 734	18565	32.6	1 571 326	15%	72.5%	12.5%	13.3
breadth-first	957	2027	38 277	13016	34.0	1 301 600	15%	80.0%	5.0%	4.9
depth-first	471	1133	18 826	6666	35.4	666 600	15%	77.5%	7.5%	6.5
trigram:										
expectation	297	1144	11 873	4872	63.1	717 127	15%	82.5%	2.5%	2.5
cosine	475	806	18 993	5769	80.0	1 464 252	15%	80.0%	5.0%	8.4
unigrams + bigrams:										
expectation	332	1406	13 275	4390	27.7	353 946	15%	82.5%	2.5%	2.5
cosine	1100	1615	43 996	12923	52.9	2 270 987	15%	75.0%	10.0%	18.8
unigrams + bigrams + trigrams:										
expectation	213	680	8 517	2718	63.2	512 489	15%	82.5%	2.5%	3.0
cosine	878	934	35 134	9896	71.2	2 456 884	15%	67.5%	17.5%	21.7
breadth-first	680	913	27 181	7034	25.9	703 400	15%	80.0%	5.0%	8.3
depth-first	332	562	13 278	4191	31.6	419 100	15%	77.5%	7.5%	8.3

Table 3.2: Efficiency results for test set: **tf.idf weights**

that result from the *idf* component of the *tf.idf* weighting scheme. Since unigram terms tend to occur more frequently in different instances, they tend to result in less discrimination between edges. Furthermore, as larger mixtures of ngrams are used, cosine-based search becomes more inefficient. Expectation-based search seems to be less affected by this.

agenda	$\overline{E}$	σE	$\sum E$	$\sum C$	$\overline{sim}$	$\sum sim$	a_{empty}	1.0	*t-out*	$\bar{t}$
unigrams:										
expectation	379	1117	15 153	5516	48.2	702 699	12.5%	82.5%	5.0%	5.6
cosine	496	1472	19 822	8667	27.9	541 374	12.5%	82.5%	5.0%	4.3
bigrams:										
expectation	1086	2655	43 426	15499	21.8	912 353	15%	75.0%	10%	7.5
cosine	1603	2326	64 082	23075	30.4	1 877 549	45%	35.0%	20%	18.2
trigrams:										
expectation	312	829	12 460	4253	86.1	1 030 521	15%	82.5%	2.5%	3.2
cosine	493	826	19 707	6124	73.1	1 395 911	15%	80.0%	5.0%	9.1
unigrams + bigrams:										
expectation	325	1194	12 997	5013	34.4	429 903	15%	82.5%	2.5%	2.8
cosine	904	1335	36 142	12031	39.5	1 406 524	15%	77.5%	7.5%	12.2
unigrams + bigrams + trigrams:										
expectation	236	660	9 450	3249	59.9	541 171	15%	82.5%	2.5%	3.4
cosine	793	1027	31 714	12052	50.9	1 587 438	15%	75.0%	10.0%	17.3

Table 3.3: Efficiency results for test set: **binary weights**

If the discriminatory power of the term weights is indeed responsible for the effectiveness of cosine and expectation-based search, binary term weights should be least efficient. Table 3.3 shows efficiency results for the different ranking variations using binary term weights. (Depth-first and breadth-first search do not need to be re-evaluated as they are not affected by the weighting scheme.) When using expectation-based priorities, binary weights are always slower than their *tf.idf* counterpart. However, cosine-based search only seems to be slower for binary weights as long as unigram terms are not part of the term representation. In fact, a system using pure binary unigram terms is significantly faster than a system using *tf.idf* weighted unigram terms.

3.3.3.1 Discriminatory power of term representations

Above, it has been suggested that the term weighting scheme – in combination with the choice of term representation which in turn influences the distribution of terms – is an important factor in determining the effectiveness of the expectation-based ranker by allowing it to distinguish between 'relevant' and 'irrelevant' instances. However, tables 3.2 and 3.3 do not seem to show a strong correlation between the average number of similarity computations and the runtime. Yet, we expect a reduced number of instances to be an indicator of efficiency of the expectation-based ranker. One explanation for the above-mentioned lack of correlation is that in some cases search continues with edges that have small relevant instance tables but that cannot be dropped, whereas in other cases, edges can be dropped early. The first case reduces the average number of similarity computations per edge although the grammar is forced to continue using the edge. On the other hand, an edge that is dropped quickly cannot help keeping the average number of similarity computations per edge down although dropping it reduces the search space of the grammar. Therefore, the overall number of similarity computations might be a better indicator of the discriminatory power of term weighting schemes. However, since different edges are produced for each run, a direct comparison between term representations is made difficult.

To investigate the ranking behaviour of different vector models more directly, we designed the following experiment: the edges produced in one parameter setting are dumped into a file (35475 edges, some marked as candidates), and we use different vector models to score exactly the same edges with the expectation-based ranker (the issue of what threshold to use is discussed below). The result allows one to determine, with respect to every edge, the size of the relevant instance table. The smaller the resulting tables, the greater the discriminatory power of the term representation. This is because the expectation-based ranker uses all instances for similarity computations for which it cannot be absolutely sure that the instances are not 'relevant' anymore. The smaller the instance table, the more precisely the ranker can focus on the relevant instances. One can liken the size of the relevant instance table to the notion of *perplexity* in speech recognition (see Jurafsky and Martin, 2000, for example) because it represents the number of choices for the nearest-neighbour when the edge is combined again.

In the experiment, we first compute the expectations for the entire instance base for each edge. Then, the expectations are checked against the threshold to determine the size of the relevant instance table. (This is necessary because the dumped edges have no predecessors from which to inherit an already reduced relevant instance table. In the real system, the updated relevant instance table is used only when the scored edge is combined with another one. In a sense, the experimental set-up therefore implies immediate reuse of the scored edge, i.e. depth-first style search. In other words, the threshold cannot change between computing the instance table and using it.)

There are two options at this point: one can use the 'real threshold' defined by the best candidate at any point a new edge is produced, or one can define a fixed threshold. A fixed threshold has the advantage of being neutral with respect to the term weighting scheme, allowing a direct comparison between term weighting schemes. On the other hand, the actual candidate score (which is affected by the term representation) should give a better indication of the real effects of the term weighting scheme.

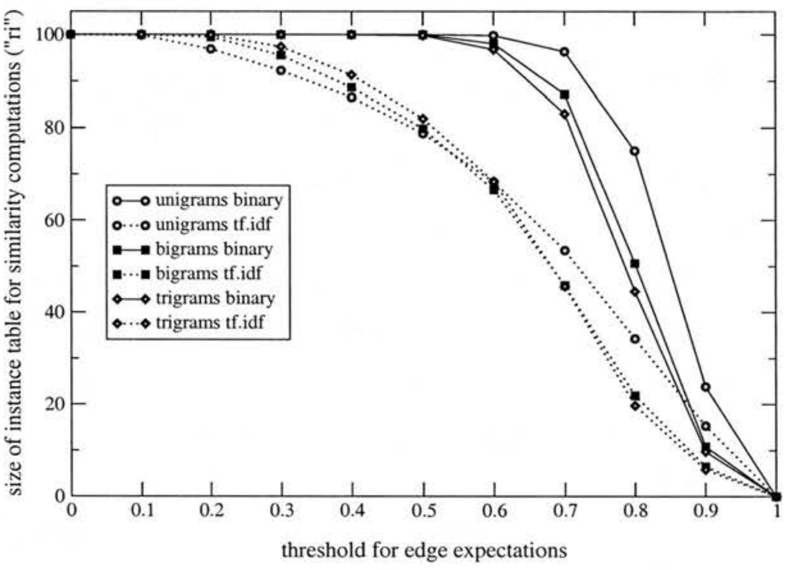

Figure 3.7: Reduction of instance tables for different fixed thresholds

Figure 3.7 shows the average numbers of instances on which the dumped edges have an expectation higher than fixed thresholds between 0.0 and 1.0 (computed for

0.0, 0.1, 0.2 etc). Three different term representations (unigram/bigram/trigram) and two term weighting schemes have been used (binary versus $tf.idf$). Detailed figures including standard deviations and the actual thresholds defined by the score of the current best candidates are given in table 3.4. Figure 3.7 is intended to give an impression of the sizes of the relevant instance tables. For all vector models, edges always have expectations larger than 0. Thus, for a threshold of 0, there are always 100 instances to consider. As the threshold increases (moving to the right on the x-axis), different vector models reduce the number of instances to different extent. In general, and this is the main result of figure 3.7, $tf.idf$ weighted terms (dotted lines) reduce the number of instances further than their binary weighted counterparts (straight lines). Moreover, the lines seem to be clustered according to the weighting scheme rather than term representation, suggesting that the weighting scheme has a stronger influence on the size of the instance set than the term representation.

threshold	**unigrams**				**bigrams**				**trigrams**			
	binary		tf.idf		binary		tf.idf		binary		tf.idf	
0.0	100	(0.0)	100	(0.0)	100	(0.0)	100	(0.0)	100	(0.0)	100	(0.0)
0.1	100	(0.0)	**99.9**	(0.4)	100	(0.0)	100	(0.0)	100	(0.0)	100	(0.0)
0.2	100	(0.0)	**96.9**	(3.0)	100	(0.0)	99.5	(0.9)	100	(0.0)	99.9	(0.4)
0.3	100	(0.0)	**92.3**	(5.1)	100	(0.0)	95.6	(4.1)	100	(0.0)	97.4	(2.6)
0.4	100	(0.0)	**86.5**	(8.4)	100	(0.0)	88.6	(6.8)	100	(0.0)	91.3	(5.9)
0.5	100	(0.0)	**78.7**	(12.7)	99.9	(0.2)	79.8	(10.5)	99.8	(0.6)	81.9	(10.0)
0.6	99.8	(0.4)	68.3	(18.6)	98.1	(3.8)	**66.5**	(17.0)	96.8	(4.9)	68.4	(15.8)
0.7	96.4	(5.5)	53.5	(25.3)	87.2	(13.0)	45.8	(24.8)	83.0	(16.8)	**45.7**	(25.2)
0.8	75.1	(20.0)	34.3	(29.0)	50.7	(29.0)	21.9	(25.3)	44.5	(31.2)	**19.7**	(25.5)
0.9	23.9	(27.2)	15.4	(25.0)	10.8	(22.8)	6.5	(18.8)	9.7	(23.5)	**5.8**	(19.3)
1.0	0.0	(0.0)	0.0	(0.0)	0.0	(0.0)	0.0	(0.0)	0.0	(0.0)	0.0	(0.0)
by cand:	54.4	(42.5)	42.8	(44.6)	68.2	(39.1)	55.7	(41.9)	76.4	(33.0)	62.5	(40.0)

Table 3.4: Number of instances used with different thresholds

Table 3.4 shows that for any term representation, the binary weighted variant needs to use the entire instance base (100 instances) up to a larger fixed threshold than its $tf.idf$ counterpart. In general, as the average size of the instance tables is reduced, the standard deviation (in brackets) increases (there is more 'room' for variation). The most efficient reduction of the instance table size is achieved by $tf.idf$ weighted unigrams for a fixed threshold up to 0.5, and by $tf.idf$ weighted bigrams for a threshold larger than 0.5. (In table 3.4, the smallest table size for each fixed threshold is shown in bold face.)

Table 3.4 also shows the (more realistic) instance numbers resulting from a threshold defined by the score of the best candidates (`'by cand'`). However, relating the instance table sizes in this 'batch experiment' to the real table sizes (and ultimately the real runtime) is not straightforward: according to the experiment (where the threshold is defined by the candidate), unigram terms should be most efficient for both term weighting schemes because the instance table sizes are the smallest (for both binary and $tf.idf$ weighted terms, respectively). This is generally not the case, as tables 3.2 and 3.3 show (with the exception of binary weighted unigram terms and cosine-based agenda ordering). An explanation might be found in the influence of agenda ordering and edge droppings on real system performance. On the other hand, figure 3.7 shows that for fixed thresholds the instance tables are largest for binary weighted unigram terms and these result indeed in a relatively bad performance of the real system. Still, we have to conclude that the efficiency of the ranker cannot directly be predicted by the discriminatory power of the term weighting scheme alone.

3.4 Lowering the expectation

A crucial prerequisite in any application of the A^* algorithm is the definition of an optimistic heuristic for the best possible future score. So far, the expectation has been computed under the assumption that all terms that are still missing in an edge w.r.t. some instance will eventually be added by future extensions of the edge. This is an overly optimistic assumption, begging the question whether the expectation can be lowered without giving up on the admissibility of the algorithm. (An alternative is to drop admissibility in favour of approximate, or 'almost admissible' algorithms. We will not pursue this line of research here.)

For the task of lowering the expectation, it is necessary to identify those instance terms that cannot possibly be added to a given edge. In general, there are two determinants that influence edge construction (and therefore also term construction): the input and the grammar. One possibility – which will not pursue here – might be to inspect the grammar and define a form of reachability for terms. However, the input itself affects the term representation in a very direct way since the input tags become part of the term representation. Therefore, we can exclude from the computation of the expectation terms that contain tags that do not occur in the input. The insight here is that the grammar will never be able to produce edges that contain semantic tags not found in the input. Consequently, instance terms that contain such semantic tags cannot be expected to be matched by some edge. We can therefore exclude them from the computation of the expectation.

This *semantic term exclusion* technique is applicable as long as the semantic input is not presented to the generator in an incremental way. As long as this can be assumed, semantic term exclusion can always be employed.

In an implementation of semantic term exclusion, the instance terms that cannot possibly be produced can be computed once when the input is known: Semantic term exclusion does not depend on the properties of edges. However, in order to take into account the excluded terms, the computation of the expectation described in section 3.2.1.1.1 needs to be changed. More specifically, the magnitude of the vector representing the best possible edge needs to be computed from scratch, making use only of non-excluded terms.

Semantic term exclusion lowers the expectation and therefore makes it harder for instances in the relevant instance tables to pass the threshold. Increased efficiency is the result. For example, the individual run examined in detail in section 3.3.1 now only generates 51 edges instead of 91 to find the first candidate with a score of 1.0.

Tables 3.5 and 3.6 show that a system employing semantic term exclusion is faster than a system without it (tables 3.2 and 3.3). In particular, expectation-driven search with unigram term representations (and both binary and $tf.idf$ weights) experiences a remarkable speed-up. This is reflected in a significantly lower number of edges, candidates and overall similarity computations. There are only two cases where semantic term exclusion seems to slow down processing (cosine-driven search with mixed ngrams and $tf.idf$ weighting). A possible explanation might be the larger term representations in combination with the large number of similarity computations which are slightly more expensive as the expectation is built up from scratch.

agenda	$\overline{E}$	σE	$\sum E$	$\sum C$	$\overline{sim}$	$\sum sim$	a_{empty}	1.0	*t-out*	$\bar{t}$
unigrams:										
expectation	89	98	3 549	447	53.3	175 775	17.5%	82.5%	0%	0.6
cosine	653	555	26 122	2705	47.8	1 198 783	17.5%	80.0%	2.5%	8.5
bigrams:										
expectation	59	36	2 374	190	77.9	178 832	17.5%	82.5%	0%	0.5
cosine	816	1847	32 653	5089	25.4	776 302	17.5%	77.5%	5.0%	7.0
trigrams:										
expectation	104	156	4 168	517	63.6	240 829	42.5%	57.5%	0%	1.2
cosine	472	1133	18 860	5767	30.5	554 000	40.0%	57.5%	2.5%	5.0
unigrams + bigrams:										
expectation	57	32	2 287	217	78.5	172 100	17.5%	82.5%	0%	0.7
cosine	806	1080	32 226	6945	42.6	1 334 113	17.5%	72.5%	10.0%	20.2
unigrams + bigrams + trigrams:										
expectation	81	127	3 256	448	67.2	205 471	17.5%	82.5%	0%	1.2
cosine	764	764	29 291	5330	49.0	1 403 193	17.5%	62.5%	20.0%	26.3

Table 3.5: Efficiency results for test set: **semantic term exclusion, tf.idf weights**

agenda	$\overline{E}$	σE	$\sum E$	$\sum C$	$\overline{sim}$	$\sum sim$	a_{empty}	1.0	*t-out*	$\bar{t}$
unigrams:										
expectation	107	160	4 272	623	59.4	237 349	15%	82.5%	2.5%	2.5
cosine	325	895	13 017	2690	23.0	284 047	15%	82.5%	2.5%	2.7
bigrams:										
expectation	165	255	6 588	1327	28.3	175 744	50.0%	50.0%	0%	1.0
cosine	1062	1995	42 464	8140	23.2	922 695	55.0%	37.5%	7.5%	9.7
trigrams:										
expectation	370	1649	14 795	6453	17.6	250 428	15%	82.5%	2.5%	2.3
cosine	535	1368	21 413	6447	15.5	324 803	15%	80.0%	5.0%	5.8
unigrams + bigrams:										
expectation	82	92	3 299	411	55.9	175 003	17.5%	82.5%	0%	1.3
cosine	612	994	24 475	6012	30.9	732 135	17.0%	82.5%	0%	9.4
unigrams + bigrams + trigrams:										
expectation	111	204	4 446	694	44.1	182 089	17.5%	82.5%	0%	2.1
cosine	595	822	23 787	6286	32.4	753 578	15.0%	75.0%	10.0%	13.8

Table 3.6: Efficiency results for test set: **semantic term exclusion, binary weights**

3.4.0.2 Discriminatory power

In section 3.3.3.1, we described a batch experiment that allows one to test the discriminatory power of vector models more directly by scoring exactly the same edges for all models. Semantic term exclusion can be expected to reduce the size of the relevant instance tables in comparison to a vector model without semantic term exclusion. To test this hypothesis, we ran the batch experiment for binary and $tf.idf$ weighted mixed ngram terms (uni/bi/trigrams) using semantic term exclusion.

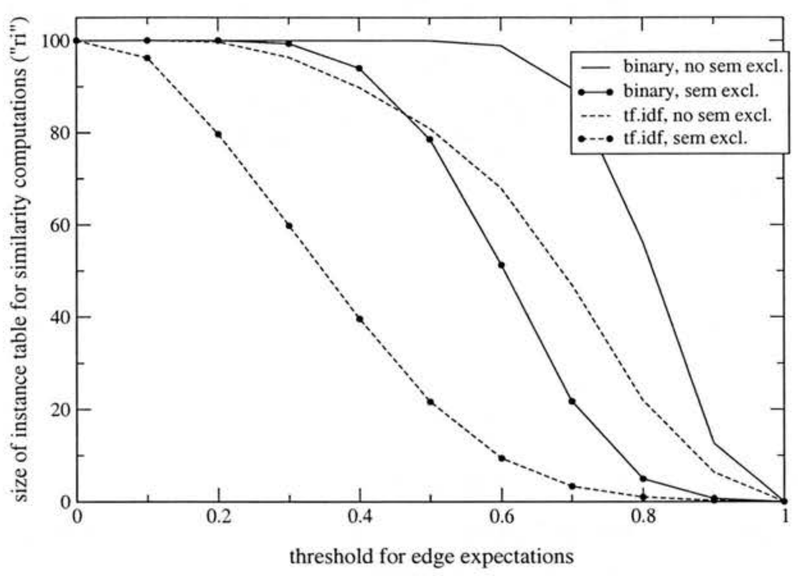

Figure 3.8: Semantic term exclusion reduces instance table sizes (mixed ngrams)

Figure 3.8 shows a clear reduction in the instance table sizes for fixed thresholds for both binary and $tf.idf$ weighted terms. This explains the efficiency gains obtained by semantic term exclusion when comparing the results of tables 3.2 and 3.3 to tables 3.5 and 3.6.

Table 3.7 details the average table sizes and standard deviations (in brackets) for fixed thresholds as well as for the actual thresholds defined by the current best candidates. For the latter, realistic thresholds, the system also shows a significant reduction of the table sizes, for example from 68.6 to 42.2 instances for binary weighted terms. In addition, table 3.7 shows the average threshold values that are used when the cosine of the best previously occurring candidate defines the threshold (`'cos thres'`). These

threshold	binary				tf.idf			
	standard		sem excl.		standard		sem excl.	
0.0	100	(0.0)	100	(0.0)	100	(0.0)	100	(0.0)
0.1	100	(0.0)	100	(0.0)	100	(0.0)	**96.2**	(4.4)
0.2	100	(0.0)	100	(0.0)	99.7	(0.8)	**79.7**	(10.7)
0.3	100	(0.0)	**99.3**	(1.4)	96.4	(3.4)	**59.8**	(14.9)
0.4	100	(0.0)	**94.0**	(6.9)	89.8	(6.4)	**39.6**	(16.9)
0.5	99.9	(0.1)	**78.6**	(14.2)	80.9	(10.2)	**21.7**	(15.9)
0.6	98.9	(2.5)	**51.3**	(19.6)	67.9	(16.1)	**9.4**	(12.1)
0.7	89.7	(11.0)	**21.7**	(17.3)	47.0	(25.0)	**3.4**	(7.5)
0.8	56.2	(28.5)	**5.0**	(9.0)	21.9	(25.5)	**1.0**	(4.3)
0.9	12.6	(24.5)	**0.7**	(2.6)	6.4	(19.0)	**0.3**	(1.8)
1.0	0.0	(0.0)	0.0	(0.0)	**0.0**	(0.0)	0.01	(0.1)
by cand:	68.6	(39.3)	**42.2**	(45.1)	60.1	(40.7)	**38.6**	(45.0)
cos thres:	0.52	(0.37)	0.52	(0.37)	0.47	(0.35)	0.47	(0.35)
expect edges:	0.92	(0.04)	0.83	(0.06)	0.88	(0.06)	0.73	(0.09)

Table 3.7: Semantic term exclusion: instance table sizes for mixed ngrams

are not affected by semantic term exclusion. On the other hand, the average expectation of the edges (`'expect edges'`) is reduced by semantic term exclusion, making it harder for the expectations to pass the thresholds.

This concludes the efficiency evaluation of the basic generation system and a first improvement. We have demonstrated that instance-based NLG can be fast. In chapter 7, we will introduce extensions motivated by the actual output of the generator. These extensions will then be re-evaluated, including considering the issue of scalability (in terms of the number of rules and instances) of the proposed generator (see section 7.5).

3.5 Generality of expectation-based ranking

The expectation-based approach to instance-based ranking makes a number of basic assumptions about the grammar and the distance metric. If these are met, the proposed algorithm should be portable to other systems.

Firstly, it is assumed that edges are always extended monotonically. More specifically, terms are not allowed to be removed or altered since this could invalidate previous pruning of instances in the relevant instance table. This assumption is met by any standard chart algorithm that forms new edges by concatenation. (It should be noted that in a unigram bag-of-words model, this requirement still leaves the possibility of word reordering by grammatical operations since this does not affect the term representation.) Moreover, as will become clear in chapter 6, an expectation-based ranker can interface with any standard chart generator. All that is required is access to newly generated edges (and their content for scoring) before they enter the agenda.

A further point concerns the applicability of the expectation-based ranking method to other distance metrics under the assumption of bag-of-words representations. The basic idea of an expectation is to add all missing instance terms to the edge representation and compute the distance of the extended edge representation to the instance. For other distance metrics to apply, it needs to be guaranteed that the extended edge representation is indeed the optimal monotonic extension of the edge. Here, we introduced a boundary on the frequency of edge terms to avoid excessive repetitions of terms. Similar measures might be required for other distance metrics as well.

3.6 Dynamic programming algorithms for ranking

Dynamic programming algorithms are commonly used in probabilistic chart parsing and also in statistical generation (Langkilde, 2000). These algorithms are similar to the *Viterbi* algorithm used for finding the best path (state sequence) in Hidden Markov Models (Huang et al., 2001). Because of their general usefulness and widespread adoption, in the following we investigate whether dynamic programming algorithms can also be employed for instance-based ranking.

The basic idea of these dynamic programming algorithms is to reuse (in most cases probabilistic) *internal scores* of partial results in larger contexts. In this sense, they are quite similar in spirit to non-probabilistic chart algorithms. The requirement for this to work is the independence from the context of the internal scores: future combinations of these results are not allowed to change the scores. The scores of the partial results can then be used as building blocks to compute the score of larger elements. For instance, in parsing with probabilistic context-free grammars (PCFGs), the probability a language model assigns to a constituent, say, *PP*, should not change when the *PP* is combined with other constituents. If this is guaranteed, the internal score of the *PP* can be reused in any number of combinations without recomputation.

In addition to internal scores which are based on independence assumptions about the language model being used, *external scores* are context-dependent. For example, if a bigram language model is used to score sequences of terminal symbols, the first and the last words of a sequence are externally relevant since they will be part of bigrams that are not yet known. In contrast, the (internal) score of the sequence in between the first and the last word can be reused.

An important advantage of probabilistic dynamic programming algorithms is their ability to safely prune away edges that have the same externally relevant features for further combinations but lower internal scores. In other words, the algorithm forms equivalence classes of edges and prunes within these classes. For example, let us assume a statistical generator has produced a number of edges with the same semantics and syntactic category. If the generator uses a bigram language model, it needs to separate these edges into classes containing the same first and last word. It can then drop all edges in their respective classes except the ones with the highest internal score. An example is given in (Langkilde, 2000). The details of forming equivalence classes depend on the grammar as well as the language model. The members of each class need to be exchangeable with respect to both the grammar and the language model: in addition to the externally relevant features for scoring by the language model, the details of the grammar formalism determine how many edge properties need to be taken into account.

Considering dynamic programming algorithms of the form just outlined, the question arises whether we can adapt them to the case of instance-based ranking. There are at least two aspects of the instance-based ranking algorithm that distinguish it from standard probabilistic chart algorithms:

1. Each edge has as many scores as there are instances in the instance base. In contrast, language models assign unique scores.

2. The similarity computations work on *global* bags-of-words: matches and mismatches between edge and instance are calculated by considering all terms of the respective term representations regardless of the place in the original structure from which the terms are derived.

In the following, we investigate the consequences of these differences for a 'Viterbi-style instance ranker':

1. **Unique versus multiple scores.** The first difference poses the following problem: In general, we are always interested in the single best score for some edge out of the larger number it is assigned. If we form grammar-based equivalence classes for edges (say, for edges that have the same semantics and syntactic category), we would prune all non-best edges regardless of the instances that 'caused' the scores. However, if two of the supposedly best edges are combined, the result needs to be scored w.r.t. to the *same* instance. As a result, at least one of the edges to be combined will most likely have to be matched against a different instance than the one for which it previously received its high score. This in turn will result in a non-optimal overall score for the new edge. In other words, without modification, a Viterbi-style algorithm is is unlikely to combine edges that are optimal w.r.t. the same instance. Figure 3.9 shows a table which holds the best cosine encountered so far for classes of edges defined by their category and semantics. We hold that such a table is non-optimal as it conflates scores derived from different instances.

   The remedy for this problem is to maintain a separate table for each instance within each class, and keep edges as long as they have a better score than previously seen on at least one instance. An example of such a table is given in figure 3.10. The changes to the chart algorithm just outlined extend it to the case of several instances, i.e. several 'evaluation functions'.

2. **Globality of bag-of-words models.** The second difference concerns the global nature of bag-of-words models. Pruning edges requires the definition of an internal score that is not dependent on any context. Bag-of-words models seem to contradict this idea. Let us assume that we are combining two edges that both have the highest score in their equivalence classes. Theoretically, this does not guarantee the best overall score of the combined edge in (at least) two situations:

   (a) If the subedges of the combined edge have **overlapping term representations**, the upper bound on matching edge terms will punish the combined

```
thresholdTable = { Cat1 -> { Sem1 -> score11,
                             Sem2 -> score12,
                             Sem3 -> score13,
                             ... },
                   Cat2 -> { Sem1 -> score21,
                             Sem2 -> score22,
                             Sem3 -> score23,
                             ... },
                   ... }
```

Figure 3.9: Incorrect: threshold scores conflating instances in Viterbi-style algorithm

```
thresholdTable = { Cat1 -> { Sem1 -> { inst1 -> score111,
                                       inst2 -> score112,
                                        ... },
                             Sem2 -> { inst1 -> score121,
                                       inst2 -> score122,
                                        ... },
                             ...},
                 { Cat2 -> { Sem1 -> { inst1 -> score211,
                                       inst2 -> score212,
                                        ... },
                             Sem2 -> { inst1 -> score221,
                                       inst2 -> score222,
                                        ... },
                             ... },
                    ... }
```

Figure 3.10: Thresholds stored per instance in Viterbi-style algorithm

edge for excessive repetitions of the same terms. This problem is a result of the globality of bag-of-words models because edge terms can be matched with instance terms representing any part of the original instance.

An example might help explaining this point. Consider an instance represented by the three terms *a*,*b*,*c* and an edge e_1 represented by *a*,*b*. Further combinations of e_1 should preferably contribute the term *c* to yield a perfect match with the instance. If we have two edges $e_2 : a, b$ and $e_3 : c$ of the same equivalence class competing to be combined with e_1, we would clearly prefer e_3. However, in dynamic programming, e_3 might have been pruned away already since e_2 has a higher overlap with the instance than e_3.

(b) **Long versus short edges:** Even if the term representations of subedges do not overlap, there is a possibility that short edges that are needed in a larger context receive a lower score. For example, with respect to an instance *a*,*b*,*c*,*d*, an edge *a* is assigned a lower score than an edge *a*,*b*,*e* despite the non-matching term in the latter.[6] If both edges compete within the same equivalence class, the shorter edge will be pruned away although it might be needed later during processing. This problem is again a result of the global nature of bag-of-words models because both edges are matched against the entire instance bag-of-words (rather than a local area, for example).

It should be emphasized that the globality problem even affects the solution for the problem of multiple scores in figure 3.10 because the use of bags-of-words representations remains unchanged.

3.6.1 Implementation experiment

To investigate these considerations, we implemented two versions of a Viterbi-style instance ranker and tested it on 10 inputs for which the expectation-based ranker is able

[6]Under the assumption of binary weighted unigram terms and a cosine distance metric, edge *a* has a cosine of 0.25 with the instance; edge *a*,*b*,*e* has a cosine of 0.57.

to exhaust the agenda. The systems were required to express the entire input to produce more edges on average. This can generally be expected to result in longer runtimes per input (see section 7.4 for a detailed discussion of completeness). Like the expectation-based ranker, but unlike breadth-first and depth-first search, the Viterbi rankers need to compute similarity scores for each edge rather than just for each candidate. The score is used to determine the agenda ordering. In contrast to expectation-based search, the Viterbi rankers have to use the entire instance base.

ranker type	$\overline{cos}$	$\sigma(cos)$	$\sum E$	$\sum sim$	$\overline{sim}$	$\bar{t}$
unigrams, tf.idf						
expectation	0.953	0.057	4 972	175 933	40.0	7.0
viterbi-single	0.81	0.11	1 960	196 000	100	1.9
viterbi-multi	0.951	0.058	16 932	1 693 200	100	24.3
unigrams, binary						
expectation	0.95	0.06	10 395	196 175	19.6	13.6
viterbi-single	0.92	0.07	2 088	208 800	100	2.0
viterbi-multi	0.95	0.06	9 203	920 300	100	11.2
bigrams, tf.idf						
expectation	0.906	0.083	9 287	134 123	15.0	9.0
viterbi-single	0.66	0.13	2 654	256 900	96.8	3.0
viterbi-multi	0.906	0.083	17 171	1 708 600	99.5	25.4
bigrams, binary						
expectation	0.894	0.11	9 170	205 281	23.4	9.0
viterbi-single	0.825	0.12	2 817	273 200	97.0	3.1
viterbi-multi	0.892	0.111	13 319	1 323 400	99.4	20.6
trigrams, tf.idf						
expectation	0.839	0.1136	14 920	203 144	14.0	14.3
viterbi-single	0.64	0.13	2 363	220 800	93.4	2.7
viterbi-multi	0.838	0.114	10 709	1 055 400	98.6	20.8
trigrams, binary						
expectation	0.827	0.145	22 653	382 361	17.5	30.1
viterbi-single	0.74	0.16	2 756	260 100	94.4	3.1
viterbi-multi	0.824	0.146	10 212	1 005 700	98.5	14.0

Table 3.8: Results for Viterbi-style and expectation-based rankers

Table 3.8 shows the results for six different term representations. Ranker type `viterbi-single` refers to a ranker that keeps a single best score for each equivalence class as depicted in figure 3.9. Ranker type `viterbi-multi` refers to a system that separates scores according to instances within each class (see figure 3.10). Our main concern is the cosine score of the best candidates. Since we know that the expectation-based ranker finds optimal candidates as long as it is able to exhaust the agenda, achieving its cosine results should be the aim of the Viterbi rankers.

3.6.1.1 Unigram term representations

We set out by testing the rankers in combination with unigram, $tf.idf$-weighed term representations. As predicted, the single-score Viterbi ranker clearly misses the best possible scores. However, it is very fast because it drops a large number of edges (in many cases, more than 50% of the newly produced edges). The multi-instance Viterbi ranker produces better candidates than the single-score version. In fact, the candidates generated by the multi-instance Viterbi ranker seem to be almost as good as those of the expectation-based ranker. In six cases, the candidates and scores were exactly the same; in four cases, the expectation-based system generates slightly better candidates. These differences result in different candidates so that they still have a real effect on the output of the generator.

For the unigram, $tf.idf$-weighed term representation, the multi-instance Viterbi ranker obviously performs surprisingly well in terms of the average cosine score although it still is much more inefficient than the expectation-based ranker. The theoretical problems of using a Viterbi-style ranker in combination with bag-of-words models mentioned above do not seem to have a big impact – if any at all. To investigate this, we can ask whether the multi-instance Viterbi system can be improved so that it generates the same candidates as the expectation-based ranker in the cases where the outputs have been different.

It turns out that improving the multi-instance Viterbi ranker is possible: in cases where an edge does not offer an improvement over another edge on the same instance in its equivalence class, it has so far been dropped. The multi-instance Viterbi ranker can be changed so that it keeps edges that are equal in score to the score in the table. As a result, the Viterbi ranker now generates exactly the same candidates as the expectation-based ranker. However, this achievement comes at a price: it now exceeds a time limit of two minutes for 3 out of the 10 inputs, and generates 30 000 edges for the inputs within this time frame (versus 4972 of the expectation-based ranker). Furthermore, the theoretical problems of using a Viterbi ranker in combination with global bags-of-words representations remain.

In terms of the overall number of similarity computations, the expectation-based ranker is the most efficient although it does not produce the least edges. It manages to reduce the average number of similarity computations per edge to 40. In contrast, the Viterbi rankers have to perform as many cosine computations for each edge as there are instances in the instance base.

Table 3.8 also shows the results for binary weighted unigram terms. Again, the single-score Viterbi ranker is fast but produces suboptimal candidates. Both expectation-based ranker and the multi-instance Viterbi ranker (without the extension just discussed) produce exactly the same candidates. The multi-instance Viterbi ranker even is faster than the expectation-based ranker. However, as the efficiency evaluations in section 3.3 have shown, binary weighted unigram terms seem to be particularly difficult for the expectation-based ranker (because discrimination between edge-instance pairs is made difficult).

A further observation concerns the tf-boundary on matching terms. The boundary affects almost exclusively unigram models. In other words, only when using unigram terms are there cases where edges have a higher number of terms of a certain kind than the instance. Obviously, individual words like determiners are more likely to be used repeatedly than higher order ngrams. Therefore, the cosine function of the Viterbi rankers also needs to take the tf-boundary into account. Apart from changing the actual candidates, this is required for a detailed comparison of the scores between Viterbi ranker and expectation-based ranker since both need to assign exactly the same score to the same candidates.

3.6.1.2 Bigram and trigram term representations

Bigram terms that are weighted according to $tf.idf$ result in exactly the same candidates for both expectation-based ranker and multi-instance Viterbi ranker. However, the expectation-based ranker is significantly faster than the multi-instance Viterbi ranker (9 versus 25.4 seconds). Again, the multi-instance Viterbi ranker is not able to reach the efficiency scores of the other rankers. The average number of similarity computations per edge is slightly lower than 100, the size of the instance base. This is because with ngrams with $n \geq 2$ there can be edges that are too short to allow the construction of at least one term, i.e. their term representation is empty. In such cases, we keep the edge since there is no reason yet to drop it.

For binary weighted bigram terms, the results are similar with the exception that for one input, the multi-instance Viterbi ranker does not produce the candidate of the expectation-based ranker but a lower scoring one. The optimal candidate can be produced by keeping new edges that score equally well as the best old ones in their class. However, processing the input in question now requires 6:05 minutes versus 12.8 seconds for the expectation-based ranker.

For trigram terms weighted according to $tf.idf$, a similar picture can be observed. Expectation-based search is faster than the multi-instance Viterbi ranker. Both differ only w.r.t. a single input. If the multi-instance Viterbi ranker is to find the same candidate as the expectation-based ranker, it needs to keep equal edges. However, the system runs out of memory after the candidate has been found.

Binary weighted trigram terms show a different picture. Again, the multi-instance Viterbi ranker produces the same candidate with one exception. It does not produce a better candidate when keeping equal edges before memory problems occur. Despite this, the previous experiments suggest that obtaining the optimal candidate should be possible.

However, for binary weighted trigram terms, the expectation-based ranker also encounters problems for one of its inputs where it does not terminate within the two minute time limit. This leads to the high average time requirement of 30.1 seconds shown in table 3.8. A closer inspection of the run in question reveals that the best candidate – although being produced after 3 seconds – does not yield a threshold high enough to prune the remaining search space sufficiently.

3.6.1.3 Time limits

The above-mentioned observation that the best candidate for one particular input was found early during processing begs the general question how long it actually takes the rankers to produce the best candidates, and whether there can be a general guideline for a sensible time limit. Tables 3.9 and 3.10 detail the runtime in seconds until the best candidate is produced for the three vector models. Time requirements greater than 10 seconds are shown in bold face.

input no	unigrams tf.idf		bigrams tf.idf		trigrams	
	expect	Vm	expect	Vm	expect	Vm
1	6	6	3	**18**	4	4
2	1	3	1	3	1	1
3	1	4	1	7	2	9
4	1	1	1	2	1	1
5	**22**	**13**	3	3	2	9
6	2	2	1	**31**	9	**11**
7	1	6	1	3	3	**10**
8	6	**42**	**10**	**13**	8	6
9	1	**33**	2	**19**	3	4
10	2	**22**	3	9	4	9
average	4.3	13.2	2.6	10.8	3.7	6.4
σ	6.5	14.4	2.8	9.5	2.8	3.7
$\overline{t_{end}}$	7.0	24.3	9.0	25.4	14.3	20.8

Table 3.9: Runtime until best candidate found: $tf.idf$ weights

The third last rows in tables 3.9 and 3.10 show the average time to the best candidate. The last rows show the average time it takes to exhaust the agenda (which is also displayed in table 3.8). Both expectation-based ranker and multi-instance Viterbi ranker are able to produce the best candidate quickly but require more time to finally exhaust the agenda. For both $tf.idf$ and binary weighted terms, the general picture is that the expectation-based ranker finds the best candidate more quickly – with the exception of binary trigram terms. There are 3 cases where the expectation-based ranker requires more than 10 seconds to produce the best candidate, compared to 16 cases for the multi-instance Viterbi ranker. The standard deviation of the times to the best candidate (second row from the bottom) tends to be lower for the expectation-based ranker. In other words, the expectation-based ranker is more consistent in its time requirements, making it easier to define a time limit. In this experiment, a time limit of less than 15 seconds seems to be appropriate for the expectation-based ranker.

input no	unigrams binary		bigrams binary		trigrams binary	
	expect	Vm	expect	Vm	expect	Vm
1	3	3	5	7	8	3
2	1	1	1	2	1	1
3	2	2	1	4	1	3
4	2	1	2	2	3	1
5	8	**12**	7	**31**	6	**12**
6	2	5	1	**15**	1	6
7	1	4	4	8	3	4
8	4	**34**	2	**57**	9	8
9	1	4	5	5	9	2
10	**13**	3	3	**22**	4	1
average	3.7	6.9	3.1	15.3	4.5	4.1
σ	3.9	10.0	2.1	17.5	3.3	3.6
$\overline{t_{end}}$	13.6	11.2	9.0	20.6	30.1	14.0

Table 3.10: Runtime until best candidate found: binary weights

3.6.2 Combining expectation and Viterbi ranker

The greater speed of the multi-instance Viterbi ranker at least in some cases raises the question whether expectation-based ranker and Viterbi ranker can be combined. However, as pointed out above, a disadvantage of the Viterbi ranker is the need to score the edges on all instances. In contrast, a key benefit of the expectation-based ranker is its ability to reduce the number of instances to be considered. A combination of the ranking methods should inherit this trait from the expectation-based ranker.

A possible solution is offered by the concept of a relevant instance table which is used by the expectation-based ranker. It implies that those instances that cannot be found in the table are deemed to be 'irrelevant'. Thus, it should not matter that a score on some irrelevant instance is unknown for a given edge. For keeping an edge, we would therefore require it to show an improvement of the score on at least one instance in its equivalence class in the Viterbi table (where adding a new instance to the class also counts as an improvement).

Implementing such an additional 'Viterbi filter' for the expectation-based ranker in the end is quite similar to a redundancy check. The Viterbi check needs to be performed *after* expectation-based ranking has taken place because it needs the results of ranking (which then always needs to compute the cosine in addition to the expectation). Thus, the Viterbi check adds another step to the ranking algorithm outlined in section 3.2.1.2 (after step 3).

ranker type	$\overline{cos}$	$\sigma(cos)$	$\sum E$	$\sum sim$	$\overline{sim}$	$\bar{t}$
trigrams, tf.idf						
expectation standard	0.839	0.1136	14 920	203 144	14.0	12.4
expectation Viterbi filter	0.839	0.1136	13 946	202 116	15.0	9.8
trigrams, binary						
expectation standard	0.827	0.145	22 881	383 833	17.4	25.7
expectation Viterbi filter	0.827	0.145	21 121	377 922	18.5	17.0

Table 3.11: A Viterbi filter can improve expectation-based search

To test the effects of the Viterbi filter on the expectation-based ranker, we used the same 10 inputs as in the previous evaluation. Table 3.11 shows the results for $tf.idf$-weighted and binary trigram terms. It should be noted that the Viterbi check needs to accept an edge even if it has a score equal to (but not better than) the best previous score (on the same instance in the same equivalence class) since this was one of the results of the previous experiments. However, this still does not guarantee optimality in general.

As table 3.11 shows, the extended expectation-based ranker improves the performance and it is still able to find exactly the same candidates as the standard expectation-based ranker. The speed-up is limited in the case of $tf.idf$-weighted trigram terms but quite substantial for binary weighted trigram terms. In particular, it is now able to avoid the time-out for one of the inputs from which the standard expectation-based ranker suffered.[7] Results for the multi-instance Viterbi ranker are not shown in table 3.11 because the earlier results were based on a ranker that drops equal-scoring edges (and keeping them turned out to be not feasible).

As a result, we conclude that the expectation-based ranker can be extended by a Viterbi filter. This may avoid problems especially in cases where the best candidate is not good enough to prune successfully. However, we still maintain that the Viterbi

[7]The average ranking times of the expectation-based ranker have changed with respect to the previous evaluation because of the machine used. This also effects the time-out run to some degree. However, experiments are always run in a row to ensure comparability.

filter is potentially liable to theoretical problems even if these could not be verified in the experiments. Thus, it should only be used after the optimal results have been established by the standard expectation-based ranker (as was done in these experiments).

3.6.3 Summary of Viterbi implementation experiment

The inputs for the evaluation in this section were chosen so that the rankers were able to produce complete candidates and exhaust the agenda. Keeping in mind that the number of inputs is limited, we draw the following conclusions:

- A Viterbi ranker that does not distinguish between instances is not able to deliver optimal candidates.
- A Viterbi ranker that separates similarity scores w.r.t. to different instances can achieve the same (optimal) results as the expectation-based ranker in most cases. However, to obtain the same results for all tested inputs, it also needs to keep edges that score as well as the best old ones in their class. This leads to prohibitive runtime requirements – unless a time limit is used (see below). Moreover, even keeping new edges that score as well as the best ones in their class does not guarantee optimality in general.
- The expectation-based ranker performs better than its competitors when $tf.idf$ term weights are used.
- For binary term weights, the expectation-based ranker can encounter efficiency problems (although this seems to be rare).
- It is possible to empirically determine a time limit for both the expectation-based ranker and the multi-instance Viterbi ranker that ensures that an optimal candidate is found in most cases. The multi-instance Viterbi ranker is less consistent in its time requirements, making it harder to define the time limit.
- It is possible to combine expectation-based ranking and Viterbi-style ranking while still avoiding similarity computations on the entire instance base. An

expectation-based ranker extended by a Viterbi filter seems to improve efficiency. However, such a combination also inherits the 'globality problem'.

- The theoretical problems of using a Viterbi-style algorithm in combination with bag-of-words instance representations identified could not be verified.

The latter point merits a further look. The theoretical problem of globality might not occur because edges in their equivalence classes have roughly the same length (after all, they cover the same semantic subset of the input). Furthermore, in general there seems to be little overlap between the edges which can be explained by the fact that terms cover different semantics, and different semantics tends to be mapped to different terms. The exception are unigram terms which have overlapping term representations. However, the tf-boundary does not seem to distort the determination of the best 'local' edge even in this case.

As a result of the evaluation experiment, we can establish that a Viterbi-style ranker that distinguishes between instances could be used as a practical alternative to the expectation-based ranker for our test data. However, the theoretical problem of non-optimality remains so that it is difficult to say how the Viterbi-style ranker will perform on other test data. Expectation-based and Viterbi-style ranker do not seem to be mutually exclusive, despite the fact that one draws on ideas of A^*-search, the other on concepts taken from probabilistic chart algorithms. Although for most term representation and weight combinations, the expectation-based ranker exhausts the agenda more quickly, binary weighted models might benefit from the Viterbi ranker, or a Viterbi filter within an expectation-based ranker. However, it should be emphasized that evaluating the Viterbi rankers was only possible because we also have an admissible algorithm that is guaranteed to find the optimal candidate(s). Only by means of the expectation-based ranker can we verify whether the multi-instance Viterbi ranker encounters problems in other domains or when using other term representation schemes.

3.7 Conclusion

In this chapter, we have presented an instance-based ranking algorithm for natural language generation. The ranker employs information retrieval methods for the computation of similarity scores and applies them to the notion of an 'expectation'. The ranking algorithm can be regarded as an extension of the A^*-algorithm to the case of several estimation functions, adapted to the chosen distance metric. The algorithm prunes away combinations of instance-edge pairs that are not expected to result in sufficiently high similarity scores. The algorithm uses the entire instance base for initial edges and restricts the number of instances as edges are combined. An edge can be discarded completely when it has no remaining 'relevant instances'. The overall architecture of the generator is based on the idea of an interaction between grammar interpreter and ranker such that both components are able to constrain each other. Furthermore, we have introduced an adaption of 'Viterbi-style' probabilistic chart algorithms to the task of instance-based ranking, and have described a combination of the two ranking algorithms. The efficiency evaluations have shown the feasibility of the proposed methods.

In the following chapters, we describe how grammar rules can be constructed automatically from a semantically annotated treebank and how those grammar rules can be executed using a Rete-based chart algorithm.

Chapter 4

Semantic annotation of a domain corpus

Having described an instance-based ranker for the proposed hybrid surface realizer, we now need to specify the rule-based component. As outlined in chapter 1, our intention is to use an existing syntactic treebank to automatically produce a generation grammar. Developing principles for doing so is intended to reduce the development effort for the grammar component. It will also help porting the system to other domains for which a treebank-like corpus can be obtained by using the results of running a statistical parser on a text corpus. However, we need to add a level of semantic annotation to the treebank which corresponds to the assumed input to the realizer. Only then will we be able to construct grammar rules that map input semantics to surface strings. This semantic annotation is domain-specific and is applied manually. It is intended to be declarative and should not impose any ordering on processing when the annotation is used as generation input. Because the level of semantics is similar to that of information extraction systems, the process of producing the resource for generation grammar construction is similar to producing training material for an information extraction system (see section 5.7). However, when annotating with the aim of providing generation input, we need to mark all information that should not be generated without appropriate generation input. This seems to be in contrast to NLP tasks like information extraction that can ignore parts of the corpus that are not of interest to the application.

We collected 144 *Who's News* articles of the *Wall Street Journal* that are part of the Penn treebank II (Marcus et al., 1993). This had to be done manually because the information about the specific origin of the texts was not preserved when the treebank was compiled. As mentioned before, we limit ourselves to the first sentences of these articles. In this work, we first annotate the unanalysed corpus and then merge the semantically annotated corpus with the treebank. (An alternative is to apply the annotation directly to the treebank.) In this chapter, we describe the first step, the annotation of corpus sentences.

4.1 Annotating generation templates

The basic idea behind the annotation scheme is to produce generation templates into which values, or 'slot fillers', can be inserted. Strings marked as slot fillers therefore need to contain information that can be provided by a content determination component which, for example, may draw on a database. Starting from such templates, our next goal will be to introduce more flexibility by taking advantage of the constituent structure provided by the treebank (this will be described in chapter 5).

To define the annotation scheme, we make a number of assumptions. Corpus strings are marked by tags. Since the generation input is supposed to be 'flat', the annotation scheme needs to be non-hierarchical as well. Thus, no nesting of tags is possible. The marked string will be the values of the generation input. It is assumed that only continuous sequences of words are marked. Generally, these can be of arbitrary length. However, since it should be possible for a content determination module to provide the marked information, we aim at marking factual information as 'narrowly' as possible. (11) gives a (generalized) example without introducing specific tag names:

(11) [TAG Pierre Vinken], [TAG 61] years old, will join the board as a [TAG nonexecutive director] [TAG Nov. 29].

Tags are just character sequences without whitespace and, like words, are treated as atoms by the ranker. Because we intend to merge syntactic treebank and annotation at a later stage, the annotation needs to be consistent with the bracketing structure of the treebank. As we will see below, in some cases treebank words need to be broken up when we do not want to mark an entire treebank word.

We generally assume that strings marked by a certain tag always have the same syntactic category. In other words, we are defining equivalence classes that make sure that a tag's slot fillers are not only semantically but also syntactically *substitutable*. This consideration points towards a general requirement of the annotation process: consistency. When annotating with the goal of producing generation templates, consistency means that marked strings of the same type (tag) need to be substitutable. Consequently, a useful tool for developing the annotation scheme is a concordancer because it allows the annotator to see the same tag in different contexts.

4.2 Defining tag names

The definition of tag names generally starts with a basic semantic category, followed by more detailed specifications. Although such a system helps the annotation process, it should be stressed again that tags are treated like atomic symbols by the generation system. Thus, the system can only test the identity of these symbols (and not their similarity at the level of more general semantic types, for example). There are four basic semantic categories in the annotation scheme that are used as first elements in the final, composed tags:

1. `POST`: tags referring to the main position of the management succession event,
2. `INPERSON`: persons occupying posts,
3. `OUTPERSON`: persons leaving posts,
4. `COMP`: company descriptions including subsidiaries.

Furthermore, there are tags pertaining to temporal and referring expressions. In the following, we introduce some of the actually used tags that are based on these main semantic categories, pointing out particular uses and difficulties. (An exhaustive list of the semantic tags can be found in appendix B.) The presented annotation scheme reflects experience gained with earlier versions of the scheme.

4.2.1 Tags pertaining to the main post

The POST tag prefix refers to the name of the post in the company or organization that is to be occupied or left. Some examples:

(12) a. ... was named an executive vice president ...

b. ... was named president.

c. ... was elected senior vice president, public affairs and advertising, ...

Marking post names requires one to handle definiteness. In (12a), the indefinite article is used to express that there is more than one 'executive vice president' at the main company. On the other hand, there is only one 'president' and one 'senior vice president, public affairs and advertising' so no determiner is used in the original texts.

One option would be to simply ignore the existence of determiners and mark all main positions just as POST. However, since the generator can only see the attributes of the inputs but not their values, i.e. the strings we are marking, it is not able to distinguish between different values and would thus generate 'was named an president', for example. Thus, we want to give the generator more guidance. For example, we might want to simply include determiners in the string marked by POST. However, this is not possible since it would require discontinuous tags due to intervening material:

(13) ... <u>the</u> 56-year old <u>chairman</u> of ...

If we want to be able to mark the age information as well, we cannot mark 'the' and 'chairman' with the same tag in a non-hierarchical annotation scheme. (Example (13) shows a different kind of post which is categorized below. However, we want to be systematic in developing the annotation scheme and apply the same treatment of definiteness throughout the annotation scheme.) Another option is to mark determiners individually by specific tags. However, we do not want to give the generator explicit word forms of non-factual information.

Our strategy to solve the problem is to mark posts narrowly, excluding any determiners, but to distinguish between tags without determiner (POST_NODET, corpus frequency f=142), indefinite post descriptions (POST_INDEF, f=36) and definite ones

(POST_DEF, f=0[1]). Thus, an annotated sentence template contains different usage patterns for these tags:

(14) ... was named [POST_NODET president], [POST_NODET chief executive officer] and a [POST_INDEF director] ...

Template (14) shows three coordinated posts. The template expresses, for example, that the indefinite post is the last one in such a template (empirically true in 5 out of 6 cases of three coordinated posts).

The POST tags just introduced expect as slot fillers singular nouns. There is also a POST_PLURAL tag (f=7) which will be discussed in section 4.3.

Furthermore, there is a designated tag for becoming a member of a company's board:

(15) a. ... was elected to the [POST_BOARD board] of this ...

b. ... was elected to the [POST_BOARD board of directors] of this ...

In cases like these, no explicit post name is mentioned and the board is referred to by a definite article. The introduction of the POST_BOARD tag (f=10) prevents the generator from producing 'was elected to the president of' or 'was named a board', for example.

The *Wall Street Journal* articles also specify a change in the number of board members which we mark in the following way:

(16) a. ... , expanding the board to [BOARD_INCR eight] members.

b. ... , increasing board membership to [BOARD_INCR 14].

The BOARD_INCR tag has a corpus frequency of 14.

The corpus sentences often provide additional information about the main post. The filler needs to be an adjective or an adjective phrase (labelled JJ and ADJP in the Penn treebank):

(17) a. ... the [POST_DESCR_ADJ additional] post of chairman.

[1]In the corpus, there is no example of a POST_DEF tag which is likely to be due to the fact that we are generating first sentences. However, we will need a _DEF suffix for cases like (13).

b. ... , a [POST_DESCR_ADJ newly created] post.

In this case, we decided to mention the syntactic category of the slot filler explicitly in the tag name. However, it should be remembered that all tags are assumed to have fillers of the same syntactic category.

```
0195:  officer, posts which had been [POST_DESCR_ADJ vacant] .
0197: ner division, was named to the [POST_DESCR_ADJ additional] post of group
0206: ent of its technology group, a [POST_DESCR_ADJ new] position.
0226: rces concern, was named to the [POST_DESCR_ADJ additional] post of chief
0237:  chief operating officer, both [POST_DESCR_ADJ newly created] posts, and
0366: stment Trust, was named to the [POST_DESCR_ADJ new] post of vice chairma
0368: tive officer, was named to the [POST_DESCR_ADJ additional] post of chair
0369: no, resident, was named to the [POST_DESCR_ADJ new] post of vice chairma
0512: staurant operator, assumed the [POST_DESCR_ADJ additional] post of chief
0548:  president, was elected to the [POST_DESCR_ADJ additional] posts of chai
0644: tive officer, was named to the [POST_DESCR_ADJ additional] post of chair
0740: en named regional president, a [POST_DESCR_ADJ new] post at the bank-hol
...
26 matches
```

Figure 4.1: Output of concordancer for annotation tag POST_DESCR_ADJ

Figure 4.1 shows the output of a simple concordancer which provides the annotator with an overview of the different uses of a tag, in this case the POST_DESCR_ADJ tag. The numbers to the left of figure 4.1 refer to the corresponding treebank files. The output of the concordancer shows that one could indeed exchange the slot fillers. There seems to be little variation within the words marked by POST_DESCR_ADJ. The POST_DESCR_ADJ tag has a corpus frequency of 25. The context of the tags also shows that the value of some POST_DESCR_ADJ tags refers to more than one post. This will be discussed in section 4.3.

Specific geographical locations of the main post are marked by the POST_LOC tag. It pertains to the place where a person occupying a post is based. The entire NP including the determiner is marked:

(18) a. ... based in [POST_LOC Tokyo].

b. ... named a vice president in [POST_LOC the Canadian head office].

The POST_LOC tag (f=4) refers to the physical location of the position, in contrast to some post names containing a location that further determines the function of the position:

(19) ... was named [POST_NODET group head investment banking in Asia], based in ...

4.2.2 Tags pertaining to persons occupying the main post

The INPERSON_FULLNAME tag (f=139) marks the name of the person assuming the main position, including initials, titles and appositives such as 'Jr.' and 'III':

(20) a. [INPERSON_FULLNAME Robert G. Walden] was elected ...

b. [INPERSON_FULLNAME Frank Carlucci III] ...

c. [INPERSON_FULLNAME Gen. Paul X Kelley] ...

d. [INPERSON_FULLNAME G. William Ryan] ...

The first sentences of the corpus articles introduce the person into the discourse. It can be assumed that the full name is given as far as it is known to the authors of the articles.

The age of the incoming person is marked by INPERSON_AGE (f=56). Only the numeral proper is marked. In some cases, this leads to marking only a substring of a word:

(21) a. ..., [INPERSON_AGE 55] years old, ...

b. ..., [INPERSON_AGE 63]-year-old chairman of ...

To be able to deal with (21b), the treebank word '63-year-old' needs to be broken up. We thus converted '(JJ 63-year-old)' into '(CD 63) (JJ -year-old)' in the treebank.

The person assuming a position frequently has other positions in the same or other companies. We mark these by _OTHERPOST. As for the POST tag, the determiners 'a', 'an' or 'the' are not included in the tagged string. Rather, the presence and type of determiner is specified in the tag:

(22) a. George W Koch, 63 years old, [INPERSON_OTHERPOST_NODET president] of Grocery Manufacturers of America Inc., was elected ... (f=59)

b. Charles S. Mitchell, a [INPERSON_OTHERPOST_INDEF vice president] with ... (f=9)

c. Erwin Tomash, the 67-year-old [INPERSON_OTHERPOST_DEF founder] of this ... (f=3)

(22c) shows that the slot filler 'founder' is treated like other ordinary posts.

The name of the company at which additional posts of a person are held is marked as INPERSON_OTHERCOMP:

(23) George W Koch, ... president of [INPERSON_OTHERCOMP Grocery Manufacturers of America Inc.], was elected ... (f=33)

There are a few special cases concerning the INPERSON_OTHERPOST tag that are due to specific slot fillers. The prepositions used to relate INPERSON_OTHERPOST and INPERSON_OTHERCOMP are 'of' (f=26), 'at' (f=3) and 'with' (f=1). Alternatively, a comma can be used (f=1). The concordances suggest that the respective slot fillers are interchangeable. A few cases, however, require a new tag to prevent unwanted substitutions:

(24) a. ... a [INPERSON_OTHERPOST_PARTNER_INDEF partner] in [INPERSON_OTHER-COMP the Washington law firm of Powell, Goldstein, Frazer & Murphy], ... (f=1)

b. ... a [INPERSON_OTHERPOST_CONSULTANT_INDEF consultant] to [INPERSON-_OTHERCOMP Drexel Burnham Lambert Group Inc.], ... (f=1)

Although we want to avoid tying tag names to individual slot fillers, we are forced to do so here because the prepositions 'in' and 'to' cannot be used in combination with the more common posts. Note that in (24a) we include the definite article in the string marked by INPERSON_OTHERCOMP. This is necessary to ensure substitutability with other, more common fillers of this tag.

The *Wall Street Journal* articles also detail previous posts of the incoming person. These tags are generally analogous to INPERSON_OTHERPOST and INPERSON_OTHERCOMP above. Again, the variety without determiner is the most frequent one:

(25) a. ..., formerly [INPERSON_PREVIOUSPOST_NODET vice president, West Coast Operations,] at this ... (f=25)

b. ... a former [INPERSON_PREVIOUSPOST_INDEF chairman] ... (f=0)

The use of the adjective 'former' in between determiner and noun is another example that motivates our treatment of determiners.

The company at which a previous position was held is marked by INPERSON-_PREVIOUSCOMP (f=10):

(26) a. ..., former president of [INPERSON_PREVIOUSCOMP Toys "R" Us Inc.], ...

b. ..., former chairman of [INPERSON_PREVIOUSCOMP the Joint Chiefs of Staff], ...

In most cases, the company name is in fact a proper name (as in 26a). In special cases like (26b), we include the determiner in the marked string.

4.2.3 Tags pertaining to persons leaving the main post

All INPERSON tags mentioned above can in principle also be used for the person leaving a position, which is marked by OUTPERSON. Below we give examples of the ones that are actually required in the first sentence. Although the frequencies of OUTPERSON tags in the first sentences tend to be low, many more examples can be found in the follow-up sentences where the articles tend to focus more on the outgoing person. However, this is outside the scope of this work. We first show some examples of tags that are a direct adaption of the ones for incoming persons:

(27) a. ..., succeeding [OUTPERSON_FULLNAME Erskin N, White Jr.], [OUTPERSON_AGE 65] years old, ...

b. ..., 51-year-old [OUTPERSON_OTHERPOST_NODET deputy chairman] of this ..., ... has retired ...

c. Christopher Whittington, ... chairman of [OUTPERSON_OTHERCOMP Morgan Grenfell & Co.], ... has retired ...

The OUTPERSON_FULLNAME tag has 23 occurrences in the corpus. Many tags that are possible following the system developed for the incoming person do not occur in the first sentences, for example OUTPERSON_PREVIOUSCOMP, OUTPERSON_PREVIOUSPOST-_NODET or OUTPERSON_OTHERPOST_INDEF. However, there are also tags that apply only to the outgoing person. For example, an outgoing person can assume a new post:

(28) ..., succeeding Delmont A. Davis, who was named [OUTPERSON_NEWPOST_NODET president] ... (f=2)

Both occurrences of OUTPERSON_NEWPOST_NODET refer to a person leaving the main post but who is assuming a new one. In principle, these persons could as well be tagged as INPERSONs. The chosen annotation reflects the focus of the article on the other person assuming the main post.

The person leaving the main post may continue in other positions in the same or another company:

(29) a. ... , who remains [OUTPERSON_POSTCONT_NODET chairman]. (f=1)

b. ... , who continues as a [OUTPERSON_POSTCONT_INDEF vice chairman] ... (f=1)

Verbs describing incoming events are not tagged. Thus, we expect the use of these verbs as part of the template always to be correct if there is an INCOMING_FULLNAME or another INCOMING tag. However, in the case of outgoing events the verb can express the reason for leaving a post. It needs to be tagged as well if we want to avoid tying the surrounding tags to the particular reasons for leaving a post in individual sentences.

The purpose or reason for leaving a post is marked by OUTPERSON_PRP_VP plus a time suffix where PRP stands for "purpose and reason" (following Santorini, 1990). A time suffix is needed to distinguish past tense, present tense and future tense verbs.

This is intended to prevent a combination of a past tense verb with a date in the future, for example:

(30) a. , who [OUTPERSON_PRP_VP_PAST resigned] to become president of ... (f=9)

b. ... [OUTPERSON_PRP_VP_PAST has retired] from his executive duties.

c. ... [OUTPERSON_PRP_VP_FUTURE will retire] next April as chairman ... (f=4)

d. ... [OUTPERSON_PRP_VP_FUTURE will take early retirement] from this steel-maker ...

e. ..., who [OUTPERSON_PRP_VP_PRESENT is retiring early]. (f=2)

In (30a), the reason for leaving the post is resignation. In the other cases, it is retirement. In (30b), we include 'his executive duties' in the generation template. We do this to give the generator greater expressibility. However, this implies that the chosen domain always deals with posts and persons at the executive level.

Strings marked by OUTPERSON_PRP_VP can include entire VPs but exclude all material that is covered by other tags. For example, in (30a) 'president' is a new post of the outgoing person and marked as OUTPERSON_NEWPOST_NODET.

4.2.4 Tags pertaining to the main company

The name of the main company is usually introduced in the headline and not explicitly mentioned again in the first sentence. Companies are frequently specified further by their field of activity, the kinds of products they produce etc.

The COMP_DESCR tag (f=121) marks descriptions of the main company or organization, including nouns such as 'company' or 'concern'. The marked string is an NP without determiner[2] but including premodifiers:

(31) a. The [COMP_DESCR closely held supermarket chain] named ...

[2]We regard determiners that precede COMP_DESCR as part of the template (see sections 4.2.7.1 and 4.2.7.2).

b. ... has been elected to this [COMP_DESCR telecommunications company]'s board.

c. ... of this [COMP_DESCR closely held publisher].

d. ... this [COMP_DESCR_RC company which primarily has interests in radio and televisions stations] ... (f=2)

The company description in (31d) requires a new tag because of the relative clause that prevents substitution into (31b), for example.

If an explicit nationality or location of a company is given, it is marked separately from COMP_DESCR by the COMP_NATIONALITY tag:

(32) of this [COMP_NATIONALITY British] [COMP_DESCR bank]. (f=5)

The location of the main company is tagged as COMP_LOC:

(33) ... of this [COMP_LOC New York] investment banking firm. (f=3)

Furthermore, there are descriptions and locations of INPERSON_OTHERCOMPs. In contrast to POST tags, company descriptions include preceding determiners:

(34) a. ... a partner of Alpha partners, [INPERSON_OTHERCOMP_DESCR a venture capital firm] based in ... (f=2)

b. ... based in [INPERSON_OTHERCOMP_LOC Menlo Park, Calif.], ... (f=6)

4.2.5 Tags pertaining to subsidiaries of the main company

The post assumed or left often is not with the main company mentioned in the headline but with a subsidiary. The proper name of the subsidiary, if given, is marked by COMP_SUBSIDIARY (f=22):

(35) a. ... was named president of the [COMP_SUBSIDIARY Atlantic Research Corp.] subsidiary.

b. ... of [COMP_SUBSIDIARY Balcor Co.], a Skokie, Ill., subsidiary of this ...

Note that nouns like 'unit' or 'subsidiary' are excluded. The description of the kind of subsidiary at which the main post is located is marked as COMP_SUBSIDIARY_DESCR. It may contain only a single word or a larger description. As for post-related tags, we encode definiteness in the tag but do not include any preceding determiner. The filler can be a single word:

(36) a. ..., the [COMP_SUBSIDIARY_DESCR_DEF investment management subsidiary] of ... (f=9)

b. ... was named president of the Clairol [COMP_SUBSIDIARY_DESCR_DEF division] of this ...

c. ... at this firm's Chemical Bank [COMP_SUBSIDIARY_DESCR_NODET unit] ... (f=9)

d. ... of Balcor Co., a Skokie, Ill., [COMP_SUBSIDIARY_DESCR_INDEF subsidiary] of this ... (f=5)

4.2.5.1 Some singleton tags

The descriptions of companies and their subsidiaries can be of considerable complexity. In many cases we can derive tags for them in a systematic way. However, since the tags tend to have single occurrences, they are of limited use: if they occur in the training set, we cannot test them; if they occur in the test set, we are not able to train on them. On the other hand, as we emphasized earlier, when annotating for NLG input it is important to mark all information that should not be assumed as always available. Thus, we will have to accept the existence of singleton tags in the corpus. In the remainder of this section, we show some examples of such singleton tags. (Their usefulness will be investigated in section 7.7.2).

Other posts of the incoming person can be located at a subsidiary of another company rather than the main company. The name of the subsidiary is marked as follows:

(37) Thomas H. Johnson, president of the [INPERSON_OTHERCOMP_SUBSIDIARY Coatedboard] division of Mead Corp., was named ...

In this case, the other position is held at a subsidiary of Mead Corp. which is labelled by OTHERCOMP. This is in contrast to more common cases where the other post is held at a subsidiary of the main company:

(38) William A. Wise, 44 years old, president of the [COMP_SUBSIDIARY El Paso Natural Gas Co.] unit of this ...

The distinction between a subsidiary of the main company and of another company resolves the problem that more than one tag seems to apply to (38) at a first glance: For the string 'El Paso Natural Gas Co.' both COMP_SUBSIDIARY and INPERSON_OTHERCOMP-_SUBSIDIARY seem to be appropriate. However, only COMP_SUBSIDIARY applies to subsidiaries of the main company.

A previous post might also have been located at a subsidiary of the main company:

(39) ... formerly vice chairman of Capital Holding Corp. and president of its [INPERSON-_PREVIOUSCOMP_SUBSIDIARY Accumulation Investment Group] ...

The main post can be located at a subsidiary of a subsidiary of the main post assumed to be present in the headline. Obviously, this is recursive and the appropriate tag can be assembled accordingly:

(40) ... was named chairman of [COMP_SUBSIDIARY_SUBSIDIARY County NatWest Investment Management Ltd.], the investment management subsidiary of County NatWest Ltd., the investment banking arm of this British bank.

Descriptions of subsidiaries of subsidiaries can be handled analogously to less embedded subsidiary descriptions.

4.2.6 Tags pertaining to temporal expressions

Dates of incoming and outgoing events are treated separately because in some cases a position opened by a leaving person is not filled immediately. Furthermore, we make a distinction between date specifications referring to future and to past succession events. The core of the tag name is constructed according to the following scheme:

{INDATE | OUTDATE}_{PAST | FUTURE}

The default slot filler is of category NP and the default person the event is referring to is the incoming person for INDATEs and the outgoing person for OUTDATEs. If the defaults do not hold, we attach appropriate extensions to the tag name.

A temporal expression indicating that a post is occupied at a future date is marked by INDATE_FUTURE (f=11). The marked string can include adverbs:

(41) a. ... will assume responsibility ..., effective [INDATE_FUTURE Nov. 1].

b. ... was named executive director, effective [INDATE_FUTURE early November].

c. ... will join the board as nonexecutive director [INDATE_FUTURE Nov. 29].

Dates only seem to be given if a decision about a management succession does not take effect immediately, i.e. the given date is different from the date of publishing the article. Most verbs are in past tense (like 41b) but some use future tense (41a,c). Therefore, this tag does not specify verb tense. In most cases, the date is preceded by 'effective' but there are exceptions as (41c) shows. There is no obvious regularity for the use of 'effective' in combination with specific verb tenses.

If only a month is given, we include the preposition 'in' in the marked string:

(42) ... will become chairman [INDATE_FUTURE in May] , ...

By including the preposition we avoid introducing a special purpose tag for months as fillers. Substitutions can yield surface strings like 'effective in May' which are grammatical but have not been seen in the corpus.

The date of an incoming event can also lie in the past. These dates seem to be mentioned in combination with outgoing events and refer to the outgoing person. The examples seen in the corpus specify months only:

(43) Kenneth J. Thygerson, who was named president of this thrift holding company [OUTPERSON_INDATE_PAST in August], resigned, ... (f=2)

The date of leaving a position is marked by OUTDATE. We distinguish between dates referring to time points in the past and those referring to time points in the future:

(44) a. ... will retire from his post .... effective [OUTDATE_FUTURE Dec. 31]. (f=4)

b. ... succeeding Thomas W. Field Jr., 55, who resigned [OUTDATE_PAST last month]. (f=3)

All three occurrences of OUTDATE_PAST are used in combination with a past tense verb and refer to the outgoing person.

In one case, the filler of a future outgoing event is a PP rather than an NP and thus requires its own tag to prevent unwanted substitutability:

(45) ... resigned as chairman of this diesel truck manufacturer, effective [OUTDATE-_FUTURE_PP upon appointment of a successor]. (f=1)

4.2.7 Referring expressions

Natural language texts contain a wealth of referring expressions. Although generating referring expressions is not the focus of this work, we need to deal with those referring expressions that occur in the collected corpus. Our general strategy is to see referring expressions as part of the template, i.e. to leave them unmarked. In other words, the generation input does not contain (instructions to use) referring expressions but the output does. This assumes that the generator uses the templates (which provide the referring expressions) only in the right context.

Since we are generating the first sentences of the *Who's News* articles of the *Wall Street Journal*, the only textual context available is the headline of the articles. As mentioned before, most headlines have not been preserved in the corpus. However, there are a few examples that indicate that the headlines always consist of a company name and a location. We can mark them accordingly (but we do not count them as 'first sentences' or use them as instances):

(46) a. [COMP INGERSOLL-RAND Co.] ([COMP_LOC Woodcliff-Lake, N.J.]) –

b. [COMP INTER-TEL Inc.] ([COMP_LOC Chandler, Ariz.]) –

Using generation templates that contain references to the main company assumes that such headlines have been generated first. In the following, we first show the referring expressions of the first sentences that refer to the main company introduced in the headline. This is the most frequent form of referring expressions. We then show a few other phenomena.

4.2.7.1 'this' (f=121)

In most cases, the demonstrative pronoun 'this' is used to refer to the main company, followed by the tag COMP_DESCR. The main company's nationality or location can precede the COMP_DESCR tag. We have already seen many examples of this use of the demonstrative pronoun (in 31 and 32, for example). A special case is the following:

(47) Clark J. Vitulli was named senior vice president and general manager of <u>this</u> U.S. sales and marketing arm of Japanese auto maker Mazda Motor Corp.

The company referred to in the headline is the subsidiary of a mother company which is specified further in the article. This is a change of perspective from the other articles which take the mother company as a reference point and then introduce subsidiaries in the article proper. However, in (47) 'this' still refers to the company in the headline. Thus, the example requires new tags to label the mother company of the company in the headline but 'this' still refers to the headline company:

(48) Clark J. Vitulli was named senior vice president and general manager of <u>this</u> [COMP_DESCR U.S. sales and marketing arm] of [COMP_MOTHER_NATIONALITY Japanese] [COMP_MOTHER_DESCR auto maker] [COMP_MOTHER Mazda Motor Corp].

4.2.7.2 'the' (f=3)

In a few cases, the definite article is used instead of the demonstrative pronoun:

(49) a. ... has been named regional president, a new post at <u>the</u> [COMP_DESCR bank-holding company].

b. ... was elected chief executive of <u>the</u> [COMP_DESCR holding company]'s two principal insurance subsidiaries.

(49a) seems to be a standard example. (49b) exhibits a possessive marker which is described below.

4.2.7.3 'its' (f=9)

The company in the headline can be referred to by 'its'. It seems that the company must have been mentioned in the left context of the first sentence already, typically by 'this `COMP_DESCR`':

(50) a. ... of <u>this transportation industry supplier</u>, increasing <u>its</u> board to six members.

b. ... of <u>the company</u> and <u>its</u> First National Bank of Toms River subsidiary.

There are two exceptions where 'its' does not refer to the headline company but to the one mentioned in the immediate left context. This confirms that the last company mentioned in left context is crucial for resolving the reference of 'its':

(51) a. ... , formerly vice chairman of <u>Capital Holding Corp.</u> and president of <u>its</u> Accumulation Investment Group, ...

b. ... was named chairman of this insurance firm's <u>reinsurance brokerage group</u> and <u>its</u> major unit, ...

In (51a), 'its' refers to a `INPERSON_PREVIOUSCOMP`, in (51b) it refers to a string marked as `COMP_SUBSIDIARY_DESCR_NODET`. However, these references should not pose a problem in a fixed template since the antecedent needs to be present.

4.2.7.4 Possessive marker *'s* (f=15)

All occurrences of the possessive marker *'s* refer to a directly preceding `COMP_DESCR` tag, and therefore to the main company:

(52) a. ... for this `[COMP_DESCR` financial and travel services concern`]` <u>'s</u> American Express Bank Ltd. subsidiary.

b. ... this `[COMP_DESCR` telecommunications company`]` <u>'s</u> board ...

In one case, the possessive marker occurs inside a slot filler. This does not seem problematic as long as the 'owner' is also part of the slot filler:

(53) ... [INPERSON_OTHERCOMP Harvard University 's Graduate School of Business] ...

4.2.7.5 References to persons

There are only two personal pronouns used throughout the corpus: 'he' (f=1) and 'his' (f=3). Obviously, their use requires an appropriate antecedent in the left context. In all cases, the antecedent is an OUTPERSON_FULLNAME. Two examples:

(54) a. .... announced that he [OUTPERSON_PRP_VP_FUTURE will retire] [OUTDATE-_FUTURE next April] as .. .

b. ... [OUTPERSON_PRP_VP_PAST has retired from his executive duties].

In (54a), the personal pronoun is part of the surface string outside tagged areas. It therefore becomes part of the domain-specific pattern of the template which records that persons in this domain are predominately male. In (54b), the possessive pronoun is part of the slot filler OUTPERSON_PRP_VP_PAST. All uses of this tag, however, follow after the outgoing person has been introduced in the discourse.

4.3 Tag indices

The annotation scheme we have developed up to this point does not account for cases where a distinction between different tags of the same type is necessary. Typical cases involve several previous posts that are located at different companies and multiple occurrences of tags when there is more than one incoming person:

(55) a. John W. Day, president of Allied-Signal Automotive and executive vice president of Allied-Signal Inc., has been named ...

b. William G. Kuhns, former chairman and chief executive officer of General Public Utilities Corp., was elected ...

c. ... was named president and chief operating officer, posts which had been vacant.

d. ... was named executive vice president and chief operating officer, both newly created posts, and a director, filling a vacancy.

e. Sheldon B. Lubar, chairman of Lubar & Co., and John L. Murray, chairman of Universal Foods Corp., were elected to the board of this engine maker.

f. Anton Amon and George Gourlay were elected vice presidents of this soft-drink company.

The examples in (55) exhibit considerable complexity. They involve phenomena like coordination, plurals and anaphora that are long-standing topics of linguistic research. In this work, we do not aim at providing a deep analysis of these phenomena. However, we need to be able to represent complex semantic relationships at the level of the annotation if we want to be able to correctly fill in the gaps in the sentence templates.

A first attempt to represent relationships between tags might be to simply add counts to the tag names to distinguish them, for example, `INCOMING_AGE_1` and `IN-PERSON_FULLNAME_1`. However, this would introduce large numbers of new tag names, changing the frequencies of tags and resulting in changes in the ranking model. For example, many succession events involve several posts. The tags `POST_1`, `POST_2` and `POST_3` would have very different frequencies, with `POST_3` being much less frequent than `POST_1`. However, there do not seem to be fundamental differences between corpus sentences that report on persons assuming one, two or three new posts. Moreover, counts in tag names could result in sentences that only contain `POST_3` tags without any `POST_1` or `POST_2` tag if not all parts of the input semantics are expressed. If the numbers indeed reflect counts, the question is whether this is appropriate.

Even without these problems, it is questionable whether counts can provide the means to represent the kind of relationship between tags that we want to express. It seems likely that counts, if introduced from the beginning of each sentence – the first occurrence of a certain tag gets the suffix `_1`, the second gets `_2` etc. – lead to inconsistencies in the annotation across different sentences. For example, a `POST_3` might be the newly occupied post of INPERSON_FULLNAME_1 in one sentence and of INPERSON_FULLNAME_2 in the other.

Therefore, we need to have a more general way of expressing the fact that two tags are related, regardless of their order of occurrence in specific corpus sentences, and without modifying the tag names. For this reason, we introduce the notion of *tag-indices*, an additional means of representation that allows one to attach indices to *pairs* of tags. Tag-indices are not interpreted as being part of the tag name and are not used in the term representation for ranking. For notational convenience, we nonetheless attach the indices at the end of the tags (separated from the tag name by '-'). Tags can carry several indices (separated by '+') because we want to be able to link a tag to several others while using indices only pairwise, i.e. there are always two occurrences of each index in any annotated sentence. Tag-indices act as variables whose material names are not important. No directionality is implied between the two occurrences of an index. As a general rule, tag-indices need to be applied whenever a tag name occurs more than once (ignoring suffixes for determiners). The sentences in (55) are annotated with indices in the following way:

(56) a. [INPERSON_FULLNAME John W. Day], [INPERSON_OTHERPOST_NODET-P1 president] of [INPERSON_OTHERCOMP-P1 Allied-Signal Automotive] and [INPERSON_OTHERPOST_NODET-P2 executive vice president] of [INPERSON_OTHERCOMP-P2 Allied-Signal Inc.], has been named ...

b. [INPERSON_FULLNAME William G. Kuhns], former [INPERSON_PREVIOUSPOST_NODET-P1 chairman] and [INPERSON_PREVIOUSPOST_NODET-P2 chief executive officer] of [INPERSON_PREVIOUSCOMP-P1+P2 General Public Utilities Corp.], was elected ...

c. ... was named [POST_NODET-P1 president] and [POST_NODET-P2 chief operating officer], posts which had been [POST_DESCR_ADJ-P1+P2 vacant].

d. ... was named [POST_NODET-P1 executive vice president] and [POST_NODET-P2 chief operating officer], both [POST_DESCR_ADJ-P1+P2 newly created] posts, and a [POST_INDEF-P3 director], filling a [POST_DESCR_VACANCY-P3 vacancy].

e. [INPERSON_FULLNAME-P2+P5 Sheldon B. Lubar], [INPERSON_OTHERPOST-_NODET-P1+P2 chairman] of [INPERSON_OTHERCOMP-P1 Lubar & Co.], and [INPERSON_FULLNAME-P4+P6 John L. Murray], [INPERSON_OTHERPOST_NODET--P3+P4 chairman] of [INPERSON_OTHERCOMP-P3 Universal Foods Corp.], were elected to the [POST_BOARD-P5+P6 board] of this [COMP_DESCR engine maker].

f. [INPERSON_FULLNAME-P1 Anton Amon] and [INPERSON_FULLNAME-P2 George Gourlay] were elected [POST_PLURAL-P1+P2 vice presidents] of this [COMP-_DESCR soft-drink company].

In (56a) indices are used to indicate pairs of OTHERPOSTs and OTHERCOMPs. In (56b) indices represent that both PREVIOUSPOSTs refer to the same PREVIOUSCOMP. Without indices, the first PREVIOUSPOST could be interpreted as referring to the main company in the headline. However, this is not the meaning we want to represent. In (56c) indices represent that POST_DESCR_ADJ applies to both POST tags. In (56d) they distinguish the new posts from the vacant one. The sentence uses the word 'both' explicitly. It should be noted that as long as we use the sentence as a fixed template, it is guaranteed that there are indeed two POSTs to which the POST_DESCR_ADJ tag refers. POST_DESCR_VACANCY is a special-purpose tag which seems to be necessary because of particular uses of the slot filler in constructions like 'filling a _'. There are five occurrences of this tag, justifying its introduction. The need for indices is also obvious in (56e). First, there are two persons with their respective OTHERPOSTs and OTHERCOMPs. Second, both persons were elected to the same board. We therefore also need to co-index the incoming persons with the post. In (56f) we use a POST_PLURAL tag in combination with co-indexing to represent that the slot filler of POST_PLURAL refers to two incoming persons.

There are a few cases in which we do not use indices although a tag occurs repeatedly. For example:

(57) Robert D. Paster, 49 years old, [INPERSON_OTHERPOST vice president] and [IN-PERSON_OTHERPOST Space Shuttle Main Engine program manager], was named ...

Since no OTHERCOMP is mentioned in the sentence, the INPERSON_OTHERPOST tags refer to the main company which is mentioned in the headline. However, in this work we limit ourselves to single sentence generation and do not add any indices in these cases.

Generally, we try to limit the use of indices. However, in a number of cases they seem to be necessary. In the corpus of 144 sentences, there are 25 articles that require the use of indices. Many have only one or two pairs of indices. In a few cases, up to 9 pairs of indices are required. In general, there seems to be a trade-off between the information encoded in tag names and in tag-indices. The more information is encoded in indices, the less complex the tags need to be. For example, one could use a single AGE tag instead of differentiating between INPERSON_AGE and OUTPERSON_AGE. The AGE tag would then always need to be co-indexed with an incoming or outgoing person tag. This would have consequences for the ranker because of changed term weights and a different distribution of tags. It would also place a bigger burden on the mechanism that checks tag-indices, which is described in section 5.5.

4.4 Problematic cases

In order to annotate generation templates, we mark all strings that should not be generated without being triggered by a particular generation input. However, there is information in the corpus sentences that seems difficult to annotate systematically:

(58) a. James T. Boone was named to the new position of chief administrative officer, [? <u>becoming second in command</u>] at this bank holding company.

b. Frank Carlucci III was named to this telecommunications company 's board, filling the vacancy [? <u>created by the death</u>] of William Sobey last May.

c. ... was named executive vice president, North America, for the Financial Times, the business newspaper [? <u>published by</u>] this ...

d. ... was named vice president and senior officer in charge of equipment leasing to municipalities, [? <u>a new effort of</u>] this bond insurer.

e. J.P. Bolduc , vice chairman of W.R. Grace and Co. , [? which holds a 83.4 % interest in] this energy-services company , was elected a director.

f. This magazine and book publisher said [? three men] were elected directors, increasing the board to 10.

g. This maker of electronic measuring devices named two new directors, increasing board membership to nine.

The ? tag marks rather arbitrary pieces of text that in most cases are not semantically or syntactically interchangeable. Information marked by ? should not be present in the generation input. There are seven such cases in the corpus of 144 sentences. As Reiter and Dale (2000) observe, there tend to be textual fragments in corpora that present data that is unavailable to the generation system. In other words, it is not always possible to fully regenerate the target language corpus. This may be due to missing input specifications and/or to knowledge gaps in the grammar. (In section 7.2.0.2.3, we discuss how ? tags affect generation inputs and instance representations in the evaluation experiments.)

We use the ? tag in (58) because it seems difficult to find a systematic annotation scheme for these sentences. There are other cases in which we can derive tags following our general scheme although the information also seems to be rather arbitrary at a first glance. For example:

(59) a. Richard W. Lock, [INPERSON_DESCR_JJ retired] vice president and treasurer of Owens-Illinois Inc., ...

b. [INPERSON_DESCR_JJ Retired] Adm. William J. Crowe, former chairman of the Joint Chiefs of Staff, ...

There are three occurrences of the INPERSON_DESCR_JJ tag in the corpus. This seems to justify the use of this tag although they all occur with the same filler.

4.4.1 Announcements

In some cases, the management succession is reported as being explicitly announced by the main company or a person. One option is to tag the corresponding verb in a way similar to the treatment of verbs like 'retired' (see tag `OUTPERSON_PRP_VP_PAST` and related above):

(60) a. This magazine and book publisher `[ANNOUNCE_COMP` said`]` three men were elected directors ...

b. Manhattan National Corp. `[ANNOUNCE_COMP` said`]` Michael A. Conway ...

c. A.F. Sloan, 60 years old, `[ANNOUNCE_OUTPERSON` announced`]` that he will retire ...

There are five such cases in the corpus. However, one could also leave the verb unmarked, assuming that an announcement of some form always takes place. The grammar would then effectively have the ability to use the announcement construction without being explicitly told to do so. It amounts to saying that we do not consider announcements as particularly meaningful. Rather, they are seen as a variation of expressing the marked pieces of information. This is the strategy used in this markup scheme. Therefore, we do not use any `ANNOUNCE` tag.

4.4.2 Punctuation

Punctuation is an important aspect of dealing with real-world texts. With respect to NLG, wrongly generating punctuation can seriously hamper the readability or even faithfulness of the generation output. Marking punctuation is made difficult because punctuation can serve rôles inside and outside the marked string at the same time. In the following examples, the last comma can either be clearly outside the tagged area (61a), clearly inside (61b) or ambiguous (61c):

(61) a. ... was named `[POST_NODET` group head investment banking in Asia`]`, based in ...

b. ... was elected `[POST_NODET` senior vice president, public affairs and advertising,`]` for this ...

c. ... was named `[POST_NODET` senior vice president, industrial systems`]`, succeeding ...

In (61a) the comma should be part of the template because it separates the marked NP from the following VP. (Both are at the same level according to the treebank analysis.) In contrast, in (61b) the comma does not have any syntactic rôle outside the marked area. However, if we replace the filler words of sentence (61a) by the ones of (61b), we end up with two commas in a row. To make matters worse, whatever possibility is taken with respect to (61c), there will be cases in which a comma is missing: excluding the comma from the marked string of (61c) results in a missing comma when the filler of (61c) is inserted into (61b); including the comma results in a missing comma when the filler of (61a) is inserted into template (61c).

When annotating such cases, we opted for excluding commas that are syntactically relevant to the surrounding context of the template (as shown in 61c). This should make sure that no template is lacking a comma that is required syntactically. It is easily possible to remove duplicate commas. However, this still leaves a problem when inserting the filler words of (61c) into (61b).

The problem also extends to full stops. For example, company names often carry the abbreviated form 'Corp.'. If 'Corp.' occurs as the last word of the sentence, its full stop serves two rôles. Similar problems as for commas can therefore arise in such cases.

Ideally, one should distinguish between template markup and generation input markup and include a comma in both, provided duplicate commas are removed during generation. We do not have this markup option here but we manually make sure that the generation input is not lacking any commas. Therefore, the output of the system needs to be polished by removing duplicate punctuation. We use the following mappings:

(62) a. $, , \rightarrow ,$

b. $, . \rightarrow .$

c. $. . \rightarrow .$

These rules are similar in spirit to the rules of 'absorption' (Nunberg, 1990) which delete 'weaker' punctuation marks when adjacent to 'stronger' ones.

4.4.3 Referring expressions

We typically deal with references to the main company by applying a COMP_DESCR tag (see 31). This assumes that there is one such input fact that needs to be expressed. However, there can also be repeated references to the main company:

(63) Christopher Whittington, 51-year-old deputy chairman of this British [COMP_DESCR investment-banking group] and chairman of Morgan Grenfell & Co., the group 's main banking unit, has retired from his executive duties.

The second reference of the group adds no further information to the sentence but rather seems to have the purpose of providing a 'link' from mother company to subsidiary. On the one hand, we would like to leave this second reference unmarked in order to avoid giving the realizer input that is too detailed. On other hand, not all companies can be referred to as 'groups'. Consequently, the second mention of 'group' needs to be marked as well. We do this by simply using another COMP_DESCR tag. This is in line with other mentions of the main company that are used in combination with the possessive marker and that do not contribute much new information:

(64) Roy E. Parrott, the [COMP_DESCR company] 's president and chief operating officer since Sept. 1, was named to its board.

Referring expressions can also be part of the slot filler. In the following example, a reference to the 'parent' company is expressed:

(65) William A. Wise, 44 years old, president of the [INPERSON_OTHERCOMP_SUBSIDIARY El Paso Natural Gas Co.] unit of this energy and natural-resources concern, was named to the additional post of chief executive officer, succeeding Travis H. Petty, 61, who continues as a [OUTPERSON_POSTCONT_INDEF vice chairman of the parent].

In contrast to (53), the slot filler of (65) refers to an antecedent outside the slot filler. We interpret 'the parent' as a reference to the main company in the headline because a subsidiary was mentioned in the left context of 'the parent'. However, rather than introducing another special purpose tag, we assume that generally there are no referring expressions in the filler (although this may cause problems when inserting this particular filler into other templates).

freq.	tag name
142	POST_NODET
139	INPERSON_FULLNAME
121	COMP_DESCR
59	INPERSON_OTHERPOST_NODET
56	INPERSON_AGE
36	POST_INDEF
33	INPERSON_OTHERCOMP
25	POST_DESCR_ADJ
25	INPERSON_PREVIOUSPOST_NODET
23	OUTPERSON_FULLNAME
22	COMP_SUBSIDIARY
14	BOARD_INCR
13	COMP
11	INDATE_FUTURE
10	POST_BOARD
10	OUTPERSON_AGE
10	INPERSON_PREVIOUSCOMP
9	OUTPERSON_PRP_VP_PAST
9	INPERSON_OTHERPOST_INDEF
9	COMP_SUBSIDIARY_DESCR_NODET

Figure 4.2: The most frequent tags in the collected corpus

4.5 A look at the annotated corpus

71 different tags are used to annotate the corpus of 144 sentences. There are relatively few high-frequency tags and relatively many low-frequency ones. The most frequent tag occurs 142 times and there are 23 singletons. Such a distribution is typical of many language phenomena. Figure 4.2 shows a list of the 20 most frequent tags.

The annotated sentences exhibit a dense markup (see 56, for example). As pointed out above, this is necessary for natural language generation since we need to make sure that no pieces of information that should not be part of the template remain. However, there are still unmarked sequences outside tagged areas. The largest one contains 9 tokens (words and punctuation). Figure 4.3 shows the largest template sequences, including their length and corpus frequency. Note, for example, the different ways of expressing an increase in the size of a company's board (many of which happen to be of the same length). The sequences in figure 4.3 obviously cross constituent boundaries and in this respect are similar to word ngrams.

length	freq.	template sequence
9	1	years old , will join the board as a
8	1	years old , has been elected to the
7	1	, increasing the number of seats to
6	3	years old , was elected a
6	2	years old , was named a
6	1	, has been named to the
6	1	years old , announced that he
5	13	years old , was named
5	3	years old , was elected
5	1	of the company and its
5	1	from his executive duties .
5	1	, will assume responsibility for
5	1	, who continues as a
5	1	, were elected to the
5	7	, was named to the
5	1	, was named to its
5	9	was elected to the
5	1	, posts which had been
5	3	, increasing the board to
5	1	, increasing its size to
5	1	, increasing its board to
5	4	, increasing board membership to
5	4	, expanding the board to
5	1	, boosting the board to
4	3	years old , formerly

Figure 4.3: The largest word sequences of the generation templates

We also measured the annotation stability by re-annotating 35 corpus sentences, using an annotated corpus of the remaining sentences as annotation examples. The two annotation files were compared by a script. Overall, about 93% of the tags were placed correctly, i.e. the same tag is used to mark exactly the same area. 24 out of 35 sentences were entirely correct, the remaining ones contained at least one error. Two errors were due to different areas of non-taggables (`[? ...]`) and one to the inclusion of punctuation. These seem to be typical points of confusion. Furthermore, in two cases either the 'gold standard' or the newly annotated sentences did not separate the location of a company from its name. Another error was not to include the preposition 'in' preceding months into the area marked by a date tag. Furthermore, in one case a slightly different singleton tag name was newly constructed. Another error was due to a lack of background knowledge:

(66) Selwyn B. Kossuth was named executive director of the commission, effective early November.

Without knowing the headline (and understanding the basics of how companies are organized), it is difficult in some cases to categorize fragments correctly. In this case, we can simply mark the entire sequence 'executive director of the commission' as a `POST_NODET` to avoid the problem.

4.6 Discussion

In this chapter, we described a declarative annotation scheme for the chosen domain corpus. It is based on the principle that equivalence classes for marked strings have two properties:

1. semantic equivalence of all class members and
2. syntactic equivalence of all class members.

From a more philosophical point of view, forming such equivalence classes can be seen as an idealization. In the words of Bloomfield (1933, p145): "If the forms are phonemically different, we suppose that their meanings are also different [...]" In the extreme, each word or sequence of words needs to form its own equivalence class. In contrast, we regard all strings marked by a tag as interchangeable although individual strings might in fact have subtle consequences for the occurrence of other words in the context. Furthermore, we treat many words that are part of the template as semantically equivalent, for example the various verbs expressing the main succession event.

An important factor in allowing us to use such equivalence classes is the availability of a ranker and the fact that the corpus and the task are domain-specific. The annotation scheme addresses the question what can be seen as semantically and syntactically equivalent for the purpose of this specific application. By abstracting away from particular filler words, the tags allow the instance-based ranker to capture certain kinds of cooccurrences even though the amount of training data is limited. Furthermore, the use of tags allows one to define generation templates which can be used with different slot fillers. The relatively simple semantic annotation of the corpus sentences can be used as generation input, reducing the complexity of the input representations. However, these simple semantic tags also need to trigger the use of relatively complex templates. This in turn requires one to assume that different templates are expressing the same semantics although they are using different ways of expressing it. Still, if there is a subtle cooccurrence of certain verbs with certain combinations of tags, this is recorded by the template.

A further consideration is the guidance we give to the realizer by providing input in the form of corpus markup. (28) has shown that the markup can represent a focus on a particular person when more than one person is mentioned. (48) changes the view on a company structure from the perspective of the mother company to that of a subsidiary. Furthermore, a POST_PLURAL tag as in (56f) embodies a decision to 'summarize' several identical posts by one mention of the post name. As examples like (63) have shown, referring expression generation is clearly outside the scope of a generator that expects corpus markup of the kind developed in this work as its input. In addition, definiteness is encoded in several tag names. It is not the goal of this work to go beyond these generation decisions. However, one should be aware that using our corpus markup as generation input implies that certain generation tasks have already been performed.

In this chapter, we assumed that we are marking the corpus to obtain fixed generation templates. In the next chapter, we show how we can introduce more flexibility by constructing grammar rules that are derived by means of the syntactic treebank.

Chapter 5

Automatic grammar construction

In this chapter, we show how a standard syntactic treebank - in combination with the semantic annotation of the unanalysed corpus described in the previous chapter - can be employed to automatically produce a generation grammar. The overall goal is to enable the developer of the NLG system to concentrate on the semantics of the system by providing the semantic annotation scheme and to concentrate less on syntactic structures, grammar rules and the organization of grammatical resources. This effectively changes the way the NLG system is being developed. In the following, we describe some characteristics of the syntactic treebank and present the algorithm for grammar construction, followed by an examination of the constructed rule set.

5.1 Syntactic treebank

We use a treebank based on constituency in the style of the Penn treebank (Marcus et al., 1993, 1994). The representational framework of the Penn treebank can be described as a "relatively impoverished flat context-free notation" (Marcus et al., 1993, p.321). It is extended by a variety of null-elements to represent trace positions of 'moved' *wh*-constituents and 'understood' subjects of infinitive and imperative verbs, amongst others. As will be seen below, we do not exploit the presence of null-elements in the Penn treebank. Furthermore, our grammar construction procedure does not require the identification of complements and adjuncts, which is not provided by the treebank (although different authors have developed heuristics to identify these, e.g. (Collins, 1999, p174)).

An example of the flat bracketing structure (taken from the compiled subcorpus of the Penn treebank II comprising texts in the management succession domain) is given in figure 5.1. Note the passive trace `*-1` which is co-indexed with the surface subject (i.e. the logical object). The tag `CLR` (`CLosely Related`) attached to the PP is an "experiment" (Marcus et al., 1994) to indicate a strong connection between PP and VP.

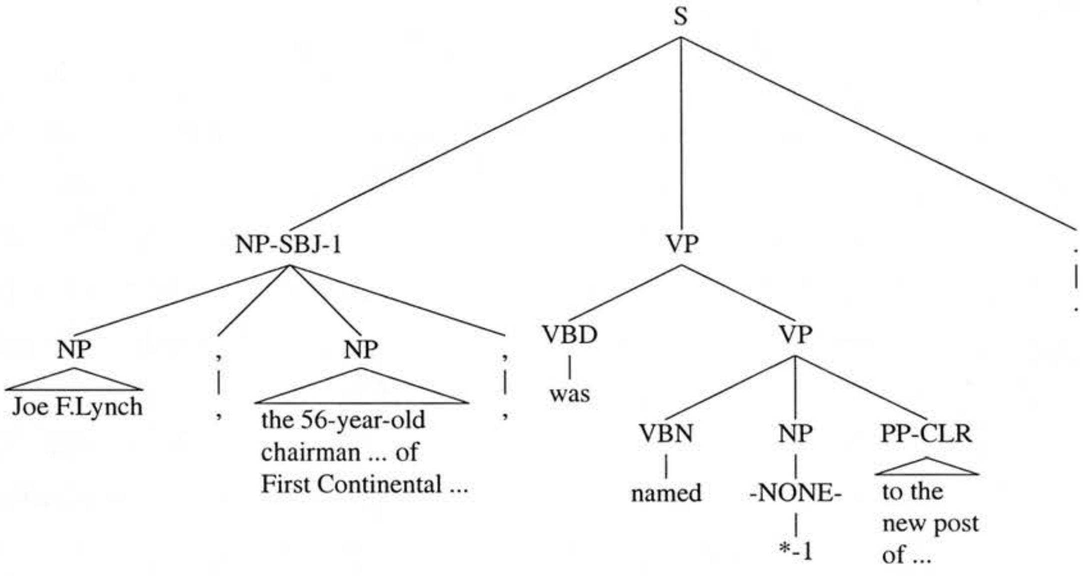

Figure 5.1: Flat tree structure (wsj_0366)

It is possible to extract a context-free grammar from the treebank parses by a recursive bottom-up procedure. As before, we are only dealing with one sentence per article. Figure 5.2 shows some of the rules that can be extracted from the tree structure in 5.1. It should be noted that such a simple grammar does not provide meaningful rules for the use of empty elements. (5.4% of lexical items are 'empty elements' in the treebank parses of the collected domain corpus.)

Figure 5.3 shows the growth of the set of different context-free rules extracted from the treebank parses of the corpus sentences in relation to the number of articles. This number seems too small to decide the question whether there is a finite grammar for the

S	$\rightarrow$	NP-SBJ-1 VP .
NP-SBJ-1	$\rightarrow$	NP , NP ,
NP	$\rightarrow$	NNP NNP NNP
NNP	$\rightarrow$	Joe
NNP	$\rightarrow$	F.
NNP	$\rightarrow$	Lynch
VP	$\rightarrow$	VBD VP
VBD	$\rightarrow$	was
VP	$\rightarrow$	VBN NP PP-CLR
...		

Figure 5.2: Some cfg rules for wsj_0366

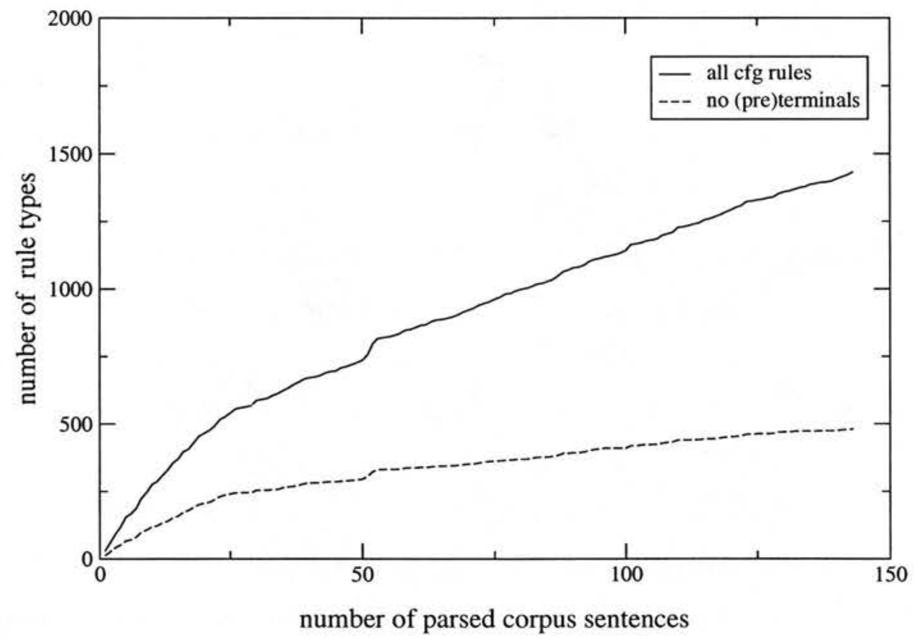

Figure 5.3: Cfg rule set growth

chosen domain, i.e. whether the number of rules stays stable at some point. However, when lexical items, i.e. rules of the form $preterminal \rightarrow terminal$, are excluded, rule set growth slows down at a faster rate than when lexical items are included.

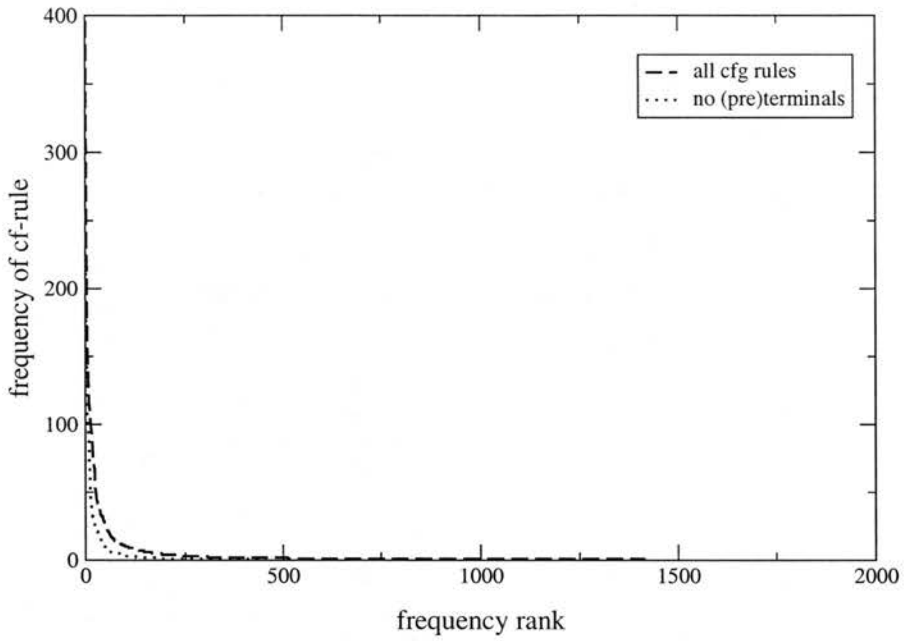

Figure 5.4: Cfg rule ranks vs frequency

A further characteristic of the extracted rule set is its 'Zipfian' distribution[1] which is typical for many language phenomena: there is only a small set of high frequency rules and a large set of low frequency ones (figure 5.4).

We can determine the 'coverage' of the extracted grammar by setting aside a number of sentences as held-out data and measuring the proportion of cf-rules in the held-out data that can be found in the training data. (Note, however, that this measure does not address the harder task of generating the held-out sentences.) Extracting cf-rules from 75% of the first sentences in the corpus (108 sentences) covers 85% of the rule tokens in the remaining 25% (36 sentences), for example. This number increases to 93% when lexical items are excluded and is smaller if rule types rather than tokens are counted. We explain this by the presence of a few high frequency rules (figure 5.4).

[1]Zipf's law states that the frequency of a word is (roughly) proportional to the reciprocal of its rank in a frequency list (Zipf, 1949).

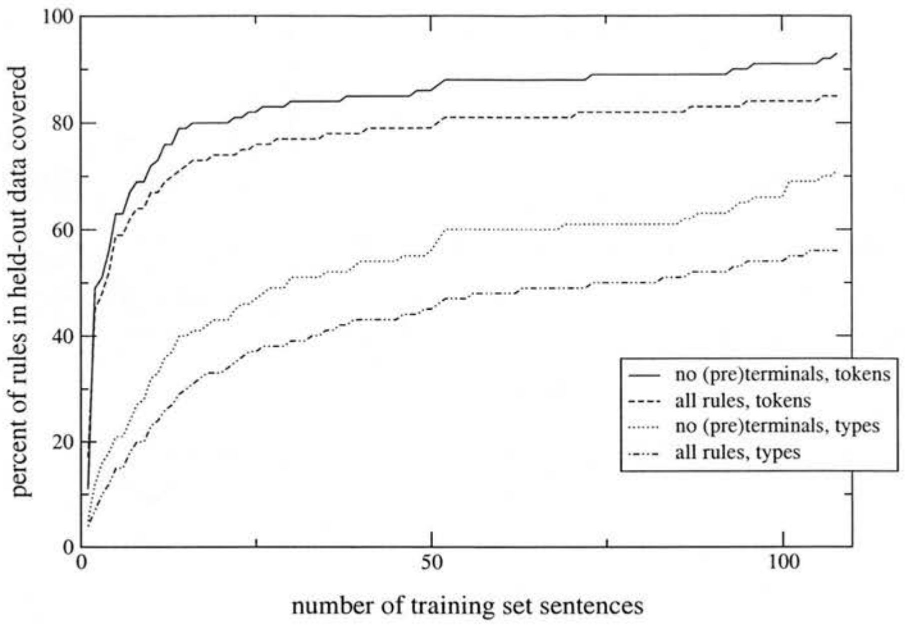

Figure 5.5: Coverage of treebank grammar w.r.t. held-out data

They are covered by even a small training set and account for a large proportion of rule tokens. Counting rule types rather than tokens decreases the relative weight of these high frequency rules. Figure 5.5 shows the increase in coverage of the rules in 25% held-out data as the number of training sentences for the different options grows. (The 'distance' between all-rules and non-(pre)terminals for either types or tokens seems to be fairly constant. This might be due to the fact that uncovered rare lexical items largely remain uncovered as the training set is enlarged.)

5.2 Merging treebank and annotation

A context-free grammar which is purely based on syntactic structures like the one just described cannot directly be used for generation. At the very least, a grammar for generation needs to be able to accept some semantic input and map this input to its grammatical resources.

We assume that the input is presented to the generator in the form of a set of attribute-value pairs according to the annotation scheme. As described in the previous chapter, the semantic annotation is applied to the raw, unanalysed corpus. Therefore, we are looking for a way of combining the annotation of the corpus sentences with the corresponding treebank structures since these are the only grammatical resources available to us. (An alternative is to apply the markup directly to the treebank, provided appropriate, possibly XML-based annotation tools.)

The annotation for the sentence corresponding to the parse tree in figure (5.1) is the following:

(67) [INPERSON_FULLNAME Joe F. Lynch], the [INPERSON_AGE 56]-year-old [INPERSON-_OTHERPOST_DEF-P1 chairman] and [INPERSON_OTHERPOST_DEF-P2 chief executive officer] of [INPERSON_OTHERCOMP-P1+P2 First Continental Real Estate Investment Trust], was named to the [POST_DESCR_ADJ new] post of [POST_NODET vice chairman] of this [COMP_DESCR bank holding company].

There are some characteristics of the annotation scheme that are relevant in this context. Most importantly, tags do not cross the bracketing structure of the treebank parses. In our experience, this property seems to fall out naturally under the assumption that relevant information should always be tagged as 'narrowly' as possible. Furthermore, it is possible for a tag to mark a word sequence within a constituent or just a part of a treebank word.

We set out by just applying the annotation tags to the words or word sequences of the parse tree. Figure 5.6 shows the result of merging the treebank parse (figure 5.1) with the annotation (67). Tags always mark pairs of terminal and preterminal nodes (and also any constituent structure that spans the terminal node sequence). For now, we ignore the treatment of tag-indices. The marked areas are surrounded by square brackets whereas the Penn treebank structure is displayed using its original round brackets. In some cases, a marked area corresponds to a full treebank constituent, for example the NP 'Joe F. Lynch' in line 3 or the NP 'vice chairman' in line 23. In the case of '56-year-old', pos-tagged in the treebank as a single adjective, the cardinal number has been separated and the appropriate pos-tag introduced (line 6). The result is a marked area that comprises only some part of the NP at the next level. The same is true for 'new' (line 20) and 'bank holding company' (line 25).

```
(S
  (NP-SBJ-1
    (NP [INPERSON_FULLNAME (NNP Joe) (NNP F.) (NNP Lynch) ] )
    (, ,)
    (NP
      (NP (DT the) [INPERSON_AGE (CD 56) ] (JJ -year-old)
        (NX
          (NX [INPERSON_OTHERPOST_DEF (NN chairman) ] )
          (CC and)
          (NX [INPERSON_OTHERPOST_DEF (JJ chief) (JJ executive) (NN officer) ] )))
      (PP (IN of)
        (NP [INPERSON_OTHERCOMP (NNP First) (NNP Continental)
                                (NNP Real) (NNP Estate) (NNP Investment) (NNP Trust)] )))
    (, ,) )
  (VP (VBD was)
    (VP (VBN named)
      (NP (-NONE- *-1) )
      (PP-CLR (TO to)
        (NP
          (NP (DT the) [POST_DESCR_ADJ (JJ new) ] (NN post) )
          (PP (IN of)
            (NP
              (NP [POST_NODET (NN vice) (NN chairman) ] )
              (PP (IN of)
                (NP (DT this) [COMP_DESCR (NN bank) (VBG holding) (NN company)] ))))))))
  (. .) )
```

Figure 5.6: Annotation applied to treebank (pre)terminals

5.3 Preparing the treebank structures for generation rule construction

As we have argued above, the annotated corpus sentences can be used as fixed templates for generation without any syntactic analysis. After merging with the treebank structure, such a template also specifies its syntactic structure. However, this in itself does not alleviate the main problem of templates, their limited flexibility. We need to find a way to introduce more flexibility and we aim at making use of the syntactic structure to do so.

To start with, we can reason about what should happen when individual tags are given to the generator. In the case of annotation tags spanning an entire constituent, we could generate just that constituent. This could result in rules like the following:

$$(68)\quad \begin{bmatrix} \text{SEM} & \{\text{IN\_NAME}(\boxed{1})\} \\ \text{SYNCAT} & \text{NP} \\ \text{PHON} & <\boxed{1}> \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SEM} & \text{IN\_NAME}(\boxed{1}) \end{bmatrix}$$

This rule corresponds to line 3 in figure 5.6. Tag names are abbreviated. We use a feature structure notation for clarity and assume a sign-based approach in that grammatical structures are characterized by a phonological string (`phon`), a syntactic category (`syncat`) and a semantic representation (`sem`).[2] However, we do not want to imply any particular grammatical theory, nor have we specified yet in which way grammatical structures are to be combined. The feature structure notation should not imply a unification-based approach, for example. Rule (68) expresses that if we have a tag `INPERSON_FULLNAME` with a slot filler stored in variable $\boxed{1}$, we can produce an NP with that tag and slot filler as its semantics in which the slot filler doubles as the surface string.

Associating a syntactic category with an annotation tag that does not mark a full constituent in the treebank parse is somewhat more difficult. As a heuristic, we take as the syntactic category of the rule left-hand side the preterminal symbol of the rightmost element of the marked sequence. However, these category assignments are only provisional and serve as a starting point for the following steps. The key question is how to

[2]The semantics of the NP is represented in a set notation. This is explained in section 5.4.

divide the entire treebank parse into substructures in such a way that the complete tree can be reconstructed when the corresponding semantic annotation is given to the generator. We therefore need to answer the question for which unmarked surface words a specific tag is 'responsible', and then cut the tree structure accordingly. We will call the surface words marked by a tag the 'marked/tagged string/area'. In unambiguous cases we will sometimes just speak of the 'tag' instead. The 'assigned string/area' will refer to the surface words that are associated with some tag but that are outside marked areas.

5.3.1 Input rules

As a first step for assigning unmarked surface words to rule areas, we extend assigned areas to the left and right of the tagged areas. This implies that tags always mark continuous areas of the surface string.

With respect to the tree in figure 5.6, the most pressing question is how to handle the largest unmarked substring, ',was named to the'. The neighbouring tags are `INPERSON_OTHERCOMP` to the left and `POST_DESCR_ADJ` to the right. If we want to assign the unmarked words to tags in accordance with the syntactic bracketing, we need to address the question how conflicts between two expanding assigned areas are handled. The answer lies in the tree structure itself: as a first step, we identify maximal nodes that dominate exactly one tag, avoiding conflicts up to this point. For example, `COMP_DESCR` is dominated by the NP node (figure 5.6, line 25) as well as the PP node (line 24). The latter is maximal since the next-level NP (line 22) also dominates `POST_NODET`. In some cases, the maximal non-conflicting node might just be the preterminal of a single word or the rightmost preterminal of a sequence of words. The maximal node dominating `INPERSON_AGE` (line 6) while not dominating any other tag is the preterminal CD.

Determining maximal nodes dominating single tags allows us to produce grammar rules that generate larger strings than before. We will call these rules *input rules* since they directly apply to input tags. There are always as many input rules for a treebank parse as there are tags in its semantic annotation. Following the same notation as above, we give some of the 8 input rules for the example parse in figure (5.6):

(69) a. $$\begin{bmatrix} \textsc{sem} & \{\text{COMP\_DESCR}(\boxed{1})\} \\ \textsc{syncat} & \text{PP} \\ \textsc{phon} & <\text{of this } \boxed{1}> \end{bmatrix} \longrightarrow \begin{bmatrix} \textsc{sem} & \text{COMP\_DESCR}(\boxed{1}) \end{bmatrix}$$

b. $$\begin{bmatrix} \textsc{sem} & \{\text{POST\_DESCR\_ADJ}(\boxed{1})\} \\ \textsc{syncat} & \text{JJ} \\ \textsc{phon} & <\text{the } \boxed{1} \text{ post}> \end{bmatrix} \longrightarrow \begin{bmatrix} \textsc{sem} & \text{POST\_DESCR\_ADJ}(\boxed{1}) \end{bmatrix}$$

c. $$\begin{bmatrix} \textsc{sem} & \{\text{IN\_NAME}(\boxed{1})\} \\ \textsc{syncat} & \text{NP} \\ \textsc{phon} & <\boxed{1}> \end{bmatrix} \longrightarrow \begin{bmatrix} \textsc{sem} & \text{IN\_NAME}(\boxed{1}) \end{bmatrix}$$

d. $$\begin{bmatrix} \textsc{sem} & \{\text{IN\_AGE}(\boxed{1})\} \\ \textsc{syncat} & \text{CD} \\ \textsc{phon} & <\boxed{1}> \end{bmatrix} \longrightarrow \begin{bmatrix} \textsc{sem} & \text{IN\_AGE}(\boxed{1}) \end{bmatrix}$$

The first and second rule contain further lexical material in their phonological representation whereas (69c) remains the same as (68) and the syntactic category of (69d) is just the preterminal symbol of the marked numeral as described above.

5.3.2 Phrase-combining rules

Obviously, we have not yet covered the entire parse tree. For example, the verb remains outside the reach of input rule areas. In figure (5.7), we represent the extended input rule areas just produced by square brackets. Any internal structure has been removed except for the root node and the words. To abstract away from particular filler words, variables (\$ + tag name) are used to represent marked words. The marked words themselves have been deleted. The uncovered parts of the tree are those outside square brackets.

We now need to find a way to split the tree further so that the remaining portions of the tree are eventually covered and appropriate rules can be extracted. We will call the rules at this next level *phrase-combining* or *phrasal* rules because they will serve to combine tree fragments generated by the input rules. In addition, phrasal rules should

```
(S
  (NP-SBJ-1
    [NP $INPERSON_FULLNAME]
    (, ,)
    (NP
      (NP (DT the) [CD $INPERSON_AGE] (JJ -year-old)
        (NX
          [NX $INPERSON_OTHERPOST_DEF]
          (CC and)
          [NX $INPERSON_OTHERPOST_DEF] ))
      [PP of
        $INPERSON_OTHERCOMP
                                  ] )
    (, ,) )
  (VP (VBD was)
    (VP (VBN named)
      (NP (-NONE- *-1) )
      (PP-CLR (TO to)
        (NP
          [NP the $POST_DESCR_ADJ post]
          (PP (IN of)
            (NP
              [NP $POST_NODET]
              [PP of
                this $COMP_DESCR] ))))))
  (. .) )
```

Figure 5.7: Input rules areas within treebank bracketing

also operate recursively on other phrasal rules. Again, the procedure will be to extend rule areas ('phrasal rule areas' henceforth) up to some stopping-criterion. We need to make sure that we do not end up with a single phrasal rule generating the entire tree since the resulting grammar would obviously lack flexibility.

The idea is to introduce and extend phrasal rule areas bottom-up by always working on the most embedded part of the tree seen so far. Since we have already identified maximal nodes dominating individual tags and collapsed them into input rules, the next-level node will encompass at least one more input rule. This node forms the root of a new phrasal rule area that can be extended further. As for input rule areas, further lexical material can always be incorporated. In contrast to the treatment of input rules, however, potential overlaps with other rule areas result in new rule areas being wrapped around the lower level ones. Before we define the algorithm in more detail, we will look at our example tree.

Working our way backwards through the tree structure in (5.7), we first encounter a surface full stop and its preterminal. They can be reduced to the full stop. Next, the `COMP_DESCR` and `POST_NODET` rules (lines 25 and 23) are wrapped in a phrasal rule area with root NP (line 22) that can be extended to encompass the surface word 'of', changing its root to PP (line 21). Incorporating the `POST_DESCR_ADJ` input rule is not allowed, however, and a new phrasal rule area needs to be wrapped around this rule and the previous phrasal rule area. The new phrasal rule area is necessary because we want to combine two areas whose root node is at the same level of embedding. Thus, none of the root nodes dominates the other. Without introducing new phrasal rules areas at these points, we would indeed end up with a single phrasal rule generating the entire tree. In this case, the new phrasal rule area is rooted in NP (line 19) and can be extended until it also encompasses the surface string 'was named to', rooted in VP (line 15). The empty surface element is ignored. Further extensions of the VP-area are not possible because it competes with NP-SBJ-1 (line 2) at the same level of embedding.

A similar bottom-up procedure applies to the subject-NP. The two `INPERSON-_OTHERPOST_DEF` input rules are combined in an area with root NX (line 7) that cannot be expanded further. The NP in line 6 dominates the `INPERSON_AGE` rule and the area rooted in NX and thus forms the root of the next-level area. To incorporate the PP node of `INPERSON_OTHERCOMP`, another phrasal area with root NP, line 5, has to be introduced. After wrapping the result of this with `INPERSON_FULLNAME` into a new area, yet another phrasal rule area wrapping the subject NP, VP and the final full stop can be formed.

We can think of this process as an operation on the tree structure that performs two actions:

1. It wraps phrasal rule markers around maximal nodes that dominate sets of input rules. This is shown in figure (5.8). Curly brackets indicate phrasal rule markers; square brackets indicate extended input rule areas.

2. It deletes all syntactic category nodes outside input rule areas that do not form the root of phrasal rule areas, i.e. it keeps only category nodes following an opening curly bracket. It also removes all remaining round brackets of the original treebank parse but keeps surface words apart from empty elements. The result is shown in figure (5.9).

Before we address the task of constructing input and phrasal rules based on reduced tree structures such as figure 5.9, we detail somewhat more formally how input and phrasal rule areas are defined.

5.3.3 Constraints on treebank marked for generation rule construction

The input to the grammar construction phase is the result of merging the treebank bracketing with the annotation, which we will call T_{tag}. The tagged areas A_{tag} in T_{tag} are sequences of terminal leaves, possibly including extra levels of structure.

Preparing T_{tag} for actually constructing the grammar rules (which is described in section 5.4) results in T_{rule}. As explained above, this involves adding additional markup for input and phrasal rule areas as well as deleting some parts of the original treebank bracketing. For the resulting T_{rule} the following constraints need to hold:

1. For each input rule area A_{input}:

```
{(S
  {(NP-SBJ-1
     [NP $INPERSON_FULLNAME]
     (, ,)
    {(NP
      {(NP (DT the) [CD $INPERSON_AGE] (JJ -year-old)
        {(NX
           [NX $INPERSON_OTHERPOST_DEF-P1]
           (CC and)
           [NX $INPERSON_OTHERPOST_DEF-P2] )}))}
       [PP of
         $INPERSON_OTHERCOMP-P1+P2
                                      ])}
     (, ,) )}
  {(VP (VBD was)
     (VP (VBN named)
       (NP (-NONE- *-1) )
       (PP-CLR (TO to)
         (NP
           [NP the $POST_DESCR_ADJ post]
          {(PP (IN of)
             (NP
               [NP $POST_NODET]
               [PP of
                 this $COMP_DESCR] ))})))))}
  (. .) )}
```

Figure 5.8: Phrasal rule markers (curly brackets) added to treebank bracketing

- A_{input} is a subtree of T_{rule}.
- The root node of the input rule area, $R_{A_{input}}$, dominates exactly one tagged area, i.e. each A_{input} corresponds to one annotation tag.
- There is no other node N in T_{rule} dominating A_{input} that also dominates root node $R_{A_{input}}$ without dominating some other root node $R'_{A_{input}}$ of an input rule area.

2. For each phrasal rule area $A_{phrasal}$:

- $A_{phrasal}$ is a subtree of T_{rule}.
- The root node $R_{A_{phrasal}}$ of $A_{phrasal}$ dominates more than one disjoint input or phrasal rule area.

```
{ S
  { NP
    [NP $INPERSON_FULLNAME]
        ,
    { NP
      { NP     the  [CD $INPERSON_AGE]      -year-old
        { NX
           [NX $INPERSON_OTHERPOST_DEF-P1]
               and
           [NX $INPERSON_OTHERPOST_DEF-P2]   } }
       [PP of
         $INPERSON_OTHERCOMP-P1+P2
                                   ] }
        ,   }
  { VP      was
              named

                   to

           [NP the $POST_DESCR_ADJ post]
          { PP     of

               [NP $POST_NODET]
               [PP of
                 this $COMP_DESCR]    }    }
      .   }
```

Figure 5.9: Reduced treebank structure

- There is no other node N dominating the same rule areas that also dominates root node $R_{A_{phrasal}}$.

5.3.3.1 Algorithm

A shift-reduce-style algorithm can be used to identify input and phrasal rule areas. It can work forwards or backwards through the treebank bracketing merged with annotation tags. The algorithm recursively removes the bracketing of the original treebank structure, inserting or changing markers for input and phrasal rule areas. It terminates when all original treebank brackets have been removed. Assuming a backward walk through the treebank bracketing, the algorithm shifts closing brackets and nodes left of it on the stack, and reduces the last closing bracket on the stack as soon as it finds an opening bracket. This is a deterministic process; no ambiguity can arise. There are as

```
Def:    treebank brackets:    ( )
        semantic annotation: < >
        input rule marker:    [ ]
        phrasal rule marker: { }

Input:  treebank structure T merged with semantic annotation,
         i.e. T contains ()s and <>s.


Start at the end of T and repeat until all ()s in T are removed:

1. Identify next subtree ST backwards in T:
     ST spans elements between first '(' from back of T and next ')' to its right.

2. Remove ( and ) around ST.

3. Cases:

  3.1 if ST does not contain any pair of <> or [] or {}:
        remove all nodes except non-empty terminals.

  3.2 else if ST contains one pair of <> and no [] or {}:
        wrap [] around ST, i.e. introduce initial input rule area.

  3.3 else if ST contains exactly one pair of [] and no {}:
        remove current [] and wrap new [] around ST,
        i.e. extend input rule area.

  3.4 else if ST contains more than one pair of [] or {}:
         wrap {} around ST, i.e. introduce new phrasal rule area.

  3.5 else if ST contains one top-level {}, possibly with internal {}s and []s:
         remove current top-level {} and wrap new {} around ST,
         i.e. extend phrasal rule area.

(3.3 and 3.5 remove all treebank nodes except non-empty terminals
 and the root node of the extended area.)
```

Figure 5.10: Pseudo-code for identifying input and phrasal rule areas

many reductions as there are constituents in the original treebank. Without going into too much detail, figure (5.10) lists the cases that need to be distinguished when treating subtrees identified in this way.

5.4 Constructing input and phrasal rules

After identifying input and phrasal rule areas in the treebank parse, we need to actually extract those rules and construct the generation rules. The reduced structure of figure (5.9), for example, can be drawn as a tree (figure 5.11; surface words are not shown; input rules are shown as abbreviated boxed tags). It should be noted that the resulting tree is not necessarily binary branching. For example, a coordination of posts might lead to higher-branching nodes. The rules that can be extracted from such a tree form the context-free backbone of the generation grammar. However, before we can do this, we need to make a number of decisions concerning category labels and the semantics of phrases.

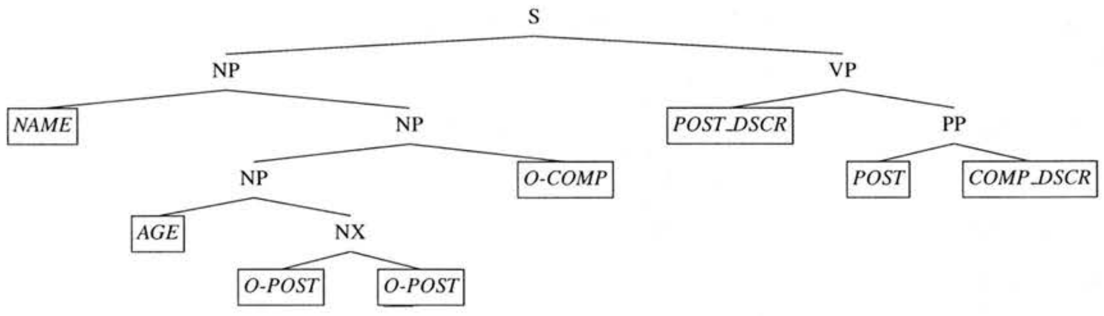

Figure 5.11: Phrasal-rule tree (input rules are boxed; surface words are not shown)

The annotated and reduced treebank trees basically provide three kinds of information: syntactic categories, surface words and semantic tags. Syntactic categories are available at every level of the treebank trees. We can make use of these node labels in the rules to be extracted. For input rule areas, there is a single semantic tag that can be assigned to that area as its semantics. However, we need to make a decision about the semantics of phrasal rules which is not directly available in tree structures such as figure 5.9. Our approach is to assign phrasal rule areas the union of the semantic tags of the areas they dominate. When the grammar rules are used for generation, this will allow us to check the flat semantic representations of the generated phrases against the equally flat semantic input. In practice, we assign a unique integer to each attribute-value pair in the input and represent the semantics of phrases as the set of such integers. Thus, input rules create phrases whose semantics is a set with just a single element, the semantic tag around which they are built (see 69a-69d). We allow each phrase built by the generator to consume each input integer only once, similar to the constraint in parsing that an input word can only be consumed once.

The next question concerning rule construction is the specification of root node categories. One extreme is to just take the syntactic category of the treebank tree. However, this would lead to serious overgeneration since all phrases with the right syntactic categories could be combined (subject to the constraint that they do not have overlapping semantic sets). To give the generator somewhat more guidance, we pick one of the semantic tags of the dominated areas and combine it with the syntactic category of the area root nodes. Our heuristic is to always choose the leftmost semantic tag. Higher-level phrasal rules take over this semantic tag from their leftmost daughter. For example, the input rule area of `POST_NODET` rooted in NP (figure 5.9, line 23) is assigned a combined category `NP-POST_NODET`. The input rule area rooted in PP, line 24, is assigned `PP-COMP_DESCR`. The phrasal rule area dominating these two input rule areas rooted in PP, line 21, adopts the tag suffix of its leftmost daughter and is thus labelled `PP-POST_NODET`.

In previous examples of input rules (69a-d), we have used a `phon` feature to represent the surface words of a phrase, including the filler words, i.e. values, of the input tag. However, for the purpose of ranking, we need to have access to the tags at the correct positions in the surface string. For example, we want the grammar to construct a string `<the INPERSON_AGE(56)-year-old INPERSON_OTHERPOST_DEF(chairman)>` rather than just `<the 56-year-old chairman>`. From the richer notation, the phonological string can easily be read-off. However, the ranker now is also able to score the mixed string `<the INPERSON_AGE -year-old INPERSON_OTHERPOST_DEF>`. This is what we intend to do to abstract away from particular slot fillers. To indicate the changed representation, the phon feature is replaced by a terms feature. (70) shows two examples of phrasal rules extracted from the reduced tree in figure 5.9:

(70) a.
$$\begin{bmatrix} \text{SEM} & \boxed{1}\cup\boxed{2} \\ \text{CAT} & \text{VP-POST\_DESCR\_ADJ} \\ \text{TERMS} & <\text{was named to}\ \boxed{3}\boxed{4}> \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SEM} & \boxed{1} \\ \text{CAT} & \text{NP-POST\_DESCR\_ADJ} \\ \text{TERMS} & \boxed{3} \end{bmatrix} \begin{bmatrix} \text{SEM} & \boxed{2} \\ \text{CAT} & \text{PP-POST\_NODET} \\ \text{TERMS} & \boxed{4} \end{bmatrix}$$

b.
$$\begin{bmatrix} \text{SEM} & \boxed{1}\cup\boxed{2}\cup\boxed{3} \\ \text{CAT} & \text{NP-AGE} \\ \text{TERMS} & <\text{the}\ \boxed{4}\ \text{-year-old}\ \boxed{5}\boxed{6}> \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SEM} & \boxed{1} \\ \text{CAT} & \text{CD-AGE} \\ \text{TERMS} & \boxed{4} \end{bmatrix} \begin{bmatrix} \text{SEM} & \boxed{2} \\ \text{CAT} & \text{NX-OPOST} \\ \text{TERMS} & \boxed{5} \end{bmatrix} \begin{bmatrix} \text{SEM} & \boxed{3} \\ \text{CAT} & \text{PP-OCOMP} \\ \text{TERMS} & \boxed{6} \end{bmatrix}$$

The $\cup$ symbol denotes set union. Phrasal rules generally share semantic and phonological information between left-hand side and right-hand side. Phrasal rules do not add any further semantic information to a phrase beyond what is being contributed by their daughters. In this sense, our approach is strictly compositional. However, phrasal rules can add further surface words to the phonological string as the left-hand sides of (70a) and (70b) show.

The representation of input rules needs to be adapted in order to carry the new syntactic-semantic category and copy tags as well as filler words into the value of the terms feature. Below, we show the updated versions of two previous input rule examples (69a,69b):

(71) a.
$$\begin{bmatrix} \text{SEM} & \{\boxed{1}\} \\ \text{CAT} & \text{PP-COMP\_DESCR} \\ \text{TERMS} & <\text{of this}\ \boxed{1}> \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SEM} & \boxed{1}\,\text{COMP\_DESCR(\$X)} \end{bmatrix}$$

b.
$$\begin{bmatrix} \text{SEM} & \{\boxed{1}\} \\ \text{SYN} & \text{JJ-POST\_DESCR\_ADJ} \\ \text{TERMS} & <\text{the}\ \boxed{1}\ \text{post}> \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SEM} & \boxed{1}\,\text{POST\_DESCR\_ADJ(\$X)} \end{bmatrix}$$

In appendix A, we show how grammar rules are actually implemented. Chapter 6 addresses the more general question how the constructed generation rules can be efficiently executed. In the remainder of this chapter, we address the question of tag-indices and investigate some characteristics of the extracted rule set.

5.5 Handling tag-indices

Tag-indices are used in the annotation scheme to represent links between tags when there is more than one occurrence of a tag (see section 4.3). The grammar construction algorithm presented so far has ignored the existence of indices. The question is how we should handle indices that are part of the input to the generator. Since tag-indices are intended to represent co-indexing, we should treat indices in the input as variables, and the grammar rules should provide a mechanism for ensuring that the variables unify. This in turn requires the rules to pass indices on until they meet, which goes beyond the basic context-freeness of the grammar. We use the local trees defined by the rules to handle indices (rather than adding an additional 'global' mechanism outside the grammar).

Since the handling of indices is based on rules that are derived from the syntactic treebank, indices will meet following this structure. Our idea is to exploit this structure-driven combination when deciding what should happen when indices are combined. When looking at the syntactic tree structure, we observe that constituents that are syntactically related are also semantically related. For example, two OTHERPOSTs are combined with their OTHERCOMP before being combined with other tags (see figure 5.9, for example). Thus, we can enforce a check for index compatibility when constituents are combined.

Similar observations of structural relatedness can also be made for coordinated persons. Figure 5.12 shows a treebank fragment initially merged with some annotation tags (corresponding to example 56f, page 118). In structural terms, the two persons are equally related to the POST_PLURAL tag. The coordination can be interpreted as expressing this. Note that in this example we use two pairs of indices to connect the coordinated persons to their posts. (Furthermore, it should be noted that the structural relations between the tags are preserved when the original treebank structure is reduced by the grammar construction algorithm.) Another example of a three-way connection between two persons and their posts can be found in sentence (56e) on page 118 where we use two pairs of indices, one for each INPERSON_FULLNAME-POST_BOARD pair. (The example also requires additional indices for the persons' other companies and posts.)

These observations suggest that the treatment of indices should be linked to the use

```
( (S
    (NP-SBJ-1
      (NP [INPERSON_FULLNAME-P1 (NNP Anton) (NNP Amon) ] )
      (CC and)
      (NP [INPERSON_FULLNAME-P2 (NNP George) (NNP Gourlay) ] ))
    (VP (VBD were)
      (VP (VBN elected)
        (S
          (NP-SBJ (-NONE- *-1) )
          (NP-PRD
            (NP [POST_PLURAL-P1+P2 (NN vice) (NNS presidents) ] )
            (PP (IN of)
              (NP (DT this) [COMP_DESCR (NN soft-drink) (NN company)]))))))
    (. .) ))
```

Figure 5.12: Coordination of incoming persons in annotated treebank parse

of coordination. We will require index pairs to meet at the next structural level after coordination has taken place, i.e. we have to collect indices when tags are coordinated, and we have to check their compatibility when the coordinated constituent is combined with other index-bearing constituents. Furthermore, we will cancel indices after they successfully meet since we will not need them again. A sentence expressing the entire input semantics should therefore not have any remaining indices.

However, we also need to be able to cancel indices when no coordination is directly involved. For example, when there is more than one INPERSON_FULLNAME, each with its respective OTHERPOST (see 56e, page 118), we need to be able to combine and cancel indices when INPERSON_FULLNAME and OTHERPOST are combined, or block the combination if the indices are not compatible. On the other hand, there are cases where tags without indices need to be combined with those that carry indices, for example COMP_DESCR and POST_PLURAL in figure 5.12. Such combinations should be possible. These considerations lead us to define the following requirements for the treatment of indices:

1. each phrase has an index set (similar to their set of semantic tags),
2. when two phrases are combined by a coordination, form the union of their index sets,
3. when non-coordinated phrases are combined, require the cancelling of at least

one index unless (at least) one of the phrases has an empty index set,

4. when a coordinated phrase is combined with another, uncoordinated, phrase, require the cancelling of at least one index of each non-empty coordinated constituent.

The third item considers the standard case of non-concatenated phrases. If none of the phrases involved contains indices, we do not require anything to happen. This refers to the rule combinations of the majority of annotated sentences that do not carry indices. However, if indices are used and two phrases with non-empty index sets are combined, we require the cancelling of at least one index. This implements the intuition that structurally related constituents are also semantically related. By requiring cancellation, the index of an AGE tag could not be passed on to the wrong person past the correct person, for example. On the other hand, we do allow the indices of a constituent simply to be passed on when the other constituent with which it is combined has an empty index set. This makes index handling somewhat more liberal since it allows the treatment of cases like the abovementioned combination of COMP_DESCR and POST_PLURAL in figure 5.12.

The fourth item considers coordinated phrases. In general, we expect each coordinated constituent to carry at least one index, and we want to be able to cancel these out at the next level upwards in the tree. However, there are cases where not all coordinated constituents carry indices. For example, there can be two OTHERPOSTs but no OTHERCOMP (see example (57), page 118). Another example is the following:

(72) This business trust company said its board elected Kieran E. Burke, ... , as [POST_NODET-P1 chief executive officer], a [POST_DESCR_ADJ-P1 new] post, and as [POST_NODET president].

In (72), POST_DESCR_ADJ needs to be combined with the left-most POST_NODET. When the two are joined, their indices can cancel each other out. When the two POST_NODETs are coordinated by the extracted rule 'PP-POST_NODET → PP-POST_NODET and PP-POST_NODET' they do not carry indices anymore, and therefore we cannot enforce a cancellation at the next level. On the other hand, if POST_DESCR_ADJ is combined with the wrong POST_NODET (yielding 'as president, a new post, and as `chief executive officer`'), the coordinated phrase still contains an index. In the proposed mechanism, this does not pose problems for the derivation as long as the index does not meet another, incompatible one (which cannot happen in this example because there is only one index pair). However, *ceteris paribus*, we would like to express a preference for edges that contain fewer indices. This preference is implemented in the redundancy check of the chart generator (see section 6.4.1.2).

In general, we want to limit the use of indices and avoid adding additional ones just for the sake of the mechanisms of index handling. However, it should be clear that our treatment of tag indices is 'shallow' and might not work in all situations. On the other hand, a shallow grammar does not necessarily lead to incorrect output because of the pruning/ranking provided by the instance-based ranker. The rules for index combination outlined above should be regarded as heuristic rules. We will evaluate them in section 7.7.1.

5.5.1 Detecting coordination

Before we can use the indices in the way just described, we need to identify the coordination of generation rule areas in the treebank. We look for sequences of nodes on rule right-hand sides that have the same syntactic category and are separated by a conjunction (a comma or 'and'). To avoid false positives, the last conjunction must be an 'and'. In general, there are two cases: the coordination can span the entire rule right-hand side or just a subsequence. If only a subsequence is coordinated, we introduce a new coordination level so that coordination is taking place in a separate rule. This is necessary because the index mechanism distinguishes between coordinated and non-coordinated rules. The left-hand side of that rule is assigned an appropriate syntactic category taken from the treebank fragment under consideration. As for other rules, the mother node reuses the tag of its left-most daughter in its combined category label. This allows for recursive coordination and avoids introducing new categories that explicitly indicate coordination, for example. (73) shows some examples of coordinated rules that span entire constituents. These coordinations are recognized (and the grammar rules are flagged appropriately) but no new coordination level rules are introduced. (74) shows rules that require the introduction of a new coordination level.

The original rule shown first is replaced by the following two new rules.[3]

(73) a. `NP-POST_NODET` $\rightarrow$ `NP-POST_NODET and NP-POST_NODET`

b. `NP-POST_NODET` $\rightarrow$ `NP-POST_NODET , NP-POST_NODET and NP-POST_INDEF`

c. `VP-POST_NODET` $\rightarrow$ `was named NP-POST_NODET and NP-POST_NODET`

d. `NP-OUTPERSON_AGE` $\rightarrow$ `NP-OUTPERSON_AGE and NP-OUTPERSON_PREVIOUSPOST_INDEF`

(74) a. `S-POST_NODET` $\rightarrow$ `NP-POST_NODET and NP-POST_NODET PP-COMP_DESCR`

$\Longrightarrow$

`S-POST_NODET` $\rightarrow$ `NP-POST_NODET PP-COMP_DESCR`

`NP-POST_NODET` $\rightarrow$ `NP-POST_NODET and NP-POST_NODET`

b. `VP-POST_NODET` $\rightarrow$ `was named NP-POST_NODET and NP-POST_NODET PP-COMP_DESCR`

$\Longrightarrow$

`VP-POST_NODET` $\rightarrow$ `was named NP-POST_NODET PP-COMP_DESCR`

`NP-POST_NODET` $\rightarrow$ `NP-POST_NODET and NP-POST_NODET`

c. `NP-INPERSON_AGE` $\rightarrow$ `a CD-INPERSON_AGE -year-old NN-INPERSON_OTHERPOST_INDEF and NN-INPERSON_DESCR_NP_INDEF`

$\Longrightarrow$

`NP-INPERSON_AGE` $\rightarrow$ `a CD-INPERSON_AGE -year-old NN-INPERSON_OTHERPOST_INDEF`

`NN-INPERSON_OTHERPOST_INDEF` $\rightarrow$ `NN-INPERSON_OTHERPOST_INDEF and NN-INPERSON_DESCR_NP_INDEF`

It should be noted that we are not looking for identical syntactic-semantic categories in order to detect coordinated phrases but only for identical syntactic category labels. This allows us to detect the coordination of NPs that are associated with different tags (see 73b and d, for example). Furthermore, it should be noted that the introduction of new coordination levels does not cross the treebank bracketing.

[3] `INPERSON` and `OUTPERSON` have been abbreviated to `IN` and `OUT`.

5.6 A look at the extracted rule set

The grammar extraction algorithm produces 1624 rule tokens for the 144 annotated treebank sentences, 896 of them input rules and 728 phrasal rules. When we remove duplicate rules, the size of the grammar is reduced to 476 rule types overall (181 input rules and 295 phrasal rules). Figure 5.13 shows the growth of the rule set when the number of annotated treebank parses used for grammar construction is increased. As for the treebank grammar (see figure 5.3), the size of the grammar seems too small to decide whether one would arrive at a finite rule set at some point.

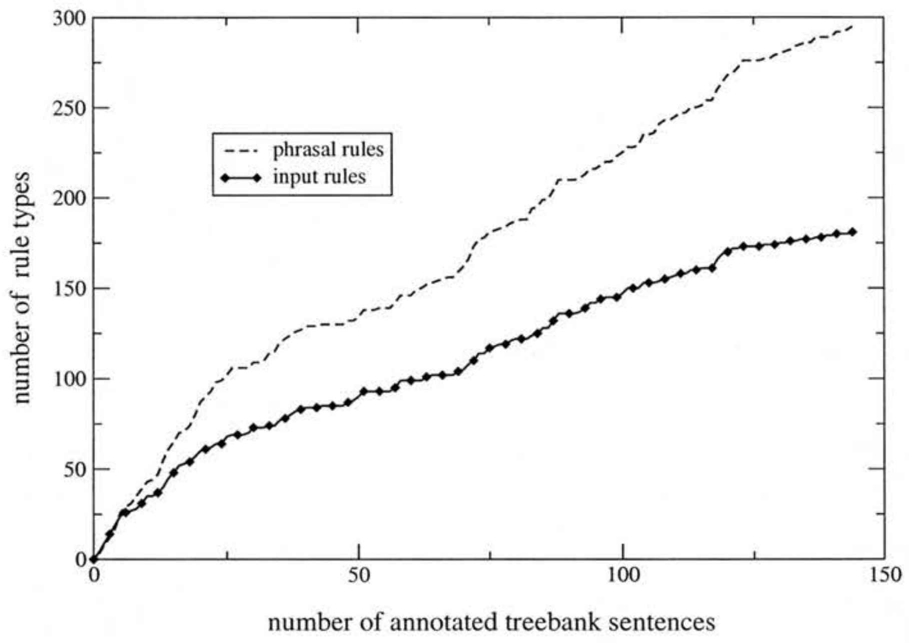

Figure 5.13: Growth of generation rule set

To conduct experiments, our corpus will be split into training and test sets, i.e. we will not be able to use a grammar based on all 144 annotated treebank parses. Figure 5.14 shows the coverage of the rules in 25% held-out data (=36 sentences) for training set grammars from 1 to 108 treebank parses. What is measured is the number of held-out rules that occur in the training set. Similarly to the coverage of cf-rules on the same held-out set (figure 5.5), relatively few training sentences are sufficient to cover the most frequent rules. This is reflected by the higher coverage values for

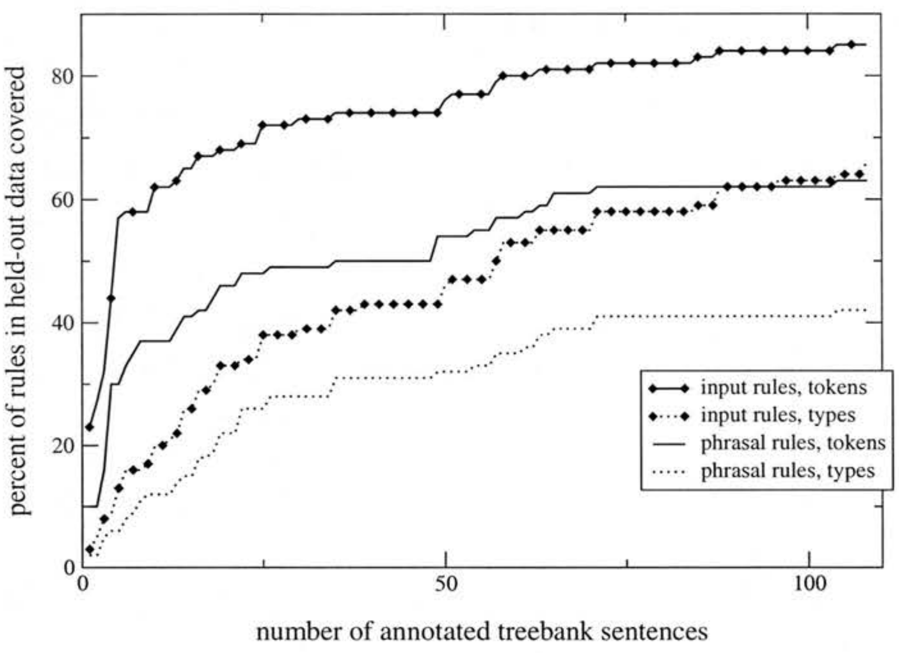

Figure 5.14: Coverage of generation grammar w.r.t. held-out data

rule tokens in the held-out data (which are 'weighted') compared to rule types (which are 'unweighted'). The extracted generation rules also contain relatively few high-frequency rules and many low frequency ones (figure 5.15). Again, this is similar to the set of simple cf-rules extracted from the treebank (figure 5.4). Generally, the coverage of input rules is higher than the coverage of phrasal rules which seems to be due to the larger number of phrasal rule types.

A closer look at the generation grammar rules reveals that 78.5% of the phrasal rules introduce additional lexical material, i.e. most phrasal rules are 'lexicalized' in this sense. Furthermore, most phrasal rules are binary branching (89%), some are ternary (10%) and 1% have four daughter nodes. Of the input rules, which are unary by definition, 49% are lexicalized. This lower number might help explain why there are fewer input rule types than phrasal rule types.

Figures 5.16 and 5.17 show some extracted generation rules.[4] The columns titled '|w|' refer to the number of additional surface words that are introduced by the rule. These added words do not affect the branching factor of the rules. The average number

[4]Again, INPERSON and OUTPERSON have been abbreviated to IN and OUT.

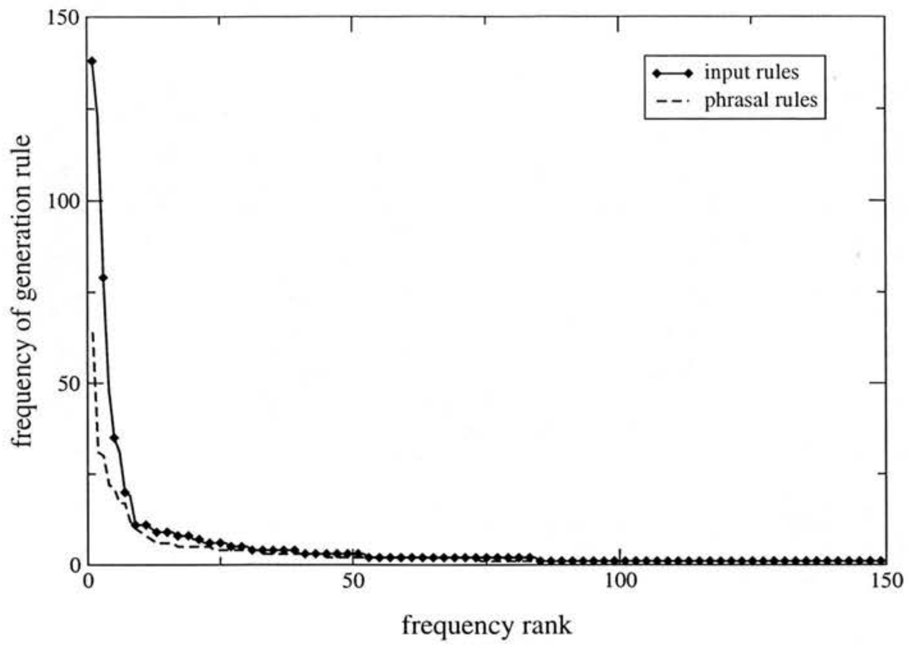

Figure 5.15: Generation rule ranks vs frequency

	|w|	input rule
(1)	6	S-BOARD_INCR → increasing the number of seats to BOARD_INCR
(2)	5	S-BOARD_INCR → increasing the board to BOARD_INCR members
(3)	5	S-BOARD_INCR → expanding the board to BOARD_INCR members
(4)	5	S-BOARD_INCR → boosting the board to BOARD_INCR members
(5)	4	VP-POST_BOARD → was named to its POST_BOARD
(6)	4	VP-OUT_PRP_VP_PAST → OUT_PRP_VP_PAST from his executive duties
(7)	4	SBAR-OUT_POSTCONT_INDEF → who continues as a OUT_POSTCONT_INDEF
(8)	4	NP-POST_DESCR_ADJ → posts which had been POST_DESCR_ADJ
(9)	3	VP-POST_INDEF → was elected a POST_INDEF
(10)	3	S-OUT_NEWPOST_INDEF → to accept a OUT_NEWPOST_INDEF
(11)	2	VP-POST_NODET → was named POST_NODET
(12)	2	VP-POST_LOC → based in POST_LOC
(13)	2	SBAR-OUT_POSTCONT_NODET → who remains OUT_POSTCONT_NODET
(14)	2	PP-COMP_DESCR → of this COMP_DESCR

Figure 5.16: Some extracted input rules

	|w|	phrasal rule
(1)	5	PP-COMP_SUBSIDIARY → of the company and its NAC-COMP_SUBSIDIARY NN-COMP_SUBSIDIARY_DESCR_NODET
(2)	4	VP-POST_INDEF → will join the board PP-POST_INDEF NP-INDATE_FUTURE
(3)	4	VP-POST_BOARD → has been elected to NP-POST_BOARD PP-COMP_DESCR
(4)	4	VP-IN_FULLNAME → said its board elected NP-IN_FULLNAME PP-POST_NODET
(5)	4	PP-POST_NODET → from his post as NP-POST_NODET PP-COMP_DESCR
(6)	3	VP-POST_RESPONSIBILITY → will assume responsibility PP-POST_RESPONSIBILITY ADJP-INDATE_FUTURE
(7)	3	VP-POST_DESCR_ADJ → was elected to NP-POST_DESCR_ADJ PP-POST_NODET
(8)	3	VP-OUT_PRP_VP_FUTURE → announced that he VB-OUT_PRP_VP_FUTURE VP-OUTDATE_FUTURE
(9)	3	SBAR-POST_NODET → who was named S-POST_NODET PP-OUT_INDATE_PAST
(10)	3	SBAR-IN_PREVIOUSPOST_NODET → who had been NP-IN_PREVIOUSPOST_NODET PP-IN_PREVIOUSPOST_DATESPAN_PAST
(11)	3	PP-COMP → of both NP-COMP and NP-COMP
(12)	2	VP-POST_DESCR_ADJ → will assume NP-POST_DESCR_ADJ PP-POST_NODET
(13)	2	NP-IN_AGE → a CD-IN_AGE -year-old NN-IN_OTHERPOST_INDEF
(14)	1	VP-POST_BOARD → joins NP-POST_BOARD PP-INDATE_FUTURE
(15)	1	VP-IN_FULLNAME → said NP-IN_FULLNAME VP-POST_NODET
(16)	1	Sup-INPERSON_FULLNAME → NP-INPERSON_FULLNAME VP-POST_NODET .

Figure 5.17: Some extracted phrasal rules

of elements on the right-hand sides of phrasal rules (words and syntactic-semantic categories) is 3.5. This is close to the average branching factor of the simple treebank grammar discussed in section 5.1, which is 3.8. On average, there are 1.7 elements on the right-hand sides of the input rules (and there is always at least one element, the input tag).

The extracted grammar rules show the effects of the annotation scheme in combination with the treebank bracketing. For example, rule (1) in figure 5.16 can be interpreted as stating that we can generate a constituent with category S-BOARD_INCR and surface words 'increasing the number of seats to BOARD_INCR' if tag BOARD-_INCR is present in the input. Guided by the treebank bracketing, the tag BOARD_INCR has been made 'responsible' for generating the surrounding surface words. Rule (1) also shows the need to introduce a distinct top-level category for candidate sentences (which we call 'Sup') since a rule such as (1) does not generate a full sentence that could be regarded as a potential output of the sentence realizer. Phrasal rule (16) in figure 5.17 shows a typical top-level rule generating NP, VP and a full stop.

Rule (6) in figure 5.16 contains the surface string `'from his executive duties'`, a result of the annotation decision to leave these pieces of information unmarked. In other words, we assume that we are only dealing with executive posts. Furthermore, the rule records that the person in question is male (which is empirically true in almost all cases).

Phrasal rules for VPs can generate word sequences like `'will join the board'` (rule 2, figure 5.17). Compared to the corresponding fixed annotation template which contains the word sequence `'years old, will join the board as a'` (see figure 4.3, page 125), the phrasal rule is shorter because the word sequence needs to be broken down according to the syntactic structure. As a consequence, it will be able to be used more flexibly.

Some example rules contain variations of the `DATE` tag. These tend to correspond to adjuncts in the treebank. However, we do not make a distinction between complements and (optional) adjuncts. Rather, the grammar rules record the actual use of the `DATE` tags, motivated by particular treebank parses. Furthermore, rules that contain verbs like `'said'` or `'announce'` (rules 4,8,15 in figure 5.17) are a result of the annotation decision to leave announcements unmarked, i.e. we assume that the explicit mention of the announcement is just a minor semantic variation. In addition, it should be noted that a rule like (11) in figure 5.17 which generates `'both'` expects exactly two coordinated `COMP` tags. It therefore records the semantically correct use of `'both'`.

A further question concerning the extracted rule set is related to the recursiveness of the grammar. We can identify 98 phrasal rules (out of 295) that are directly recursive, i.e. the mother category reappears on the rule right-hand side. They involve 32 different syntactic-semantic categories. The recursiveness of the grammar is obviously influenced by decisions about node labelling. (As described above, we use the tag of the left-most daughter node as part of the mother node label.) Some examples of directly recursive phrasal rules have been shown in (73a,b,d) and (74a,b,c).

5.7 Related work on combining treebank and annotation

There is work on simply reading-off a context-free grammar from a syntactic treebank (Charniak, 1996). Krotov et al. (2000) show a method for compacting such a treebank grammar by removing redundant rules, where a rule is defined as redundant when it can be parsed by other rules.

However, (syntactic) treebank grammars alone are not sufficient for NLG because they lack semantic representations. Combinations of treebank and semantic annotation have been explored for information extraction (IE). Chelba and Mahajan (2002) show how a structured language model can be trained on a treebank augmented with semantic frames. The task of the IE system is to determine the correct frame and fill in the appropriate slots. Similar to our approach, the semantic annotation is 'flat'. However, we do not have dedicated frame labels for specific combinations of slot tags.

(Miller et al., 2000) also describe a statistical approach to information extraction that combines manual semantic annotations with syntactic structures. The semantic annotation is more complicated, comprising nested tags and relations between them. In contrast to our approach and the one of (Chelba and Mahajan, 2002), additional syntactic nodes have to be introduced into the treebank to make syntax and semantics match.

It seems that the (manually constructed) resources of these approaches are generally quite similar to the resources developed in our approach. However, the task is analysis rather than generation. An issue for possible future work is the question of whether it is possible to have a reversible system for NLG and IE (see also section 8.6.2).

5.8 Conclusions

In this chapter, we presented an approach to automatically derive the context-free backbone of a generation grammar from a treebank and a manual semantic annotation. Our approach assumes a syntactic treebank based on constituency in the style of the Penn treebank (Marcus et al., 1993). The reason for this choice is the widespread use of this treebank and the availability of state-of-the-art parsers trained on it (see Collins, 1999, for example). It should therefore be possible to obtain syntactic structures for new corpora that are collected in the context of NLG system development efforts by using such parsers. Our approach provides a way of taking advantage of the continuing progress in statistical parsing.

It should be noted that the manipulation of the rule structure to account for coordination is based on general heuristics that apply to the entire treebank. No individual grammar rule is written or altered manually. Similarly, the treatment of indices is based on a few general considerations.

By using grammar rules rather than templates, we increase the robustness and flexibility of the generation system. On the other hand, this also extends the search space of the candidate generator. Section 3.3 gave some idea of the number of edges involved in finding the output candidates; chapter 7 will show examples of the candidate sentences that are actually produced by the system. In the next chapter, we show how the constructed grammar rules can be used to build an efficient candidate generator for our hybrid generation system.

Chapter 6

Rete-based chart generation

The rule extraction phase described in the previous chapter produces a large number of input and phrasal rules. In this chapter, we describe a bottom-up chart generation algorithm which is able to handle several hundreds of rather 'shallow' rules.

One issue concerning the organization of large numbers of grammar rules is the question of rule generalization. Finding common characteristics within sets of rules can improve processing and might also lead to linguistic insights. In parsing, techniques have been developed that exploit the fact that rules may share a number of daughter nodes. For example, Evans and Weir (1997, 1998) describe a structure-sharing parser that merges finite-state automata derived from LTAG trees. As a result, a bottom-up parser is able to simultaneously traverse several trees at once by matching nodes in the merged automaton.

We use a method that also enables the sharing of daughter nodes, albeit in a different way. Rete networks (Forgy, 1982) are commonly used in production system languages such as OPS5 (Forgy, 1981) and CLIPS ("C Language Integrated Production System", (Riley, 1999)) to organize large sets of rules (or 'productions').

Knowledge-based expert systems are a typical application of production systems. A match of a node in a Rete network partially 'activates' a whole set of rules that share that node. This avoids cycling over large numbers of rules in turn and repeatedly matching the same elements. Once all elements required in a rule are matched, the rule 'fires' by executing a function call.

In the following, we introduce Rete networks and relate them to the chart algorithms of NLP. We then present a bottom-up chart generator that uses a Rete network to share partial matches on grammar rules and show empirical results of the amount of structure sharing for the grammars used in this thesis. The actual encoding of grammar rules as productions is shown in appendix A.

6.1 Rete networks

Knowledge-based expert systems basically consist of rules and a knowledge base of facts (KB). Rules express logical implications and facts contingent truths. Facts match (possibly partially defined) fact descriptions, or ‘conditions’, in the antecedent of rules, and if all conditions required in a rule antecedent are satisfied, the consequent follows. This usually involves the assertion or retraction of facts to/from the KB. The basic outline of knowledge-based expert systems is similar in spirit to declarative rule systems based on the logic programming paradigm in NLP in which the order of rule execution does not alter the result (Pereira and Warren, 1983). In particular, the knowledge base serves as a store for proved facts and is hence comparable to a chart data structure in NLP. The knowledge base corresponds to a passive chart as it only contains completed chart edges representing linguistic phrases.

Standardly, a forward chaining interpreter is used to execute expert system rules. This corresponds to bottom-up tree traversal algorithms in computational linguistic terminology. Productions are often characterized as ‘condition-action rules’ which implies forward-chaining. However, backward-chaining (top-down) processing is also possible. The ‘conflict resolution strategy’ in expert systems addresses the question which of the ‘activated’ rules should fire and in what order. In chart algorithms, this question is dealt with by means of an agenda and a function that determines its ordering. Figure 6.1 relates expert system terminology to computational linguistic terminology. This relationship will be explored throughout the rest of this chapter, in particular with respect to active edges and Rete networks.

The question we are concerned with here is how production system methods can help deal with a large number of rules in the context of an NLG system. This issue arises because of the potentially large number of rules produced by the automatic

Expert Systems	NLP
productions in Rete network	grammar rules
working memory/ knowledge base	passive chart
facts in knowledge base	passive edges
partially activated productions (inside Rete network)	active edges
conflict resolution strategy	agenda (of passive edges) with ordering function

Figure 6.1: Comparison of terminologies

grammar construction algorithm described in chapter 5.

A simple approach would be to cycle over all rules and check the conditions in the antecedents against the facts in the knowledge base. There are two sources of inefficiencies when computing inferences on a large set of rules in such a way (see Jackson, 1990, for example):

1. *Within-cycle redundancies:* Matches of facts against rule antecedents are repeatedly performed because many rules have at least partially similar conditions. In other words, the rule set exhibits *structural similarities*.

2. *Between-cycle redundancies:* Facts that remain in the KB from cycle to cycle are matched against the same rules at each cycle. This is called *temporal redundancy*.

With respect to the issue of organizing rules we are mainly concerned with structural similarities. However, we also have to address the question to what extent temporal redundancies are a problem in NLP. We need to make sure that the assumptions about production systems that underly solutions to the handling of large sets of rules also hold for NLP.

6.1.1 A simple network

A solution to the problem of within-cycle redundancies is to exploit structural similarities between rule antecedents by creating a network that allows facts to match antecedents in several rules at once. Forgy (1982) introduced Rete networks which have been widely used in expert system applications ('rete' is Latin for 'net'). The idea is to create tests for conditions in rule antecedents and to share the results between rules. For example, consider the following two simple rules (more complex ones are discussed in the following sections):

(75) $$\begin{array}{l} a \leftarrow b\ c \\ d \leftarrow a\ b \end{array}$$

We use left arrows in these context-free rule to emphazise a logic-based view of rules. In this case, the rule right-hand side is the antecedent and the left-hand side the consequent. (As we will see below, production systems use a different rule notation. A logic-based characterization of rules allows one to abstract away from notational issues.) There are only three different symbols in the two rule antecedents. A match on *b* should be attributed to both rules at once. A Rete network for these two rules makes this possible (figure 6.2).

```
          start
          / | \
         b  c  a       [pattern network]
         |  |  |
         a1 a2 a3      [alpha memories]
         |  |  |
         |\/   |
         \/\  /
         j1 j2         [join nodes]
          |  |
          b1 b2        [beta memories]
          |  |
   FIRE:  a  d         [rule consequent]
```

Figure 6.2: Simple Rete network

In the graph structure in figure 6.2, new facts enter the network at the start node and are then tested against 'one-input tests'. In this case, these are just tests for the atomic symbols *b*, *c* and *a*. The test results for two one-input nodes are joined into 'two-input nodes' (or 'join nodes'): *b* and *c* are joined into `j1` and lead to a consequent node that asserts fact *a* into the knowledge base. In the same way, *b* and *a* are joined into `j2` and connect to a node asserting *d*. The network organization allows the result of the test for *b* to be shared between the two rules. This addresses the problem of structural similarities between rules and the resulting computational redundancy.

Furthermore, temporal redundancies are avoided by associating memories with the nodes in the network. Many accounts of Rete-based production systems (Miranker, 1990; Nayak et al., 1988; Doorenbos, 1993, for example) distinguish between memories for one-input nodes, called 'alpha memories', and those for two-input nodes, called 'beta memories'. Alpha memories store information about the set of individual facts that satisfy the conditions they represent. Beta memories, sometimes called 'instantiation memories' (Bouaud, 1993), store information about compatible joins of their left and right input. This is related to the use of variables in production languages which will be discussed below. Other accounts, including (Forgy, 1982), mention only memory of join nodes.

In the example at hand (figure 6.2), if processing starts by asserting *b* and *c*, the antecedent of the first rule is satisfied and fact *a* can be asserted. Join node `j1` remembers the facts on its left and right inputs; `j2` memorizes the fact on its left input. Let us assume that a new *a*-fact matches the right-hand side of the second rule in the following cycle. Since `j2` remembers its left input, the new right input is sufficient to fully activate the second rule and assert fact *d*. Because of this memoing of test results, Rete algorithms belong to the class of 'state saving match algorithms'.

The example shows that there is no predetermined order in which facts need to be presented to the network. Parsing sequences of facts can proceed in any order of elements. On the other hand, Rete networks have been developed under the assumption of forward-chaining interpreters (although backward-chaining is available in some implementations too). Traversing a Rete network from the start node always results in bottom-up processing. Still, the order of processing daughter nodes is less fixed than in the automaton-directed parsing approach of (Evans and Weir, 1997, 1998) mentioned above.

More realistic examples of productions add more issues that need to be addressed when constructing the Rete network. In the following, we give examples for those issues that are relevant for encoding grammar rules in our approach. A more detailed description of production system languages can be found in (Giarratano and Riley, 1993) or (Jackson, 1990), for example.

6.1.2 Structured facts

Facts can have more structure than just consisting of a single atomic symbol. They can be defined by template definitions that specify a number of attributes for a specific 'type' or 'class' of fact. These attributes have values which are atomic symbols, variables (possibly with some tests attached to them) or lists of these. However, attribute-value pairs are not recursive: a value cannot be assigned another attribute-value pair. Thus, one cannot represent arbitrary feature structures straightforwardly. On the other hand, this makes it easier to match facts against conditions. These are split into a finite number of nodes, called a 'pattern'. The part of the Rete network matching facts in isolation is called the 'pattern network', as opposed to the 'join network' which combines the results of the pattern network tests.

Figure 6.3 shows conditions matching facts that are structured as attribute-value pairs. We use the standard functional notation of production system languages. Antecedents are shown on rule left-hand sides; no consequent has been specified in the example. The network in figure 6.3 shows the maximal amount of node sharing that is possible. Depending on the internal order of attributes, practical production system implementations may split the network at other points depending on the order of attributes chosen. There are fewer performance improvements possible in this kind of structure sharing since new facts always have to filter through the pattern network from the beginning. This is because facts are distinct objects so that subtests for values of different facts cannot be pooled together. Therefore, a fact is only remembered by an alpha memory in such a network if it passes all tests of the condition it represents.

```
(defrule rule-1
  (sign (phon man)  (syn n) (num sing))
  (sign (phon girl) (syn n) (num sing))
  =>
  )

(defrule rule-2
  (sign (phon men)   (syn n) (num plur))
  (sign (phon girls) (syn n) (num plur))
  =>
  )
```

```
             start
               |
type=        sign
               |
syn=           n
             /   \
num=     sing     plur
        /   \    /   \
phon=  man girl  men  girls
        |    |    |     |
        a1   a2   a3    a4
         \  /      \  /
          j1        j2
          |         |
          b1        b2
          |         |
FIRE:    rule-1   rule-2
```

Figure 6.3: Matching structured facts in Rete network

6.1.3 Variables in conditions

Variables are part of expert system languages. Consequences for the construction of Rete networks follow in cases where the same variable is used in different conditions in a rule's antecedent. This requires the introduction of join nodes that test whether the instantiation of the variables in one fact is consistent with the instantiation in another. (Although this points to fundamental questions of programming language design such as how "equality" between data structures is defined, it should be remembered that we are dealing with relatively simple objects here. In addition to their non-recursive structure, in production systems, facts are not allowed to contain variables. On the other hand, they do not need to have specified values for all their attributes. The test for compatibility required to match partially defined facts of the KB against partially defined fact descriptions in rule antecedents is nonetheless much simpler than unification for arbitrary recursive feature structures.) Some authors emphasize the role of join nodes for checking variable consistency across facts (Giarratano and Riley, 1993; Miranker, 1990). However, as we have seen above, join nodes are also introduced for combining test results of facts without any variables. Generally, the structure of the Rete network is fixed at compile time. Maintaining the bindings of variables (if they are used) needs to be done at runtime.

```
(defrule rule-3
    (sign (syn np) (num ?n))
    (sign (syn vp) (num ?n))
    =>
    )
                      start
                         |
                     type=sign
                      /   \
                  syn=np syn=vp
                      |     |
                      a1   a2
                      |     |
                      \    /
                       \ /
                        j1:
                      num=?n
                         |
                        b1
                         |
                   FIRE: rule-3
```

Figure 6.4: Conditions with variables in Rete network

A simple example for the introduction of a join node due to the use of variables (notation: `?..`) is given in figure 6.4. In many production system languages, explicit tests in the form of function calls can be attached to variables in conditions. Rete networks make the results of these tests reusable for re-occurring variables. For example, a test attached to the first occurrence of `?n` – for instance, that it should only be instantiated by certain values – also applies to the second occurrence of `?n` due to the structure of the network.

6.1.4 Ordering of conditions

In Rete networks, join nodes are binary, i.e. they have two inputs. In larger networks, many of them receive input from other join nodes rather than one-input nodes. Standardly, a join node combines the alpha memory associated with a single-input node on its right input with a beta memory associated with a join node on its left input. (Combining the first two alpha memories in a rule is obviously a special case and is often dealt with by assuming a dummy beta memory as the top node to keep the structure uniform.) In other words, in standard Rete networks, join nodes never combine input from two join nodes. For example, the network in figure 6.5 first combines alpha memories for the antecedents a and b in a join node (the special case), and then combines the output of this join node with an alpha memory for antecedent c. The alpha memories of d and e feed into join nodes that integrate the join nodes of the antecedents to their left. However, there is no join node that directly combines d and e, or b and e, for instance.

In general, there are three activities in the join network (Doorenbos, 1993): computing the left activation of join nodes, computing their right activation, and proliferating the facts to the associated beta memory if both inputs are non-empty. When facts b, c, d and e are presented to the network in figure 6.5, there are no facts in the beta memories since there are no join nodes with two non-empty inputs. On the other hand, presenting facts a, b, c and d to the network results in a chain of three join nodes being fully activated. The associated beta memories then remember the facts that arrived at the left and right inputs of these join nodes and proliferate the facts to their child nodes.

The example in figure 6.5 shows that the order of conditions in rule antecedents has consequences for the structure of the network. This in turn affects the efficiency of the Rete algorithm. Highly volatile facts matching conditions at the beginning of rule antecedents tend to result in frequent recomputations of joins. If facts b to e are given in the above example and a is frequently asserted and retracted, the state of all beta memories is affected by each such action. This is because the entire network has to be

```
(defrule rule-4
 (a)                                 start
 (b)                             /  /  |  \  \
 (c)                            a   b  c  d  e
 (d)                            |  /  /  /  /
 (e)                           a1 a2 a3 a4 a5
 =>                             |/  /  /  /
 )                              j1 /  /  /
                                b1 |  |  |
                                | /  /  /
                                j2  /  /
                                b2  |  |
                                |  /  /
                                | /  /
                                |/  /
                                j3 /
                                b3 |
                                |  |
                                | /
                                j4
                                b4
                                |
                           FIRE: rule-4
```

Figure 6.5: Chain of join nodes in Rete network

updated to the current state of the KB after each assert or retract. Therefore, a general guideline for writing expert system rules is to place conditions that match volatile facts last (Giarratano and Riley, 1993, p492). This can contradict another guideline, to place most specific conditions and those matching infrequent facts first, if the facts matching those specific conditions are volatile. The problem is more severe if productions have many conditions so that long chains of join nodes are built in the Rete network (as in figure 6.5).

Figure 6.6 shows how the order of conditions influences the structure of the network for sets of two similar rules, one of which has the specific condition $c1$, the other has condition $c2$. Memory nodes are omitted for simplicity. In the first example in figure 6.6, `j1` is shared because both rule antecedents share the prefix *ab*. In the second example, the specific conditions $c1$ and $c2$ are placed at the first position, not allowing any sharing of join nodes. This also requires five join nodes rather than just four.

```
(defrule rule-5                     start
 (a)                            /  /  \  \  \
 (b)                            a  b  c1 d  c2
 (c1)                           | /  /   /  /
 =>                             |/  /   /  /
 )                             j1  /   /  /
                                |\/   /  /
(defrule rule-6                j2/\  /  /
 (a)                            |  \/  /
 (b)                   FIRE rule-5 j3 /
 (d)                                |/
 (c2)                               j4
 =>                                 |
 )                                 rule-6
----------------------------------------------

(defrule rule-7                      start
 (c1)                            /  /  \  \  \
 (a)                            c1 a   b  c2 d
 (b)                            | / \ /|  |  |
 =>                             |/   \ |  |  |
 )                             j1   / \___j3 |
                                |  /   | /   |
(defrule rule-8                 | /    |/   /
 (c2)                          j2     j4   /
 (a)                            |      |  /
 (b)                       FIRE rule-7 | /
 (d)                                   j5
 =>                                    |
 )                                   rule-8
```

Figure 6.6: Effects of condition ordering in Rete networks

Figure 6.6 shows that Rete networks only share join nodes for identical prefixes of conditions.

The example shows that Rete does not automatically minimize the number of join nodes (for example, by joining *a* and *b* and sharing that join node between the two rules). On the other hand, this allows one to influence how the network is built in the light of application data. In section 6.5.2 we show differences in node sharing for different condition orderings.

6.1.5 Suitability of Rete networks for NLP

Rete networks seem to be most useful if there is a large number of rules with repeated occurrences of antecedents since this allows the network to share match results. In addition, Forgy (1982) establishes a number of general conditions that must be satisfied for Rete to be employed successfully. Firstly and probably most importantly, the knowledge base must change "relatively slowly". The reason for this is that there is a cost involved in updating the Rete network which grows with the number of changes to the knowledge base, i.e. the number of asserts and retracts. This cost must be lower than the alternative, cycling over rules which still can use efficient indexing techniques, for example. If the knowledge base is used as a store for completed chart edges, it is only monotonically increased. Usually, chart edges are not removed once they have been added to a chart (unless one wanted to claim back memory space or model human forgetting, for example). Therefore, for chart-based NLP there is no problem involving volatile edges. There are no volatile edges since there is no retracting of facts from the KB. Thus, there does not seem to be a reason why maintaining state across cycles should involve any additional cost apart from memory requirements and the need to compute the spread of facts through the network. It is always possible that the result of a successful matching operation in one cycle is eventually used at a later one.

Furthermore, Forgy (1982) mentions that Rete networks require facts in the knowledge base to be immutable. If facts were allowed to change, references to facts in the network could become incorrect. This poses no additional constraints on chart algorithms since immutability of chart edges is generally assumed. Furthermore, in production languages, facts do not contain variables that could be bound to outside objects which may result in changes to the fact's match results.

6.1.6 Other state saving match algorithms

Rete networks are the most widely used method for organizing production rules. Several alternatives have been proposed. Many of them address the problem of join ordering, i.e. of sequentializing the network. In the following, we give an overview of some of the proposals.

The Treat algorithm (Miranker, 1990) is probably the most well-known alternative to Rete. It has one-input nodes with associated alpha memories but does not have a fixed network of join nodes. Instead, the join order is determined dynamically. Treat places less emphasis on memoing and does more recomputation of joins. Furthermore, it aims at reducing the amount of work to be done for retracting facts which in Rete generally is the same as for asserting facts. A sketch of a Treat network is given in figure 6.7.

```
(defrule rule-9
 (a)                        start
 (b)                    /  /  |  \  \
 (c)                    a  b  c  d  e
 (d)                    |  |  |  |  |
 (e)                  { any join order }
 =>                     \  \  |  /  /
 )                       FIRE: rule-9
```

Figure 6.7: Structure of Treat network

The performance of Rete versus Treat has been evaluated in the context of the Soar project (Laird et al., 1987). Nayak et al. (1988) report that Rete outperformed Treat on several tasks within the Soar project. Their analysis of the strengths and weaknesses of the algorithms is relevant for language processing as well since there are similarities between Soar and chart-based NLP: According to (Nayak et al., 1988), productions cannot retract facts from the knowledge base in Soar. In other words, the Soar knowledge base is monotonically increasing just as a chart is. This makes any recomputation of joins unnecessary and therefore gives an advantage to Rete which does more memoing. Furthermore, the Treat algorithm has to spend effort on determining the order of join computations at runtime. Rete, on the other hand, fixes the network at compile time. As a result, Rete indeed seems to be the better choice for chart-based NLP.

There are a number of other approaches that should be mentioned. Ishida (1988) aims at automatically optimizing the total cost of join operations based on execution statistics of earlier program runs. Bouaud (1993) presents the Tree algorithm, a state-saving match algorithm that has been developed for production systems with a restricted rule format. In contrast to Rete, where the network directly reflects the ordering of elements in the antecedent, Tree tries to reduce the size of the join search space heuristically. Furthermore, there has been theoretical work on 'Generalized Rete networks' that are characterized by arbitrary join structures. Lee and Schor (1992) define update algorithms for these generalized networks.

There is substantial work on scaling up production systems to large sets of rules. Dewan (1994) addresses the problem with respect to data bases and parallel processing. Doorenbos (1993, 1994) shows that production systems can deal with 100 000 productions and do not need to slow down as more rules are added. The key idea is to *unlink* left and right inputs to join nodes if the opposite input is empty. For example, if the left memory input of a join node is empty, there is no need to activate its right memory even if there is a matching fact in the associated alpha memory since that fact has no chance of filtering further through the network. However, if the left input becomes non-empty, the right input is linked into the network again. Doorenbos (1994) discusses the question how to combine left and right unlinking without disabling a join node completely. An interesting point about this approach is the assumption that productions are automatically learned which requires addressing scalability problems. NLP using much larger sets of (possibly automatically derived) rules could draw on this work.

6.2 Chart algorithms

Chart-based techniques explore the search space by keeping track of already performed subcomputations. Pereira and Warren (1983) characterise chart parsing within the logic programming paradigm. They describe Earley deduction as a general deduction procedure for definite clauses based on the Earley parsing algorithm (Earley, 1970). It uses unification to perform matching. We take Earley deduction in the following as an example of a principled approach to chart algorithms for NLP.

The Earley deduction procedure distinguishes between *program* and *state*. The program consist of grammar rules ('non-unit clauses') and lexical entries ('unit clauses').

Clauses are of the general form

(76) $P \leftarrow Q_1 ... Q_n$

P is the *head* or *positive literal* of the clause. Q_1 to Q_n are *negative literals*. Non-unit clauses have at least one negative literal; unit clauses have none. Literals are terms, i.e. predicate argument structures that are possibly nested. Non-unit clauses represent logical implications and are therefore very similar to productions in production systems under the assumption that the consequent of a production always asserts exactly one fact. Derived unit clauses correspond to derived facts in production systems. They represent proved consequences of a given set of initial unit clauses that are assumed to be true. In contrast to production systems which use relatively simple data structures and matching operations, Earley deduction uses term unification to perform matching. This is a major difference between the two approaches that effects how the notion of 'redundancy' is defined (see 6.4.1.2).

In Earley deduction, the state is a chart of unit and non-unit clauses that reflects the state of the inference process. The 'fundamental rule' combines items and performs inference proper. Non-unit clauses (corresponding to active edges – see figure 6.1, page 162) are reduced by unifying one of their negative literals with some unit clause (passive edge) which has no negative literals (anymore).

The fundamental rule only works on items in the state whereas the 'scanning rule' and 'prediction rule' introduce new items from the program into the state. The scanning rule adds unit clauses for lexical items to the state. The prediction rule introduces non-unit clauses from the program into the state but does not perform any inference steps proper. It selects those grammar rules (non-unit clauses) from the program whose positive literal unifies with the selected literal of the non-unit clauses of the state. Thus, it performs top-down prediction of grammar rules. Depending on the selection function for the next negative literal of non-unit clauses to be matched, left-to-right, head-driven or other strategies result.

In Earley deduction, an initial goal statement (stating, for example, that some string is a sentence) is proved successfully if it can be deduced from the program by repeated rule applications. A dot-notation is common for parsing a string from left to right, implying that the literal to be matched next is the one to the right of the dot. For example, a non-unit item $[NP \leftarrow Det \bullet N < 0, 1 >]$ states that we have found a *Det* between positions 0 and 1 and we are looking for an *N* to the right of *Det* next. For head-driven strategies, for example, one could require literal *N* to be matched first, requiring slight changes in notation.

Earley deduction mixes top-down and bottom-up processing as in the original Earley parsing algorithm. Other tree-traversal strategies like pure top-down and bottom-up can be implemented within this paradigm by modifying the order of inference rule applications, i.e. by varying the degree of 'eagerness' of the algorithm. For example, pure bottom-up processing only selects lexical entries by means of the scanner and uses the fundamental rule to recursively apply them to non-unit clauses of the grammar. It does not need to use the prediction rule (Erbach, 1997). Shieber et al. (1995) describe a common framework for various parsing strategies within the paradigm of *parsing as deduction*. Often, an agenda is employed to determine the next item to be processed. If the agenda is used as a last-in, first-out stack, the interpreter performs depth-first search. Breadth-first search is modelled by a queue, i.e. first-in, first-out. Other ordering functions can be used for best-first search.

The distinction between program and state in the Earley deduction framework gives a clear separation between those things that change during processing and those things that do not. In principle at least, chart items are always copies of program items.[1] They become further instantiated as items are combined. However, this does not affect the grammar rules and lexical entries in the program. For example, an input word might match a lexical entry of the grammar/program. This results in the introduction of a copy of the corresponding unit clause into the state. In a similar way, non-unit clauses of the state are always copies of non-unit clauses of the program. If they are selected by unifying a unit clause of the state with their head, for example, the copy of the non-unit clause of the program which is added to the state is further instantiated by this unification.

[1]In this discussion, we assume a direct copying approach for passive and active edges to highlight the difference between program and state. There are also methods based on structure sharing that might be more efficient for implementing the underlying programming languages.

6.3 Relating chart algorithms to production systems

To investigate the relationship between Rete-based algorithms and chart algorithms, it is useful to ask what parts of the Rete-based algorithm are constant and what are mutable. Obviously, the knowledge base of facts is part of the state. It is also clear that the compiled network of rules is part of the program in the sense defined above. However, rules in the Rete network can be further instantiated by matching facts just as a non-unit clause can when one of its literals is unified with a unit clause. As we have seen above, rules in production systems are allowed to contain variables. A Rete network needs to keep track of these instantiations to determine consistent matching of facts in the antecedent and to pass them on to the rule consequent, if required. Obviously, variable instantiations are part of the mutable state rather than the program.

Partial instantiations/activations of rules in the Rete network correspond to the active edges of chart algorithms. Since these instantiations are handled internally by the Rete network, they are more difficult to observe than the active edges of chart algorithms. (However, production languages such as JESS (Java Expert System Shell, Friedman-Hill, 2000) are able to display the fact combinations that activate the rules.)

When a production fires, only the consequent is asserted into the knowledge base. The facts matching the antecedent and instantiations of variables in the antecedent are 'left behind'. This is different from standard chart items which monotonically add information when combined with other items. In other words, passive edges, which are the best characterization of fully activated rules, are treated slightly differently in Rete-based chart algorithms. Only an instantiated copy of the consequent is added to the knowledge base rather than an instantiated copy of the entire production.

6.3.1 Agenda and conflict resolution strategy

A further point of comparison between Rete and chart algorithms concerns the notion of an agenda. In chart algorithms, the agenda can contain passive as well as active edges. In Rete networks, the output of the matching phase for a new fact presented to the network is a 'conflict set', i.e. the set of all newly activated rules. The production system has to make a decision about the order in which the rules should fire.

Like in chart algorithms, common strategies are depth-first and breadth-first search. Furthermore, some production languages allow the user to specify a pre-determined or dynamic rule ordering. In other words, Rete networks have an internal agenda of activated rules. In our implementation, we introduce an additional agenda outside the Rete network. This agenda collects all newly generated facts before they are presented to the network which allows us to freely manipulate the ordering in which new facts are used.

In chart algorithms, it is assumed that all inferences can eventually be drawn, which guarantees the completeness of the algorithm. The same is true for production systems: regardless of the internal agenda ordering (or our external agenda ordering), all facts (passive edges) can eventually be given to the network. An important point for using a production system instead of a standard chart algorithm for overgeneration-based NLG is that the space of possible realizations can be fully explored.

6.3.2 Structural redundancies

As described above, the basic idea of both Rete networks and chart algorithms is to use memoing techniques in order to deal with repetitive computations. For example, after matching *b* against the rules in (75), a chart algorithm has the following two active edges (matched elements are shown in bold face):[2]

(77)
$$a \leftarrow \mathbf{b}\ c$$
$$d \leftarrow a\ \mathbf{b}$$

A chart algorithm needs to match *b* against each rule individually. In contrast, in a Rete-based algorithm, *b* is matched only once. The key difference between Rete-based processing and standard chart algorithms therefore is the reduction of structural redundancies. This addresses the problem of within-cycle redundancies (see page 163). On the other hand, both Rete and chart algorithms reduce between-cycle redundancies by maintaining state across cycles. In section 6.5.2 we quantify the reduction of structural redundancies of our Rete-based chart algorithm with respect to our rule set.

[2]Matching in Rete is non-directional. In the context of this work, we assume the same for chart algorithms.

6.4 Bottom-up chart generation with Rete networks

To define a chart generator in a production system, we make a number of basic assumptions. The automatically constructed generation grammar rules are compiled into a Rete network. In the overall system architecture, the chart generator is interleaved with the ranker (depicted in figure 1.2, page 15). Both communicate via an agenda into which newly produced passive edges are added after they have been scored by the ranker. (As described in section 6.3, the rule instantiations that correspond to the active edges are tied to the Rete network.)

```
(1) Initially:
(2)     - for all input tags:
(3)         - add fact for tag to agenda

(4) Repeat until agenda is empty:
(5)     - move top-most agenda item (passive edge) to Rete network
(6)     - compute rule activations for item, i.e. move all the dots
(7)     - for all new items (passive edges) generated by
          fully activated rules:
(8)         - check redundancy
(9)         - if item not redundant:
(10)            - score item
(11)            - if item 'promising':
(12)                - add item to agenda
(13)                - adjust agenda ordering
```

Figure 6.8: Bottom-up chart generation algorithm

Figure 6.8 gives the basic outline of the algorithm.[3] Generation starts by adding facts representing individual input tags and their indices to the (external) agenda. Agenda items (passive edges) are then repeatedly matched against the Rete network, partially or fully activating grammar rules. This corresponds to 'moving the dots' in chart parsing. The top-most fact (passive edge) on the agenda is only presented to the Rete network after all consequences of the previously presented fact have been drawn, i.e. all possible dot movements have been computed. Thus, rules are executed in a breadth-first manner. All rules that can fire as a consequence of a new fact (passive

[3]We use the terms 'fact', 'item' and 'edge' interchangeably in the appropriate contexts. They all refer to passive chart items.

edge) being presented to the network do so before any more facts are taken from the agenda. Any Rete-internal rule ordering does not affect our algorithm, i.e. does not affect the result. The algorithm stops when the agenda is empty.

The implementation language we use is JESS (Friedman-Hill, 2000). It is closely related to the CLIPS expert system language (Riley, 1999). (The implementation of grammar rules as productions is described in appendix A.) The algorithm works bottom-up and can use the forward-chaining interpreter of production systems directly. As discussed in section 6.3, matching a fact (passive edge) against the Rete network corresponds to applying the fundamental rule in chart algorithms (in cases where the rule application is initiated by a passive edge). The generation algorithm is a variation of a standard bottom-up chart parser (Gazdar and Mellish, 1989).

Epsilon productions (i.e. rules of the form $X \rightarrow \varepsilon$) generally pose a problem for bottom-up processing since empty elements can be stipulated at any point without any restriction by top-down predictions. However, this is not a problem in our approach because the grammar construction algorithm eliminates all treebank traces.

Furthermore, bottom-up processing avoids termination problems due to left-recursion. There is no prediction rule so that there cannot be a prediction loop. Therefore, techniques that remove left-recursion from context-free grammars do not need to be applied (see Johnson and Roark, 2000, for example).

6.4.1 Changes to standard production systems

In order to use a standard production system for chart generation, we need to make a number of changes and additions. As pointed out already, our agenda is an additional data structure different from the Rete-internal one. We bypass the internal conflict resolution strategy by collecting all newly generated passive edges first before going into the next cycle. Presenting a fact (passive edge) to the Rete network can simply be done by taking a fact from the agenda and adding it to the knowledge base of the production system. This is implemented by a simple rule of lower preference than the grammar rules that requests a new agenda item after the grammar rules have finished firing.

The use of an (external) agenda enables one to carry out experiments using different agenda orderings for passive edges (see section 3.3, for example). The agenda ordering can depend on a simple 'time stamp' or a more elaborate ranking function (instance-based or otherwise). For time stamp-based search strategies like depth-first or breadth-first, it is assumed that all items are potentially useful. A more elaborate ranker might decide that an item is not 'promising enough' to be considered further so that it can be dropped. Like the agenda itself, the function that determines the agenda ordering is outside the realm of standard production systems. In the following, we discuss two further mechanisms that need to be implemented in order to use a production system for chart generation.

6.4.1.1 Non-overlapping semantics

The basic definition of grammar rules comprises semantic, syntactic and surface form features. An example is given in (78) (see section 5.4 for a more detailed description of the rule format and appendix A for the actual encoding of grammar rules as productions):

$$(78)\quad \begin{bmatrix} \text{SEM} & \boxed{1} \uplus \boxed{2} \\ \text{CAT} & \text{VP-POST\_DESCR\_ADJ} \\ \text{TERMS} & \langle \text{was named to } \boxed{3}\,\boxed{4} \rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SEM} & \boxed{1} \\ \text{CAT} & \text{NP-POST\_DESCR\_ADJ} \\ \text{TERMS} & \boxed{3} \end{bmatrix} \begin{bmatrix} \text{SEM} & \boxed{2} \\ \text{CAT} & \text{PP-POST\_NODET} \\ \text{TERMS} & \boxed{4} \end{bmatrix}$$

When edges are combined, a check needs to be performed to ensure that the resulting edge does not cover the same semantic input tag twice. For example, in (78) we need to make sure that $\boxed{1}$ and $\boxed{2}$ do not contain the same input tag. This check serves the same purpose as the constraint in parsing to consume each input word only once. However, since we are dealing with (unordered) sets of semantic tags rather than (ordered) lists of words, this check needs to be performed slightly differently. Our implementation uses set functions built into the production language. An alternative is to use bit vector representations of the semantics covered by the edges (Kay, 1996), and only allow the combination of edges whose bit vectors have empty intersections.

6.4.1.2 Redundancy checking

Redundancy checking is performed before newly generated facts are added to the agenda. This saves the potentially expensive effort of ranking facts that are blocked by the redundancy check anyway. We do not need to check items in the chart for redundancy since all those items had been added to the agenda before and therefore have been checked already. This begs the question when an edge should be considered redundant.

Generally, if only atomic symbols are used in the grammar (as in context-free grammars, for example), checking for equivalence between items is a simple identity check. However, if definite clauses (or feature structures encoded in definite clauses) are used, redundancy checking needs to block those new items that are subsumed by some chart or agenda item. This is because non-ground items are abbreviations of all their ground instances: if a new item is subsumed by some chart or agenda item, then the ground instantiations of the new item are a subset of the ground instantiations of this chart or agenda item. In other words, the new item would add no new information to the chart or agenda.

In our encoding of grammar rules as productions, we assume that the values of all attributes of facts representing chart edges are instantiated and redundancy is defined as identity of the slot values for `sem`, `cat` and `terms`. Therefore, redundancy checking is much less costly in this approach than in chart generation with unification grammars. The implementation of grammar rules as productions also makes use of other attributes, for example unique ids for facts (see appendix A). These are not considered by the redundancy check. However, the use of the `terms` slot needs to be further qualified. If it is used directly for determining edge equality, a problem arises in the case of generation input containing some tag more than once. For example, consider the following values of the `terms` slot of a VP:

(79) a. `was named [ POST_NODET vice president ] and [ POST_NODET assistant to the chairman ] of this [ COMP_DESCR maker of alcoholic beverages and consumer products ]`

b. `was named [ POST_NODET assistant to the chairman ] and [ POST_NODET vice president ] of this [ COMP_DESCR maker of alcoholic beverages and consumer products ]`

The only difference between the two edges is the order of the slot fillers for tag `POST_NODET`. The question arises whether these two strings should be treated as identical for the purposes of redundancy checking if they occur in two edges that otherwise have identical syntactic-semantic category and multi-sets of semantic indices. On the one hand, the phonological string of these edges is different, and one could argue that they should therefore be treated as dissimilar. On the other hand, this would allow all permutations of filler words for the same tag to be generated. There are examples in the domain corpus that have three posts, resulting in a corresponding extension of the search space.

The answer to this question needs to be given with respect to the ranking function. If the ranker scores the term representations in (79) differently, the corresponding edges should be regarded as different by the redundancy check. However, in our approach the ranker considers only tags and words outside tags but not filler words. We therefore ignore filler words for redundancy checking. The above examples are then both presented in the following way for the purpose of redundancy checking:

(80) `was named POST_NODET and POST_NODET of this COMP_DESCR`

Using this representation, one of the edges in (79) will be blocked if the other is already in the chart or agenda. It should be noted that linear order still matters in this representation. A different word order of non-filler words is still treated as being different:[4]

(81) a. `was named [ POST_NODET vice president ] of this [ COMP_DESCR maker of alcoholic beverages and consumer products ] and [ POST_NODET assistant to the chairman ]`

b. `was named POST_NODET of this COMP_DESCR and POST_NODET`

Since the representation in (81b) – which is used for redundancy checking – is different from the one in (80), the corresponding edges cannot block each other.

The redundancy check can be efficiently implemented using a tree-structured representation, first checking the syntactic-semantic category, then the semantic indices, then the reduced content of the `terms` slots.

[4]We are not considering nuances in semantics at this point.

However, if index checking is performed, we also store the number of tag indices associated with the edge in question. This allows us to prefer edges with less indices (see section 5.5).

Finally, it should be noted that the derivation is not considered by the redundancy check. This means that the derivation of blocked edges is lost, and only the one of the blocking edge which is already in the chart or agenda survives. If we were interested in keeping the derivations of all edges, we could store information about derivations separately and recover them after processing has finished. This technique is used in two-step parsing (see Shieber et al., 1995). Alternatively, one could introduce disjunctive representations for the derivation encoding of chart edges. In such an approach, the blocking edge keeps the derivation of the blocked edge in addition to its own derivation.

6.5 Match savings of Rete networks

Employing a Rete network for rule matching avoids cycling over all rules in turn. In the following, we aim to examine – at a reasonable level of abstraction – how much work is actually being saved in comparison to standard chart algorithms. We first look at a simple example and show how we estimate the expected matches in standard chart algorithms. We then empirically investigate the number of matches with respect to an automatically derived generation grammar.

6.5.1 Scalability of Rete networks

In Rete networks, the number of matches in the pattern network increases linearly with the number of facts whereas the number of matches in the join network increases quadratically. This is because in the pattern network, a successful match result for a fact can be used in all productions that share the corresponding condition. In contrast, in the join network, the cross product of all combinations of facts needs to be checked in principle.

```
(defrule rule-10                          start
  (a ?X)                                  /  \
  (b ?X)                                  a   b
 => )                                     \  /
                                           ?X
                                            |
                                     FIRE rule-10

Facts in KB:
            (a 1)    (b 1)   |  (a 1)     (b 1)   |  (a 1)     (b 1)
            (a 2)    (b 2)   |  (a 2)     (b 2)   |  (a 2)     (b 2)
                             |  (a 3)     (b 3)   |  (a 3)     (b 3)
                             |                    |  (a 4)     (b 4)
                             |                    |
number of facts:     4       |       6            |        8
                             |                    |
pattern matches:     4       |       6            |        8
join computations:   4       |       9            |       16
 (successful joins:     2    |          3         |             4)
total:               8       |      15            |       24
```

Figure 6.9: Scaling w.r.t. the number of facts in Rete network: matching one rule

6.5.1.1 Matching with a single rule

Figure 6.9 shows a simple example (memory nodes are omitted). Production `rule-10` requires two ordered facts that share variable `?X` to match its conditions. Two `a`-facts and two `b`-facts in the KB require four matches in the pattern network and four variable comparisons in the join network. The pattern network matches the facts `(a 1)`, `(a 2)`, `(b 1)` and `(b 2)`. (We only count successful matches in the pattern network, not match attempts. We believe this is justified under the assumption of a simple indexing scheme for the atomic heads of facts and conditions.) The join network checks the compatibility of the variable instantiations for the pairs of facts `(a 1)-(b 1)`, `(a 1)-(b 2)`, `(a 2)-(b 1)` and `(a 2)-(b 2)`. Two of these four checks are successful. Doubling the number of `a`-facts and `b`-facts results in 8 pattern matches and 16 join computations (4 of which are successful).

The question now is how these observations translate into chart algorithms. For the purpose of comparing algorithms, wc rcgard matching in chart algorithms as also consisting of two phases: matching facts against conditions in isolation, and checking variable consistency. For a grammar that only contains a single rule as in figure 6.9, the matching effort for Rete and chart algorithms will be the same.

A possible point of confusion when comparing Rete and a chart algorithm in this way is the creation of new active edges in chart algorithms. For example, after rule `rule-10` matches two `a`-facts, there are two active edges (`a1.b` and `a2.b` in dot-notation) that wait for matching `b`-facts. The Rete network does not create such active edges (see section 6.3). However, it also keeps track of the rule instantiations (and maintains them across cycles similar to chart algorithms). It is difficult to estimate which of the two methods of representing rule instantiations/activations is more efficient. For the purpose of this comparison, we assume that both involve equal costs.

6.5.1.2 Matching with a more than one rule

The previous example did not exhibit any differences between Rete and chart algorithm. Since the main difference between Rete networks and chart algorithms is the removal of structural redundancies, we need to look at an example that exhibits such redundancies.

Figure 6.10 shows a Rete network for the rule of the previous example and one more rule that has the conditions of the first as its prefix. For the same sets of facts in the KB, the matching cost in a Rete-based approach is exactly the same as in the single rule case since all network nodes belonging to the first rule are shared.

In contrast, if we interpret chart algorithms as approaches that have separate networks for each rule and do not allow any structural sharing between them, each network needs to be traversed individually. This is depicted in figure 6.11.[5] As for the Rete network, we assume indexing of fact/condition-heads and count only successful pattern matches. The chart algorithm has to perform twice as many pattern matches as the Rete algorithm since there are two grammar rules. For example, for 4 input facts we now perform 8 successful pattern matches rather than just 4. A similar situation arises w.r.t. the join computations, i.e. the variable unifications: for 4 input facts (2 `a`-facts and 2 `b`-facts), we perform 4 comparisons in each rule network, resulting in 8 comparisons overall (rather than just 4 for Rete). For larger numbers of input facts, the

[5]Figure 6.11 lists the facts that match the pattern network of each rule separately. However, the KB contains exactly the same facts as in the Rete case (see figure 6.10).

```
Productions:

(defrule rule-10           (defrule rule-11
  (a ?X)                     (a ?X)
  (b ?X)                     (b ?X)
 => )                        (c)
                            => )
----------------------------------------------------------------------------
Rete:              start
                  /  |  \
                  a  b  c
                  \  /  |
                   ?X---|
                   |    |
         FIRE rule-10 rule-11

Facts in KB:
           (a 1)    (b 1)  |  (a 1)    (b 1)   |  (a 1)    (b 1)
           (a 2)    (b 2)  |  (a 2)    (b 2)   |  (a 2)    (b 2)
                           |  (a 3)    (b 3)   |  (a 3)    (b 3)
                           |                   |  (a 4)    (b 4)
                           |                   |
number of facts:     4     |        6          |        8
                           |                   |
pattern matches:     4     |        6          |        8
join computations:   4     |        9          |       16
 (successful joins:     2  |           3       |           4)
total:               8     |       15          |       24
```

Figure 6.10: Scaling in Rete network: matching two rules

```
Chart algorithm: individual networks

  start                  start
  /  \                  /  |  \
  a  b                 a   b  c
  \  /                  \  |  /
   ?X                    ?X--|
   |                         |
 rule-10                  rule-11

Facts in KB matching rule patterns (only one instance of each fact in KB):
rule-10:  (a 1)    (b 1)  | (a 1)    (b 1)  | (a 1)    (b 1)
          (a 2)    (b 2)  | (a 2)    (b 2)  | (a 2)    (b 2)
                          | (a 3)    (b 3)  | (a 3)    (b 3)
                          |                 | (a 4)    (b 4)
rule-11:  (a 1)    (b 1)  | (a 1)    (b 1)  | (a 1)    (b 1)
          (a 2)    (b 2)  | (a 2)    (b 2)  | (a 2)    (b 2)
                          | (a 3)    (b 3)  | (a 3)    (b 3)
                          |                 | (a 4)    (b 4)
                          |                 |
number of facts:    4     |        6        |        8
                          |                 |
pattern matches:    8     |        12       |        16
join computations:  8     |        18       |        32
 (successful joins:    4  |          6      |           8)
total:             16     |        30       |        48
```

Figure 6.11: Scaling in chart algorithm: matching two rules

hypothesized chart algorithm always performs twice the number of joins computations (variable unifications) than the Rete case.

The total number of match savings achieved by the Rete network is expected to become more dramatic if we increase the number of rules. It should be noted, however, that the above example allows the Rete network to share the maximal number of nodes. On the other hand, for larger rule sets, even non-maximal sharing will be beneficial. This will be investigated empirically in section (6.5.2).

6.5.2 Match savings for grammar rules

The previous section discusses match savings for Rete networks that can be expected in the general case. In this section, we quantify match savings with respect to a Rete network of 381 productions that were automatically constructed from 104 annotated treebank sentences. The productions are described in more detail in appendix A. Here, we are only interested in the matching behaviour of the Rete network built from these rules. In order to compare Rete and chart algorithms, we again assume that chart algorithms perform matching in a pattern and a join network. However, as in the previous section, we assume that standard chart algorithms do not allow any sharing of match results across sets of rules in the same 'cycle', i.e. for the same fact (passive edge) that is taken from the agenda and added to the chart.

6.5.2.1 Pattern network

The amount of sharing in the pattern network depends on the frequency of the different types of conditions in the grammar. These conditions match the corresponding types of unordered facts. For the most frequent type of condition, `NP-POST_NODET` occurring 46 times, for example, employing a Rete network means that matching is performed only once rather than than 46 times for a single edge matching this condition. Put another way, a single match partially activates 46 productions. This is what we set out to achieve by employing a Rete network. In contrast, a standard chart algorithm needs to perform matches on all rules individually. On the other hand, there are no savings in the pattern network for conditions that only occur once in the grammar. Thus, measuring the actual amount of savings in a generation system runs requires one to look at the actual facts asserted into the KB. There will be more savings if facts tend to match frequent conditions rather than infrequent ones.

After generation has finished for some semantic input, we can examine the facts in the KB and relate them to the frequency of the conditions they match. Since the frequencies of conditions in productions can be interpreted as the number of matches required in standard chart algorithms, the sum of the condition frequencies for the facts in the KB represents the number of pattern matches for chart algorithms. In contrast, Rete networks only perform as many pattern matches as there are facts in the KB.

input	A	B	C	D	E	F	G	H	I	J
KB edge facts	86	144	189	166	98	162	232	237	266	337
α-matches: Rete	92	150	195	172	103	167	238	242	271	342
α-matches: chart	885	1510	2578	2893	1214	2254	2096	2204	2021	2335
Rete savings	793	1360	2383	2721	1111	2087	1858	1962	1750	1993
in %	89.6	90.1	92.4	94.1	91.5	92.6	88.6	89.0	86.6	85.4

Figure 6.12: Pattern matches in Rete network versus chart algorithm

Figure 6.12 details the number of edges for individual system runs and the number of matches in the pattern network for Rete and chart algorithms ("α-matches"). The example runs were carried out using the expectation-based ranker and standard system settings as in the experiments described in section 3.3. Here, the number of facts in the KB is most relevant since these are presented to the Rete network.

The number of pattern matches in the Rete network is the same as the number of chart edges (unordered facts of the KB) plus the number of initial facts for the input tags (usually about six). The initial facts do not count as chart edges because they were not generated by the grammar (see also our encoding of input rules in appendix A). The difference between Rete and hypothesized chart matches shows how many matches in the pattern network have been saved with respect to individual generation inputs by using a Rete network. For example, input A yields 86 facts in the KB. These facts, together with 6 input facts, result in 92 matches of facts against conditions in the pattern network. In contrast, a standard chart generator needs to perform 885 matches since it looks at all rules in isolation. Therefore, using a Rete network saves 793 matches, or 89.6%. The average savings for all inputs are 90%. Differences between inputs with respect to the number of saved matches for similar numbers of KB-facts can be explained by the varying frequency of conditions and of heads in KB-facts.

6.5.2.2 Join network

The figures for match savings in the pattern network have to be seen in the context of the work required in the join network. One example of checks performed in the join network is the test for non-overlapping semantics of chart edges ('consumes-check', see section 6.4.1.1) which obviously requires information from more than one edge. Generally, the number of consumes-checks corresponds to the number of join tests if all grammar rules are binary. This is largely true for the automatically derived grammars in this approach (see section 5.6). We can count how often the consumes-check is performed in the generation runs (and also how often it is successful or fails). This measures the number of combinations of facts that are tested by the consumes-check for those facts that filter successfully through the pattern network.

input	A	B	C	D	E	F	G	H	I	J
α-matches: Rete	92	150	195	172	103	167	238	242	271	342
consumes checks	863	1742	11415	11542	2162	6797	5710	6153	4554	6892
success	233	336	1102	468	507	1473	1686	543	547	680
failure	630	1406	10313	11074	1655	5324	404	5610	4007	6212

Figure 6.13: Computations in the join network: placing unique condition first ('chart')

Figure 6.13 gives the number of consumes checks for the same inputs and system parameter settings as in the previous table. For example, there are 863 consumes-checks for input `A` (233 of which succeed whereas 630 fail). In contrast, there are only 92 matches in the pattern network for the same input.

Since combinations in the join network scale quadratically in general, it is not surprising that the number of consumes-checks tends to be higher than the number of pattern matches even for a chart algorithm (see figure 6.12).

6.5.2.2.1 Condition ordering The counts in figure 6.13 have been obtained by using rules that place a specific condition matching a unique rule-id first (for example: '`input-rule-23`'; see appendix A). This effectively simulates the performance of chart algorithms with respect to the join network since productions never share a common prefix. We can also ask how many join nodes are actually shared for the rule set at hand, and what effect this has on the number of consumes-checks if the unique conditions are placed last.

fact-id $\longrightarrow$	none	last ('rete')	first ('chart')
α-unshared	485	1251	1494
α-shared	1328	1328	1073
β-unshared	238	621	639
β-shared	20	20	0

Figure 6.14: Number of shared and unshared nodes in Rete network

Figure 6.14 gives the number of shared nodes in pattern and join network (α- and β- networks, respectively) for the different condition orderings for the grammar we are using throughout these experiments. (Note that the sharing in the Rete network does not depend on the generation input.) The high overall number of nodes can be explained by the fact that the JESS rule compiler produces nodes for subtests on facts. For example, a pattern matching a structured fact is represented by two nodes in the pattern network.

If no conditions matching rule-ids are used at all (first column in figure 6.14), the number of shared join nodes is the same as if they are placed last (second column). This is because rule-ids are unique and the tests cannot be shared. There is no sharing at all if rule-ids are placed first (third column) because there can be no common prefix among productions. In this case, rule-ids act as *guards* or *control facts*.[6]

Placing the unique condition last results in 20 join nodes out of a total of 641 being shared. This seems to be a fairly small number. In principle, join nodes may be shared across a large number of rules so that they may save more work than this number suggests. However, when we compare the number of shared and unshared β nodes in chart and Rete networks in table 6.14, we see that the savings are indeed small. In general, join node-sharing takes place in two cases:

[6]It is also noteworthy that the number of shared and unshared nodes in the pattern network seems to change slightly depending on the position of the rule-id. We attribute this to the specific Rete compiler used in the experiments.

1. The sequence of conditions in one production is a prefix of another production. For example, a ternary rule might share its first two daughters with a binary rule.

2. Productions have identical conditions but different consequents. For example, two binary rules might match the same daughters but have different mother nodes.

The low number of shared join nodes can be explained by the fact that, excluding rule-ids, most rules are binary (see section 5.6) and they are obviously not identical since identical rules are filtered out by the grammar construction procedure. For the majority of rules, only sharing in the second case is a possibility.

input	A	B	C	D	E	F	G	H	I	J
α-matches: Rete	92	150	195	172	103	167	238	242	271	342
consumes checks	756	1742	11383	11542	2090	5667	4938	5753	3832	6492
success	216	336	1070	468	471	1183	1439	538	488	675
failure	540	1406	10313	11074	1619	4484	3499	5215	3344	5817
% id last vs id first	-12.4	0	-0.3	0	-3.3	-16.6	-13.5	-6.5	-15.9	-7.3

Figure 6.15: Computations in the join network: placing unique condition last ('Rete')

6.5.2.2.2 Effects of condition ordering We can compare the number of consumes checks for productions that place rule-ids last to those that place them first, effectively disabling any join sharing. Figure 6.15 shows the changed number of consumes-checks for the same inputs and parameter settings as in the previous tests. The last row in figure 6.15 shows the reduction in the number of consumes-checks for the two condition orderings. Placing rule-ids last is always at least as good as placing them first, and in some cases up to 16% better. The average improvement is 7.6%. These savings are a systematic effect that can be explained by the sharing of join nodes in the Rete network. Differences for specific inputs are due to the fact that not all productions are able to share results.

The differences between the two condition orderings seem to be rather small. This finding is in line with other studies on relatively small sets of rules showing that improved sharing in the join network only produces very limited speedups (Doorenbos, 1993, p294). It may be the case that for language processing with (largely) binary rules, the amount of join node sharing that is possible remains fairly limited. However, especially for larger rule sets, even a small amount of join sharing is beneficial.

Although for our grammar of 381 productions the differences between the Rete network (estimated by placing rule-ids last) and chart algorithms (estimated by placing rule-ids first) are relatively small concerning join computations, gains in the pattern network are still an advantage of Rete networks. The 7.6% improvement in the join network seems rather modest compared to the 90% improvement in the pattern network.

From a processing point of view, sharing in the pattern network could be exploited best when the linguistic representations used in the grammar formalism become more complex. Furthermore, it is preferable to limit the use of shared variables and try to do as much work as possible in the pattern network. In our approach, shared variables are only used to perform tests that require information from two facts, most notably the check that edges have no overlapping semantics. The tag index treatment (see section 5.5) is another candidate for this. In the practical implementation, the test is performed in the rule antecedent since it rarely fails (see appendix A). In general, however, shared variables will be necessary to model agreement, for example, unless one wanted to use only ground values for the corresponding attributes in conditions which in turn would require a larger number of productions.

6.6 Related work on NLG using production systems

The `Ana` stock report generation system (Kukich, 1983, 1988) is an early example of a knowledge-based approach to NLG and was implemented in a production system language (OPS5). `Ana` is an example of the explicit-choice model of generation. The input to the system are stock quotes. `Ana` first generates the facts to be expressed, then identifies appropriate "messages", groups these according to topic and realizes them with a phrase-based grammar. Although the general processing strategy is top-down, realization is performed bottom-up by a "clause-combining grammar" that builds up sentences from phrases originating from a phrasal lexicon. As we have seen in this chapter, the forward-chaining interpreter of production language encourages this kind of processing strategy.

Our automatically constructed grammar shares some characteristics with a phrasal grammar such as the one used by `Ana`. The elementary units of processing are often larger than single words, limiting the number of rule applications required for building up a sentence. In our system, the rules are derived systematically and conform to the bracketing structure of the treebank.

A production system language was also used to analyse and generate natural language in the NL-Soar project which formed part of the larger Soar project (Laird et al., 1987). The grammatical framework is based on Chomsky's Government and Binding theory. NL-Soar distinguishes between 'utterance model', used for building up syntactic structure, and 'situation model', used for semantic interpretation.

In the utterance model of NL-Soar, each node in a syntax tree is represented by an unordered fact. The fact's head represents a unique node-id, attribute names are `bar-level`, `category`, `head` and `spec` amongst others. In contrast to the representations used in our approach, which basically associate facts with edges and therefore with local trees, NL-Soar 'distributes' tree representations across facts. A result of this choice of representation is that more work needs to be done in the join network. To match nodes that are part of the same tree, instantiations across facts need to be identical. As described in section 6.5.1, this involves more work than matching in the pattern network.

6.7 Conclusion

The idea put forward in this chapter is to address the matching problem for large sets of automatically generated grammar rules by employing techniques developed for production systems. To our knowledge, there has been little overlap between the NLP community and the production system community although both have developed mature techniques over the years. To bridge the gap, we systematically relate chart parsing algorithms to production systems and show empirically that Rete networks have advantages over standard chart algorithms with respect to the amount of matching they perform. This is due to the exploitation of structural similarities between rules in Rete networks. The gains in the pattern network of Rete-based rule systems are larger than those in the join network, especially if the rules are mostly binary. For this reason, it seems advisable to choose grammar representations that limit the use of shared variables in productions and to try to maximize join node sharing.

Chapter 7

Evaluating the instance-based generator

In the previous chapters, we described the two basic components of the hybrid generation system: the instance-based ranker on the one side and the rule-based candidate generator on the other. In this chapter, we investigate the output of the system and its general behaviour. We divide our experiments into two categories: 'gold standard' experiments and experiments based on manual exploration. Gold standard experiments divide the corpus into training and test set and use the test set semantics as input to the generator. This evaluates the system in a way that is empirically justified with respect to both the individual inputs and the distribution of these inputs.

The experimental setup of gold standard experiments is depicted in figure 7.1. The figure is simplifying matters a bit because it does not show the interleaved architecture of the generation system (see section 1.2.5, figure 1.2). As in previous experiments, we extract inputs from the semantically annotated test corpus and match them against the grammar rules. Instances and grammar rules are derived from the semantically annotated training set. The edges produced by the grammar rules are scored with respect to the instance base. The resulting candidates can be compared to the original sentences (this is discussed in section 8.1). An important characteristic of our approach is the separation of individual corpus examples (or the resources derived from them) throughout the generation system: the items of the instance base as well as the grammar rules are always individually motivated by annotated treebank parses.

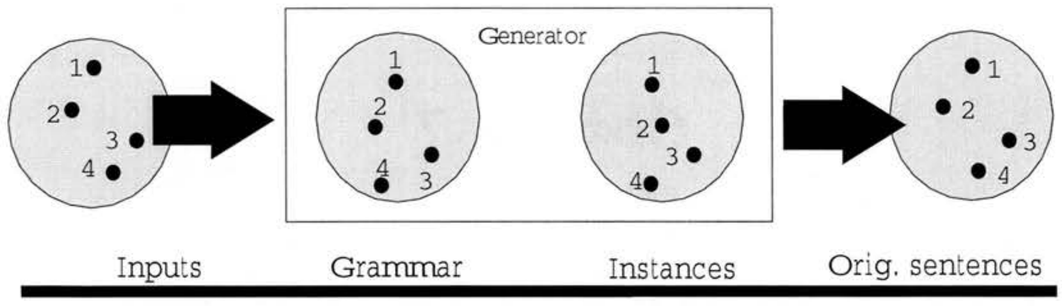

Figure 7.1: General experimental setup

However, the empirical gold standard methodology does not test the system on ill-formed input and it does not probe the generator systematically by searching for particular weaknesses. Such experiments need to be conducted by using hand-crafted input. This is necessary to obtain a more complete picture of the capabilities of the generation system.

7.1 The need for ranking

The grammar interpreter generates large numbers of candidates depending on the size of the input and the individual tags involved. For example, inputs of size 6 typically yield about 10000 candidates. In chapter 3 we investigated the efficiency implications of such an overgeneration approach. In this chapter, we look at the actual candidates chosen by the ranker. The need for ranking arises from the fact that many candidates are ungrammatical, unintelligible or at least non-fluent. Thus, we expect the ranker to identify the good candidates among the many 'bad' ones.

The generated candidates contain intelligible pieces since our grammar is phrase-based, i.e. many grammar rules generate several words at once (see section 5.6). Furthermore, the slot fillers also often contain several words. As a consequence, bad candidates typically combine phrases in the wrong order but they do not exhibit random permutations of the words available to the grammar.

Lord Chilver , 63 to English China Clays PLC , , chairman , , was elected an nonexecutive director of the chemical company .
Lord Chilver , chairman , at English China Clays PLC , 63 , was elected an nonexecutive director of the chemical company .

James L. Pate , 54 , , this oil concern 's , was elected a director .
James L. Pate , from this oil concern , , 54 years old , was elected a director .

from this metals and industrial materials maker said its board elected Michael Henderson of chairman .
Michael Henderson , group chief executive , , , This metals and industrial materials maker , , 51 , was appointed chairman , effective in May , succeeding Ian Butler .
this metals and industrial materials maker 's group chief executive said Michael Henderson was named chairman succeeding Ian Butler .

Scott C. Smith , formerly former chief financial officer was elected senior vice president at the media concern .
Scott C. Smith , formerly vice president, finance, , , formerly chief financial officer , was elected senior vice president at the media concern .

Charles A. Pearce, 66, will retire This bank holding company Dec. 31.
Charles A. Pearce , chief executive officer , , 66 , , will retire this bank holding company 's Dec. 31 .

Frank Nicastro, this closely held supermarket chain's, was named treasurer.
Frank Nicastro, This closely held supermarket chain, was named vice president.
Frank Nicastro, from this closely held supermarket chain, was named treasurer.
Frank Nicastro, from this closely held supermarket chain, was named treasurer, and vice president.
the closely held supermarket chain's said Frank Nicastro was named treasurer.
from this closely held supermarket chain said Frank Nicastro was named treasurer.
from this closely held supermarket chain said its board elected Frank Nicastro of treasurer and of vice president.

Drug Emporium Inc. 's said its board elected Gary Wilber , This drugstore chain , , former chief operating officer , of chief executive officer .
Drug Emporium Inc. 's said its board elected Gary Wilber , formerly formerly former president and formerly chief operating officer of chief executive officer .
Drug Emporium Inc. said its board elected Gary Wilber , formerly formerly formerly chief operating officer and formerly president as chief executive officer .

Terry L. Haines , this plastics concern 's , has been named vice president, North American sales, , both new posts .
Terry L. Haines , from this plastics concern , was named vice president, North American sales, , the new posts .

Figure 7.2: Some rather bad candidates

Figure 7.2 shows some bad candidates for a number of inputs. They often contain sequences of punctuation marks. Some punctuation marks are part of the slot fillers and are not visible to the ranker. Most, however, are generated by the grammar rules. Generally, the ranker treats punctuation marks like words since they are separate tokens in the instance representation. We do not apply any text polishing during generation.[1] However, in the following evaluations, we assume a simple text polishing function that applies the rules of absorption.

Finding particularly bad examples can be a problem in itself since in many cases exhaustive search is not possible due to memory problems. In these experiments, we tricked the expectation-based ranker into preferring bad edges by defining an inverted expectation $exp_{inv} = 1 - exp$. We then ran the system without pruning and a time limit of 3 minutes. (However, even this time limit leads to memory problems in some cases.) We then looked at the resulting candidate list from the bottom.

7.2 Reproducing the training set

A first test of an instance-based system should evaluate whether it has correctly learned the training set instances. In our hybrid generation system, this implies a number of issues:

1. Does the rule system reproduce the annotated training set template?
2. Does the ranker choose the correct candidate?

Quite obviously, training set reproduction is a gold standard experiment. It can be conducted at different levels of difficulty depending on how many rules and instances are available to the system. In the simplest case, we only allow the generator to use rules that were extracted from the treebank parse in question, and restrict the instance base to a single instance, the corresponding annotated training template. This effectively excludes issues of search and comes closest to testing the implementation

[1]The rules of absorption for punctuation (see section 4.4.2) only apply when the final candidates are presented to the user. The proposed duplication of punctuation in the markup scheme does not affect the system internal instance/edge representations since the additional commas are inserted into the slot filler and are therefore not visible to the ranker.

of grammar rule construction and vector-based matching between candidate and instance. We can then increase the number of rules and instances and test whether the system still reproduces the training set instance in question.

In order to carry out these experiments, we need to be able to block and unblock grammar rules of particular treebank parses. The general mechanism for this is to use unique facts for rule names in rule antecedents. To unblock a particular rule, its rule name needs to be added to the chart as a fact (see appendix A). Since a grammar rule can be derived from more than one treebank parse but is only spelled-out once in the grammar, we associate each training set parse with a set of rule names where a rule name can occur in more than one such set. To unblock a particular training set parse, we release the rule names associated with that parse into the chart.

7.2.0.2.3 Dealing with non-taggables When evaluating training set reproduction, it makes sense to test the system on as many inputs as possible. To this end, we treat the entire corpus as a training set and produce grammar rules for the entire treebank.

However, the question arises how to deal with sentences that contain areas regarded as non-taggable (see section 4.4). Generally, we do not want '?'-tags to be part of the generation input although it should be technically possible to use them like any other tag in the rule system. (Grammar construction does not regard them as special.) Except for rules that expect the presence of a '?'-tag, the grammar rules extracted from parse trees of sentences containing non-taggables are still generally useful.

Furthermore, we need to deal with instance representations containing '?'-tags. One option is to just omit terms that contain a non-taggable tag and keep the rest of the instance representation. However, such a reduced instance would not correspond to any corpus sentence and might even be ungrammatical. If we decide to keep the instance, the question arises how to weight terms containing non-taggable tags. Simply using the common $tf.idf$ weighting scheme (see section 3.1.2) does not seem to be justified because the weights of '?'-containing terms should not depend on the occurrence of other '?'-tags in the corpus. Thus, a high, fixed weight, for example one corresponding to a single occurrence of the non-taggable tag in the corpus might be appropriate. This allows us to still use the instance for similarity computations although we will never be able to get a similarity score of 1.0.

For training set reproduction, however, sentences containing non-taggable tags do not seem to be suitable. There are 6 of these in the corpus which leaves us with 138 corpus sentences for the reproduction experiments.

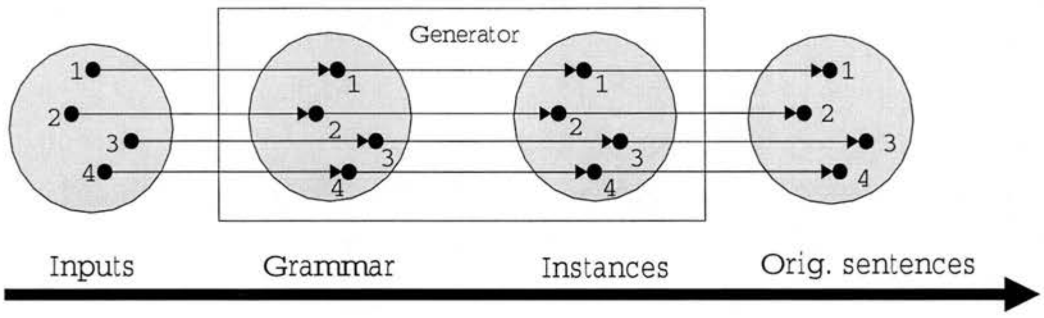

Figure 7.3: Experimental setup of narrow reproduction experiment

7.2.1 Reproduction with narrow rule set and instance base

The first set of reproduction experiments unblocks the grammar rules associated with the instance in question, and uses this instance as the only member of the instance base. This is depicted in figure 7.3; the numbers correspond to individual corpus sentences (and their parse trees and annotations). Even when only rules extracted from the instance parse in question are used, the rule system produces more than one candidate in 121 out of 138 inputs. In order to make the overall generation process transparent, we present one such example in somewhat greater detail.

Figure 7.4 shows a number of representations involved in testing the reproduction of original sentence (82) which is given the semantic annotation (83):

(82) **original:** Robert P. Bulseco, 44 years old, was named president and chief administrative officer of this regional commercial bank.

(83) **annotation:** `[INPERSON_FULLNAME` Robert P. Bulseco`]`, `[INPERSON_AGE` 44`]` years old, was named `[POST_NODET` president`]` and `[POST_NODET` chief administrative officer`]` of this `[COMP_DESCR` regional commercial bank`]`.

(84) **template:** `INPERSON_FULLNAME , INPERSON_AGE years old , was named POST_NODET and POST_NODET of this COMP_DESCR .`

(85) **instance terms:** `{<INPERSON_FULLNAME ,>, <, INPERSON_AGE>, <INPERSON_AGE years>, <years old>, ... }`

(86) **generation input:**

COMP_DESCR	regional commercial bank
POST_NODET	administrative officer
POST_NODET	president
INPERSON_AGE	44
INPERSON_FULLNAME	Robert P. Bulseco

(87) **input rules:**

```
PP-COMP_DESCR → of this COMP_DESCR
NP-POST_NODET → POST_NODET
ADJP-INPERSON_AGE → INPERSON_AGE years old
NP-INPERSON_FULLNAME → INPERSON_FULLNAME
```

phrasal rules:

```
NP-POST_NODET → NP-POST_NODET and NP-POST_NODET
VP-POST_NODET → was named NP-POST_NODET PP-COMP_DESCR
NP-INPERSON_FULLNAME → NP-INPERSON_FULLNAME , ADJP-INPERSON_AGE ,
Sup-INPERSON_FULLNAME → NP-INPERSON_FULLNAME VP-POST_NODET .
```

(88) **generation output:**

a. (cos=1.0) Robert P. Bulseco, 44 years old, was named chief administrative officer and president of this regional commercial bank.

b. (cos=0.88) Robert P. Bulseco, 44 years old, was named chief administrative officer of this regional commercial bank.

c. (cos=0.88) Robert P. Bulseco, 44 years old, was named president of this regional commercial bank.

d. (cos=0.59) Robert P. Bulseco was named chief administrative officer and president of this regional commercial bank.

e. (cos=0.38) Robert P. Bulseco was named chief administrative officer of this regional commercial bank.

f. (cos=0.38) Robert P. Bulseco was named president of this regional commercial bank.

Figure 7.4: Example input for ‘narrow’ reproduction experiment

From the annotated sentence (83) we extract generation input (86). It should be noted that generally the ordering of the attribute-value pairs is not important. In this case, we presented the input in reverse ordering to the generator. The goal of the generator is to produce a template (84) which matches the corresponding instance in the instance base. However, instances and edges are not matched directly but transformed into sets of terms. For example, in the current experiment we used bigram terms, and hence the instance as well as the content of the `terms` slot of the edge are represented as bigrams (85).

The chart generator uses generation rules based on the context-free grammar (87). In this case, the rule system produces 6 candidates which are shown in (88). These candidates are produced by inserting the slot fillers into the generated templates. All six candidates are grammatical – which is not always the case, as we will see below – and are faithful to the input although not all candidates express the entire input semantics. The candidates have different similarity scores w.r.t. the only instance available. Since we only have a single nearest neighbour, more dissimilar but grammatical sentences like (88e,f) are bound to have a low score. The generated candidates show that a grammar based on a single parse tree is able to produce 'summarized' versions of the original sentence.

In order to obtain all candidates, we used a simple ranker without expectation that does not prune edges in this 'narrow' experiment. The experiment confirmed that the rule system is able to reproduce all 144 templates with a cosine of 1.0. We also wanted to know whether it reproduced the exact surface string of the original sentence. It turns out that this is not always the case because slot fillers of identical tags may be exchanged. For example, 'chief administrative officer and president' in the best candidate in (88) is not literally the same as 'president and chief administrative officer' in the original string (82). However, their template representation is identical and, thus, the redundancy check filters out one or the other, depending on the order in which the edges are generated. This in turn can vary on the ordering of the input (which otherwise has little effect). When we switch off redundancy checking, all 138 original strings are reproduced. However, the system also generates many permutations. The largest number of candidates for a single input now is 900, in contrast to 152 candidates for the same input when redundancy checking was used. Thus, in an overgeneration approach to generation with larger grammars, the redundancy check is indispensable.

Exchanged slot fillers, like the POST_NODET fillers of the first candidate in (88) for example, show that surface string reproduction is difficult to achieve when the chart generator does redundancy checking. However, the ranker is working on the template level, so that template regeneration should be what we are aiming for. In addition, there are practical problems in measuring string reproduction, for example whether or not punctuation is attached to the previous word or how special symbols like '&' are rendered ('"&"' does not literally match '&'). Unlike (88), exchanging slot fillers can also result in faithfulness problems which motivated the introduction of tag-indices (see sections 4.3 and 5.5). These are evaluated in section 7.7.1.

7.2.2 Reproduction with broad rule set and instance base

Having confirmed that the system is able to reproduce the original sentences when using the associated grammar rules, we now broaden the grammar and the instance base to all 138 corpus sentences. We still expect the original templates to be reproduced but search will be harder. We will therefore need to use the expectation-based ranker. The experiments described in chapter 3 give some impression of the number of candidates that can be generated by the rule system when search is not expectation-based. However, exhaustive candidate production is difficult to achieve in practice because the system tends to run out of memory after about 10 000 candidates. This demonstrates the need for efficient search algorithms like expectation-based search.

Figure 7.5 illustrates the changed experimental setup. The tags of input 1 are matched to rules derived from all treebank parses and edges can be scored using the entire instance base. The goal is to reproduce the template of sentence 1.

For reasons that will become apparent in section 7.4, in this experiment we require the generator to express more of the input semantics than it would naturally do. Since the instance base and the rule base in this experiments are an extension of those of the previous experiment, we know that template reproduction should still be possible. Running the generator on the 138 inputs resulted in 118 template reproductions. Of the 20 remaining inputs, 10 resulted in candidates that have a perfect score and express the

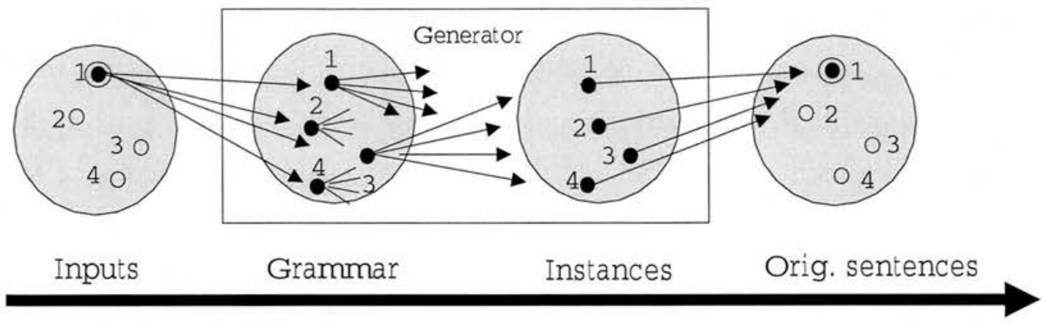

Figure 7.5: Experimental setup of broad reproduction experiment

entire input semantics – however, their nearest neighbour was not the original template (therefore that template was not regarded as reproduced). In other words, the generation system found another instance that expressed the input semantics perfectly well. Some examples of original sentence and best candidate:

(89) a. **candidate:** Charles D. Way, president, was named to the additional post of chief executive officer of this restaurant operator.

b. **original:** Charles D. Way, president of this restaurant operator, assumed the additional post of chief executive officer.

(90) a. **candidate:** Frank Nicastro was named treasurer and vice president of this closely held supermarket chain.

b. **original:** The closely held supermarket chain named Frank Nicastro vice president and treasurer.

In (89) and (90), the original sentence is effectively recast by another sentence. Other examples involve simply exchanging 'increasing' and 'expanding' or 'named' and 'elected'. This is possible because the semantics of the original sentence is also expressed by another sentence. In fact, we might as well have just exchanged the slot fillers in the annotation templates without any generation. Our annotation scheme is suggesting that the output sentences are mere paraphrases of the sentences from which the input is taken. The examples show an important general point: reproducing the original template cannot be the aim of the generator. Rather, reproducing some template that fully expresses the semantics should be considered a 'successful reproduction'. For the reproduction experiments, we can assume that at least one such a template exists.

The remaining 10 best candidates in this reproduction experiment also had a perfect similarity score with the original nearest neighbour but still did not reproduce the template. This is possible because the ranker measures similarity at the level of ngram term representations of the generated template sequences, not of these sequences themselves. There can be cases in which a candidate is not identical to the nearest neighbour template but still gives rise to the same term representation. In this experiment, we represented edge contents and instances as bigram terms. In the following examples, some bigrams of the original sentence (at the template level) have been exchanged:

(91) a. **candidate:** George L. Manzanec, senior vice president of Texas Eastern Corp., 53 years old, was elected a group vice president of this natural-gas-pipeline concern.

b. **original:** George L. Manzanec, 53 years old, senior vice president of Texas Eastern Corp., was elected a group vice president of this natural-gas-pipeline concern.

In (91), the string ', 53 years old,' seems to have been moved to another position and the candidate does not seem to be as fluent as the original sentence. The problem occurs because bigrams cannot look beyond the comma so that subordinate clauses can be 'moved' if the overgenerating grammar licences it. In the following example, a similar swap of parts of the original template seems to take place:

(92) a. **candidate:** Sheldon B. Lubar, and John L. Murray, chairman of Universal Foods Corp., chairman of Lubar & Co., were elected to the board of this engine maker.

b. **original:** Sheldon B. Lubar, chairman of Lubar & Co., and John L. Murray, chairman of Universal Foods Corp., were elected to the board of this engine maker.

In (92b), the word sequence ', chairman of Lubar & Co.,' has been misplaced so that the resulting sentence is ungrammatical. This case is more difficult because it also involves tag-indices (which we are not using yet) to indicate the relationship between the persons and their OTHERPOSTs. However, the sentence is ungrammatical in any case and again this seems to be due to the use of bigram term representations.

In (93), the two incoming persons and their OTHERPOSTs have been swapped as well:

(93) a. **candidate:** F. Warren McFarlan, a professor at Harvard University's Graduate School of Business, and Leon J. Level, chief financial officer and vice president of this computer services concern, were elected directors, increasing board membership to nine.

b. **original:** Leon J. Level, vice president and chief financial officer of this computer services concern, and F. Warren McFarlan, a professor at Harvard University's Graduate School of Business, were elected directors, increasing board membership to nine.

The underlying templates of (93a) and (93b) are not the same because one of the persons has two OTHERPOSTs whereas the other has one OTHERPOST and an OTHERCOMP tag. Therefore, the redundancy check cannot block one at the expense of the other. However, once candidate (93a) has been found, it can be used as a threshold in the A^*-algorithm to prune away the other one, as has happened in this case. We can prevent this by using a non-strict threshold, i.e. by only pruning similarity computations that have an expectation strictly lower than the current threshold (see the ranking algorithm in section 3.2.1.2). This results in the generation of both (93a) and (93b). However, this leaves open the question how to make a decision between the two. On the other hand, candidate (93a) seems to be just as fluent as original sentence (93b).

7.2.2.1 Reproduction with trigram terms

The above observations suggest that trigram terms might be more appropriate representations because of the extended local context they provide. Running the reproduction experiment with trigram term representations yields 128 reproduced templates, i.e. 10 more than for bigram terms. The examples (91-93) now are all reproduced at the template level. The remaining 10 inputs that have no template reproduction are those that have a perfect match with another nearest neighbour (like (89) and (90)).

7.2.2.2 Reproduction with unigram terms

We have seen that bigram term representations can result in reproductions that exhibit 'moved' bigrams. Thus, for unigram terms we expect even more such movements. In the following, we give two examples of candidates whose word order has been changed:

(94) a. **candidate:** Roy E. Parrott, the company's since Sept. 1 chief operating officer and president, was named to its board.

b. **original:** Roy E. Parrott, the company's president and chief operating officer since Sept. 1, was named to its board.

(95) a. **candidate:** Joseph L. Dionne, chief executive officer of McGraw-Hill Inc. and chairman, was elected to the board of directors of this electronics manufacturer.

b. **original:** Joseph L. Dionne, chairman and chief executive officer of McGraw-Hill Inc., was elected to the board of directors of this electronics manufacturer.

Candidates (94a) and (95a) contain exactly the same words as the original strings. However, they are still not extremely different from the original sentences. In principle, a unigram-based ranker assigns the same score of 1.0 to any permutation of the words of the original sentence. On the other hand, the fragments which the grammar combines are relatively large and rule combinations are constrained by the syntactic-semantic category labels (as well as the check for non-overlapping semantics, in general) so that most permutations are not generated.

7.2.3 Summary

In this section, we investigated whether the instance-based generator is able to reproduce the training set. Our experiments show that the generator is indeed capable of producing candidates that exactly match the instance term representations. Table 7.1 summarizes the reproduction results for bigram and trigram term representations. The experiments show an important aspect of our approach: there can be more than one instance that completely expresses the input semantics. If the best candidate has a perfect match with a different instance from the one from which the input is taken, this is indicated by '`other NN`' in table 7.1. We regard these cases as equally 'correct' as a perfect match with the original sentence template.

	bigrams	trigrams
Reproduction (orig. NN)	118	128
Reproduction (other NN)	10	10
changed ngram order	10	-
sum	138	138

Table 7.1: Reproduction results (broad rule set and instance base)

If lower order ngram term representations are used, problems concerning exchanged ngrams can arise: the surface form of the candidate sentence template does not exactly match that of the original sentence although their term representations match perfectly. Table 7.1 shows that bigram term representations encounter problems of exchanged ngrams whereas trigram term representations do not. (It is difficult to obtain reliable figures for unigram terms because these can lead to efficiency problems for some inputs. In general, we expect a higher number of exchanged ngrams for unigrams than for bigrams.) Thus, the experiments suggest that trigram terms are preferable over bigram terms. However, there seems to be a trade-off between the amount of context – which points toward using ever larger ngrams – and the usability of the ngrams when matching unseen sequences – which points toward using short ngrams. We will take this point up again in section 7.3.1 where we introduce mixed-order ngram representations.

7.3 Evaluating the generator on test data

Proper evaluation of the proposed generation system needs to distinguish between training and test data. Figure 7.6 shows the resulting experimental setup. The main difference between (broad) reproduction experiments (figure 7.5) and test set evaluation is the need to exclude all test data from the generation system. In principle, one could again insist on reproducing the original sentence (as figure 7.6 suggests). However, this requirement seems too strict in practice. After all, we expect the generator to produce novel output for previously unseen input. At the very least, there can be more than one way of expressing the input as the repetition of some tag combinations in the corpus has shown. Thus, we will have to look at the actual candidates to get a picture of the output of the generation system. (The issue of test set reproduction is taken up again in section 8.1).

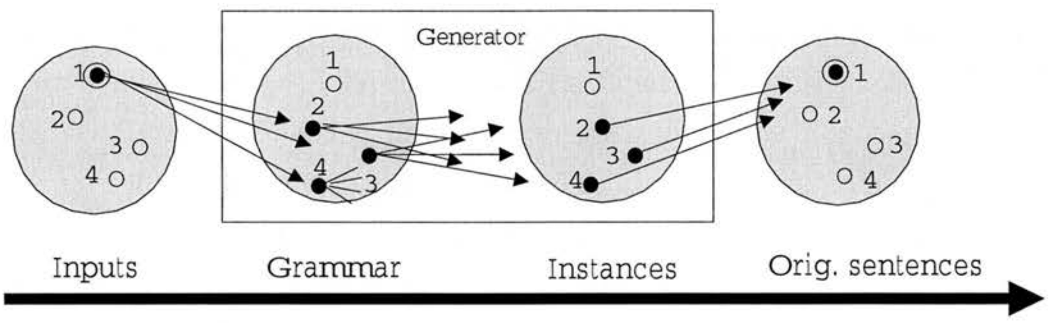

Figure 7.6: Experimental setup of test set experiments

We have already seen that our approach allows us to block and unblock particular grammar rules and instances. This is because both rules and instances can always point to the corpus examples from which they were derived. In order to evaluate the generation system on test data, it might seem that we could also proceed by blocking and unblocking grammar rules and instances of at least the corpus sentence from which the input is taken. This would allow one to do 'leave-one-out'-testing without the need to compile new instance models. However, there is a problem concerning $tf.idf$-based term weighting: the idf-computation depends on the 'instance frequency' of the terms, i.e. it does matter what and how many instances are considered to be part of the training set when the weights are computed. We therefore need to compile new instance weights for particular training sets to maintain a clean separation between training and test set. For most of the time, the generation system had been developed with a training set of 104 sentences and a test set of 40 sentences where grammar and instance base were compiled for the purpose.

7.3.1 Examining the output of the basic generation system

We first trained a basic version of the generation system on 104 sentences and tested it on 40 test set inputs. (96) in figure 7.7 shows the best candidates of a number of test inputs. (97) shows the corresponding original sentences. (96) also details for every candidate the similarity score ('cos') and the number of input tags expressed ('cov').

Almost all of the output sentences in (96) are syntactically correct. Furthermore, they are semantically faithful compared to the original sentences (97). Most candidates have a cosine score of 1.0. This means that their term representation exactly matches the instance representation. The overwhelming tendency of the ranker seems to be to choose perfectly matching candidates that do not fully express the input semantics. However, the basic version of the generation system also produces 9 candidates that have a cosine of less than 1.0. For outgoing events, the system settles for the same nearest neighbour in two cases. (96e) and (96f) are both motivated by an instance representation of (98) (we show the surface string and the template from which the term representation is extracted):

(98) a. A.F. Sloan, 60 years old, announced that he will retire next April as chairman and chief executive officer of this snack food and bakery products maker.

b. `OUTPERSON_FULLNAME` , `OUTPERSON_AGE` years old , announced that he `OUTPERSON-_PRP_VP_FUTURE` `OUTDATE_FUTURE` as `POST_NODET` and `POST_NODET` of this `COMP-_DESCR` .

The similarity scores for (96e) and (96f) w.r.t. (98) are different because they express different semantic tags. The input for (96f) does not provide any age information and does not express the company descriptor 'regional banking company'. In contrast, (96e) expresses the entire input semantics although the system cannot produce a candidate that matches the instance perfectly.

(96f) exhibits a syntactic error by using the wrong preposition in 'and of president'. We know that the context-free grammar overgenerates and does not have sophisticated grammatical knowledge. However, we would expect the ranker to make a better choice. A possible explanation for the preference for candidate (96f) can be found in the term representation. The trigram representation of the candidates yields many unknown

(96) **generation output:**

a. James L. Pate was named a director of this oil concern. (cos=1.0, cov=3/6)

b. Michael Henderson was named chairman of this metals and industrial materials maker. (cos=1.0, cov=3/10)

c. John F. Barrett, 40 years old, was named chief operating officer and president. (cos=1.0, cov=4/7)

d. Gary Wilber was named chief executive officer of this drugstore chain. (cos=1.0, cov=3/8)

e. Charles A. Pearce, 66 years old, announced that he will retire Dec. 31 as chief executive officer of this bank holding company. (cos=0.89, cov=6/6)

f. John F. McNair III, announced that he will retire on Dec. 31 as chief executive officer and of president. (cos=0.68, cov=5/10)

g. Arthur Price resigned from his executive duties. (cos=0.42, cov=2/7)

(97) **original sentences:**

a. James L. Pate, 54-year-old executive vice president, was named a director of this oil concern, expanding the board to 14 members.

b. Michael Henderson, 51-year-old group chief executive of this U.K. metals and industrial materials maker, will become chairman in May, succeeding Ian Butler, 64, who is retiring.

c. John F. Barrett, 40, formerly executive vice president and chief financial officer, was named president and chief operating officer, posts which had been vacant.

d. Drug Emporium Inc. said Gary Wilber, 39 years old, who had been president and chief operating officer for the past year, was named chief executive officer of this drugstore chain.

e. Charles A. Pearce, 66 years old, will retire from his post as chief executive officer of this bank holding company effective Dec. 31.

f. First Wachovia Corp. said John F. McNair III will retire as president and chief executive officer of this regional banking company's Wachovia Corp. and Wachovia Bank & Trust Co. subsidiaries on Dec. 31.

g. Arthur Price resigned as president and chief executive officer of MTM Enterprises Inc., a Studio-City, Calif., entertainment concern.

Figure 7.7: Some outputs of the basic generation system (using trigram term representations) and the original test sentences

terms so that the ranker only has a choice between candidates with large numbers of unknown terms. The ranker can obviously not distinguish between different unknown terms.

In general, one has to distinguish between *unknown* terms which have not been seen in the training set and *non-matching* (but known) terms which are not present in the instance representation at hand but do occur in other instance representations. It is not possible for the grammar to generate unknown unigram terms since all words and tags that can be used by the grammar are present in the instance base.[2] However, previously unseen higher order ngrams can be generated as the result of grammar rule combinations.

On average, we find that 40% of all terms extracted from the generated edges in this experiment (this includes edges that never contribute to forming a sentence) have not been seen in the training data. As explained in section 3.1.4, we deal with these cases by assuming a document (instance) frequency of 1 for the missing terms. A closer look at the non-matching terms of the best candidates revealed that the majority are actually not unknown. It is the low scoring edges and candidates that cause the high number of unknown terms. However, some of the non-matching terms of the best candidates have not been seen in the training set either. This shows that the ranker can select candidates that contains unknown terms. It is not confined to the known ones. From the perspective of the ranker, unknown terms are just like any other non-matching term.

To reduce the number of unknown words, we wanted to give the ranker a more fine-grained representation and compiled a new instance model that contained mixed unigrams, bigrams and trigrams. The idea is to allow the ranker to 'back-off' to decisions based on lower-order ngrams even if the trigrams involved are unseen in the training data. This reduces the number of unknown terms in the test set run to 10%.

Running the system with the mixed ngram model yields the following two best candidates for the problematic input:

(99) a. John F. McNair III, announced that he will retire on Dec. 31 as president. (cos=0.81,

[2]This is true as long as we use rules and instances derived from the same examples. Initial semantic ranking (section 7.6) can in principle choose subsets of different size of the grammar and the instance base so that the rules can generate words that are not present in the instance base. However, the only question that needs to be answered in such a case is – as before – how to weight the unknown terms.

cov=4/10)

b. John F. McNair III, announced that he will retire on Dec. 31 as chief executive officer. (cos=0.81, cov=4/10)

In this case, the system produces two best candidates (which is relatively unusual). Although the two candidates have the same template representation, the redundancy check does not block one at the expense of the other because they consume different sets of input tags. Compared to (96f), candidates (99a,b) avoid the syntactic error as expected but also express one tag less.

Since our choice of a trigram model was based on the reproduction experiments (section 7.2.2.1), we re-evaluated reproduction with the mixed ngram model and found that reproduction was as good as for the pure trigram model (see table 7.1).

The lowest scoring candidate in the trigram experiment is (96g). It realizes only 2 out of 7 input tags. The nearest neighbour of (96g) is shown below in its template form:

(100) OUTPERSON_FULLNAME , OUTPERSON_AGE -year-old POST_NODET of this COMP_NATIONALITY COMP_DESCR and POST_NODET chairman of COMP_SUBSIDIARY , the COMP_DESCR 's COMP-_SUBSIDIARY_DESCR_DEF , OUTPERSON_PRP_VP_PAST from his executive duties .

Candidate (96g) is not only very different from the original string but also from the nearest neighbour (as the low similarity score suggests). The sequence 'from his executive duties' in (96g) is part of the generation template and had not been marked in the annotation scheme (see (30b) in section 4.2.3). (96g) also lacks two POST_NODET tags that are present in the generation input although nearest neighbour (100) also contains POST_NODET tags. Again, the pure trigram model is the culprit. The rule system does produce candidates that express the POST_NODET tags of the input but these are ranked lower w.r.t. nearest neighbour (100) because the trigrams do not match. The output obtained from the system using the mixed ngram model supports this explanation:

(101) Arthur Price, chief executive officer and president, resigned from his executive duties. (cos=0.56, cov=4/7)

In contrast to (96g), candidate (101) realizes the post tags. In this case, the system sticks to the same nearest neighbour for both term representations. The increased number of matching terms results in a higher cosine score for (101). (However, cosine scores are generally not directly comparable when different representations are used.)

In one case there is no output at all. This seems to be due to the presence of a singleton tag `POST_INF` in the input the original sentence of which is shown in (102):

(102) Directors elected R. Marvin Womack, currently vice president/product supply, purchasing, to `[POST_INF` head`]` the company's Washington, D.C., office.

The `POST_INF` tag does not occur in the training data, and hence there is no input rule to build an initial phrase for that tag. This example shows that at least for gold standard testing singleton tags are of limited use. In other cases the system might be able to build candidate sentences without the unknown tag.

7.3.1.1 Explaining the lack of completeness

We have not yet explained why the ranker tends to choose candidates that match some instance representation perfectly but as a result also convey considerably less information than the original sentences. This is generally true irrespective of the term representation. In the trigram experiment, the candidates express only 55% of the input tags on average. Only 7.7% are complete. In the mixed ngram experiment, 56% of the input tags are expressed and 10% of the candidates are complete. The average sentence length of the 40 best candidates in the trigram experiment is 13.2 words and 13.9 in the mixed ngram experiment, in contrast to 21.9 words of the original candidates. The average template length (which replaces filler words by tags and keeps punctuation outside tags as separate tokens) is 9.1 tokens for the best candidates versus 16.3 for the original annotated sentences. Furthermore, there does not seem to be much variation in the generator output. For example, (97d) has been 'normalized' to the more standard (96d). 35 best candidates use 'named' in contrast to just 26 in the original sentences.

The explanation for the lack of completeness[3] of the chosen candidates lies in the nearest neighbours that are used by the system. The grammar is able to flexibly com-

[3]We use the words 'coverage' and 'completeness' interchangeably. The context should make clear whether we are referring to the coverage/completeness of individual candidates or the grammar.

bine input tags in various ways. This allows it in many cases to produce candidates that exactly match some instance, as demonstrated by the cosine score of 1.0 for the candidates (96a-96d). Thus, the instance-based ranker tends to prefer candidates conveying subsets of the input semantics that have been expressed by some training set instance. This tendency is furthered by the A^*-search algorithm which uses short candidate sentences that have a high cosine score to prune away longer but lower scoring candidates. A^*-search does make sure that we do not prune candidates that may turn out to be better than already obtained ones. However, once a short candidate with a similarity score of 1.0 has been found, the remaining edges are pruned away because there is no edge with an expectation larger than the threshold. Given this preference of the ranker, candidate (96e) which is complete and has a cosine score of less than 1.0 seems to be rather an exception. Obviously, the grammar was not able to generate a perfectly matching but incomplete candidate in this case.

Counting the number of *support instances* verifies our observation about variation in the output sentences: 11 different nearest neighbours have been used for the 40 test set inputs. In principle, the ranker could have chosen 40 different nearest neighbours from the instance base of 104 instances. Furthermore, only 44 different *support rules* have been used to construct the candidates out of 384 rules specified in the grammar.

7.3.2 Summary

Running the instance-based generator on test data showed that it is capable of handling unseen input and produce (mostly) fluent and grammatical output sentences. (An efficiency evaluation of the basic system on test data can be found in section 3.3.2). We also identified a major characteristic of this basic version of the instance-based generator: it tends to produce incomplete candidates that have a perfect match with some instance in the instance base. This can be seen as a problem as the generator does not fully complete the task that is given to it. In the following section, we address this problem and present an improved version of the instance-based generator.

7.4 Fluency and completeness

In order to obtain more complete candidates, it helps to modify the A^*-search so that it does not prune edges that have an expectation larger *or equal* to the cosine of the best already obtained candidate (rather than insisting on an expectation that is strictly larger than the threshold; see section 3.2.1.2). Instead of just one best candidate, we now often obtain several ones. An example:[4]

(103) a. James L Pate was named a director of this oil concern . (cos=1.0, cov=3/6)

b. James L Pate , 54 years old , was elected a director of this oil concern , expanding the board to 14 members . (cos=1.0, cov=5/6)

c. James L Pate , 54 years old , was elected a director of this oil concern , increasing the board to 14 members . (cos=1.0, cov=5/6)

Using the more liberal definition of the threshold in A^*-search effectively eliminates a peculiarity of this search procedure. However, the fundamental problem remains unsolved: the instance-based ranker chooses candidates that express subsets of the input semantics and are a perfect match for one of the instance representations. Only rarely does the system find a candidate that expresses the entire input semantics or has a cosine score of less than 1.0.

A fundamental assumption underlying instance-based NLG is the idea that in order to be fluent the output should be as similar to human-authored corpus sentences as possible. Being similar to the corpus sentences is a desirable property and a general characteristic of corpus-based approaches – the goal is to mimic some aspects of the corpus. However, maximizing only this similarity can be a problem. Candidates (103a-103c) could in fact be generated by a system that simply has a complex template corresponding to each instance. In other words, the creativity of the rule-based grammar has not been exploited by the instance-based generator. The grammar rules might be able to generate previously unseen candidates that better express the input. However, these are not preferred by the ranker.

[4]These experiments were carried out using $tf.idf$-weighted mixed ngram term representations.

One obvious attempt to address this problem is to extend the definition of a 'candidate' from any edge of syntactic category '`Sup`'[5] to also require the complete realization of the input. This yields the following best candidate for the input corresponding to original sentence (97a):

(104) James L. Pate, 54 years old, executive vice president, was named a director of this oil concern, increasing the board to 14. (cos=0.97, cov=6/6)

This sentence has a cosine score lower than 1.0 and has therefore previously lost out against its shorter but higher scoring competitors (103a-103c). (The system is not able to fully reconstruct the original sentence (97a) because it lacks the grammar rule to produce *54-year-old executive vice president* in the required syntactic context.) However, it cannot generally be assumed that the generator is always able to express the entire input semantics in one sentence although in this case we were lucky. For example, there are 7 tags in the test set that do not occur in the training set. Furthermore, there is no guarantee that a given combination of tags known to the grammar can be expressed in a single sentence. This is part of a general problem that has been described as the "generation gap" Meteer (1990). The fundamental question is how a generation input can be constructed by some external component without knowing exactly what the grammar is able to express.

7.4.1 Combining cosine score and coverage

In this thesis, we are not trying to decide whether or not the generation gap can be bridged but rather address the practical question how we can influence the coverage of the input semantics in an instance-based approach to candidate ranking without hardwiring a completeness constraint. Our first proposal is to use a linear combination of the cosine score and a coverage score to compute a new edge score:

(105) $(1-\lambda) \cdot cos(E,I) + \lambda \cdot \frac{|Sem_E|}{|Sem_{Input}|} \; [\lambda \in \{0 \ldots 1\}]$

[5]Recall that we introduce a special top-level syntactic category during grammar construction (see section 5.6).

The cosine between edge E and some instance I (to the left of 105) is combined with a coverage score which is computed by dividing the size of the semantics of E by the size of the input.[6] In principle, the coverage score could take into account different weights for parts of the semantic input. A high value of λ places more emphasis on completeness, a low value yields candidates that more closely resemble instances.

Choosing different values for λ results in candidates of different sentence lengths and input coverages. For example, one can only obtain the complete candidate (104) with $\lambda > 0.31$. The nearest neighbour of (104) is a training set template that lacks the `INPERSON_OTHERPOST_NODET` tag:

(106) `[INPERSON_FULLNAME` William C. Ballard Jr.`]`, `[INPERSON_AGE` 48`]` years old, was elected a `[POST_INDEF` director`]` of this `[COMP_DESCR` distilled beverages concern`]`, expanding the board to `[BOARD_INCR` 11`]` members.

The linear combination of cosine score and coverage score effectively 'squeezes' the missing `INPERSON_OTHERPOST_NODET` tag ('executive vice president' in (104)) into the nearest neighbour template. (However, the system does not directly adapt templates to new inputs but rather uses grammar rules derived from an analysis of the instances to produce the output candidates.)

There are always certain values of λ at which the system 'switches' between different best candidates. The switching points and number of best candidates for different settings of λ can vary greatly for different inputs. Another example:[7]

(107) a. **λ= 0.0-0.32:** David A. DiLoreto was named group vice president, packaging products, of this packaging, industrial and aerospace products concern. (cos=1.0, cov=3/10)

b. **λ= 0.33-0.7:** David A. DiLoreto, president of metal container division, has been named group vice president, packaging products, a additional post at the packaging, industrial and aerospace products concern. (cos=0.86, cov=6/10)

c. **λ= 0.71-1.0:** David A. DiLoreto, president of metal container division, was named to the additional post of group vice president, packaging products, of this packaging,

[6]When we talk about the 'length' of an edge, we are referring to the size of the semantic set. This broadly correlates with the length of the surface string (see table 7.2).

[7]The cosine of the candidates `'cos'` in this and the following examples is always the pure similarity score even if a linear combination is used.

industrial and aerospace products concern, succeeding Delmont A. Davis. (cos=0.61, cov=7/10)

d. **original:** David A. DiLoreto, president of metal container division, was named to the additional post of group vice president, packaging products, at this packaging, industrial and aerospace products concern, succeeding Delmont A. Davis, who was named president and chief operating officer in August.

(107) shows 3 candidates that are optimal for certain ranges of λ. The example suggests that there is a trade-off between fluency (cosine) and completeness (coverage). When more emphasis is placed on completeness, sentences are generated that have a greater distance from the nearest neighbours of the instance base. In fact, this generally turns out to be the case when the generator is run with different values of λ.

λ	cosine	std.	coverage	std.	complete	len(templ)	std.	len(surf)	std.
0.0	0.94	0.12	0.67	0.21	15.4%	10.5	3.3	14.6	3.7
0.1	0.94	0.12	0.68	0.20	15.4%	10.7	3.5	14.9	3.7
0.2	0.94	0.12	0.69	0.20	18.0%	11.0	3.6	15.2	4.0
0.3	0.94	0.13	0.71	0.21	23.1%	11.4	3.8	15.8	4.5
0.4	0.90	0.13	0.77	0.19	33.3%	12.6	3.9	17.1	5.0
0.5	0.88	0.13	0.79	0.19	35.9%	13.0	3.8	17.6	5.0
0.6	0.77	0.17	0.88	0.16	56.4%	14.5	3.9	19.5	4.9
0.7	0.75	0.16	0.90	0.16	64.1%	15.2	3.7	19.8	4.8
0.8	0.73	0.18	0.90	0.14	64.1%	15.4	3.7	20.0	5.0
0.9	0.73	0.18	0.90	0.14	64.1%	15.4	3.7	20.0	5.0
1.0	0.73	0.18	0.90	0.14	64.1%	15.4	3.7	20.0	5.0

Table 7.2: Trade-off between fluency (cosine similarity) and completeness

Table 7.2 shows average values of cosine and coverage for the test set for different values of λ. It clearly indicates that the cosine decreases as the coverage weight λ increases.[8] This is also depicted in figure 7.8. Increasing λ has the desired effect of increasing the average candidate length as measured by the number words ('`len(surf)`') and by the length of the template representation ('`len(templ)`'). The maximal candidate length of 20.0 words comes close to the average length of the original sentences which is 21.9 words. Maximal coverage is reached at λ=0.7 where the average cosine

[8] As a reference point at λ=0.0, we take the longest candidate if there are more than one.

score has not yet reached its minimum. We will use this value of λ in the following experiments when we want to encourage the system to produce long sentences.

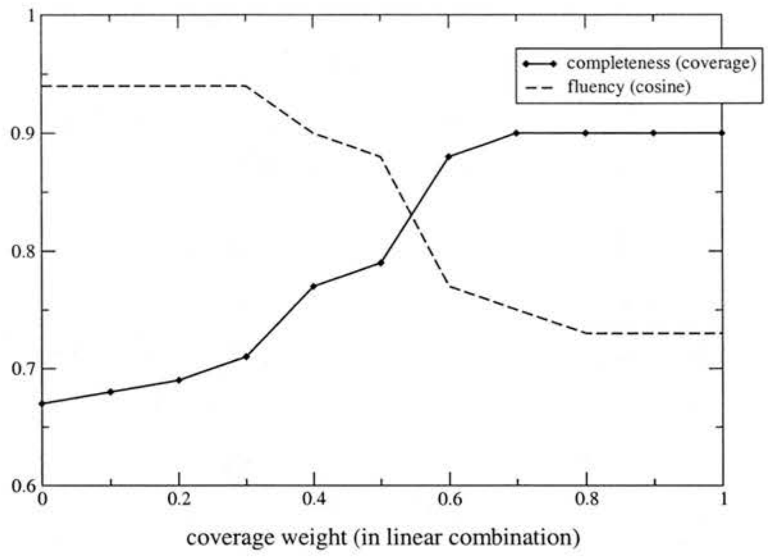

Figure 7.8: Trade-off between fluency (cosine similarity) and completeness

The number of support rules and instances increases when a linear combination is used. For λ=0.7, 29 different instances are used (in contrast to 11 for λ=0.0) and 117 different rules (versus 44 for λ=0.0). The increase in the number of rules can be explained by the need to produce candidates that express larger sets of tags.

When the similarity score is less than 1.0, the candidate representation contains terms that are not present in the instance representation or it is missing terms. From the definition of the distance metric (section 3.1.3) it follows that the ranker prefers missing or non-matching terms that are as 'light' as possible in order to minimize this error. If the weighting scheme contains an *idf*-component as is the case here, these additional terms tend to be frequent ones, i.e. terms that occur in many documents/instances.

Using a linear combination, we can now look again at the candidates for the inputs of the original sentences in (97). (108) shows candidates chosen by a mixed ngram model and λ=0.7 (which is where the maximum coverage is reached according to table 7.2):

These candidates cover considerably more input tags than those in (96). For example, (108b) now expresses 8 out of 10 tags instead of just 3. It still lacks the slot

(108) **generation output:**

a. James L. Pate, 54 years old, executive vice president, was named a director of this oil concern, increasing the board to 14 members. (cos=0.82, cov=6/6) =(104)

b. Michael Henderson, 51 years old, group chief executive, was appointed chairman of this metals and industrial materials maker, effective in May, succeeding Ian Butler, who is retiring. (cos=0.83, cov=8/10)

c. John F. Barrett, 40 years old, formerly chief financial officer, was named chief operating officer, both vacant posts, and president. (cos=0.592, cov=6/7)

d. Drug Emporium Inc. said Gary Wilber, 39 years old, was elected chief executive officer of this drugstore chain. (cos=0.63, cov=5/8)

e. Charles A. Pearce, 66 years old, announced that he will retire Dec. 31 as chief executive officer of this bank holding company. (cos=0.95, cov=6/6)

f. John F. McNair III, announced that he will retire on Dec. 31 as president and of chief executive officer of Wachovia Bank & Trust Co. of this regional banking company. (cos=0.71, cov=7/10)

g. Arthur Price, chief executive officer and president of MTM Enterprises Inc., resigned from his executive duties. (cos=0.51, cov=5/7)

fillers 'UK' and '64' whose tags the grammar cannot combine with the other input tags. Similarly, (108d) realizes 5 out of 8 tags rather than just 3. One of the missing tags is a singleton. The other two can only be forced into the candidate at the expense of omitting one of the tags of (108d). Candidate (108e) remains the same as (96e). (Note that its cosine score is different due to the change of the instance model.)

All but two candidates in (108) are syntactically and semantically correct. Compared to original sentence (97f), (108f) does not seem to be very fluent. The generator (correctly) tries to express that 'Wachovia Bank & Trust Co.' is a subsidiary of the main company (described as a 'regional banking company') by using prepositions ('of ... of'). The nearest neighbour of (108f) is still (98). It is the same that was chosen by the basic ranker without linear combination for (96f) and does not contain any tag related to subsidiaries. There are a number of corpus examples that deal with subsidiaries similarly to (108f). However, these clearly indicate the status of the subsidiary by using words like 'unit' or 'division'. An example:

(109) ... of the [COMP_SUBSIDIARY Clairol] [COMP_SUBSIDIARY_DESCR_DEF division] of this [COMP_DESCR pharmaceuticals and health-care company].

However, in the case of (108f) the information about the subsidiary is given to the generator in the form of a singleton tag COMP_SUBSIDIARY_DESCR_PLURAL. Since this tag only occurred in the test set, it was not available in the training set and cannot be expressed.[9] Thus, under the given circumstances, the generator makes the best use of the available tags. Unfortunately, it also seems to be forced to buy the wrong preposition in 'and of chief executive officer' (again).

(108c) exhibits a syntactic error by misplacing a constituent. The system correctly chooses 'both' (the original sentence does not use this word) but seems to get the word ordering wrong. There is no explicit representation in the grammar that expresses that 'both' needs to refer to exactly two referents. Despite this, the ranker chooses a nearest neighbour that uses 'both' correctly:

(110) ... was named executive vice president and chief operating officer, both newly created posts, and a director, filling a vacancy.

However, the ranker seems to get 'confused' by the presence of the NP at the end of (110). A correct candidate can be found at rank 5:

(111) John F. Barrett, 40 years old, formerly chief financial officer, was named president and chief operating officer, both vacant posts. (cos=0.586, cov=6/7)

Candidate (111) consumes the same number of input tags but has a slightly lower cosine score on the same nearest neighbour using the mixed ngram model (and also the trigram model).

7.4.1.1 Effects on A^*-search algorithm

A linear combination of cosine and coverage score affects the computation of the expectation: we need to be optimistic about the coverage score (as we are about candidate-instance similarity) and assume that the expectation for the coverage score

[9]A practical solution in this case might be to recast the unknown plural tag by other, known tags.

is always maximal (1.0). We have to make this assumption since we do not know what combinations of tags are licensed by the grammar – again a manifestation of the fundamental problem of expressibility. The combined expectation therefore is higher than the cosine alone, making it easier for edges to pass the threshold set by the combined score of already existing candidates. This extends the search space but does not lead to problems in practice except for extreme values of λ. Typical average runtimes range from 1.3 seconds for the basic system with λ=0 to 11.5 secs for λ=0.7. For high values of λ (>0.8), the A^*-search algorithm tends to search for ever larger candidates so that we need to apply a time limit (30 seconds). However, as in previous experiments the expectation-based agenda ordering ensures that good candidates are generated early during processing, i.e. we do not seem to miss any better candidate although the search space has not been fully explored before the time limit applies. Only in the extreme case of λ=1.0 is there no guidance by the expectation since (combined) score and expectation are fully dominated by the coverage values.

An advantage of using a linear combination in contrast to incorporating a completeness constraint into the candidate definition is related to the A^*-algorithm: the linear combination allows the system to obtain candidates quickly and use their similarity score as a threshold. This is more efficient than having to wait until the first complete candidate has been produced despite the fact that the candidate score may have been discounted because of a lack of completeness. In the reproduction experiments with the large grammar and instance base (section 7.2.2) we therefore used a linear combination rather than a completeness constraint to prevent the system from just expressing some subset of the input semantics. (In section 7.4 we did not use the linear combination because we wanted to present a basic version of the instance-based generator. The experiences we made with this basic system were our main motivation to introduce the linear combination.)

7.4.2 Generating candidates into several ranked lists

In general, it is difficult to determine ahead of time how many different candidates there are for all values of λ, where the switching points are, or whether increasing λ would result in larger coverage. It may be possible to choose a heuristically determined value for λ if the goal is simply to express as much input as possible without compromising fluency (i.e. similarity between candidate and nearest neighbour) too much.

We would like to take a step back at this point and ask what we actually expect the output of a realizer to be in the context of an overgeneration approach. For text generation, or any input for which we are unsure whether it can really be expressed in a single sentence, there is a possibility that we do not want to express a maximal number of tags in the first sentence since we do not know whether the remaining semantics can be expressed in the follow-up sentences. We would rather aim at generating a globally best text than a locally best first sentence.

We therefore propose that a realizer used in the context of a text generation system produce several ranked candidate lists, each one for a different subset of the input semantics. Follow-up sentences would be generated for the remaining semantics of these candidate lists which again return a set of ranked lists. The ranked lists could be organized in a lattice or a similar data structure. The text generator would then choose a path though this data structure.

In this thesis, we concentrate on sentence realization but believe that maintaining several dynamically created candidate lists for different semantic sets is a useful output of a sentence realizer if it is unclear how much semantics should be expressed in the individual sentences.

As an example, we take again the input corresponding to original sentence (107d). The system dynamically creates 14 ranked lists ('bins') according to the semantic sets of the candidates that have been produced. In (112-125, figure 7.9) the best candidates in each bin are shown in combination with their semantic 'footprint' (a set of unique integers that represents the input tags consumed by the candidates). Behind every shown candidate there is a list of lower ranked candidates that express the same semantics (which are not shown). Note that all candidates are syntactically and semantically correct.[10]

There are more bins for this input than there are best candidates for different settings of λ (see 107). Generating into different bins has the advantage of making locally non-optimal candidates available for rescoring in a larger context. This also addresses

[10]In (113), 'a additional' should be changed into 'an additional' which can be done by simple post processing.

another problem of imitating the instance base too closely: if there is a good match for a long candidate, the system might not be able to find a shorter, lower scoring one. For example, candidate (125) which expresses only two input tags also has a low cosine score and is therefore not optimal for any setting of λ.

7.4.2.1 Effects on A^*-search algorithm

The use of multiple rankings allows candidates only to compete within their bins. However, the question arises which threshold to use when pruning the instance-edge pairs used for the similarity computations. In principle, we cannot know how an edge is going to be extended so that the correct bin from which to take the threshold remains unknown. Therefore, in these experiments we used an expectation-based ranker that does no pruning of similarity computations. This requires one to use a time limit (in practice, 30 seconds is usually sufficient). However, we are helped (again) by the expectation-based agenda ordering: the best candidates tend to be produced early so that shutting down the generator after 30 seconds does not have a large effect on the top-scoring candidates. It is possible to define a 'dynamic time limit' that applies when no new best candidate (in any bin) has been produced in a given time span.

time limit	n bins	av bins	std.	n cands	av cands	std.	av time	std.
dynamic 5 secs	417	10.4	11.8	2629	67	66.4	10.9	7.8
absolute 60 secs	506	12.7	14.6	9213	236	184.7	40.0	25.6

Table 7.3: Generating into semantic bins: comparing stopping criteria

The number of bins that are generated depends on the size of the input and the combinations of tags that can be produced by the grammar. The maximum number of bins generated for one test set input with an absolute time limit of 60 seconds is 64 bins, the minimum number is 2 and the average is 12.7 bins. We compared a system run that uses an absolute time limit with one that uses a dynamic time limit of 5 seconds (in combination with an absolute limit of 60 seconds). Only in 9 out of 40 inputs does the dynamic time limit return fewer bins than the absolute time limit. Table 7.3 shows the results of the comparison. The dynamic time limit reduces the average runtime by almost 30 seconds at the expense of losing 18% of the bins (for the entire test set).

(112) 4 5 9: David A. DiLoreto was named group vice president, packaging products, of this packaging, industrial and aerospace products concern. (cos=1.0, cov=3/10)

(113) 4 5 6 7 8 9: David A. DiLoreto, president of metal container division, has been named group vice president, packaging products, a additional post at the packaging, industrial and aerospace products concern. (cos=0.86, cov=6/10)

(114) 4 5 6 9: David A. DiLoreto was named to the additional post of group vice president, packaging products, of this packaging, industrial and aerospace products concern. (cos=0.84, cov=4/10)

(115) 5 6 9: David A. DiLoreto was named to the additional post of group vice president, packaging products. (cos=0.84, cov=3/10)

(116) 5 6 7 8 9: David A. DiLoreto, president of metal container division, assumed the additional post of group vice president, packaging products. (cos=0.84, cov=5/10)

(117) 3 4 5 9: David A. DiLoreto was named group vice president, packaging products, of this packaging, industrial and aerospace products concern, succeeding Delmont A. Davis. (cos=0.81, cov=4/10)

(118) 3 4 5 7 8 9: David A. DiLoreto, president of metal container division, was named group vice president, packaging products, of this packaging, industrial and aerospace products concern, succeeding Delmont A. Davis. (cos=0.74, cov=6/10)

(119) 4 5 7 8 9: This packaging, industrial and aerospace products concern said its board elected David A. DiLoreto, president of metal container division, as group vice president, packaging products. (cos=0.68, cov=5/10)

(120) 3 5 9: David A. DiLoreto was named group vice president, packaging products, succeeding Delmont A. Davis. (cos=0.64, cov=3/10)

(121) 3 4 5 6 7 8 9: David A. DiLoreto, president of metal container division, was named to the additional post of group vice president, packaging products, of this packaging, industrial and aerospace products concern, succeeding Delmont A. Davis. (cos=0.61, cov=7/10)

(122) 3 4 5 6 9: David A. DiLoreto was named to the additional post of group vice president, packaging products, of this packaging, industrial and aerospace products concern, succeeding Delmont A. Davis. (cos=0.61, cov=5/10)

(123) 3 5 7 8 9: David A. DiLoreto, president of metal container division, was named group vice president, packaging products, succeeding Delmont A. Davis. (cos=0.59, cov=5/10)

(124) 5 7 8 9: David A. DiLoreto, president of metal container division, was named group vice president, packaging products. (cos=0.47, cov=4/10)

(125) 5 9: David A. DiLoreto was named group vice president, packaging products. (cos=0.41, cov=2/10)

Figure 7.9: Semantic 'bins' produced by generation system

The mechanisms that are used for generating candidates into semantic bins can also be used for determining the optimal candidate for different values of λ in the linear combination. To this end, we define bins by the coverage value rather than a semantic footprint. Only the best candidates in each bin have a chance of being optimal for some value of λ. To identify these optimal candidates, we iterate over a number of values for λ and compute the linear combination for the best candidate in each bin after generation has finished. The number of bins generated for different coverage values tends to be lower than the number of semantic bins because different semantic sets of the same size are mapped to the same coverage value (as long as all tags are assigned the same coverage weight).

7.4.2.2 Semantic bins and n-best search

The A^*-search algorithm can be extended so that n best candidates are found (see section 3.2.1.5). Keeping n best candidates in each bin when generating into multiple ranked lists might be useful if macroscopic properties are scored in text generation (see Oberlander and Brew, 2000). For example, one might want to approximate a certain global distribution of verbs in the text, and this might require the system to choose candidates that are non-optimal according to the sentence-based nearest neighbour ranker. Similarly, issues of discourse coherence in multi-sentence generation may require a re-ranking of the locally best sentences. Alternatively, discourse coherence could be ensured by means of grammar constraints. In other words, additional knowledge needs to be added to either the ranker or the rule-based part of the hybrid generation system (see also section 8.4.1).

7.4.3 Faithfulness and completeness

Realizing the generation input only partially can result in non-intended interpretations. An example of a faithfulness error due to the omission of certain tags:

(126) George L Manzanec, senior vice president, was named a group vice president of this natural-gas-pipeline concern.

(127) George L Manzanec, senior vice president of Texas Eastern Corp., was elected a group vice president of this natural-gas-pipeline concern .

Only the second realization is faithful to the input since Manzanec is senior vice president at another company, not the natural-gas-pipeline concern as (126) implies. Similar examples can be found for previous posts that can be located at either the current company of a person or at a former company, for example.

Solutions to this faithfulness problem need to consider that the only semantic information available to the system is the annotation of the corpus sentences. We do not have a deep semantic model that is able to reason about possible interpretations of candidates.

7.4.3.1 Cooccurrence constraints on semantic tags

One approach to address this problem is to manually specify hard cooccurrence constraints on semantic tags that effectively act as additional filters on candidates. For example, they can state that both `IN_OTHERCOMP` and `IN_OTHERPOST_NODET` need to be expressed together. This leaves the system the freedom of not expressing any of the two as this can result in a short but still faithful sentence:

(128) George L Manzanec was named a group vice president of this natural-gas-pipeline concern .

Tag coocurrence constraints can be applied to two or more tags which can be regarded as single tags that are allowed to be realized discontinuously.

An efficient implementation of tag cooccurrence constraints is possible by only checking for those semantic indices that are actually present in the input. The constraints are specialized to semantic indices when the input is given to generator. This means that constraints that do not apply (since not all tags that need to cooccur are present in the input) need never be checked during generation. For example, consider the following input and coocurrence constraints:

(129)

IN_FULLNAME	Bruno DeGol
IN_OTHERPOST_NODET	chairman
IN_OTHERCOMP	DeGol Brothers Lumber
IN_OTHERCOMP_LOC	Gallitzin Pa
POST_INDEF	director
COMP_DESCR	bank-holding company
BOARD_INCR	11

(130)

```
coocurrenceCst(["IN_OTHERCOMP","IN_OTHERPOST_NODET"]).
coocurrenceCst(["IN_OTHERCOMP","IN_OTHERPOST_DEF"]).
coocurrenceCst(["INPERSON_PREVIOUSCOMP",
                "INPERSON_PREVIOUSPOST_NODET"]).
```

When input (129) is given to the generator, only the first constraint is translated into requiring either both IN_OTHERCOMP and IN_OTHERPOST_NODET or none of these. The other constraints do not apply, avoiding unnecessary overhead when individual candidates are checked. Furthermore, constraint specialization for tag coocurrence constraints needs to consider tag-indices if these are used in the input. A coocurrence constraint only applies to those pairs of tags that 'belong together'. In some cases, this can result in the same constraint applying more than once. An example is given in section 7.7.1 below.

7.4.3.2 Improving the annotation scheme

A related faithfulness problem can be traced back to an ambiguity in the annotation scheme. In the following example, the best candidate places the post of the incoming person at the main company:[11]

(131) William J Russo was named senior vice president public affairs and advertising, of this financial and travel services concern .

However, the original sentence specifies that the post is located at a subsidiary (we only show the annotation of the relevant parts of the sentence):

[11]Recall that the main company is assumed to have been introduced into the discourse context already, usually by a headline to the article about the management succession.

(132) William J. Russo was named senior vice president, public affairs and advertising, for this [COMP_DESCR financial and travel services concern]'s [COMP_SUBSIDIARY American Express Bank Ltd.] [COMP_SUBSIDIARY_DESCR_NODET subsidiary].

The annotation does not clearly specify the location of the post, and in a sense the ranker does not choose an "unfaithful" candidate. The problem can be solved by marking the post location explicitly, for example by adding '*' to each tag specifying the location of the main post. If this is done consistently for the entire corpus, the system will be able to clearly distinguish between COMP_DESCR – as we will find it in the revised annotation of (132) – and COMP_DESCR* for corpus sentences like (133):

(133) Bruno DeGol, chairman of DeGol Brothers Lumber, Gallitzin, Pa., was named a director of this bank-holding company, expanding the board to 11 members.

This example demonstrates that we can influence the output of the generation system by changing the annotation that forms the basis of the ranker (and the grammar, of course).

7.4.4 Human evaluation of the test set candidates

Although the similarity values and the completeness of the candidates can be measured automatically, faithfulness errors and other problems need to be observed by human judges. To this end, two people (including the author) evaluated the output of the generator for the test set of 40 inputs. We also observed other types of errors such as fluency and syntactic problems. The results in this section were obtained using a trigram instance model.

Figure 7.4 shows the results of the manual evaluation of the improved system. Semantic errors include faithfulness errors and other semantic problems. We distinguish syntax errors from fluency errors. Some examples:

(134) *Fluency:* Francis D. John, *35 years old, president,* was elected to ... .

(135) *Syntax:* This bank holding company said its board *elected* John W. Day.

(136) *Semantics:* ... was named vice president,..., *the new post/both new posts ...*

λ	Semantic errors	Syntax errors	Fluency errors	correct candidates
0.0	3	0	0	92.5%
0.2	3	0	0	92.5%
0.4	5	0	3	80.0%
0.6	5	2	10	62.5%
0.8	5	4	12	57.5%
1.0	5	4	12	57.5%

Table 7.4: Human evaluation of best candidates (using faithfulness improvements)

(134) is considered a fluency error by the two judges. In fact, no pattern like the one in (134) can be found in the corpus. More fluent phrases include 'the 35-year-old president' and '35 years old, president of this ...', for example. (135) is a syntactic error as it violates the subcategorization frame of the ditransitive verb 'elected'. (136) is a semantic error - the correct use of 'both' is beyond the capabilities of the shallow grammar.

We observed an increase in the number of fluency and syntactic errors for lower average cosine scores – confirming our basic assumption that cosine similarity is a good indicator of fluency. As a result of tag cooccurrence constraints and improved annotation, the number of faithfulness errors is relatively small, even for short candidates. An unmodified version of the system yields 7 faithfulness errors due to omissions of tags for λ=0.2 (versus 3 faithfulness errors for the new system). The overall number of correct sentences is maximal for λ=0.0 and λ=0.2 and decreases when more weight is given to completeness. When using the additional faithfulness constraints, the number of semantic errors follows the general trend of the fluency and syntactic errors: it is low for candidates that have a high similarity score and high for candidates that have a low similarity score.

7.4.5 Summary

We started with the observation that many candidates produced by a basic version of the instance-based generator do not completely express the input semantics. We then presented two techniques to deal with this problem. First, using a linear combination of cosine and coverage score allows one to trade candidate-instance similarity against candidate-input similarity (in semantic terms). Second, maintaining multiple candidate lists circumvents the problem of choosing a candidate that expresses the right portion of the input altogether by delaying the final decision until a larger context is known. We also demonstrated that faithfulness problems can arise if parts of the generation input are omitted, and proposed tag cooccurrence constraints as well as refining the annotation scheme as potential solutions. Finally, the results of a human evaluation of the system output showed that 57.5 - 92.5% of the candidates were perfectly correct, depending on the completeness of the candidates which can be controlled by the setting of λ in the linear combination. Section 7.8.1 will discuss the generality of the trade-off between the goals of fluency and completeness in NLG.

7.5 Scalability

To investigate how the generation system scales with respect to the size of the grammar and the instance base, we compiled a number of sub-grammars/instance bases and tested them on the 40 test set inputs. To keep the generator from replicating the instance base too closely we used a linear combination with λ=0.7 and trigram terms throughout the experiments but also tested a basic system without linear combination.

sent.	inst.	rules	solutions	λ=0.7				λ=0.0			
				cos	std.	cov%	tags all	cos	std.	cov%	tags all
2	2	17	1	(0.65)	(0)	(0.7)	5	(0.65)	(0)	(0.7)	5
3	3	26	11	0.46	0.07	50.8	32	0.47	0.08	49.2	31
4	4	36	11	0.45	0.08	50.8	32	0.46	0.08	49.2	31
5	5	47	24	0.45	0.16	70.3	97	0.53	0.11	56.5	78
10	10	81	27	0.45	0.27	81.2	125	0.58	0.20	65.6	101
20	20	149	35	0.52	0.24	79.4	162	0.63	0.21	63.7	130
30	30	178	36	0.54	0.25	81.4	171	0.67	0.22	66.2	139
40	40	186	36	0.66	0.26	81.0	170	0.90	0.19	57.6	121
50	50	227	36	0.68	0.22	84.8	178	0.94	0.16	52.9	111
70	70	345	37	0.71	0.21	84.3	182	0.93	0.16	58.8	127
90	90	359	38	0.70	0.20	85.2	190	0.94	0.15	55.6	124
104	104	384	39	0.70	0.21	85.4	199	0.93	0.15	54.9	128

Table 7.5: Results of scalability experiments (trigram terms)

Table 7.5 shows the results of the experiments. The left columns show the number of instances and rules for different numbers of corpus examples. There are always as many instances as there are annotated corpus sentences. The growth of the rule set is slower (it is described in more detail in section 5.6).

The first issue to be investigated is the coverage of the grammar: for how many of the 40 test set inputs does the grammar produce some candidate? As table 7.5 shows, 5 examples are sufficient to yield at least one candidate for more than half of the inputs (‘`solutions`’). The largest grammar generates candidates for 39 inputs. (One input contains an unknown `POST` tag; see example (102) on page 216). These results are the same for both sets of experiments. Not entirely surprisingly, the experiments suggest that more corpus examples will further increase the coverage of the grammar.

The next issue concerns the similarity score of the candidates. We measured the average cosine score for those inputs that result in at least one candidate. The general tendency for both sets of experiments is an increase of the similarity score as the grammar and instance base grow larger. This increase is smaller for λ=0.7 than for λ=0.0 because of the higher emphasis on coverage (‘`cov`’). This demonstrates again the trade-off between fluency and coverage. In addition, table 7.5 shows the total number of tags realized by the candidates across the test set (‘`tags all`’). Thus, it factors in that there is no candidate for some inputs. The test set provides 238 input tags. The maximal number of tags expressed by the generator is 199 for the largest grammar with linear score combination. This corresponds to 83.6% overall coverage of the generator.

Small decreases of the average cosine with increasing corpus size can be explained by the larger number of inputs for which a candidate can be obtained. For λ=0.7, the cosine score peaks at 70 corpus examples. However, more training material still increases the completeness of the candidates. A minor oddity occurs when moving from a corpus of 3 to a corpus of 4 examples. In both cases, exactly the same candidates are produced. Despite this, the average cosine decreases slightly. This can be explained by the changed term weights. In addition, the results for a corpus of size 2 which yields only one solution can be regarded as unreliable. Furthermore, there is no solution for the extreme case of a single instance in the instance base: in this case, all idf-weights are 0 since all terms occur in all instances, i.e. the only one available.

(137) a. **corpus=5:** David A. DiLoreto, president of metal container division, was named group vice president, packaging products, of this packaging, industrial and aerospace products concern. (cos=0.24, cov=5/10, nn=wsj_0009)

b. **corpus=10:** David A. DiLoreto, president of metal container division of this packaging, industrial and aerospace products concern, was elected group vice president, packaging products, a additional position. (cos=0.33, cov=6/10, nn=wsj_0196)

c. **corpus=20:** David A. DiLoreto, president of metal container division, was named to the additional post of group vice president, packaging products, of this packaging, industrial and aerospace products concern. (cos=0.73, cov=6/10, nn=wsj_0368)

d. **corpus=30:** David A. DiLoreto, president of metal container division of this packaging, industrial and aerospace products concern, assumed the additional post of group vice president, packaging products. (cos=0.78, cov=6/10, nn=wsj_0512)

e. **corpus=40-90:** David A. DiLoreto, president of metal container division, has been named group vice president, packaging products, a additional post at the packaging, industrial and aerospace products concern. (cos=0.82-0.84, cov=6/10, nn=wsj_0740)

f. **corpus=104:** David A. DiLoreto, president of metal container division, was named to the additional post of group vice president, packaging products, of this packaging, industrial and aerospace products concern, succeeding Delmont A. Davis. (cos=0.61, cov=7/10, nn=wsj_1459)

(138) a. **corpus=3,4:** Alvin W. Trivelpiece was elected a director. (cos=0.42, cov=2/6, nn=wsj_0005)

b. **corpus=5:** Alvin W. Trivelpiece, director of Oak Ridge National Laboratory, was elected a director. (cos=0.68, cov=4/6, nn=wsj_0005)

c. **corpus=10-30:** Alvin W. Trivelpiece, director of Oak Ridge National Laboratory, was elected a director of this optical-products concern. (cos=1.0, cov=5/6, nn=wsj_0185,wsj_0378)

d. **corpus=40-104:** Alvin W. Trivelpiece, director of Oak Ridge National Laboratory, Oak Ridge, Tenn., was elected a director of this optical-products concern. (cos=1.0, cov=6/6, nn=wsj_0729)

Figure 7.10: 'Evolution' of candidates with growing resources

(137) and (138), figure 7.10, show the 'evolution' of candidates as the training set increases (settings: λ=0.7, trigram terms). The numbers preceding the candidates indicate the size of the resources that have been used. Generally, the coverage of the candidate increases with the size of the corpus. In (137), the ordering of the information as well as the main verb changes at the different stages if this is necessary to obtain a higher similarity score. These changes happen even if they do not lead to increased completeness (see 137b-137e). The system is not just modifying the 'same' candidate but actually switches between different nearest neighbours ('nn' in figure 7.10). The reordering of information becomes apparent when we look at the placement of 'additional'. The ranker uses global bag-of-terms representations and, thus, reordering happens globally as well: the goal of the ranker is to optimize the overall similarity score of the entire sentence. (138) shows that the system can move from one perfect match with a nearest neighbour to another, more complete one as more grammar rules and nearest neighbours become available. This is because the combined score of the non-complete candidate in the linear combination is below 1.0. In (138), the candidates always use the same main verb although different nearest neighbours are used. Furthermore, (138c) is equidistant to two nearest neighbours.

7.5.1 The benefits of a large instance base

A variation of the previous scalability experiment is to keep the number of grammar rules fixed and only vary the size of the instance base. Table 7.6 shows the results of such an experiment. The main finding is that fluency as well as efficiency increase as the instance base grows larger. For very small instance bases, the grammar is less likely to generate a high-scoring candidate, and thus, less pruning can take place. When the instance base is large, new inputs tend to have a higher similarity to the tags expressed by some instance, and thus the generator is more likely to produce a candidate that is similar to the surface template of that instance. This in turn increases the similarity score and allows the system to prune more.

In this experiment, we used again a 30 second time limit. For very small instance bases the time limit is reached frequently. As before, for an instance base of size 1 there is no solution due to *idf* term weighting. The minimum runtime is reached at an

instances	rules	λ=0.7, trigram terms					
		cos	std.	cov%	tags all	time av	cands all
2	384	0.08	0.14	84.6	197	13.6	11759
3	384	0.21	0.24	82.8	193	13.0	9838
4	384	0.32	0.26	82.0	187	13.5	9687
5	384	0.28	0.23	81.5	190	12.4	6143
10	384	0.35	0.28	81.5	190	10.8	4996
20	384	0.46	0.27	84.6	197	11.1	5447
30	384	0.50	0.27	84.6	197	10.3	6093
40	384	0.59	0.31	83.7	195	9.6	5516
50	384	0.62	0.27	85.0	198	9.5	5361
70	384	0.66	0.25	84.6	197	9.2	4883
90	384	0.68	0.22	84.6	197	9.4	4764
104	384	0.70	0.21	85.4	199	9.8	4327

Table 7.6: Results of scalability experiments with fixed grammar

instance base of size 70 and increases slightly for the larger instance bases despite the fact that the average cosine score still grows. This may indicate that from 70 instances onwards the sheer size of instance base slows the generation system down. In that case, the technique described in the next section becomes relevant.

In sum, our experiments indicate that the instance-based generator scales well. A particularly interesting finding is the increase in efficiency as instance base and grammar grow larger. However, the number of available training examples is too small to draw conclusions about much larger instances bases.

7.6 Dynamic rule and instance selection

'Initial semantic ranking' is a technique that increases the efficiency of the system by reducing the number of grammar rules and instances before generation starts. It computes the cosine similarity between the semantic input and the instances represented by their semantic tags only. Since we do not assume any ordering of the semantic input tags, we use unigram terms to represent input and instance semantics. These unigrams are $tf.idf$-weighted to avoid missing rare tags. The similarity computation only involves a single cycle over the instance base. We can then select the n-best instances for all further similarity computations.

Moreover, we can characterize each instance by a bitvector representation of the grammar rules derived in the grammar construction phase. We determine the rules corresponding to the *n* semantically closest instances by the union of their bitvector representations. Generation can then proceed as before with a reduced number of instance tables and grammar rules from the beginning of processing. This effectively adds some top-down information into the bottom-up generation process in the sense that information about the goal (the input semantics) is used to restrict search.

A major advantage of initial semantic ranking is the increase in efficiency due to the reduced number of grammar rules and instances. We conducted a number of experiments in which we allowed the generator to dynamically select subsets of different sizes from a large grammar/instance base of 104 examples. Again, a trigram instance model with $tf.idf$ term weighting was used.

window	solutions	cos	std.	cov%	tags all	time av	cands sum
1	24	0.80	0.21	76.2	109	0.1	39
2	26	0.81	0.22	76.7	122	0.2	113
3	29	0.77	0.23	81.1	142	0.4	185
4	29	0.78	0.23	82.9	145	1.0	229
5	30	0.78	0.22	82.8	149	1.1	254
10	34	0.77	0.21	82.2	171	5.4	1126
20	36	0.73	0.22	85.7	186	4.0	2835
30	36	0.69	0.23	87.6	190	7.2	5805
40	36	0.69	0.23	87.6	190	7.3	4994
50	36	0.70	0.24	87.6	190	8.2	5213
70	39	0.70	0.21	85.4	199	9.0	4137
90	39	0.70	0.21	85.4	199	9.6	4159
104	39	0.70	0.21	85.4	199	9.8	4327

Table 7.7: Results of initial semantic ranking (λ=0.7, trigrams)

Table 7.7 compares the results of initial semantic ranking to a generation system that uses all available resources (window size: `104`). Compared to the small but fixed grammars/instance bases in the scalability experiments (see table 7.5), initial semantic ranking selects more appropriate resources: it provides more solutions for the inputs for the same grammar/instance base sizes, and the candidates have higher average cosine and coverage values. As the window size increases, the results for both sets of experiments begin to look more similar, which is due to the larger overlap between the resources. For example, for a window of size 90, there can only be a maximal divergence of 14 examples. As before, increasing the number of solutions can bring the average cosine and coverage scores down. However, the total coverage of all candidates increases steadily until it reaches a maximum at a window size of 70. From then on, no further changes happen. The available data therefore suggests that a window size of 70 is sufficient to obtain the best possible results.

The efficiency results for initial semantic ranking are encouraging: although increasing the window size leads to increased runtime requirements, the growth is modest. For example, moving from a window size of 50 to 104 examples only increases the average runtime from 8.2 to 9.2 seconds. Furthermore, a larger window can even decrease the number of candidates that are generated (which peaks at a window size of 30). This is because more suitable grammar rules and instances become available which allow the generation of better candidates. However, if speed is more important than returning a solution to the maximal number of inputs, dynamic rule and instance selection is a practical solution. Moreover, it could be used with much larger grammars and instance bases to limit the runtime requirements of the generator. In addition, performing initial semantic ranking is very cheap: it takes less than 50 msecs on average. This is because computing the similarity between input and (semantic) instance base is in general no more costly than scoring a single chart edge.

In sum, initial semantic ranking appears to be a technique that allows the instance-based ranker to use much larger instance bases and automatically constructed grammars than we have done in this work. This technique would be especially useful if it should turn out that the results of the scalability experiments reported in section 7.5 do not apply beyond the currently available corpus.

7.7 Manual experiments

7.7.1 Testing tag-index checking

In section 4.3 we introduced tag-indices as an additional means of representation to indicate a close relationship between two tags and avoid confusion when a tag name occurs more than once in the input. Section 5.5 details the general mechanism that is employed when edges carrying tag-indices are combined. In this section, we look at some candidates that are generated from input that specifies tag-indices. We use mixed ngram representations with λ=0.7.

7.7.1.1 Repeated OTHERPOST and OTHERCOMP tags

One of the major sources of tag repetitions is the reoccurrence of OTHERPOST and OTHERCOMP tags. (139) shows a simple manually created input that contains these tags:

(139)

index	tag	slot filler
1	IN_OTHERPOST_NODET	other-post-1
1	IN_OTHERCOMP	other-comp-1
2	IN_OTHERPOST_NODET	other-post-2
2	IN_OTHERCOMP	other-comp-2
-	IN_FULLNAME	George Miller
-	POST_NODET	president
-	COMP_DESCR*	natural-gas-pipeline concern

Without tag-index checking, the generator produces unwanted candidates like the following:

(140) George Miller , other-post-2 of other-comp-1 , other-post-1 of other-comp-2 , was named president of this natural-gas-pipeline concern . (cos=0.48, cov=7/7)

Apart from not being faithful, candidate (140) also does not seem to be particularly fluent (note the relatively low cosine). However, it does express all the input tags. Using tag-index checking, we generate the following candidates, among others:

(141) a. George Miller, other-post-1 of other-comp-1, other-post-2 of other-comp-2, was named president of this natural-gas-pipeline concern.

b. George Miller, other-post-1 and other-post-2 of other-comp-2 was named president of this natural-gas-pipeline concern.

c. George Miller, other-post-2, becomes president of this natural-gas-pipeline concern.

Candidate (141a) combines the input tags in the desired way. However, the other two candidates do not. (141b) places 'other-post-1' at 'other-comp-2'. As described in section 7.4.3, we consider a candidate like (141c) unfaithful because 'other-post-2" needs to be interpreted as belonging to a company already introduced into the discourse context.

Structurally, (141b) first correctly combines 'other-post-2' with 'other-comp-2' and cancels off the matching index 2. However, it then also combines the result with 'other-post-1' to its left. This means that resulting sentence still contains an index that has not been cancelled off. One way of dealing with this problem is to exclude sentences with a non-empty index set as valid candidates. This would also prevent candidate (141c). However, in more complex cases with several indices this would be too strict because faithful but non-complete candidates can also have non-empty index sets. In some cases, this would effectively amount to a hard-wired completeness constraint. The solution to the problem lies in the tag cooccurrence constraints that have not been used here (see section 7.4.3.1). They make sure that certain pairs of tags are always expressed together. In addition to (141a), the generator produces the following tags when cooccurrence constraints are applied:

(142) a. George Miller, other-post-1 of other-comp-1, was appointed president of this natural-gas-pipeline concern.

b. George Miller, other-post-2 of other-comp-2, was appointed president of this natural-gas-pipeline concern.

c. George Miller was named president.

(142c) shows that the system can still choose not to express any of the indexed tags.

These examples show that it is the combination of tag-index checking with tag cooccurrence constraints that finally yields the desired output. The observations about

OTHERPOST and OTHERCOMP tags also apply to PREVIOUSPOST and PREVIOUSCOMP tags which are used in a similar way in the corpus. To verify that the mechanisms work correctly, one needs to look down the entire candidate list since hard constraints should apply to all candidates.[12]

7.7.1.2 Repeated AGE and FULLNAME tags

Another source of repetitions in the corpus are AGE tags in combination with several FULLNAME tags. In this experiment, we would like to test whether the system associates person names correctly with their ages. (143) shows an input that is an extension of a training set example (see 56e, page 118). This input also tests whether the INPERSON_OTHERPOST and INPERSON_OTHERCOMP are combined with the correct persons:

(143)

index	tag	slot filler
-	COMP_DESCR*	engine maker
5,6	POST_BOARD	board
3	INPERSON_OTHERCOMP	Universal Foods Corp.
3,4	INPERSON_OTHERPOST_NODET	chairman
4,6,8	INPERSON_FULLNAME	John L. Murray
1	INPERSON_OTHERCOMP	Lubar & Co.
1,2	INPERSON_OTHERPOST_NODET	chairman
2,5,7	INPERSON_FULLNAME	Sheldon B. Lubar
7	INPERSON_AGE	55
8	INPERSON_AGE	44

As for the previous input (139), tag cooccurrence constraints apply to the OTHERPOST and OTHERCOMP tags but there are no further relevant constraints. (144) shows some of the generated candidates:

(144) a. John L. Murray, chairman of Universal Foods Corp., and Sheldon B. Lubar, 55 years old, chairman of Lubar & Co., were elected to the board of this engine maker. (cos=0.95, cov=9/10)

b. John L. Murray, 44 years old, and Sheldon B. Lubar, chairman of Lubar & Co. were elected to the board of this engine maker. (cos=0.68, cov=7/10)

[12]In principle, one also needs to check the candidates that are pruned away by expectation-based search.

c. Sheldon B. Lubar, 55, was elected to the board of this engine maker. (cos=0.75, cov=4/10)

d. Sheldon B. Lubar, 55 years old, were elected to the board of this engine maker. (cos=0.57, cov=4/10)

Candidate (144a) is ranked highest and exhibits the maximal coverage the generator seems to be able to obtain. (144b) expresses fewer input tags but also correctly combines the realized ones. We did not observe any incorrect combination of tags in the candidate set, for example 'Sheldon B. Lubar, 44 years old, ...'. Furthermore, the plural verb form has been used correctly in (144a) and (144b). These candidates are motivated by the nearest neighbour from which the input was derived. They therefore use the same verb form. (144d) incorrectly uses the plural verb form but it is beaten in the ranking by (144c) which is motivated by a nearest neighbour that also realizes only a single `FULLNAME` tag. Similarly, a version of (144a) that uses a singular verb is ranked lower than the correct version.

These examples should suffice to convince us that the basic index mechanism serves its intended purpose. It is not claimed that it will produce correct results in all circumstances. However, this cannot be the aim of such a shallow mechanism. Furthermore, it should be noted that the (largely) context-free grammar does not employ grammatical agreement constraints and decisions about verb number can be incorrect in some cases.

7.7.2 The usefulness of singleton tags

Singleton tags cannot be tested in gold standard experiments because they are missing in either training or test set. However, we can of course use singleton tags of the training set in combination with manually constructed inputs. The more interesting question is whether the grammar rules and the instances that use the singleton tag are applicable beyond the particular example from which they were derived. To investigate this, we identified singleton tags in the training set and tested variations of the inputs in which they occurred. In other words, we took reproduction as a starting point and then deviated incrementally from there.

We found that the system tends to realize the singleton tag in the highest ranked candidate, motivated by its closeness to the original nearest neighbour and the high weight of rare terms in *idf*-based term weighting. However, while extensions of the original input were usually possible, changes or omissions of tags could render the realization of the input impossible. This is because the grammar rules record patterns of tags in the node labels. An example rule and the corresponding fragment of the original annotated sentence:

(145) a. `S-POST_INDEF` $\rightarrow$ `NP-POST_INDEF PP-COMP_DESCR_RC*`

b. ... was elected a `[POST_INDEF` director`]` of this `[COMP_DESCR_RC*` company, which primarily has interests in radio and television stations`]`, ...

The `PP-COMP_DESCR_RC*`[13] tag only occurs in the form of (145b) in the corpus, and (145a) is the only phrasal rule that can combine this tag. (Node `PP-COMP_DESCR_RC` is generated by an input rule). Thus, the grammar also needs a `POST_INDEF` tag for rule (145a) to fire. For example, we cannot express 'was elected president of this company, which ....' since this requires a `POST_NODET` tag. A practical solution might be to generalize rules such as (145a) to match any `POST`-related tag (see also section 8.3.1.2). On the other hand, the current rule format still allows the realization of a large variety of subject NPs in combination with a VP that incorporates the `S-POST_INDEF` node generated by rule (145a).

In sum, the system is able to realize input tags that only have a single occurrence in the training corpus in novel combinations of tags. However, this often requires certain other tags to be present in the input as well. This is explained by the fact that the grammar rules record patterns extracted from specific treebank parses. In other words, the less frequent a tag in the training set, the less flexibly it can be used to express novel input. On the other hand, frequent tags will tend to have many different realizations.

[13]The `PP-COMP_DESCR_RC*` tag is also mentioned in (31d) in chapter 4. In (31d) it does not yet carry the '*' marker we introduced in section 7.4.3.2.

7.7.3 Learning exceptions

It is generally claimed that instance-based models learn exceptions well (e.g. Daelemans et al., 1999a). To investigate this issue we need to clarify what characterizes exceptionality. Generally, we would liken 'exceptional' events to 'rare' or 'infrequent' ones. In our approach, these can be tags, template words or cooccurrence patterns of these. The previous section has indicated that the system is able to use singleton tags beyond the original sentence to some degree. It tends to use the instance representation of the original sentence as its nearest neighbour because of the high weight of rare terms in *idf*-based term weighting. The system thus learns the particular usage pattern of that rare instance as well.

Generally, there is no explicit distinction between 'exceptional' and 'regular' cases in the instance base. Each instance is valid in its own right, and serves as a potential nearest neighbour for the generated candidates. For example, the well-known first sentence of the Penn treebank, although looking quite 'normal', is also exceptional in the sense that it is the only sentence in our corpus that uses 'will join':

(146) `[INPERSON_FULLNAME` Pierre Vinken`]`, `[INPERSON_AGE` 61`]` years old, will join the board as a `[POST_INDEF` nonexecutive director`]` `[INDATE_FUTURE` Nov. 29`]`.

There is one other use of 'join' in the corpus (in the present tense):

(147) ... joins the `[POST_BOARD` board`]` of this `[COMP_DESCR*` cement products company`]` `[INDATE_FUTURE` on Dec. 1`]`.

From (146) and (147) we might conclude that there is a hidden regularity in the corpus that wants to realize 'join' verbs in combination with `INDATE_FUTURE` tags. In fact, when we remove the `INDATE_FUTURE` tag from the input in these two cases, the system switches to other nearest neighbours, using 'was elected' and 'has been elected to the board' instead of 'join'. The question we have to ask is whether it is the instance model or the rule system that is responsible for this 'regularity'. It turns out that it is the rule system in this case:

(148) a. `VP-POST_INDEF` $\rightarrow$ `will join the board PP-POST_INDEF NP-INDATE_FUTURE`

b. `VP-POST_BOARD` → `joins NP-POST_BOARD PP-INDATE_FUTURE`

However, the instance base plays a rôle as well. To show this, we experimentally removed the suffixes that define definiteness of `POST` tags from the grammar to widen the range of candidates. The rule system now generates the following candidates for a changed version of the input that can be extracted from (146):

(149) a. Pierre Vinken, chairman, will join the board as a nonexecutive director Nov. 29. (cos=0.91, cov=4/4, nn=wsj_0001)

b. Pierre Vinken, chairman, was named a nonexecutive director, effective Nov. 29. (cos=0.43, cov=4/4, nn=wsj_2135)

The best candidate in this experiment is (149a), still using the verb 'join' and the instance representation of (146) as its nearest neighbour. Candidate (149b), which can be found at rank 6, is the first one to use another verb.

The corpus has more uses of `INDATE_FUTURE` tags than there are 'join' verbs, and thus there are cases when the `INDATE_FUTURE` tag is expressed in combination with other verbs. Vice versa, it is unlikely that a 'join' verb is used without a `INDATE_FUTURE` tag. A combination of the two will score high on the instances that contain both but a candidate that uses 'join' without `INDATE_FUTURE` is bound to loose against its competitors. In other words, the instance-based ranker learns that there is an 'exceptional' cooccurrence pattern of terms containing `INDATE_FUTURE` and 'join'.

In conclusion, the ability to learn exceptions is built-in in the instance-based generator. In some sense, every instance is its own 'exception'. Regularity only arises when several instances exhibit the same pattern. However, regularity is never explicitly represented.

7.7.4 Robustness

The 'robustness' of a generation system has a number of dimensions. In the following, we discuss a few of them. The first issue cannot be tested by gold-standard experiments:

- **ill-formed input:** The input format is a simple set so that there should be no danger for some external component to get the input structure wrong. The leaves the potential problem of ill-formed members of the input set. This is relevant to the grammar component which needs to match the input against the rules. The bottom-up chart algorithm is able to deal with ill-formed input tags by simply ignoring the ill-formed tags. From the point of view of the grammar, unknown input tags are no different from 'ill-formed' input tags. Furthermore, it seems difficult to conceive ill-formed fillers since, at least in principle, the fillers do not need to be used by the generation system until the selected candidate templates are completed and presented to the user.

  Another possibility of ill-formed input is input that contains semantically incorrect information. For example, we can specify an input that assigns two `AGE` tags to the incoming person. The generator is able to express that information (for example: 'Pierre Vinken, 58-year-old chairman, 61 years old, ...') but it is beyond the capabilities of the system to reason whether this actually makes sense.

- **ill-formed grammar rules:** Rules that are 'ill-formed' can be grammar rules that are obviously incorrect syntactically, for example. An earlier version of the grammar construction algorithm introduced the following coordination level rule into the grammar:

  (150) `VP-POST_NODET` $\rightarrow$ `NP-POST_NODET and NP-POST_NODET`

  This allowed the generation of candidates such as (151)

  (151) Clark J Vitulli senior vice president and general manager.

  However, candidates without verbs never showed up among the first candidates. The explanation can be found in the instance base: there is no instance without a verb, and thus, a candidate like (151) never wins against a competitor that contains a (matching) verb.

- **ill-formed instances:** 'Ill-formed instances' is another term for 'noisy data'. Since the generator makes its decision based on the instance base, it is indeed sensitive to noise which it cannot distinguish from well-formed exceptions. However, an instance base for generation is likely to be carefully selected and annotated. Furthermore, we can detect the ill-formed instance if it contains an unwanted candidate (and if that instance is never used, the noise is not harmful).

Improving the annotation scheme can be seen as a way to make the instance base less noisy. For example, generating 'was elected to the president' was a typical error of an earlier version of the annotation scheme. The introduction of the `POST_BOARD` tag (see section 4.2.1) prevented the generation of the above-mentioned VP. By refining the annotation scheme we have removed some 'noise' from the instance base.

7.8 Conclusions

In this chapter, we have reported on a number of experiments that served to evaluate different aspects of the implemented instance-based generation system. First, we identified the ability to reproduce the training set as an important property of an instance-based generator. The experiments have shown that the system is indeed capable of reproducing the training material. This is not necessarily the case for more eager approaches, for example those that convert the training corpus into a statistical model. In contrast, a (correctly implemented) instance-based generator never performs worse than a rote learner on the training data. Furthermore, we observed that there can be more than one instance that completely expresses the input semantics. In such cases, a candidate reproducing any of these instances should be seen as a correct solution to the reproduction task.

Next, we ran the instance-based generator on test data and observed that it largely produces fluent, grammatical and faithful output. However, the basic version of the generator also tends to produce candidates that do not fully express the input semantics although they perfectly match with some instance. This problem of completeness results from the combination of a flexible rule system and an instance-based learner. We proposed two techniques to deal with this problem and have evaluated the improved generation system in a number of ways. By preventing the generator from realizing mere subsets of the input semantics, the linear combination with a coverage value takes better advantage of the inherent 'exceptionality' of many members of the instance base. This manifests itself in the increase of support instances, for example. The linear combination allows one to trade fluency against completeness. We discuss the generality of this fluency-completeness trade-off in section 7.8.1 below. In addition, we proposed maintaining multiple candidate lists which allows the system to delay the final decision about the completeness of the output candidates until a larger context is known.

We observed that faithfulness problems can arise if the generation output is incomplete and introduced a number of measures to improve the faithfulness of the candidates. The results of a human evaluation of the system output (using the linear combination) showed that up to 92.5% candidates are perfectly correct for short candidates (low values of λ), decreasing to 57.5% for more complete candidates (high values of λ). As a general tendency, candidates that have a high similarity score exhibit less errors than those having a low similarity score, confirming our basic assumption that the generator should aim at producing candidates that are similar to the training set instances.

Although the overall corpus size is small, the results summarized above indicate the validity of our approach. In fact, being able to work with limited amounts of training data is one of the strengths of this approach. We also conducted a number of experiments that addressed the scalability of the instance-based generator. The results were encouraging: in addition to improving fluency (as measured by the instance-candidate similarity score), the efficiency of the system increases when moving from a very small instance base of only a few examples to an instance base of size 70. It still seems possible that a much larger instance base (and grammar) will slow the system down. However, in that case, subsets of the instance base and the grammar can be dynamically selected to increase efficiency. Thus, we conclude that instance-based generation is technically feasible, scalable and reasonably efficient.

In a number of manual experiments, we convinced ourselves that the mechanism for tag-index checking serves its intended purpose of preventing combinations of tags with incompatible tag-indices. In addition, we found that the system is able to realize tags that occur only once in the training set in novel combinations of tags, albeit with a limited degree of freedom. Furthermore, the instance-based generator is robust with respect to ill-formed input and even ill-formed grammar rules. The biggest danger for the instance-based generator probably lies in ill-formed instances. However, we claim that in our approach this is not a serious practical problem (see also section 7.8.2 below).

Throughout this chapter, the discussion of individual candidates illustrated the workings of the instance-based generation system (see, for example, figure 7.10 which shows that the system switches between nearest-neighbours as the instance base grows larger). The nearest neighbours of the chosen candidates can be regarded as 'explanations' of the decisions made by the system. In principle, the derivation tree of the candidates should also be seen as part of an explanation of how the output came about (as the explanation of the rule-based part of the hybrid system).

In the following, we draw conclusions about the generality of the completeness-fluency trade-off and take up a number of fundamental issues concerning instance-based learning.

7.8.1 Generality of the completeness-fluency trade-off

Ranking models for NLG that are based on corpus similarity alone (even if the corpus contains additional syntactic or semantic information) make the assumption that all candidates are (equally) complete and (equally) faithful. If this assumption is relaxed – as we have done in this work – the possibility of a trade-off between fluency and other goals arises. This does not depend on the search strategy or the generation architecture and also applies to systems in which grammar interpreter and ranker are non-interleaved.

In instance-based ranking, the tendency towards replicating training set sentences is easily observable because the original examples are available as nearest neighbours which can be imitated directly. In statistical approaches, the corpus material is generally not accessible. However, in case statistical NLG systems do not ensure completeness, they can face similar problems of comparing candidates. For example, ngram models tend to prefer sequences of words that are much shorter than any corpus sentence. Knight and Hatzivassiloglou (1995) describe using a function which increases with sentence length to counter this behaviour. Avoiding minimising sentence length can be seen as an indirect way to approximate a completeness criterion.

7.8.2 Instance-based learning

The experiments in this chapter are related to a number of general claims about instance-based approaches that were mentioned in section 1.2.3. Here, we would like to point to the results of this research concerning these claims:

- **Dealing with sparseness of data.** The evaluation of scalability (section 7.5) shows that very few examples are indeed sufficient to generate solutions for combinations of input tags not seen in the training corpus. The fluency and the coverage of the output sentences increases as more training material becomes available. Generally, the ranker is able to deal with edges that contain large numbers of unknown terms (see section 7.3.1).

- **Runtime efficiency.** This issue becomes particularly important when an instance-based learner is combined with rule-based overgeneration. Expectation-driven A^*-search turns out to be a practical solution to this problem. The scalability results of section 7.5 are particularly encouraging.

- **Flexibility.** The instance base is flexible in the sense that it can be adapted to the current input. Instances (and rules) can be selected dynamically as the technique we call 'initial semantic ranking' shows (see section 7.6).

- **Learning exceptions and sensitivity to noise.** By treating instances individually, the instance-based system ensures that their 'exceptionality' remains available as a blueprint for the output candidates (see section 7.7.3). On the other hand, this poses the danger of also learning noisy data. However, as we have argued in section 7.7.4 ('ill-formed instances'), our semantic annotation is generally carefully crafted. In addition, in an instance-based approach erroneous examples can always be identified and corrected.

This concludes our evaluation of the implemented instance-based generator. In the next chapter, we discuss a number of general issues and directions for future research. Section 8.1 addresses the question of test set reproduction, an issue that arises naturally in a corpus-based approach to NLG that separates training from test set.

Chapter 8

Discussion, future work and applications

In this chapter, we take up a number of issues concerning our approach to instance-based natural language generation. We start by discussing an issue concerning the automatic evaluation of the instance-based generator. We then relate our approach to statistical NLG in general. A working realization system has many facets, and we will discuss some extensions and alternatives. Naturally, these options do not belong to the tried-and-tested methods of the previous chapters and need to be investigated further. Finally, we discuss possible applications of our approach.

8.1 The question of test set reproduction

The general experimental setup for test set evaluation (depicted in figure 7.6, section 7.3) suggests as a natural evaluation metric a comparison between the generation output and the original annotated sentence from which the input was extracted. Automatic evaluation with respect to reference strings bears close similarities with an instance-based approach to candidate ranking since it also directly compares pairs of strings/templates. Since we have argued that the cosine similarity of surface templates is a suitable metric for the ranker, we should use that same metric for the comparison between output and original sentence. Using a different metric would imply that we believe the newly introduced metric to be more suitable for our similarity computations – and in that case, the ranker should be able to use that same metric.

We conducted a test set reproduction experiment in which we reused candidates that were generated by a system trained on 104 sentences and tested on 40 test set inputs. In that earlier experiment, we employed mixed term representations and $tf.idf$ term weighting (see section 7.4.1). In order to measure the reproduction of the original test sentences, we needed to compute the similarity of the generated candidates to these original sentences. At this point, the question arises what instance model to use for the similarity computations. Since generation has finished and we are examining the result, we are allowed to use the entire corpus for similarity computations (rather than just the training corpus). Thus, we computed a new instance model for the entire corpus, using again mixed ngrams and $tf.idf$ weighting.

	training set (104)				globally best score (144)			score on orig		
λ	cos.	std.	cov.	compl.	cos	std.	nn=orig	cos	std.	repro
0.0	0.94	0.12	0.67	15.4%	0.94	0.13	2.6%	0.26	0.23	2.6%
0.1	0.94	0.12	0.68	15.4%	0.94	0.13	2.6%	0.27	0.23	2.6%
0.2	0.94	0.12	0.69	18.0%	0.94	0.13	2.6%	0.27	0.23	2.6%
0.3	0.94	0.13	0.71	23.1%	0.93	0.13	2.6%	0.28	0.23	2.6%
0.4	0.90	0.13	0.77	33.3%	0.90	0.12	5.1%	0.32	0.24	2.6%
0.5	0.88	0.13	0.79	35.9%	0.88	0.14	5.1%	0.33	0.24	2.6%
0.6	0.77	0.17	0.88	56.4%	0.80	0.16	18.0%	0.37	0.25	5.1%
0.7	0.75	0.16	0.90	64.1%	0.76	0.17	23.1%	0.39	0.26	5.1%
0.8	0.73	0.18	0.90	64.1%	0.75	0.17	23.1%	0.41	0.26	5.1%
0.9	0.73	0.18	0.90	64.1%	0.75	0.17	23.1%	0.41	0.26	5.1%
1.0	0.73	0.18	0.90	64.1%	0.75	0.17	23.1%	0.41	0.26	5.1%

Table 8.1: Similarity scores of generated candidates on training set, entire corpus and original sentences

Table 8.1 shows the results of the test set reproduction experiment. The column `training set (104)` just repeats the results of the earlier experiment in which the system was trained on 104 sentences and tested on 40 test set inputs (see table 7.2).[1] Column `globally best score (144)` in table 8.1 shows the cosine score of the 40

[1] In table 8.1, `cov.` is the average coverage value of the candidates; `compl.` is the number of complete candidates for the 40 inputs.

best candidates with the closest of the 144 corpus examples. This globally best score decreases from 0.94 to 0.75 as the candidates become more complete, which can be explained by the fact that the generator is forced to express more rare tags and tags combinations.

The globally best scores are generally quite similar to the ones for the training set model, suggesting that the training set model makes good use of the available data. For larger values of λ, the globally best score remains higher than the training set score. (However, the similarity scores are not directly comparable because they are obtained using term weights computed for different corpus sizes.) Column '`nn=orig`' in table 8.1 shows in how many cases (out of 40) the globally best score of the candidate was actually obtained by using the original sentence as the nearest-neighbour. This number increases with the completeness of the candidates but generally stays fairly low (the maximum is 23.1%).

Column '`score on orig`' in table 8.1 shows the similarity score of the generated candidates with the original examples. This score is relatively low but increases slightly with the coverage of the candidates which can be explained by the fact that at least the unigram terms containing the input tags match the ones of the original sentence template. However, exact reproduction is rare: only in maximally 5.1% does the generator reproduce the original test set sentence.

These results indicate that the system is using the training data well (high training set similarity scores) but that the system is not reproducing the test corpus to a large extent. In other words, the system is able to produce fluent candidates that are quite unlike the original sentences/templates from which the generation input was extracted. Obviously, there is more than one 'good' way of expressing the input semantics, and in many cases, our system chooses realizations that are different from the original sentences.

In general, automatic evaluation could significantly facilitate the development of NLP systems. Measures for automatic evaluation in statistical NLG and Machine Translation have been proposed by (Bangalore et al., 2000) and (Papineni et al., 2001), for example. Unfortunately, the results we obtained from the experiment described above do not seem to indicate that automatic evaluation could greatly contribute to the development of our NLG system, especially if automatic evaluation is based on the idea of reproducing the original sentences. We had hoped that we might be able to determine an optimal value of λ, for example. Instead, the similarity scores of the candidates and the original sentences just suggest that the candidates should be as complete as possible. Furthermore, the generation output seems to be of a better quality than the low reproduction rate and the low similarity score (with the original sentences) suggest. As a consequence, we remain skeptical as to the possibility of fully automatic evaluation. However, more research is certainly needed to clarify this point.

8.2 An interpretation of statistical generation

In chapter 2 we discussed a number of approaches to statistical realization. In this section, we demonstrate that statistical realization can be described by a noisy channel model similar to speech recognition and statistical machine translation. We identify a theoretical problem and discuss its consequences for instance-based realization.

8.2.1 Applying Bayes' rule to statistical NLG

A statistical generator tries to find the best realization S for a given meaning input M. Thus, it needs to find

$$\arg\max_S P(S|M) \tag{8.1}$$

We can transform this conditional probability by using Bayes' rule:

$$\begin{align} \arg\max_S P(S|M) &= \arg\max_S \frac{P(M|S) \cdot P(S)}{P(M)} \tag{8.2} \\ &\approx \arg\max_S P(M|S) \cdot P(S) \tag{8.3} \end{align}$$

The denominator on the right-hand side in (8.2) can be omitted since M is fixed, i.e. dropping it does not affect the relation between competing S. As a result of this transformation, the statistical generator has to compute two quantities:

1. The *prior probability* $P(S)$. This is the probability of the candidates without taking into account the meaning input. The 'language model' of statistical NLP systems computes prior probabilities.

2. The conditional probability $P(M|S)$. This translates into asking whether the generation input M is a likely interpretation of the candidate S. In the context of NLG, we can call $P(M|S)$ a 'semantic model'.

Using Bayes' rule to transform an initial conditional probability is done in many areas of statistical NLP (e.g. Manning and Schütze, 1999; Jurafsky and Martin, 2000). It adheres to the metaphor of a *noisy channel*. A prototypical application is statistical machine translation (MT) where it is assumed that a noisy channel distorts a transmitted target language string in such a way that it arrives as a source language string, and the task of the MT system is to reconstruct the input of the noisy channel (the target language string) given the output (e.g. Berger et al., 1994). Likewise, in speech recognition the original task of determining the best word sequence for a given speech signal is transformed into determining the product of the likelihood of the word sequences, i.e. the prior probability, and the channel probability of the candidate word sequences generating the noisy input to the channel.

The rationale behind applying Bayes' rule to statistical NLP lies in the fact that the resulting probabilities tend to be easier to compute than the original conditional probability. The language model can be computed from a training corpus of target texts without knowing anything about the input to the statistical NLP system. A typical example is a ngram language model trained on a large corpus. The conditional probability in (8.3) can be obtained from a corpus of parallel texts (in MT) or from a corpus of speech signal and word sequence pairs (in speech recognition).

The channel model also offers an interpretation of statistical realizers in natural language generation. Ranking output candidates by means of a statistical model trained on target language texts corresponds to using the prior probability of (8.3). However, this ignores the semantic model. This is spelled out explicitly by Knight and Hatzivassiloglou (1995) who employ a bigram language model for candidate ranking:

> "Suppose that according to our knowledge bases, input I may be rendered as sentence A or sentence B. If we had a device that could invoke

new, easily obtainable knowledge to score the input/output pair $< \mathtt{I,A} >$ against $< \mathtt{I,B} >$, we could then choose A over B or vice-versa. An alternative is to forget I and simply score A and B on the basis of fluency. This essentially assumes that our generator produces valid mappings from I, but may be unsure as to which is the correct rendition." (Knight and Hatzivassiloglou, 1995)

In other words: The idea is to require all candidates to express the desired semantics in a sufficient way and then to omit the conditional probability $P(M|S)$.

8.2.2 A potential problem for statistical NLG

Scoring generation candidates only by means of a language model implies that the interpretation M is equally likely for all candidates. This raises the question whether this simplification can lead to undesirable choices by the generation system. Is a human reader likely to be led 'up the wrong track', i.e. towards an unintended interpretation, as a result of the simplification of the statistical model? For example, consider the following two candidate realizations:[2]

(152) S_1 : John was believed to have been shot by Bill.

S_2 : John was believed by Bill to have been shot.

If we assume that the system is given an input conveying a BELIEVED-BY-BILL meaning rather than a SHOT-BY-BILL meaning, the second sentence is the better choice because it makes the intended interpretation more likely. However, if the language model gives a higher probability to the first sentence, i.e. $\mathrm{P}(S_1) > \mathrm{P}(S_2)$, it is clear that S_1 will always win, regardless of the meaning input. In other words, the statistical ranker has no means to take into account that $\mathrm{P}(\mathtt{BELIEVED\text{-}BY\text{-}BILL}|S_2) > \mathrm{P}(\mathtt{BELIEVED\text{-}BY\text{-}BILL}|S_1)$. However, this is exactly what a channel model would contribute if it were available.

The example shows that a candidate sentence may have unwanted interpretations which are more likely than the desired one. In general, the assumption that all candidates convey the intended meaning equally well does not seem to hold. However, the human ability to interpret ambiguous sentences in context may play an important

[2]The example is taken from (Collins, 1999).

role in limiting its effect. In the above example, it may be obvious from the context that Bill cannot be the murderer of John and hence the chosen realization S_1 is given a BELIEVED-BY-BILL interpretation.

8.2.3 Consequences for instance-based generation

Instance-based generation does not convert its training examples into a probabilistic model but rather retains them for direct use in candidate-instance similarity computations. As a consequence, one might be tempted to conclude that the interpretation of statistical NLG given above (including the potential pitfall) does not apply to instance-based NLG. However, we believe that this is not the case. The instance base fulfills the same rôle as the language model in statistical NLG: it serves to judge candidates by means of their 'likelihood' of belonging to a training corpus and it does not take into account the specific generation input.

A possible point of confusion is the fact that in our approach the training set examples are semantically annotated and the candidates contain semantic tags as well. Therefore, we know which tags the candidates can possibly contain. However, this should not be confused with the overall meaning of the candidate sentences. As the investigation of faithfulness problems in section 7.4.3 has shown, the interpretation of tags (once they have been replaced by filler words) can depend on the context of other parts of the sentence. In addition, producing output that contains semantic tags is not a unique feature of the instance-based realizer. The statistical surface realizer of (Ratnaparkhi, 2000) is also trained on a semantically annotated corpus and its candidates contain tags.

If the instance base can be interpreted as performing the rôle of a prior probability, it follows that the interpretation problem described in section 8.2.2 also applies to instance-based generation. In fact, we might argue that faithfulness problems like those originating from the omission of certain tags are caused by the lack of a semantic model. Such a model would need to be able to detect the unfaithful interpretation of the candidate from a parser's perspective. In contrast to the rather subtle ambiguity of example (152), the unfaithful candidates described in section 7.4.3 often seem to have only one (unwanted) interpretation.

We can interpret the introduction of additional semantic constraints on candidates (section 7.4.3.1) as an attempt to control the output of the generator which is otherwise too 'loose' and only oriented toward fluency. However, this may be a price worth paying. In general, the lack of a semantic model in today's ranking-based surface generation systems seems to be due to the difficulty of obtaining the relevant training material. It should be stressed that simplifying assumptions are often the key to practical success and may thus be necessary.

8.3 The division of labour between knowledge sources

The hybrid generation system developed in this thesis employs two sources of knowledge: a rule-based grammar and an instance base populated by training examples. Both components can be regarded as placeholders for directions of science that are often seen as incompatible: the rationalist tradition and the empiricist tradition. It is the lack of rationalist knowledge (in the form of explicit input and grammar specifications) that makes a combination of the two paradigms seem worthwhile. The rule-based grammar contributes the ability to produce previously unseen structures by means of recursive rules ('creativity'). The instance base provides empirical knowledge about the use of language which is often difficult to codify by means of grammar rules alone. On the other hand, using only empirical data tends to lead to conservativeness in the sense that observations of the past are repeated in the future.

Both analytical and empirical knowledge run into problems of completeness in their coverage of language use. These have been termed the 'knowledge acquisition bottleneck' and the 'sparse data problem', respectively. A hybrid approach like the one explored in this thesis combines two incomplete sources of knowledge in the hope of obtaining more complete knowledge than provided by one of the sources alone. Our interpretation of such a hybrid system is that of a *division of labour between knowledge sources*. The instance base represents empirically known data points, and the rule system generates new data points in the space between these. The decision between alternative candidates, i.e. newly proposed data points, is made on the basis of their distance to the empirically justified data points. Furthermore, the candidates are generated by rules that result from an analysis of the instances. Therefore, an important test for the appropriateness of the rule system is the ability of the rules to reproduce the known data points (see section 7.2).

As Knight and Hatzivassiloglou (1995) have argued, corpus-based knowledge can fill in the "knowledge gaps" of a rule-based grammar. This increases the robustness of the generation system by producing output in situations where input or grammar specifications are missing. It can be regarded as a form of "emergent choice" (Ward, 1994). In most cases, robustness is a desirable property. However, there may be applications for which no output is better than an approximately correct one.

Ideally, the knowledge sources are complementary and, when combined, result in 'complete' knowledge at least of the domain in question. In practice, knowledge gaps may remain, resulting in errors in the system output. This is our interpretation of the remaining errors made by the implemented generator. Consequently, there are generally two locations where improvements can be made: the grammar component and the ranker. Generally, fluency problems should be dealt with in the ranking component. Syntactic problems could be dealt with in either the grammar or by making the ranker aware of syntactic structures (see sections 8.3.1.1 and 8.3.2.2 below). Semantic problems seem to require improvements in the annotation scheme, possibly in combination with additions to the grammar component (like the additional filtering implemented by tag cooccurrence constraints). In the following, we indicate a number of options for changing grammar and ranker that could be pursued in the future.

8.3.1 Improving the grammar component

8.3.1.1 Extending the current approach

The current approach of automatically constructing a generation grammar from a syntactic treebank could be extended in a number of ways:

- We could add extra knowledge to the existing mechanisms. For example, we could encode that 'both' needs to refer to exactly two coordinated constituents. Furthermore, grammar construction could make a distinction between singular and plural VPs, taking advantage of the recognition of coordination.

- The grammar could use recasting rules for semantic tags to increase its paraphrasing power. This will involve not only a change of the tag names but in many cases also of the slot filler, as the latter always needs to be of a specific syntactic category. For example, a singleton tag such as `POST_INF` ('infinitive') could be recast by other `POST` tags that require a noun or a noun prefixed by an adjective. Other recasting rules could be applied to slot fillers such as 'vacant' and 'vacancy'.

- Additional information about heads and complement/adjunct distinctions could be employed to extract subcategorization frames.

- We could make use of the long-distance dependencies (traces) of the Penn treebank which are currently ignored.

- One could use a treebank based on a principled grammar theory such as Combinatory Categorial Grammar (Hockenmaier and Steedman, 2002).

- The semantic markup could be based on (typed) feature structures. Generation using such a formalism requires a unification-based grammar formalism (see next section).

8.3.1.2 Unification-based grammar formalisms

Unification seems to be well-suited for overgeneration approaches to NLG because of its ability to represent underspecified structures and the possibility of fully exploring the space of candidates within a given set of constraints. Underspecification in unification-based grammar formalisms is a means of encoding uncertainty. It could be exploited to allow the instance-based generator to accept input at different levels of linguistic description, making it more versatile. Input specifications would generally act as constraints on the space of possible candidates. Although the grammar format used in this work seems to be adequate for the task, applications like summarization or machine translation (see section 8.6) might benefit from input that contains not only semantic but also lexical and syntactic constraints.

To be practically interesting for our approach, the grammar construction procedure should provide a richer representation of grammatical descriptions than we currently obtain from our annotation scheme and the Penn treebank II. On the other hand, if a large number of rules – based on typed feature structure representations, for example – can indeed be automatically constructed, the rule-edge matching problem needs to be rephrased with respect to unification. Under the assumption that many rules partially share some feature structure description, subtests for feature structure unifications might be shared across rules. In effect, the general unification operation would be split into subtests whose results are reusable. These tests could be organized in a Rete network in such a way that sharing is maximized.

The interleaved architecture proposed in this work also allows an instance-based ranker to be combined with a chart generator that uses a hand-crafted unification-based grammar. The main requirement would be to give the ranker access to newly generated edges and to their *phon* feature. We would then be able to investigate whether the grammar can be simplified due to the availability of an instance base. This in turn might simplify manual grammar development since examples would be used to avoid overgeneration. In other words, the research question would be whether providing examples can help avoiding excessively fine-grained features.

Furthermore, it may be possible to map the simple 'flat' semantic input used in this work to underspecified representations in unification grammars, for example using Minimal Recursion Semantics (Copestake et al., 1999). This in turn would offer a way of combining a shallow NLP component (for example, an information extraction system) with a principled grammar formalism for realization (see also Crysmann et al., 2002).

8.3.2 Improving the instance-based ranker

8.3.2.1 Extending the current approach

The three basic ingredients of an instance-based ranking function are the term representation, the term weights and the distance metric (section 3.1). Many variations are possible for these parameters. Concepts from information retrieval can be applied to instance-based NLG because of the close connection between the fields. In sections 8.3.2.2 and 8.3.2.3, two alternatives are discussed briefly. Section 8.3.2.4 then discusses how slot fillers could be incorporated in the ranking function. Finally, section 8.3.2.5 discusses the combination of similarity scores from a number of instances.

8.3.2.2 Syntactic features

The surface-based representation of edges and instances can be extended to also include syntactic features. However, it is unclear whether syntactic information actually improves candidate ranking for generation. Experience in information retrieval has shown that improving on the standard retrieval methods by adding syntactic features is difficult (Lewis and Jones, 1996). Furthermore, it should be noted that the incorporation of syntactic information would decrease the portability of the generator in the sense that grammar formalism and instance base needed to provide matching syntactic features. On the other hand, the ranking tasks in NLG and IR are different enough for this to merit further investigation. Furthermore, results presented by Daume III et al. (2002) indicate that, with respect to statistical language modelling, syntactic information could improve candidate ranking for NLG over ngram models.

In order to obtain syntactic structures in the style of the original treebank during generation, the grammar rules would need to specify the original treebank fragments which would be combined when edges are combined. This way, one could recover the original treebank parse if the appropriate semantic input is given to the generator. The syntactic structure might be useful not only for the ranker but also as input to further processing steps. However, in our approach, representing the full treebank parse is not necessary.

8.3.2.3 Latent Semantic Indexing (LSI)

LSI (Deerwester et al., 1990) is a technique for reducing the dimensionality of high-dimensional spaces that maps coocurring document terms onto the same dimensions, and non-coocurring terms onto different dimensions. Applying LSI to instance-based NLG would mean mapping instances as well as edge contents to a reduced vector space before the conventional similarity metric is used. LSI seems to be most useful when applied to large, heterogeneous collections of documents (Manning and Schütze, 1999). However, it remains an open question whether LSI is also suitable for small, domain-specific corpora because it amounts to losing some of the valuable information about the coocurrence of surface words that is available.

8.3.2.4 Slot filler preferences

In the current version of the system, instances and edge contents are represented in such a way that the ranker only sees the semantic attributes but not their values, or 'slot fillers'. This makes good sense for small corpora since it generalizes from the training data and avoids learning the wrong exceptions. For example, many proper names occur only once in a small training corpus. If an input attribute happens to contain the same proper name as one of the training examples, this would wrongly gear the generation process towards this nearest neighbour. On the other hand, the current approach cannot learn realization differences for certain slot fillers of the same attribute (this has also been noted by Ratnaparkhi, 2001). Apart from particular filler words, the length of the filler string will play a rôle in determining the overall length of a sentence, for example. Therefore, the introduction and appropriate weighting of features pertaining to slot fillers is worth further investigation.

8.3.2.5 Mixing instances

In chapter 7, we saw that the decisions of the ranker are always based on a single instance at a time, combined with 'light' terms which add a statistical (frequency-based) component to the decisions of the ranker if the term weighting scheme is based on inverse document frequency. An alternative is to use as nearest-neighbours combinations of instances which are chosen with respect to the input. This does not necessarily mean that instances are merged in practice but rather that their similarity scores are combined, resulting in an adaptive k-nearest neighbour model. Using combinations of examples has also been explored for other tasks such as the translation of line drawings from one style to another (Freeman et al., 1999). For generation, we would aim at finding combinations of instances whose semantics fits the input and use them as nearest neighbours for candidate surface forms. Furthermore, the combined score could incorporate a measure of *dissimilarity* to unwanted instances, for example those used for a previous input, in order to increase variation.

8.4 Extending the scope of instance-based generation

8.4.1 Text generation

In this work, we have limited ourselves to the generation of the first sentences of the articles in the chosen domain. Some of the articles indeed only contain a single sentence. The majority, however, include a number of follow-up sentences. The average text length is 2.8 sentences. The follow-up sentences tend to provide background information about the main succession event reported in the first sentence and become increasing diverse as the article continues. Two examples are given in (153).

(153) a. Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29. Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group. `(wsj_0001)`

b. Mark Q. Huggins was named executive vice president and chief financial officer. Mr. Huggins, 39 years old, formerly was controller and chief accounting officer at Harte-Hanks Communications Inc. Management Co. manages entertainers and produces, markets and finances entertainment. `(wsj_0731)`

The current instance base contains the first sentences of these articles. The annotation scheme developed for the first sentences would be able to mark the follow-up sentences in (153). However, the treatment of referring expressions such as *Mr. Vinken* is a major requirement for text generation. Furthermore, extensions of the basic annotation scheme would be required to annotate the follow-up sentences of other articles. In the following, we indicate how issues of text generation could be addressed within our approach.

- **Annotation and candidate generation.** Referring expressions would be annotated as pointing towards the antecedent tag they refer to. For example, *Mr. Vinken* could be marked a `TITLE_NAME` pointing to a specific `INPERSON_FULLNAME`. The grammar would only be allowed to produce referring expressions that refer to tags present in the input. Since the generator cannot consume a part of the input more than once, rules that produce referring expressions should be non-consuming. One could investigate whether the concept of tag-indices can be applied to referring expression generation.

- **Ranking.** The extended grammar would (over-)generate texts including referring expressions. In this setting, it is the task of the ranker to choose not only fluent but also coherent text.

  There is, in general, a large number of options for instance-based text ranking. One is to continue to use an instance base of sentences but to populate it with sentences from different positions in original articles. However, a last sentence of the original text obviously should not serve as a nearest neighbour for a first sentence. To impose a global text structure onto the local sentence model, one could experiment with representing the original texts by ngrams of their tags, crossing sentence boundaries. Such ordering information would indirectly express that articles start with a headline or that referring expressions appear to the right of their antecedent. Furthermore, one could experiment with discontinuous ngrams. An alternative approach is to simply represent an article as a single bag-of-ngrams and treat full stops (sentence boundaries) just like words.

- **Efficiency.** Since the grammar adheres to the principle that any part of the input can only be consumed once, sentence-sized candidates should be generated into semantic bins (see section 7.4.2) with follow-up sentences trying to consume the remaining input. However, text generation in an overgeneration approach places additional requirements on efficiency. We might thus be forced to generate incrementally sentence-by-sentence and prune the number of bins at each step to keep the number of continuations tractable.

- **Completeness.** Inputs in the single-sentence approach which yield low fluency (i.e. cosine) values if emphasis is placed on completeness suggest that the content should better be distributed across two sentences. However, the fundamental problem of completeness versus fluency will repeat itself on the text level. On the other hand, at text level there are more ways of expressing the content which should ease the tension between the two goals.

Above, we have sketched a relatively knowledge-poor text generation system following the approach of this work. Basically, we are confident that we could over-generate texts as well and see the challenge in providing the appropriate ranker. An alternative view is to improve the grammar component in such a way that only coherent text is generated in the first place. However, it should be emphasized that further research is required to verify the feasibility of these ideas.

8.4.2 Scalability and clustering

In this work, we assume that edges are compared with individual instances. If a training set considerably larger in size is used, it may improve efficiency to cluster instances off-line and change the A^*-based search algorithm to work in two phases. In the first, search works with cluster centroids to narrow down the number of "relevant clusters". The expectation would need to be more tolerant to account for all material members of the cluster. In the second step, clusters are replaced by their members and the algorithm works as usual. The point of switching between the two phases could simply be defined in terms of the maximum number of instances that can be used in the second phase.

Identifying clusters based on the input could also help narrowing down the number of grammar rules for candidate construction, just as we are already doing when selecting rules before the start of candidate generation. In general, the use of IR techniques for candidate ranking in NLG allows one to draw on the techniques for working with large document collections developed in IR and adapt them to the needs of NLG.

8.4.3 Incorporating other domains

An important issue in extending the coverage of the instance-based ranker is the addition of further domains. As a first step, we propose to deal with new domains separately and in a way similar to our treatment of texts in the management succession domain. As a second step, we propose to merge the derived grammars and instance bases. Apart from the technical aspect of narrowing down the search space by clustering instances and selecting a subset of the available grammar rules (see section 7.6), issues of consistency concerning treebank and semantic annotation would arise.

A combination of domain corpora may involve overlapping annotation schemes. In general, we may be faced with problems concerning the combination of 'ontologies' (Hovy, 1998). However, we do not see tags that occur in more than one scheme as a problem even if they have different intended meanings. The instance-based ranker treats tags just like words which are disambiguated in the context of a bag-of-words. In other words, there does not seem to be a need for insisting on a globally unambiguous semantic annotation scheme.

On the other hand, consistency of the syntactic corpus analysis seems to be required if candidates are to be generated by rules derived from different domain corpora. However, in our approach this could be achieved by using the same statistical parser across domain corpora. The alternative is to keep domains completely separate and to chose the relevant domain before generation starts. Note, however, that this might still require the computation of semantic similarity between input and domain clusters which in turn would take advantage of the fact that tags do not need to be unambiguous.

The identification of domains suitable for instance-based generation might make use of clustering techniques as well. Possibly preceded by named entity recognition, the clustering of large general purpose corpora could at least be an aid to an incremental extension of the instance-based ranker to other domains.

8.4.4 Many-to-many mapping

In chapter 1, we introduced our basic assumption of realization as a one-to-many mapping combined with ranking techniques. A logical extension of this concept is to view generation as a many-to-many mapping. This would yield 'fat pipelines' in which the content determination module does not need to commit itself to a single semantic representation, enabling the system to trade content against fluency whilst preserving the conceptual simplicity of a pipeline. Extending the presented realizer to account for disjunctive input would require an adaption of the check for non-overlapping semantics when chart edges are combined. The idea is to throw all tags mentioned in the input at the rule system, generate bottom-up as before but prevent edges from combining that would result in a semantic representation incompatible with the input. Furthermore, the disjunctive parts of the generation input could have weights attached to them which are taken into account by the coverage score.

8.5 Future scientific work

8.5.1 Comparison to statistical ranking

The goal of this thesis has been to investigate the technical feasibility and properties of instance-based realization (the results are summarized in section 7.8). Based on this investigation, we can only draw limited conclusions about the characteristics and relative advantages of alternative approaches to candidate ranking in NLG. In principle, a large number of machine learning methods can be applied to the task. Among those, the statistical methods used in other ranking approaches to NLG play a prominent rôle (see chapter 2).

What seems to set instance-based learning apart from statistical models is its flexibility concerning the use of the training set. As we have shown, the system can adapt itself to new inputs by choosing 'relevant' subsets of the instance base. Furthermore, the ability of instance-based methods to learn exceptions has been confirmed in our experiments (see section 7.7.3). However, the importance of this property might depend on the regularity of the chosen corpus.

The investigation of the generator output has demonstrated the importance of the issue of input coverage. Instance-based methods naturally represent the length of the training set examples towards which candidate generation is directed. Simple ngram models do not exhibit this property and tend to prefer very short word sequences. To counter this behaviour, Knight and Hatzivassiloglou (1995) employ a function which increases with sentence length. Although this method of correcting candidate length seems rather ad-hoc, statistical frameworks like maximum entropy modelling (Berger et al., 1996) are able to include features that represent sentence length explicitly (proposed in Oberlander and Brew, 2000).

A further notable property of the instance-based generator is its ability to rank candidates based on very few instances. An investigation of alternative ranking approaches should ask questions about the minimally required training material. An ngram language model, for example, will be based on exactly the same ngram counts as the corresponding instance model if both are trained on the same corpus. Although this might allow one to carry out a fairly direct comparison between the ranking methods, issues of input coverage are likely to interfere. On the other hand, our treatment of coverage by means of a linear combination should be applicable to other models too: in a statistical ranker, the overall score of a candidate would be a compromise between the minimal structure preferred by the statistical model and the maximal structure preferred by the coverage score. Furthermore, such research might point to principled combinations of instance-based and statistical techniques. We can ask, for example, whether it is possible to use the similarity score of the instance-based learner as a feature in the maximum entropy framework.

8.5.2 Cognitive models of language production

Exemplar-based models, as well as combinations of exemplar- and rule-based models are popular in Cognitive Science, for example in category learning (Johansen and Palmeri, 2002) and dual-route models of morphology (Pinker, 1991). The hybrid approach to NLG proposed in this work is also an attempt to combine rule-based and example-based methods and might offer a starting point for psycholinguistic research in human language production. For instance, some human production errors might be explained by wrongly mixing or adapting examples. However, we should emphasize that the present work does not make any claims about cognitive models of language production.

8.5.3 Human evaluation in-the-large

In the present work, we judge candidates mainly by criteria that we believe can be applied with reasonable accuracy (like syntactic correctness and faithfulness). A large scale evaluation involving human acceptability judgements should allow one to also draw fine-grained conclusions about the fluency decisions of the generator. In our approach, these are based on the cosine distance metric which has been extensively used in information retrieval. In practice, the cosine tends to rank more fluent candidates higher than less fluent ones. To reliably determine the usefulness of other distance metrics, for example, considerable resources seem to be required.

8.6 Applications of the proposed methods

This thesis assumes a scenario in which the realization system obtains a semantic input from another component that determines the content by drawing on a knowledge base. The assumptions about the input are more relaxed than in most 'classical' NLG systems which expect highly structured representations meeting the specific requirements of the grammar formalism used for sentence realization. In this section, we indicate other possible applications of the techniques developed in this thesis. In the first two applications, we consider tasks in which natural language strings form the input as well as the output.

8.6.1 Machine Translation

Translation Memories can be seen as a starting point for more flexible instance-based machine translation in the same way as fixed template-based generation systems can be regarded as a starting point for instance-based NLG. A similar line of reasoning with respect to Example-based Machine Translation can be found in (Somers, 1999). To perform fully automatic translation, we would envision a hybrid system consisting of rule-based candidate construction and instance-based ranking similar to our approach to realization in NLG. The off-line grammar construction phase needs to derive grammar rules that map source language strings to target language strings. The input to the grammar construction algorithm would need to be an aligned corpus and a syntactic analysis of at least one of the languages involved. In contrast to Example-based Machine Translation, there would be no separate source language analysis, transfer and generation phases. Rather, we would aim at generating translation candidates as directly as a parser generates semantic forms.

The translation process should be regarded as a many-to-many mapping (see section 8.4.4), enabling the preservation of ambiguities. Chart-based algorithms for achieving this are described in (Shemtov, 1998b,a). Translation candidates would be scored by their similarity to selected target language sentences, taking into account the similarity between the source language counterparts of these target language sentences and the source language input to the translation system. The latter would be functionally equivalent to our treatment of coverage. Again, the system needs to reconcile the two goals of fitting the input and maximizing fluency and could thus trade target language fluency against source language faithfulness.

8.6.2 Summarization

In recent years, document summarization has attracted increasing interest in the NLG community. We believe that at least some of the techniques developed in this thesis could be fruitfully applied to the task of summarization. The chosen level of semantic representation in this work is similar to that of many information extraction (IE) applications. Thus, IE system and generator could be combined in a straightforward pipeline. In the end, this is not very different to obtaining the generation input from a database (which might have been populated by IE techniques in the first place). Combinations of information extraction and NLG techniques have been explored in (White et al., 2001), for example. An extension of the approach presented in this thesis to account for disjunctive input (see section 8.4.4) would allow the IE system to keep its options open. We would like to mention two other issues that would need to be addressed in an adaption of our approach to summarization:

- **Integrating IE phase and generation grammar.** The output of the IE phase could consist of constituents labelled with semantic roles, possibly annotated with confidence scores. The generation grammar would need to be able to 'replay' these constituents, and combine or paraphrase them in novel ways. Our experiments[3] indicate that the automatically derived grammar seems to be able to perform sentence compression (Knight and Marcu, 2000).

- **Instance base.** Following the approach developed in this thesis, an instance base for summarization needs to consist of a collection of 'good' summaries. Knowing the source documents that have been summarized allows one to train

[3]See the experiment described in section 7.2.1, for example.

content selection decisions as well. Providing appropriate training material and making good use of it do seem to be difficult tasks. However, these are problems shared with many machine learning approaches to summarization.

8.6.3 Integrating expert systems and case-based reasoning

Our hybrid approach to natural language generation uses techniques that have been successfully employed in other fields of Artificial Intelligence. The grammar is implemented in a production system language which is commonly used for expert system applications. The instance-based ranker makes use of the same principles that underly case-based reasoning systems (Aha, 1998). Thus, the techniques developed in this thesis should allow one to combine the two paradigms beyond natural language generation.

The basic approach in applying our work to case-based reasoning would be that the expert system is proposing some structure – for example a piece of legal or medical reasoning – and the case-based ranker judges the structure. Again, the underlying idea is to overgenerate because the explicitly codified knowledge of the expert system is incomplete.

Golding and Rosenbloom (1996) describe a system that combines rule-based and case-based reasoning. Similar to our approach, the case-based reasoner serves as a critic to the solution proposed by rule applications. However, the system has to make a decision whether to trust the rule-based or the case-based component. Furthermore, the latter uses a case base of negative examples of rule applications. In contrast to our approach, Golding and Rosenbloom (1996) assume that rule application is deterministic.

Chapter 9

Conclusions

In this thesis, we developed a novel approach to natural language surface realization that is based on the idea of an instance-based ranking of output candidates. The approach is hybrid in nature since it combines a rule-based grammar with a corpus-based ranker. We showed that instance-based generation is technically feasible, reasonably efficient and in most cases produces sentences that are fluent, syntactically correct and faithful. Furthermore, experimental evidence suggests that instance-based generation also scales well. In the following, we summarize the main contributions of this work and draw final conclusions.

A considerable part of this work has been devoted to efficiency issues since these are particularly pressing in an overgeneration approach. A naive version of an instance-based generator might just compare each output candidate to the entire instance base. Using an A^*-based search algorithm and interleaving chart generator and instance-based ranker significantly improves efficiency and makes practical use of instance-based generation a plausible perspective: in our implementation, the best candidates are usually found after a few seconds. The interleaved architecture should allow a wide class of grammar formalisms to be used in combination with instance-based ranking without the need to construct special purpose data structures for ranking. Thus, instance-based NLG does not make strong assumptions about the nature of the grammar. Our approach can be seen as a general framework for combining a component in the rationalist research tradition (characterized by the use of rules) with an (empirical) instance-based learner.

The evaluation of a basic instance-based generator on test data revealed that the system tended to generate incomplete candidates that exactly replicate some instance of the instance base. Having identified this problem of completeness, we introduced a linear combination of the instance-edge similarity score and a coverage score that allows the system to trade the goal of fluency against the goal of producing complete output. This trade-off can be observed empirically. Moreover, we claim that the trade-off between fluency and completeness is general and applies to other overgeneration approaches as well. The possibility of trade-offs between competing goals offers promising prospects for future research on text generation and for applications such as machine translation and summarization. (A more detailed summary of the evaluation experiments can be found in section 7.8.)

We proposed a new approach to the development of generation grammars that exploits a corpus of domain texts in two ways: to populate the instance base and to automatically construct the grammar rules. The only manual effort that is required is the provision of a semantic markup scheme (and possibly the definition of additional faithfulness constraints on candidates). Furthermore, the domain corpus does not need to be very extensive for our approach to work. The output of the generation system shows that such a grammar, in combination with an instance-based ranker, produces mostly fluent, syntactic and faithful candidates, depending on the emphasis that is placed on coverage. This is achieved without manually writing a single grammar rule.

However, as a word of caution, it should be emphasized that the development of a consistent annotation scheme can be challenging, and in practice there are limits to the number of distinctions that can be made. What these distinctions are should depend on the application and the capabilities of the NLP component that provides the input to the realizer. On the one hand, it may be tempting to develop ever more fine-grained semantic annotation schemes to guide the realizer. On the other hand, it should be kept in mind that the presented approach uses a corpus precisely for the reason that fine-grained explicit choice has its limitations.

In conclusion, the main contributions of this work are:

- an architecture that interleaves grammar-based overgeneration with instance-based ranking (chapter 1),

- an efficient A^*-based search algorithm for instance-based ranking (chapter 3),
- a novel method for automatically constructing a generation grammar from a syntactic treebank which has been extended by semantic annotations (chapters 4,5),
- a bottom-up chart algorithm for candidate generation that uses a Rete network, which makes it particularly suitable for generation with large numbers of grammar rules (chapter 6),
- several improvements of an initial version of the instance-based generator: using a linear combination of the fluency score with a coverage score, generating into several ranked lists and additional filtering of candidates by means of (hard) faithfulness constraints (chapter 7).

The instance-based sentence realizer performs a number of tasks that have become increasingly separated in explicit-choice models of NLG. In addition to sentence realization, our approach performs lexical choice and some aspects of aggregation. Moreover, sentence realization proper here involves more work than in many rule-based realizers because its input is semantic rather than syntactic in nature. The processing model that seems to emerge from this is that of a candidate generator of reduced complexity that maps its input as directly as possible to concrete word forms. The resulting candidates are then evaluated by a ranker. We indicated above that this model could be extended to text generation, machine translation and other tasks, possibly within a single module or alternatively by a 'fat pipeline' of simple modules. In effect, such a system would encompass the entire 'how-to-say-it' phase of natural language generation since this is what fluency is about.

Appendix A

The encoding of grammar rules as productions

The implementation language used for the chart generator is JESS (Java Expert System Shell, Friedman-Hill, 2000). We use structured (or 'unordered') facts of attribute-value pairs to define chart edges. The following representations are produced fully automatically after the context-free backbone of the generation grammar has been constructed (see chapter 5). Given the grammar rules (productions) the JESS compiler produces a Rete network (see chapter 6) which is part of the runtime generation system.

A.0.3.0.1 Input rules Figure A.1 shows an input rule encoded as a production as used in the actual implementation and the corresponding representation in a more abstract format. Input rules generally need to match simple facts that represent input tags and their index. These so-called 'ordered facts' are unstructured and contain a list of atoms and variables. In figure A.1, the input fact to be matched is `(input ?sdx COMP_DESCR ?cx $?fillerPhon)`. The head `input` is followed by a variable for the semantic index of the input tag, followed by the tag proper, the tag index and the slot filler, i.e. the surface words associated with the input tags given as input to the generator.

The semantic index is unique for each input tag and allows one to determine which input tags a chart edge consumes. It can be interpreted as a reference to an object in the real world. In contrast, the tag represents the semantics of this object and is not

$$\begin{bmatrix} \text{SEM} & \{\boxed{2}\text{COMP\_DESCR}(\boxed{1})\} \\ \text{CAT} & \text{PP-COMP\_DESCR} \\ \text{TERMS} & <\text{of this } \boxed{2}> \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SEM} & \text{COMP\_DESCR}(\boxed{1}) \end{bmatrix}$$

```
(defrule input-rule-23
  (input ?sdx COMP_DESCR ?cx $?fillerPhon)
  (input-rule-23)
 =>
  (add-to-agenda
       (PP-COMP_DESCR
           (idx (bind ?idx (npt)))
           (coidx ?cx)
           (syn PP)
           (consumes (create$ ?sdx))
           (terms  of this [ COMP_DESCR $?fillerPhon ] ))
           (instances (new-instance-table))
           (deriv  [ PP i23-0014 ?idx  of this COMP_DESCR ] )
           (fired-by input-rule-23))))
```

Figure A.1: Input rule and its encoding as production

necessarily unique.

The antecedent of input rules also contains a condition that matches a simple unique fact for the name of the rule `input-rule-23`. In this way, input rules can be blocked and unblocked dynamically. Only those rules are able to fire whose name has been asserted into the KB.

If all conditions are met, an unordered fact that has the syntactic-semantic category as its head is added to the agenda. A unique fact-id is created by the function `npt` as the value of the `idx` slot. Slot `coidx` takes the tag index given in the input. The syntactic category `syn` is represented separately to allow identification of (candidate) sentences by the ranker. The `consumes` slot takes the unique semantic index of the input tag and creates a 'multi-field' for multi-sets of such indices. The `terms` slot contains the marked-up surface string of the edge. The content of this edge is used by the ranker. The `terms` slot contains the filler words of the input explicitly (stored in the variable `$?fillerPhon`). Strictly speaking, these are not needed during generation and could be recovered from the semantic index after generation has finished. The `instances` slot contains a reference to all those instances that should be considered for ranking. These are the 'relevant instances' of the ranker (see 3.2.1.2). The `fired-by` slot contains the rule name

The `deriv` slot represents the derivation for the edge. It details the merged and reduced treebank structure that is the basis for this grammar rule, i.e. each constructed grammar rule forms a local tree. In the above example, the syntactic category of the mother node is a `PP` and the daughter nodes are the sequence `'of this COMP_DESCR'` (the slot filler is not represented in the value of `deriv`). The `deriv` slot also specifies the fact-id (`?idx`) and the name of the input rule and the treebank sentence by which it is motivated (`i23-0014`).

```
New fact added to agenda:
     (PP-COMP_DESCR
        (idx 15)
        (coidx <External-Address:java.util.HashMap>)
        (syn PP)
        (consumes 6)
        (terms  of this [ COMP_DESCR industrial conglomerate ])
        (instances <External-Address:java.util.Hashtable>)
        (deriv [ PP i23-0014 15 of this COMP_DESCR ])
        (fired-by input-rule-23))

Facts matched on conditions:
 (input 6 COMP_DESCR <External-Address:java.util.HashMap> industrial conglomerate)
 (input-rule-23)
```

Figure A.2: Fact generated by input rule

An example of a fact produced by the production in figure A.1 is given in figure A.2. It shows a fully instantiated `terms` value, `'of this [ COMP_DESCR industrial conglomerate ]'`. The content of the `coidx` and `instances` slots contains references to objects outside the realm of the production system language.

A.0.3.0.2 Phrasal rules In contrast to input rules, phrasal rules match structured facts representing chart edges in their conditions. Figure A.3 shows a production for a phrasal rule that matches facts with heads `NP-POST_DESCR_ADJ` and `PP-POST_NODET`. It also requires a simple ('ordered') fact in order to be unblocked ('`phrasal-rule-83`').

$$\begin{bmatrix} \text{SEM} & \boxed{1} \uplus \boxed{2} \\ \text{CAT} & \text{VP-POST\_DESCR\_ADJ} \\ \text{TERMS} & \langle \text{was named to } \boxed{3}\,\boxed{4} \rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SEM} & \boxed{1} \\ \text{CAT} & \text{NP-POST\_DESCR\_ADJ} \\ \text{TERMS} & \boxed{3} \end{bmatrix} \begin{bmatrix} \text{SEM} & \boxed{2} \\ \text{CAT} & \text{PP-POST\_NODET} \\ \text{TERMS} & \boxed{4} \end{bmatrix}$$

```
(defrule phrasal-rule-83
  (NP-POST_DESCR_ADJ (idx ?i0) (coidx ?cx0) (syn ?s0) (consumes $?c0)
                     (terms ?t0) (instances ?i_table0) (deriv $?d0))
  (PP-POST_NODET (idx ?i1) (coidx ?cx1)
                 (syn ?s1) (consumes $?c1&:(set (create$  $?c0 $?c1)))
                 (terms ?t1) (instances ?i_table1) (deriv $?d1))
  (phrasal-rule-83)
 =>
  (if (bind ?cxc (combine-index-sets  ?cx0 ?cx1 ))
      then (add-to-agenda
             (VP-POST_DESCR_ADJ
                 (idx (bind ?idx (npt)))
                 (coidx ?cxc)
                 (syn VP)
                 (consumes (create$ $?c0 $?c1))
                 (terms  was named to ?t0 ?t1)
                 (instances (combine-instance-tables ?i_table0 ?i_table1))
                 (deriv  [ VP p83-0366 ?idx was named to $?d0 $?d1 ] )
                 (fired-by phrasal-rule-83)))))
```

Figure A.3: Phrasal rule and its encoding as production

Matches for facts in conditions of phrasal rules are largely confined to matching fact heads. Slot values are mainly only picked up by variables and passed on to the rule consequent. However, there is an exception. The condition matching the second edge in figure A.3 performs a test to prevent a combination of edges that express overlapping semantics, based on the unique semantic indices in the `consumes` slot. In order to perform these tests as early as possible, they are called for each non-first condition in principle. Since the example has two daughters, there is only one such test to be performed. This is indeed the most frequent case since most phrasal rules are binary.

On the consequent side of phrasal rules, a further check related to the treatment of tag indices is performed (see section 5.5). If this test is also successful, a new fact is created and then added to the agenda. Its `consumes` slot contains the union of the values of the daughter's `consumes` slots. This implements the concept that the semantics of a phrase is the union of the semantic information of its daughters. Furthermore, a new value of the `instances` slot is computed on the basis of the corresponding values of the daughter edges: the instance table of the newly created edge is the intersection of the relevant instance tables of the daughter edges, combined with the lowest expectation for the instances (this is a cheap initial approximation of the true expectations – see step 1 of the ranking algorithm, section 3.2.1.2). Finally, the `deriv` slot embeds the derivation information of the daughter edges, building up structure in this way.

Appendix B

A complete list of annotation tags

This part of the appendix shows the final list of semantic tags, including their frequencies. The tags are grouped by their name. This list reflects the changes to the annotation scheme described in section 7.4.3.2. Therefore, some tag counts differ from the ones given in chapter 4. The overall number of tags in the final list below is 78.

freq.	**tag name**
14	BOARD_INCR
7	COMP
6	COMP*
19	COMP_DESCR
102	COMP_DESCR*
1	COMP_DESCR_RC
1	COMP_DESCR_RC*
1	COMP_DESRC*
3	COMP_LOC
1	COMP_MOTHER
1	COMP_MOTHER_DESCR
1	COMP_MOTHER_NATIONALITY
5	COMP_NATIONALITY
2	COMP_SUBSIDIARY
20	COMP_SUBSIDIARY*
2	COMP_SUBSIDIARY_ADDINFO_JJ
8	COMP_SUBSIDIARY_DESCR_DEF
1	COMP_SUBSIDIARY_DESCR_DEF*
5	COMP_SUBSIDIARY_DESCR_INDEF
6	COMP_SUBSIDIARY_DESCR_NODET
3	COMP_SUBSIDIARY_DESCR_NODET*

freq.	tag name
1	COMP_SUBSIDIARY_DESCR_PLURAL
1	COMP_SUBSIDIARY_DESCR_PLURAL*
3	COMP_SUBSIDIARY_LOC
3	COMP_SUBSIDIARY_SUBSIDIARY*
1	COMP_SUBSIDIARY_SUBSIDIARY_ADDINFO_JJ
1	COMP_SUBSIDIARY_SUBSIDIARY_DESCR_DEF
1	COMP_SUBSIDIARY_SUBSIDIARY_DESCR_DET
1	COMP_SUBSIDIARY_SUBSIDIARY_DESCR_NODET
1	INDATE_ENDURING
11	INDATE_FUTURE
1	INDATE_FUTURE_ENDING
56	INPERSON_AGE
1	INPERSON_DESCR_ADJP
3	INPERSON_DESCR_JJ
2	INPERSON_DESCR_NP_INDEF
139	INPERSON_FULLNAME
33	INPERSON_OTHERCOMP
2	INPERSON_OTHERCOMP_DESCR
6	INPERSON_OTHERCOMP_LOC
2	INPERSON_OTHERCOMP_SUBSIDIARY
2	INPERSON_OTHERCOMP_SUBSIDIARY_DESCR_DEF
1	INPERSON_OTHERPOST_CONSULTANT_INDEF
3	INPERSON_OTHERPOST_DEF
9	INPERSON_OTHERPOST_INDEF
1	INPERSON_OTHERPOST_LOC
59	INPERSON_OTHERPOST_NODET
1	INPERSON_OTHERPOST_PARTNER_INDEF
10	INPERSON_PREVIOUSCOMP
1	INPERSON_PREVIOUSCOMP_SUBSIDIARY
1	INPERSON_PREVIOUSPOST_DATESPAN_PAST
25	INPERSON_PREVIOUSPOST_NODET
4	OUTDATE_FUTURE
1	OUTDATE_FUTURE_PP
3	OUTDATE_PAST
10	OUTPERSON_AGE
23	OUTPERSON_FULLNAME
2	OUTPERSON_INDATE_PAST
1	OUTPERSON_NEWPOST_INDEF
2	OUTPERSON_NEWPOST_NODET
1	OUTPERSON_OTHERPOST_DEF
1	OUTPERSON_POSTCONT_INDEF
1	OUTPERSON_POSTCONT_NODET
2	OUTPERSON_PREVIOUSPOST_INDEF
4	OUTPERSON_PRP_VP_FUTURE
9	OUTPERSON_PRP_VP_PAST
2	OUTPERSON_PRP_VP_PRESENT
1	OUTPERSON_PRP_VP_SADV_PAST

freq.	tag name
10	POST_BOARD
25	POST_DESCR_ADJ
5	POST_DESCR_VACANCY
36	POST_INDEF
1	POST_INF
4	POST_LOC
142	POST_NODET
7	POST_PLURAL
2	POST_RESPONSIBILITY
6	?

Bibliography

Aha, D. W. (1998). The Omnipresence of Case-Based Reasoning in Science and Application. *Knowledge-Based Systems*, 11(5-6):261–273.

Aha, D. W., Kibler, D., and Albert, M. (1991). Instance-based Learning Agorithms. *Machine Learning*, 7:37–66.

Appelt, D. (1985). *Planning English Sentences*. Cambridge University Press, Cambridge, Massachusetts.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison Wesley, Reading, Massachusetts.

Bangalore, S. and Joshi, A. K. (1999). Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265.

Bangalore, S. and Rambow, O. (2000a). Corpus-based Lexical Choice in Natural Language Generation. In *Proceedings of ACL-2000*, Hongkong.

Bangalore, S. and Rambow, O. (2000b). Exploiting a Probabilistic Hierarchical Model for Generation. In *Proceedings of COLING-00*, pages 42–48.

Bangalore, S., Rambow, O., and Whittaker, S. (2000). Evaluation Metrics for Generation. In *Proceedings of the 1st International Conference on Natural Language Generation (INLG 2000)*, Mitzpe Ramon, Israel.

Berger, A. L., Brown, P. F., Pietra, S. A. D., Pietra, V. J. D., and etc (1994). The Candide System for Machine Translation. In *Proceedings of the 1994 ARPA Workshop on Human Language Technology*.

Berger, A. L., Pietra, S. A. D., and Pietra, V. J. D. (1996). A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1).

Bloomfield, L. (1933). *Language*. British edition: Allan & Unwin Ltd, London, 1935.

Bouaud, J. (1993). Tree: a heuristic driven Join Strategy of a Rete-like Matcher. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 496–502.

Brew, C. (1992). Letting the Cat out of the Bag: Generation for Shake and Bake MT. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, pages 610–616, Nantes, France.

Brown, R. D. (1996). Example-based Machine Translation in the Pangloss System. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*.

Brown, R. D. (1999). Adding Linguistic Knowledge to a Lexical Example-based Translation System. In *Proceedings of the Seventh International Conference on Theoretical and Methodological Issues in Machine Translation (TMI)*, pages 22–32.

Burke, R., Hammond, K., Kulyukin, V., Lytinen, S., Tomuro, N., and Schoenberg, S. (1997). Natural Language Processing in the FAQ Finder System: Results and Prospects. In *Papers from the 1997 AAAI Spring Symposium on Natural Language Processing for the World Wide Web.*, Stanford University, California.

Cahill, L., Doran, C., Evans, R., Kibble, R., Mellish, C., Paiva, D., Reape, M., Scott, D., and Tipper, N. (2000). Enabling Resource Sharing in Generation: an Abstract Reference Architecture. In *Proceedings of LREC*, Athens.

Cahill, L., Doran, C., Evans, R., Mellish, C., Paiva, D., Reape, M., Scott, D., and Tipper, N. (1999). In Search of a Reference Architecture for NLG Systems. In *Proceedings of the European Workshop on Natural Language Generation (EWNLG-99)*, Toulouse, France.

Calder, J., Reape, M., and Zeevat, H. (1989). An algorithm for generation in Unification Categorial Grammar. In *Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics*, pages 233–240.

Carroll, J., Copestake, A., Flickinger, D., and Poznanski, V. (1999). An efficient Chart Generator for (semi-)lexicalist Grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation*, pages 86–95, Toulouse, France.

Charniak, E. (1996). Tree-bank Grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1031–1036. MIT Press.

Charniak, E. (2000). A Maximum-Entropy-Inspired Parser. In *Proceedings of the North American Meeting of the Association for Computational Linguistics (NAACL-00)*.

Chelba, C. and Mahajan, M. (2002). Information Extraction using the Structured Language Model. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP-01)*.

Cheng, H., Poesio, M., Henschel, R., and Mellish, C. (2001). Corpus-based NP Modifier Generation. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-01)*, pages 9–16, Carnegie Mellon University, Pittsburgh, PA, USA.

Chu-Carroll, J. and Carpenter, B. (1999). Vector-based Natural Language Call Routing. *Computational Linguistics*, 25(3).

Cohen, W. W. (1995). Fast Effective Rule Induction. In *Proceedings of the 12th International Conference on Machine Learning (ML-95)*, pages 115–123, Lake Tahoe, CA. Morgan Kaufmann.

Collins, M. (1999). *Head driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, Philadelphia.

Collins, M. and Miller, S. (1998). Semantic Tagging using a Probabilistic Context Free Grammar. In *Sixth Workshop on Very Large Corpora*. ACL.

Copestake, A., Flickinger, D., Pollard, C. J., and Sag, I. A. (1999). Minimal Recursion Semantics: An Introduction. ms. CSLI, Stanford University.

Cover, T. M. and Hart, P. E. (1967). Nearest Neighbor Pattern Classification. *IEEE Trans. on Information Theory*, 13(1):21–27.

Crysmann, B., Frank, A., Kiefer, B., Krieger, H.-U., Müller, S., Neumann, G., Piskorski, J., Schäfer, U., Siegel, M., Uszkoreit, H., and Xu, F. (2002). An Integrated Architecture for Shallow and Deep Processing. In *ACL-2002*.

Daelemans, W. (1999). Memory-based Language Processing. Introduction to the Special Issue. *Journal of Experimental and Theoretical AI*, 11(3):287–467.

Daelemans, W., Bosch, A. V. D., and Zavrel, J. (1999a). Forgetting Exceptions is Harmful in Language Learning. *Machine Learning*, 34:11–43.

Daelemans, W., Buchholz, S., and Veenstra, J. (1999b). Memory-Based Shallow Parsing. In *Proceedings of the EACL'99 workshop on Computational Natural Language Learning (CoNLL-99)*, Bergen, Norway.

Danlos, L. and Namer, F. (1988). Morphology and Cross Dependencies in the Synthesis of Personal Pronouns in Romance Languages. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING-88)*, volume 1, pages 139–141.

Daume III, H., Knight, K., Langkilde-Geary, I., Marcu, D., and Yamada, K. (2002). The Importance of Lexicalized Syntax Models for Natural Language Generation Tasks. In *Proceedings of INLG 02 (International Natural Language Generation Conference)*, New York.

Deerwester, S., Dumais, S. T., K.Landauer, T., Furnas, G. W., and Harshman, R. A. (1990). Indexing by Latent Semantic Analysis. *Journal of the Society for Information Science*, 41(6):391–407.

Dewan, H. M. (1994). *Runtime Reorganization of Parallel and Distributed Expert Database Systems*. PhD thesis, Columbia University, Department of Computer Science.

Doorenbos, R. B. (1993). Matching 100,000 learned rules. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*.

Doorenbos, R. B. (1994). Combining Left and Right Unlinking for Matching a Large Number of Learned Rules. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*.

Duboue, P. and McKeown, K. (2001). Empirically Estimating Order Constraints for Content Planning in Generation. In *Proceedings of ACL-EACL'01*.

Earley, J. (1970). An Efficient Context-Free Parsing-Algorithm. In *Communications of the ACM*, volume 13-2, pages 94–102 [Reprinted in Grosz *et al.* (1986, S.25–33)].

Elhadad, M. (1991). FUF User Manual - Version 5.0. Technical Report CUCS-038-91, Columbia University, Dept. of Computer Science.

Elhadad, M. (1993). *Using Argumentation to Control Lexical Choice: a Functional Unification-based approach*. PhD thesis, Department of Computer Science, Columbia University, New York.

Erbach, G. (1997). *Bottom-Up Earley Deduction for Preference-Driven Natural Language Processing*. PhD thesis, Universität des Saarlandes, Saarbrücken.

Evans, R. and Weir, D. (1997). Automaton-based Parsing for Lexicalized Grammars. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, Cambridge, Mass.

Evans, R. and Weir, D. (1998). A Structure-sharing Parser for Lexicalized Grammars. In *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics*, pages 372–378.

Forgy, C. L. (1981). OPS5 User's Manual. Technical Report CMU-CS-81-135, Computer Science Department, Carnegie Mellow University.

Forgy, C. L. (1982). Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem. *Artificial Intelligence*, pages 17–37.

Freeman, W. T., Tenenbaum, J. B., and Pasztor, E. (1999). An Example-Based Approach to Style Translation for Line Drawings. Technical Report MERL-TR-99-11, MERL (Mitsubishi Electric Research Laboratory).

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. (1998). An Efficient Boosting Algorithm for Combining Preferences. In *Machine Learning: Proceedings of the Fifteenth International Conference*.

Friedman-Hill, E. (2000). *JESS - the Java Expert System Shell, Version 6.x.* Sandia National Laboratories, Software available at http://herzberg.ca.sandia.gov/jess/.

Gazdar, G. and Mellish, C. (1989). *Natural Language Processing in PROLOG. An Introduction to Computational Linguistics*. Addison-Wesley, Wokingham.

Giarratano, J. and Riley, G. (1993). *Expert Systems: Principles and Practice*. PWS Publishing, Boston, 2nd edition.

Golding, A. R. and Rosenbloom, P. S. (1996). Improving Accuracy by Combining Rule-Based and Case-Based Reasoning. *Artificial Intelligence*, 87(1-2):215–254.

Hockenmaier, J. and Steedman, M. (2002). Acquiring Compact Lexicalized Grammars from a Cleaner Treebank. In *Proceedings of Third International Conference on Language Resources and and Evaluation (LREC)*, Las Palmas.

Hovy, E. H. (1998). Combining and Standardizing Large-Scale, Practical Ontologies for Machine Translation and Other Uses. In *Proceedings of the 1st International Conference on Language Resources and Evaluation (LREC)*, Granada, Spain.

Huang, X., Acero, A., and Hon, H.-W. (2001). *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall.

INLG (2002). *International Natural Language Generation Conference (INLG-02)*, New York.

Ishida, T. (1988). Optimizing Rules in Production System Programs. In *AAAI-88*, pages 699–704.

Jackson, P. (1990). *Introduction to Expert Systems*. Addison-Wesley, Reading, Massachusetts, 2nd edition.

Johansen, M. and Palmeri, T. (2002). Are there Representational Shifts during Category Learning? *Cognitive Psychology*. In Press.

Johnson, M. and Roark, B. (2000). Compact Non-Left-Recursive Grammars using the Selective Left-Corner Transform and Factoring. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 355–361.

Jordan, P. and Walker, M. (2000). Learning Attribute Selections for Non-Pronominal Expressions. In *Proceedings of ACL-2000*, Hong Kong.

Joshi, A. K. (1987). The Relevance of Tree Adjoining Grammar to Generation. In Kempen, G., editor, *Natural Language Generation*, pages 233–252. Martinus Nijhoff Press, Dordrecht, The Netherlands.

Jurafsky, D. and Martin, J. (2000). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall.

Kay, M. (1985). Unification in Grammar. In Dahl, V. and Saint-Dizier, P., editors, *Natural Language Understanding and Logic Programming*, Amsterdam. North-Holland.

Kay, M. (1996). Chart Generation. In *Proceedings of ACL-96*, pages 200–204.

KI-99 (1999). *Proceedings of the KI-99 Workshop on May I Speak Freely: Between Templates and Free Choice in Natural Language Generation.*

Knight, K. and Hatzivassiloglou, V. (1995). Two-level, Many-paths Generation. In *Proceedings of the 33th Annual Meeting of the Association for Computational Linguistics (ACL-95)*, Boston, MA.

Knight, K. and Luk, S. (1994). Building a Large-Scale Knowledge Base for Machine Translation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Knight, K. and Marcu, D. (2000). Statistics-Based Summarization — Step One: Sentence Compression. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Krotov, A., Hepple, M., Gaizauskas, R., and Wilks, Y. (2000). Evaluating two Methods for Treebank Grammar Compaction. *Journal of Natural Language Engineering*, 5(4). (Also in Proceedings of the COLING-ACL'98 Joint Conference.).

Kukich, K. (1983). *Knowledge-Based Report Generation: A Knowledge Engineering Approach to Natural Language Report Generation*. PhD thesis, Department of Information Science, University of Pittsburgh.

Kukich, K. (1988). Fluency in Natural Language Reports. In *Natural Language Generation Systems*, pages 280–311. Springer, New York.

Laird, J. E., Newell, A., and Rosenbloom, P. (1987). SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, 33:1–64.

Langkilde, I. (2000). Forest-based Statistical Sentence Generation. In *Proceedings of the North American Meeting of the Association of Computational Linguistics*, pages 170–177.

Langkilde, I. (2002). An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of INLG-02 (International Natural Language Generation Conference)*, New York.

Langkilde, I. and Knight, K. (1998a). Generation that Exploits Corpus-based Statistical Knowledge. In *Proceedings of COLING/ACL-98*, pages 704–710, Montreal, Canada.

Langkilde, I. and Knight, K. (1998b). The Practical Value of N-grams in Generation. In *Proceedings of the 9th International Workshop on Natural Language Generation*, Niagra-on-the-Lake, Ontario.

Lavoie, B. and Rambow, O. (1997). A Fast and Portable Realizer for Text Generation Systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, Washington, DC.

Lee, H. S. and Schor, M. I. (1992). Match Algorithms for Generalized Rete Networks. *Artificial Intelligence*, 54(3):249–274.

Lewis, D. D. and Jones, K. S. (1996). Natural language processing for information retrieval. *Communications of the ACM*, 39(1):92–101.

Malouf, R. (2000). The Order of Prenominal Adjectives in Natural Language Generation. In *Proceedings of ACL-2000*, Hongkong.

Mann, W. C. and Matthiessen, C. M. I. M. (1985). Nigel: A Systemic Grammar for Text Generation. In Benson, R. and Greaves, J., editors, *Systemic Perspectives on Discourse: Selected Papers Papers from the 9th International Systemics Workshop*, London. Ablex. Also available as USC/ISI Research Report RR-83-105.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

Marcus, M. P., Kim, G., Marcinkiewicz, M., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schlasberger, B. (1994). The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the 1994 Human Language Technology Workshop*, pages 110–115.

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

McDonald, D. D. (1993). Issues in the Choice of a Source for Natural Language Generation. *Computational Linguistics*, 19:191–197.

McDonald, D. D. and Meteer, M. W. (1988). From Water to Wine: Generating Natural Language Text from Today's Applications Programs. In *Proc. of the Second Conference on Applied Natural Language Processing*, pages 41–48, Austin, TX.

McKeown, K. R. (1985). *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Studies in Natural Language Processing. Cambridge University Press.

Mel'cuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press, New York.

Mellish, C., Evans, R., Cahill, L., Doran, C., Paiva, D., Reape, M., Scott, D., and Tipper, N. (2000). A Representation for Complex and Evolving Data Dependencies in Generation. In *Proceedings of ANLP-00*, Seattle.

Mellish, C., Knott, A., Oberlander, J., and O'Donnell, M. (1998). Experiments using Stochastic Search for Text Planning. In *Proceedings of the 9th International Generation Workshop*, pages 98–107, Niagara, Canada.

Meteer, M. W. (1990). *The Generation Gap – The Problem of Expressibility in Text-Planning*. PhD thesis, University of Massachusetts, Amherst, MA, USA.

Miller, S., Fox, H., Ramshaw, L., and Weischedel, R. (2000). A Novel Use of Statistical Parsing to Extract Information from Text. In *Proceedings of the North American Meeting of the Association for Computational Linguistics (NAACL-00)*, pages 226–233.

Minnen, G., Bond, F., and Copestake, A. (2000). Memory-Based Learning for Article Generation. In Cardie, C., Daelemans, W., Nedellec, C., and Tjong Kim Sang, E., editors, *Proceedings of CoNLL-2000 and LLL-2000*, pages 43–48. Lisbon, Portugal.

Miranker, D. P. (1990). *TREAT: a new and efficient match algorithm for AI production systems*. Morgan Kaufmann, San Mateo, California.

Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

MUC-6 (1995). *Proceedings of the Sixth Message Understanding Conference*. Morgan Kaufmann.

Murata, M., Ma, Q., Uchimoto, K., and Isahara, H. (1999). An Example-Based Approach to Japanese-to-English Translation of Tense, Aspect, and Modality. In *Proceedings of the 8th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-99)*.

Nayak, P., Gupta, A., and Rosenbloom, P. (1988). Comparison of the Rete and Treat Production Matchers for SOAR (A Summary). In *Proceedings of the 1988 Conference of the American Association for Artificial Intelligence*.

Neumann, G. (1994). *A Uniform Computational Model for Natural Language Parsing and Generation*. PhD thesis, Universität des Saarlandes, Saarbrücken.

Neumann, G. (1998). Interleaving Natural Language Parsing and Generation through Uniform Processing. *Artifical Intelligence*, 99:121–163.

Nicolov, N., Mellish, C., and Richie, G. (1996). Approximate Generation from Non-Hierarchical Representations. In *Proceedings of the 8th International Workshop on Natural Language Generation*, Herstmonceux Castle, UK.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.

Norvig, S. R. P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, New Jersey, 1st edition.

Nunberg, G. (1990). *The linguistics of punctuation*. CSLI Lecture Notes 18, Stanford, CA.

Oberlander, J. and Brew, C. (2000). Stochastic Text Generation. In *Philosophical Transactions of the Royal Society of London, Series A*, volume 358, pages 1373–1385.

Oberlander, J., O'Donnell, M., Knott, A., and Mellish, C. (1998). Conversation in the museum: experiments in dynamic hypermedia with the intelligent labelling explorer. *New Review of Hypermedia and Multimedia*, 4:11–32.

O'Donnell, M. (1996). Input Specification in the WAG Sentence Generation System. In *Proceedings of the 8th International Workshop on Natural Language Generation*, Herstmonceux Castle, UK.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2001). Bleu: a Method for Automatic Evaluation of Machine Translation. Technical Report RC 22176, IBM Research Division.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, Massachusetts.

Pereira, F. C. and Warren, D. (1983). Parsing as Deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 137–144.

Pinker, S. (1991). Rules of Language. *Science*, 253(5019):530–535.

Popowich, F. (1996). A Chart Generator for Shake and Bake Machine Translation. In *Proceedings of AI'96 - 11th Canadian Conference on Artificial Intelligence*, Toronto.

Rambow, O., Rogati, M., and Walker, M. A. (2001). Evaluating a Trainable Sentence Planner for a Spoken Dialogue System. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-2001)*, pages 434–441.

Ratnaparkhi, A. (2000). Trainable Methods for Surface Natural Language Generation. In *Proceedings of the North American Meeting of the Association for Computational Linguistics (NAACL-00)*, pages 194–201.

Ratnaparkhi, A. (2001). Modeling Informational Novelty in a Conversational System with a Hybrid Statistical and Grammar-Based Approach to Surface Natural Language Generation. In *Proceedings of the NAACL 2001 Workshop on Adaptation in Dialogue Systems*, Carnegie Mellon University, Pittsburgh, PA, USA.

Reiter, E. (1994). Has a Consensus NL Generation Architecture Appeared, and is it Psycholinguistically Plausible? In *Proceedings of the Seventh International Workshop on Natural Language Generation (INLG-94)*, pages 163–170, Kennebunkport, Maine, USA.

Reiter, E. and Dale, R. (2000). *Building Applied Natural Language Generation Systems*. Cambridge University Press, Cambridge, UK.

Reiter, E. and Mellish, C. (1992). Using Classification to Generate Text. In *Proceedings of the 9th COLING*.

Resnik, P. (1992). Probabilistic Tree-Adjoining Grammar as a Framework for Statistical Natural Language Processing. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, pages 418–424, Nantes, France.

Rich, E. and Knight, K. (1991). *Artificial Intelligence*. McGraw Hill, 2nd edition.

Riley, G. (1999). CLIPS: A Tool for Building Expert Systems. http://www.ghg.net/clips/CLIPS.html.

Salton, G. (1988). Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24:513–523.

Salton, G. (1989). *Text Processing: the Transformation and Retrieval of Information by Computer*. Addison-Wesley, Reading, Massachusetts.

Salton, G. and McGill, M. (1983). The SMART and SIRE Experimental Retrieval Systems. In Jones, K. S. and Willett, P., editors, *Readings in Information Retrieval*, pages 118–155. McGraw-Hill, New York.

Santorini, B. (1990). Part-of-speech Tagging Guidelines for the Penn Treebank Project. Technical Report MS-CIS-90-47, Department of Computer and Information Science, University ofPennsylvania.

Sato, S. (1995). MBT2: A Method for Combining Fragments of Examples in Example-Based Translation. *Artificial Intelligence*, 75:31–49.

Sato, S. and Nagao, M. (1990). Towards Memory-based Translation. In *Proceedings of COLING-90*.

Schabes, Y. (1992). Stochastic Lexicalized Tree-Adjoining Grammars. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, pages 426–432, Nantes, France.

Shaw, J. and Hatzivassiloglou, V. (1999). Ordering among Premodifiers. In *Proceedings of ACL-99*, pages 135–143, University of Maryland, USA.

Shemtov, H. (1998a). *Ambiguity Management in Natural Language Generation*. PhD thesis, Department of Linguistics, Stanford University.

Shemtov, H. (1998b). A Method for Preserving Ambiguities in Chart Generation. In *Proceedings of the 1st Workshop on Tabulation in Parsing and Deduction (TAPD'98)*, Paris, France.

Shieber, S. M. (1988). A Uniform Architecture for Parsing and Generation. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 614–619, Budapest, Hungary.

Shieber, S. M., Schabes, Y., and Pereira, F. C. (1995). Principles and Implementation of Deductive Parsing. *Journal of Logic Programming*, 24(1-2):3–36.

Shieber, S. M., van Noord, G., Pereira, F. C., and Moore., R. (1990). Semantic-head-driven Generation. *Computational Linguistics*, 16(1):30–42.

Somers, H. (1999). Review article: Example-based Machine Translation. *Machine Translation*, 14:113–158.

Stone, M. and Doran, C. (1996). Paying Heed to Collocations. In *Proceedings of the Eighth International Workshop on Natural Language Generation*, pages 91–100.

The XTAG Research Group (1998). A lexicalized tree adjoining grammar for english. Technical Report IRCS-98-18, Institute for Research in Cognitive Science (IRCS), University of Pennsylvania, Philadelphia.

Thompson, H. (1990). Best-first Enumeration of Paths through a Lattice – an Active Chart Parsing Solution. *Computer Speech and Language*, 4:263–274.

Trujillo, A. (1997). Determining Internal and External Indices for Chart Generation. In *7th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-97)*.

Uchimoto, K., Sekine, S., and Isahara, H. (2002). Text Generation from Keywords. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING)*, Taipei, Taiwan.

Varges, S. (2002). Fluency and Completeness in Instance-based Natural Language Generation. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING)*, pages 1058–1064, Taipei, Taiwan.

Varges, S. and Mellish, C. (2001). Instance-based Natural Language Generation. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-01)*, pages 1–8, Carnegie Mellon University, Pittsburgh, PA, USA.

Walker, M. A., Rambow, O., and Rogati, M. (2001). SPoT: A Trainable Sentence Planner. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-01)*, pages 17–24, Carnegie Mellon University, Pittsburgh, PA, USA.

Ward, N. (1994). *A Connectionist Language Generator*. Ablex, Norwood, N.J.

White, M., Korelsky, T., Cardie, C., Ng, V., Pierce, D., and Wagstaff, K. (2001). Multi-document Summarization via Information Extraction. In *Proceedings of HLT-2001 (Human Language Technology Conference)*, San Diego, CA.

Wilcock, G. and Matsumoto, Y. (1998). Head-Driven Generation with HPSG. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Wilson, D. R. (1997). *Advances in Instance-Based Learning Algorithms*. PhD thesis, Department of Computer Science, Brigham Young University.

Woods, W. A. (1982). Optimal Search Strategies for Speech Understanding Control. *Artificial Intelligence*, 18:295–326.

Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*. Addison Wesley, Cambridge, MA.