Context Dependent Probability Adaptation in Speech Understanding in the Philips Automatic Inquiry System

Erwin Drenth, University of Groningen
Cognitive Science and Engineering
Master's Thesis
1996

Supervisors:
Bernhard Rüber (Philips Research, Aachen, Germany)
Tjeerd Andringa (University of Groningen, the Netherlands)

Table of Content

1. Introduction

In this thesis, we will look at ways to improve speech understanding using dialogue context information (i.e. the past dialogue history) in the Philips Automatic Inquiry System. Several different methods of language modelling will be discussed: bigrams, trigrams and concept set estimations. Combinations of these models will also be investigated, as well as different ways for modelling context information. We will see that context dependence can bring some remarkable improvements, especially when taking into account the circumstances we will have to deal with.

In chapter 2, a general description of the system will be given, chapters 3 and 4 will be concerned with system adaptations and data setup. In chapter 5, we will define contexts manually, and in chapter 6 we will use these contexts to train context dependent bigram models. In chapter 7 we will again use these contexts for training what we will call *graph based* models: We will estimate joint probabilities of concepts using graph theory. In chapter 8 we will train bigram models again, only then we will use contexts that are selected automatically, by a clustering algorithm. Some previous results of a similar investigation into context dependencies will be discussed in chapter 9, and we will also try to reproduce these results. Finally, we will state our conclusions in chapter 10.

Regretfully, there will be no room to perform all experiments to our complete satisfaction, as not all methods at our disposal are particularly suited for the system we work with. We are also constrained in our conclusions, due to the lack of data. So some questions will be left unanswered. Still, this thesis will be a thorough survey of the possibilities there are to model context dependencies in this particular system, and many conclusions will also hold for similar systems.

2. TABA, The Philips Automatic Inquiry System: An Overview

2.1 Introduction

The following overview of the Philips Automatic Inquiry System follows [3] to a certain extent, though it will be somewhat more detailed where matters especially important for this thesis are concerned.

Automatic inquiry systems are systems that people can consult to obtain some sort of information without the involvement of a human operator. The input from the user is used to create a database query and the results are returned to him or her. This input can have several forms, such as pressing keys on a touch tone telephone or speaking a few words the system can understand (e.g. "yes", "no" and/or some digits).

The automatic inquiry system (called TABA: *Telefonische Automatische Bahnfahrplan-Auskunft*[1]) developed at Philips Research Laboratories in Aachen, Germany is a system that provides train timetable information over the telephone and which allows users to talk in fluent, natural speech, like one would use when talking to a human operator. The system is currently operational with the Swiss Railway and provides information about more than 1700 railway stations and their connections in Switzerland.
A typical dialogue goes as follows[2] (translated from German into English):

System: Good morning. This is the automatic timetable information service. From where to where would you like to go?
Caller: Good morning. I would like a connection from Aachen to Bonn.
System: When would you like to go from Aachen to Bonn?
Caller: Tomorrow morning.
System: At what time would you like to travel tomorrow morning?
Caller: Around seven o'clock will be fine.
System: So you want to travel at eleven o'clock in the morning?
Caller: No, at seven o'clock!
System: So you want to travel at seven o'clock in the morning?
Caller: That's right!
System: There is the following connection: With the CityExpresss, departure at Aachen Main Station at 7:12, arrival at Köln Main Station at 8:09, there you take the CityExpress leaving at 8:19, arrival at Bonn Main Station at 8:44. Would you like me to repeat the connection?
Caller: No, thanks.
System: We wish you a pleasant journey. Thank you for calling, good bye.

As can be seen, there is a lot of similarity between the way this call is handled by the system and the way a human operator would handle it. Error recovery is possible because the system always checks the new knowledge it has obtained, both explicitly ("So you want to travel at seven o'clock in the morning") and implicitly ("When would you like to go from Aachen to Bonn?"). At this point the caller has the opportunity to correct the system. If this fails several times, the caller is referred to a human operator.

The system architecture consists of four parts, viz. the speech recognition module, the speech understanding module, the dialogue control module, and the speech output module. These modules are executed sequentially, see figure 2.1. In the next paragraphs we will discuss each of these modules separately.

---

[1] In English: Telephonic Automatic Train Timetable Information.
[2] To chat with the Philips prototype TABA yourself, dial +49 241 60 40 20. Information about more than 1000 German train stations can be obtained. You can also call the on-line system in Switzerland: +41 512 20 23 23.

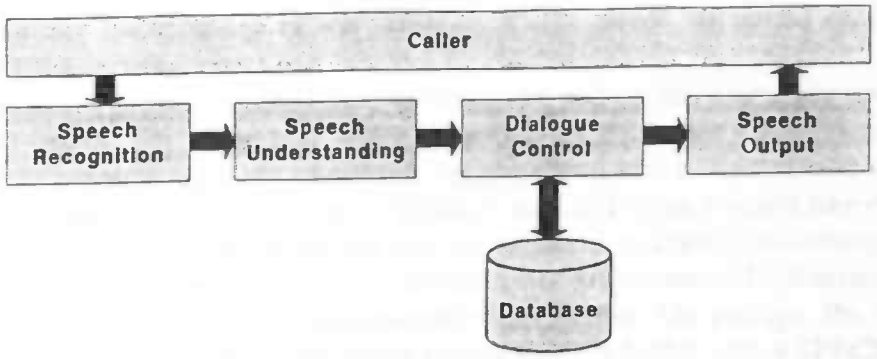


Figure 2.1: The system architecture.

2.2 Speech Recognition

The first module handles the speech recognition. For this, the PHICOS system developed at Philips Research is used. It uses Hidden Markov Models with continuous mixture densities, 6-state left-to-right phoneme models, and a tree-organised beam search. A detailed description of this system is beyond the scope of this thesis, so we refer to [23] for a more in-depth discussion.

The output of this module is a *word graph*, which is a compact way of representing many possible sentences. A graph contains nodes and edges. A word graph is a directed acyclic graph in which the nodes represent points in time and the edges are labelled with a word and its acoustic score (see figure 2.2). The score of a word is the (suitably scaled) negative natural logarithm of its probability, which can be computed using estimation techniques that are described in [5].

Each path through the graph represents a possible sentence. For each spoken word, several thousand word hypotheses are computed, but with proper pruning and optimisation, an average of about 10 edges per word can be reached, which gives a satisfactory performance.

The word graph thus created is then sent to the following module, the speech understanding module, on which we will focus next.

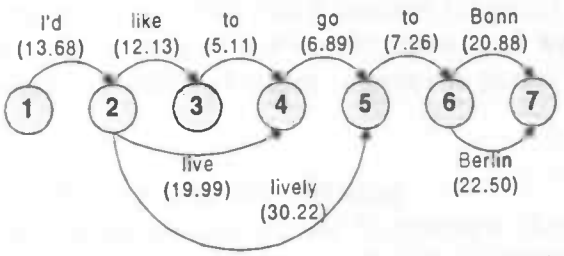


Figure 2.2: An example of a (strongly simplified) word graph. The numbers between parentheses are the scores, the negative logarithms of the acoustic probabilities computed by the speech recogniser.

2.3 Speech Understanding

When we want to understand a spoken sentence, we want to determine the meaning of what was said. For the purpose of creating a database query, it is not necessary to determine the meaning of every single word. What we are looking for are certain (parts of) phrases which express a part of a request to a train timetable information system. These phrases are called *concepts*. For example, in the sentence "I would like to go from Aachen to Bonn" there are two concepts[3] , viz. <origin>[4] ("Aachen") and <destination> ("Bonn"). These concepts

[3] Actually, there are four concepts. "I would like" and "go" are also concepts for reasons explained later.

can be in arbitrary order and there can be words in between which are meaningless with respect to computing the meaning of the sentence. These words are called *filler words*, e.g. "hello", "thank you", "ehm" etc.

So, instead of processing the entire sentence, we end up processing the concepts in that sentence. This has the advantage that sentences which are not entirely grammatically correct, which of course happens regularly in spoken language, can still be understood without many problems. For instance, from the sentence "Hello, I want to I mean I would like ehm.. to go to Bonn no from Bonn to Aachen" there can still be extracted a destination-concept and an origin-concept, although grammatically the sentence is not fully correct. Furthermore, this technique has the advantage that it is computationally inexpensive: On average, the system needs 21 ms to understand a sentence[5] on an Alpha processor (225 MHz) with a SPECfp95 value of 5.71. To perform these computations, we use a program library called SUSI (*Speech Understanding Software Interface*) developed at Philips, which is a package containing several tools for processing language.

To compute the meaning of the concepts we use an *attributed stochastic context-free grammar*. To determine the most likely sequence of concepts we also use this grammar, in combination with a *concept bigram model*. A bigram is a special kind of *N-gram*, with $N = 2$. An N-gram model is the probability for a certain event e_N ,given the (N-1) previous events:

$$P(e_N) = P(e_N | e_1 \ldots e_{N-1}) \tag{2.1}$$

So, when we use a bigram model for concepts, every concept c_i gets assigned a probability, for all possible predecessors $c_1 \ldots c_K$, with K the number of concepts, so that

$$\forall\ j = 1,\ldots,K \quad \sum_{i=1}^{K} P(c_i | c_j) = 1 \tag{2.2}$$

For instance, if we have the concepts <origin> and <destination>, we can compute the bigram probability of <origin>, given that the previous concept is <destination>, $P(< origin >|< destination >)$, by counting how often this particular bigram occurs in the training set[6].

This concept bigram model we shall call a *concept language model*, and concept language models will be the main subject of our investigation, and we will elaborate on them later. For a different approach, in which an entire sentence is parsed and its meaning is computed, we refer to [6].

2.3.1 Extracting Concepts and Computing their Meaning

In our version of the TABA system, we use 76 concepts (Appendix I). The extraction of these concepts is done by parsing the word graph for all the different concepts, with a stochastic context-free grammar. The word graph is transformed into a *concept graph*. Computing the meaning of the concepts is done with an attributed grammar, i.e. an attributed stochastic context-free grammar is used. This will be explained next.

---

[4] Concepts will be written in between angled brackets from now on: <conceptname>.

[5] That is, when using N-gram search. N-best search is slower. More on this later.

[6] In practice it is often not possible to 'just count how often a bigram occurs'. But for the sake of simplicity, we leave it at this for now.

2.3.2 Attributed Stochastic Context-Free Grammar

We create a concept graph, of which the nodes are the same as in the word graph, but the edges are now concept instances instead of words and all entries of the original graph that do not contribute to a concept are missing, as is seen in figure 2.3. The scores of these concepts are the summed acoustic scores of the words contained in these concepts, plus the scores from the rules of the attributed stochastic context-free grammar.

In a context-free grammar, every rule has the form $A \rightarrow \psi$, where A is a non-terminal symbol and ψ is any string, possibly empty, from the union of the non-terminal and terminal alphabets [16].

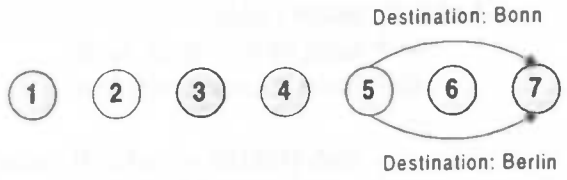


Figure 2.3: Concept graph created from the word graph of fig. 2.2.

2.3.2.1 Stochastics

A stochastic context-free grammar [8] is used to model the concepts. In this kind of grammar, every rule has a probability which indicates how likely it is to be applied, given the left-hand-side non-terminal, where concepts serve as distinct start symbols of the grammar [2]. Every possible derivation is computed and inserted into the concept graph. A rule can use other rules for its description, so the grammar can be regarded as a set of rules that may share subordinate rules. The probability of deriving a word sequence $w_{1,i} \ldots w_{n,i}$ given a concept c_i, which we shall call the *derivation probability*, is:

$$P(w_{1,i} \ldots w_{n,i} | c_i) \tag{2.3}$$

Furthermore, the speech recogniser module has delivered the acoustic probabilities $P(\mathbf{O}|\mathbf{W})$, which denotes the probability of finding the acoustic vector $\mathbf{O}$, given a word sequence $\mathbf{W} = w_1 \ldots w_m$. For a certain concept sequence $\mathbf{C} = c_1 \ldots c_n$ this amounts to the probability of this concept sequence, *given an acoustic vector* $\mathbf{O}$ (the user utterance), $P(\mathbf{C}|\mathbf{O})$, which can be estimated using Bayes' rule [5]:

$$\tilde{\mathbf{C}} = \arg\max_{\mathbf{C}} P(\mathbf{C}|\mathbf{O}) \tag{2.4}$$

$$P(\tilde{\mathbf{C}}|\mathbf{O}) = \max_{\mathbf{C}} \left\{ \frac{P(\mathbf{C}) \cdot P(\mathbf{O}|\mathbf{C})}{P(\mathbf{O})} \right\} \tag{2.5}$$

and

$$P(\mathbf{O}|\mathbf{C}) \approx \max_{\mathbf{W}} \{P(\mathbf{W}|\mathbf{C}) \cdot P(\mathbf{O}|\mathbf{W})\} \tag{2.6}$$

Since $P(\mathbf{O})$ does not depend on $\mathbf{C}$, maximising $P(\mathbf{C}|\mathbf{O})$ is equivalent to maximising the likelihood $P(\mathbf{C},\mathbf{O}) = P(\mathbf{C})\ P(\mathbf{O}|\mathbf{C})$. So, this leaves us with $P(\mathbf{C})$ (because $P(\mathbf{O}|\mathbf{W})$ is given by the recogniser module and $P(\mathbf{W}|\mathbf{C})$ is the derivation probability), which requires a concept language model, that assigns a probability to every concept sequence $\mathbf{C}$. As already stated, the

standard system uses a bigram model. This produces the following concept language model probability for a certain sequence $c_1 \ldots c_n$ (for sentence end and begin we use the @):

$$P(c_1 \ldots c_n) = P(c_1 | @) \cdot P(c_2 | c_1) \cdot \ldots \cdot P(c_n | c_{n-1}) \cdot P(@ | c_n) \qquad (2.7)$$

Thus, a probability can be estimated for every concept sequence C.

The following are example rules which are written in HDDL, a language specially designed for automatic inquiry systems [1]. It is taken from the grammar we used in our experiments[7].

```
<basic_time> ::= (0.01) <hour_0_24> Uhr
        start_time := <1>.hour * 60
        end_time := <1>.hour * 60

<hour_0_24> ::= (0.003) drei
        hour := 15
```

Example 2.1: An excerpt from the grammar.

The right hand side contains the definition of the (left hand side) non-terminals, which are written between angled brackets (and therefore we also write the concepts between angled brackets, as they serve as distinct start symbols for grammar, and as such are non-terminals). The number between parentheses is the probability. Then follow the terminals and/or non-terminals, which form the syntactic part of the rule. On the second and third line there are the *attributes* (with assignment operator ':='), which form the semantic part, which we will explain next.

2.3.2.2 Attributes

With this stochastic grammar, we can determine the possible concept sequences. However, in order to create a database query, we need the actual meaning of the concepts, rather than their textual representation. For instance, in the example above, the string "drei" cannot be used for calculations, it needs to be transformed into the integer 15 (i.e. 3 p.m.)[8].

The best time to derive these meanings is during the construction of the concept graph; a meaning is then associated with all concept instances and not just with those in the chosen path. This has the advantage that one could use semantic information for additional constraints on the graph, e.g. "two destinations observed in one sentence must be equal" or "two different destinations in one sentence are very unlikely" [2]. For this derivation we use an attributed grammar: Each non-terminal can have any number of attributes, of which we saw an example above.

So, "start_time := <1>.hour * 60" means that the attribute *start_time* (belonging to <basic_time>) receives the value of the attribute *hour* belonging to the first element of the right hand side of the rule (designated by <1>), which is <hour_0_24>. When we look at the rule for <hour_0_24> we see that if we find the terminal "drei", the attribute *hour* will receive the value 15. This value is then used for *start_time*, which will become $15 \cdot 60 = 900$ (minutes past midnight). This way, an expression is parsed top-down, and its meaning is computed bottom-up.

The attributes belonging to the concepts also form the basis for our error calculation, which is defined as the *attribute error rate*, on which we will elaborate in the next chapter.

---

[7] Some translations: "Uhr" means "o'clock", and "drei" means "three".

[8] How we distinguish between a.m. and p.m. is a matter we will not discuss here.

2.3.3 Filler Arcs

In general, there is no connection from the start node to the end node of a concept graph. It is also possible that, due to recognition errors, concept instances appear which are not part of the spoken sentence. So, when searching through the graph to find the optimal path, we have to be able to bypass concepts as well as to bridge gaps.

For this purpose, *filler arcs* are introduced. They are labelled with an empty concept and with the acoustic score of the (acoustically) optimal path between their start and end node.

For every concept edge, a bypassing filler arc is inserted (because of the possibility of incorrect recognition). Also, every concept end is connected to each following concept begin. The result is a connected concept graph which serves as a suitable basis for determining the most probable path (figure 2.4).

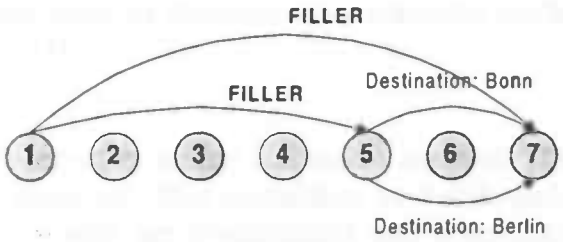


Figure 2.4: The connected concept graph.

2.3.4 Searching for the Best Path

We now have the completely connected concept graph through which we have to find the path with the lowest score. During this search we not only use the acoustic scores, but also a concept language model[9] (in the standard version of TABA a bigram model) and the derivation scores, as was explained above.

To establish a better coverage of the sentence by the concept graph, we create a few concepts for standard utterances as there are "I would like" and "Hello". This reduces the risk of misunderstanding a concept, because the chance of confusing it with an 'important' concept, e.g. <origin>, is smaller.

However, when we want to determine the best path, it will always consist of one filler arc only, as is obvious because this will only have the optimal acoustic score of the path between its start and end. To avoid this, a filler penalty is added to the score of each filler arc. This penalty is time proportional, to express the decreasing likelihood of long sequences of filler words. The value of this penalty is empirically established through optimisation.

The search through the graph is done using an *N-gram* search algorithm when we use the bigram concept language models, and an *N-best* search algorithm when we use alternative language models.

2.3.4.1 N-gram Search

The N-gram algorithm searches through a graph the best path using dynamic programming techniques [14]. In its search, it incorporates acoustic scores, derivation scores and language model scores to find the optimal path, which it is guaranteed to find.

2.3.4.2 N-best Search

The N-best algorithm performs an exhaustive search for the N-best concept sequence hypotheses in the concept graph [24]. The algorithm produces an exact N-best list of the concept sequences, based on the acoustic and derivation scores. The consequences of using this algorithm will be discussed extensively in 7.6.1.

---

[9] Fillers are not taken into account on this level (i.e. they are not in the concept language model).

2.4 Dialogue Control

When all concepts and their meaning are known, a database query can be created. But, it does not often happen that all information can be extracted from one utterance at once (and correctly). Of course, there are often things missing, ambiguous or contradictory. And, because the recogniser still has a significant error rate (see the word graph error rates in chapter 4), it is important the system can verify the information it has obtained (both implicitly and explicitly).

The obvious solution is to set up a dialogue with the user, and thus obtain the extra information needed. To make this conversation natural, flexible, and capable of adjusting to the utterance of the user, we use the already mentioned language HDDL [1]. With this language we can declaratively state how the system response should be in a relatively easy manner. It would go beyond the scope of this thesis to explain this mechanism in full, and the interested reader is referred to [1].

2.5 Speech Output

For speech output, recorded phrases and words are used. This is possible because the vocabulary is a relatively fixed one. This is stored on hard disk, and whenever output is to be created, the appropriate segments are concatenated into sentences and replayed. Though it would be possible to use synthesised speech (which is more flexible), it showed that people responded better to the more natural sounding recorded speech. However, current techniques may allow synthesised speech to be used within short time.

3. Recreating the Original System

3.1 Introduction

Our goal is to compare different concept language models. We want to do this on the dialogue level, as opposed to the (more common) sentence or word level: A comparison is made between models in a certain *dialogue state*. This dialogue state is the whole of the system's beliefs at a certain time; it represents what the system knows at a particular point in time and what it still needs to obtain and verify to come to a database query. From now on we will call this the *system state* or *context*. On the basis of this state the new system prompt is generated. When the system is not on-line and used for experiments, this system state is given as output after the prompt, so inspection of it is quite easy, as the next example will show:

System : From where to where would you like to go?
System state : 000 000 000 000 000 000 000 **080 080** 000 000 000 000 000 000 000 000
User : I would ehm like to go from Bern to Basel.
Concepts found[1]: @ FILLER <origin> <destination> <kommen> @

System : When would you like to go from Bern to Basel?
System state : 000 000 **080 080 080** 000 000 **004 004** 000 000 000 000 000 000 000 000

The system state is printed as a hexadecimal string. For now it suffices to notice that the eighth and ninth triplets (printed in bold) have changed their values from 080 to 004. This indicates the system has gone from a state in which it wanted to obtain an origin and destination, to a state in which it will implicitly verify these concepts, and will try to obtain date and time (triplets 3, 4 and 5). The exact meaning of these numbers is not relevant now, see Appendix II for a more detailed description of what they mean.

So we make our comparison on the basis of the same system state. But how do we make sure the system is in the state in which we want it to be?

3.2 Reproducing the Original System Prompts

If we want to compare our new models to the original one, it means that we compare output on the basis of the same input. In this case, the input are word graphs, as delivered by the recognition module of the on-line Swiss system. The output are the best paths that are found, represented as a concept graph with only one possible path, and these we will compare. An example of such a graph is figure 3.1. This is the format[2] in which SUSI represents the concept graphs. The first two numbers are the node numbers, then follows the concept or filler preceded by the symbol @ to distinguish them from normal words, then the score of this concept. On the next lines we find the actually found words (*text*), and the attributes and the values belonging to them. BEGIN_LATTICE and END_LATTICE speak for themselves.

If, however, we feed the system using our new model, the resulting system state will not always be the same as it was in the original system, so any comparison from that point on will be worthless! This is a problem when evaluating inquiry systems in general [4] and when we want to evaluate on the dialogue level in particular: How to compare results?

Of course, the best way would be to create a new model, do a field test for a few months and then compare[3] things as transaction successes, number of turns that are necessary

[1] <kommen> is a concept for words like "travel"; the single symbol @ represents the sentence end and begin; the word sequence differs somewhat in German.
[2] A few things are omitted for readability, but they are not relevant here.
[3] Assuming everything else will have remained the same: the group of users, the recognition error rates, no station names are added, etc.

per user, dialogue duration and correction rate [4]. This is a good way to test the system, but of course totally unsuitable for (relative) fast evaluations of different language models. That is why we have chosen for the following solution of *parallel input*.

```
BEGIN_LATTICE
   1    7   @origin                668.3259
            text                   von Schaffhausen
            origin                 Schaffhausen
   7   19   @destination           595.6432
            text                   nach Basel
            destination            Basel
  19   27   @FILLER                713.2900
            text                   aus der
END_LATTICE
```

figure 3.1: a concept graph with only one path: from node 1 to 7, 7 to 19 and 19 to 27.

3.2.1 Parallel Input

Instead of using only one input stream, we have created a second, parallel input stream: One stream to bring the system in its original state, and one stream with which our new models are tested. The detailed explanation of this process can be found in Appendix III. Here, it suffices to know that the new concept language model is evaluated in the same system state as the system was in when it received the original input on-line.

This solution has its weaknesses of course, the main one being that we now do not profit from the (possibly) improved understanding, because we force the system in a certain state. This will be clear with the next example, which is a possible part of a dialogue:

System: From where to where would you like to go?
User : From Aachen to Bonn.

At this point, the original system could have understood e.g. "From Aachen to Berlin", and the follow up question would then be "When would you like to go from Aachen to Berlin?", which would then be corrected by the user. But, it is very well possible that the system with our new model would understand the sentence correctly, which would give rise to a different follow up prompt and a different user response, so the system state would no longer be the same. Beside that, the next input would be the word graph representing the user correction, which would make no sense at all to the system if it had understood the sentence correctly in the first place. It could be that, with the new model, the dialogue would terminate successfully faster, *when tested in the field*. But for us, there is no way of knowing so we have to ignore this possibility. It is a compromise we had to do. Even if we would track the changes of the dialogue state, we still would not be able to say something about a possible faster completion of the dialogue most of the time, because we would have the possibility to observe only one change in dialogue state, which is not enough to make any statements about a faster completion.

3.3 Error Rates

The establishing of the error rates needs some explanation, because it is somewhat different from the usual way this is done. In speech recognition, it is common to determine *word error rates* by minimising the Levenstein distance between the transcribed and recognised sentences. This is done by aligning the transcribed sentence with the recognised sentence, thus determining how many words were inserted, deleted and substituted. For instance,

Spoken : This is a test sentence.
Recognised: This a test rental.

would yield one deletion ("is") and one substitution ("rental" for "sentence"), i.e. two[4] errors which amounts to a word error rate of 2/5 = 40%.

When we want to determine the attribute error rate[5] (AER), however, we do not perform this alignment. The attributes are put into two sets and these sets are compared:

Spoken:	*Understood*:
origin: Aachen	destination: Berlin
destination: Bonn	origin: Aachen
date: tomorrow	

The above example would yield one substitution (in the 'destination' attribute) and one deletion (the 'date' attribute), i.e. two errors, which would amount to an AER of 2/3 = 67%.

However, our concept language models only have an effect on the concepts in a graph, as the name already suggests. They do not influence the attributes directly. We therefore also compute the *concept error rate* (CER), which is computed similarly to the word error rate: The spoken and understood 'sentence' (i.e. concept sequence) are aligned and an error rate is computed (we ignore the fillers):

Spoken:		<origin>	<destination>		<date>
Understood:	FILLER	<origin>	<kommen>	FILLER	<date>

This understood sentence would yield one substitution, so the CER = 1/3 = 33%.

The reason we used the AER in the first place, is that the attributes are the components that influence the system performance directly: A database query is built up with these attributes. Beside this, there is often a one-to-one mapping between concepts and attributes. E.g., <origin> has one attribute, viz. 'origin'. There are however some concepts that have more than one attribute. One last thing one has to bear in mind upon inspecting these concept error rates, is that the system is optimised using the AER, in order to optimise system performance. The CERs are therefore non-optimised values. But this will be brought forward again when we discuss the actually achieved error rates.

[4] Of course we could say that "is" is substituted by "a", "a" is substituted by "test" etc. But we minimize on the number of errors.

[5] Also called *concept error rate* in [4], but to prevent confusion we use the term attribute error rate.

4. Setting Up the Data

4.1 Introduction

Our data was provided by Philips Dialogue Systems (Aachen, Germany) and consisted of material collected from November 2nd 1995 until March 29th 1996 in Switzerland by the Swiss Railway. This material comprised:

- The system prompts
- The time and date of each dialogue
- The transcriptions of the user utterances
- The word graphs which were the output of the speech recogniser
- The grammars which were used during this period

4.2 Selection

This material could not be used in its raw form. Because we wanted to reproduce the dialogues, for reasons explained in the previous chapter, we could only use the consecutive turns of each dialogue. This is the only way we would be able to recreate the original system state later. When for instance a user utterance was missing because it was nonsense (and consequently not transcribed), we could not use the dialogue from that point on.

The data came divided into four periods, in each of which a different, newly trained grammar was used. The trainings were performed using the data that was collected over the previous periods. The first period is from November 2nd 1995 until February 25th 1996, the last three periods covered March.

We were able to use 79.5% of the total of the consecutive turns, the rest was lost because of differences between the original prompts, and those produced by our system. This was caused by little differences between the on-line system in Switzerland and the system used by Philips Aachen.

The four periods have comparable *perplexities*, but varying attribute error rates, which we will discuss in the next paragraphs.

Finally, we created one new corpus, consisting of the four periods, which we divided into a test set and a training set. The training set consists of 6.1 hours[1] of spoken material and 3704 dialogues; the test set consists of 50 minutes spoken material and 530 dialogues.

4.3 Perplexity

Perplexity is defined as the probability that a language model produces for a sequence of words $w_1 \ldots w_N$, normalised with respect to the number of words N by taking the N-th root and the inverse [15]:

$$PP = P(w_1 \ldots w_N)^{-1/N} \tag{4.1}$$

Using the definition of conditional probabilities,

$$P(w_1 \ldots w_N) = \prod_{n=1}^{N} P(w_n | w_1 \ldots w_{n-1}) \tag{4.2}$$

and taking the natural logarithm, we obtain the *log perplexity*[2]

$$\log PP = -\frac{1}{N} \sum_{n=1}^{N} \log P(w_n | w_1 \ldots w_{n-1}) \tag{4.3}$$

[1] Estimated at 120 words per minute.

[2] Also called the *estimated entropy* (p. 450, [19]).

Perplexity can be seen as the average difficulty or uncertainty in each word based on the language model. Or, in other words, perplexity can be considered as "the average number of possible words following any string of (N-1) words in a large corpus based on an N-gram language model" (p. 450, [19]). Apart from the constant ($-1/N$) the corpus perplexity is identical to the probability or likelihood. Therefore minimising the corpus perplexity is the same as maximising the log-likelihood function.

In our case, the four periods had comparable perplexities with an average of 12.5, using a bigram language model trained on all four corpora[3]. The fact that the periods do not differ in perplexity significantly is important, because it indicates that the people who called the system, did not change their choice of words when asking for information. It would have been problematic, for instance, if the perplexity had lowered over these periods, because this would be an indication that people are talking in a different (easier) style, which would probably be due to a change of users.

4.4 Error Rates

During the four periods, there is a decrease in attribute error rate from 39.14% in the first period, to 22.39% in the fourth. This improvement is mostly due to the increasing success rate of the speech recogniser; the word graphs delivered contain less recognition errors, thus making the task of finding the right concepts easier for the speech understanding module. This was confirmed by computing the error rates for the first period, with the grammar used in the last period. This grammar was the best trained one, but the attribute error rate of the first period did not change significantly. These differences are not a problem for us, because we focus on relative improvements.

4.4.1 Word Graph Error Rates

The word graph error rates of the four periods, i.e. the minimal error rates taken over all possible sentences in the word graph, are shown in table 4.1. They also show the increase in speech recognition performance, although they are still quite high. The first period shows clearly the bootstrapping problems one has with such a system: Many station names are not trained well or not at all.

Period	Word Graph Error Rate
1	20.23%
2	17.72%
3	12.95%
4	11.65%

Table 4.1: Word graph error rates over the four periods.

4.4.2 Concept Graph Error Rates

Analogous to the word graph error rates, the concept graph error rates are computed for the test and training set. This gives a measure as to what we can reach, if we were to do everything right. The numbers are shown in table 4.2. We have to keep in mind as well, that the concept graph error rates are also dependent on the word graph error rates. As we could see in table 4.1, the word graphs do not always contain the entire spoken sentence. This is also

[3] Please note that the language model was trained on the same corpus as we computed the perplexity of, so the actual perplexity may be different, but we are only interested in the possible *differences* between the four corpora.

important to keep in mind when we inspect our improvements in language modelling in the following chapters.

	AER	CER
Training	15.54%	4.06%
Test	15.52%	3.80%

Table 4.2: The concept graph error rates for the test and training set.

One additional remark about the attribute error rate of the concept graphs needs to be made. They are not computed entirely according to the explanation in the previous chapter, i.e. the set comparison without alignment (3.3). In this case, we first performed an alignment of the concepts, and *then* the attributes are compared, because otherwise it would require the enumeration of all paths through the concept graphs, whereas the aligned versions can be obtained by dynamic programming. This results in an estimation for the AER which is higher than it would be when calculated according to 3.3, because the position of the attribute is now also taken into account implicitly (through the concept position). So these numbers are to be interpreted as an upper bound[4] on the AERs for the concept graphs.

4.5 Significance of the Error Rates

Because the sizes of the different data sets will become smaller once we define our contexts (in the next chapter), it is important to give an indication of how to interpret the established error rates. We do this by giving the 95% confidence interval of the attribute error rates (i.e., the error rates are given with a certainty of 95% in the designated interval), computed according to [21], pages 258 and 259. For this, we had to assume a binomial distribution of the error rates. This is not entirely correct, as the error rates are a composition of three different kinds of errors (insertions, deletions and substitutions), so there are not really only two possible outcomes. Beside this, it is very well possible to achieve a total error rate of over 100%, if the number of insertions is great enough. However, for the sake of simplicity we will assume a binomial distribution, so the intervals are not to be taken as an exact number, but rather as an approximation. A final simplification is that we give only one value for the upper and lower boundary, thereby suggesting the interval is symmetric, which is not so. But the differences are so small that most of the time, this simplification is justified.

Furthermore, we write two digits after the dot when the data set comprises more than 2000 elements, and one digit for sizes between 100 and 2000 elements (p. 258, [21]).

[4] Though this upper bound will not be very different from the actual number.

5. Establishing Different Contexts Manually

5.1 Introduction

We want to establish different concept language models for different contexts. This means we have to divide the training and test corpora into different parts, according to our selection, and train new concept language models with them. This selection is done manually, in chapter 8 we will look at the possibilities of automatically finding relevant contexts.

5.2 Establishing Different Contexts

As was already mentioned, we use 76 concepts in out speech understanding system. We trained a context independent concept language model and a new grammar on the training set, and established the following error rates which serve as a baseline, according to which our context dependent models should perform better (table 5.1). We also give the error rates for the system without concept language model, which clearly show the influence the model has.

	AER	95% Int	CER
Bigram	34.83%	±1.6%	17.28%
Unigram	39.22%	±1.7%	22.96%
No LM	48.36%	±1.9%	31.52%

Table 5.1: Error rates for context independent bigram and unigram, and no language model.

We trained both a unigram and a bigram, but as can be seen, the bigram performs significantly better than the unigram. This is not surprising, as it is intuitively obvious that concept order can be an important constraint (e.g., when confronted with the question *from where to where do you want to go*, most people will answer with the place of departure first, and then the place of arrival (*priming*). So when you find an origin, it will be likely that the next concept will be a destination. This is reflected in the bigram models, but not in the unigram models. These last ones do not profit from the structure of the utterance). As was already mentioned, bear in mind that these results are obtained through optimisation on the AER, *not* on the CER.

When we inspect the data manually, there are some situations, or rather *contexts*, that are immediately noticed as being relevant, because they often entail the same kind of user response, and therefore also the same set of concepts. This means that training on this situation could lead to a bias towards this set of concepts, which could improve the understanding (in [22] a similar idea is pursued, though it is not based on defining a certain context, but on the mutual information of words).

Of course, this is only a manual selection, which could have many disadvantages. But we felt that some contexts were so obvious, that a manual selection was justified[1]. We came to the following division:

I The basic *where to / where from* questions, as there are:
- the opening sentence: "Good morning. From where to where do you want to go?"
- second connection, or not understood: "From where to where do you..."
- when time is already known: "From where to where after X o'clock..."
- when destination or arrival is already known: "Where do you want to go from ..."

II The basic *when* questions, as there are:
- "When do you want to go from X to Y?"

[1] And, as we shall see in chapter 8, our selection is not a bad one.

III The basic *time* question:
- "At what time tomorrow do you want to go from X to Y?"

IV The confirmation question:
- "So you want to go from X to Y at Z o'clock today?"

V The check whether the caller wants the information repeated:
- "Would you like me to repeat the information?"

These five contexts comprise 26 system states (Appendix IV). Their distribution on the training corpus is as shown in table 5.2. The distribution on the test corpus is similar, see table 5.3 (Context 0 means not covered by one of the five predefined contexts; a separate language model is trained on this 'rest' set). We also give the *average number of concepts per turn*, which shows that there are not many concepts to be found each turn, which will certainly have its effect on the language modelling.

Context	No. of Turns	%	Av.No. of Concepts per Turn	No. of Attributes
0	3872	28	1.2	4967
I	5362	38	1.9	9050
II	2813	20	1.8	6383
III	565	4	1.1	1700
IV	600	4	1.2	877
V	804	6	1.0	861
Total	**14016**	**100**	**1.6**	**23838**

Table 5.2: Distribution of the different contexts on the training corpus.

Context	No. of Turns	%	Av.No. of Concepts per Turn	No. of Attributes
0	509	26	1.1	615
I	795	40	1.7	1295
II	396	20	1.7	910
III	86	4	1.4	266
IV	89	4	1.2	123
V	111	6	1.1	116
Total	**1986**	**100**	**1.5**	**3325**

Table 5.3: Distribution of the different contexts on the test corpus.

5.2.1 Concept Graph Error Rates

Again, we computed the concept graph error rates (chapter 4), this time for the different contexts, which can be seen in table 5.4. It shows that, in the test set, certain contexts (IV and V) contain the right path almost 100% of the time. However, it also shows that we have to interpret these numbers cautiously, as the error rate for context IV is 7.4% on the training set.

	Test			Training		
context	AER	95% Int	CER	AER	95% Int	CER
0	10.7%	±2.3%	4.4%	11.3%	±0.9%	4.6%
I	23.2%	±1.9%	4.1%	22.2%	±0.9%	3.8%
II	14.2%	±2.6%	3.9%	14.5%	±0.9%	4.5%
III	7.1%	±5.1%	3.2%	7.1%	±1.3%	4.1%
IV	0.8%	-0.8% - +4.2%	1.0%	7.4%	±1.8%	3.3%
V	0%	+3.2%	0%	2.3%	±1.1%	1.8%

Table 5.4: Concept graph error rates for the different contexts.

6. Context Dependent Language Modelling Using Bigrams

6.1 Introduction

Since we have established the different contexts now, we can use them to create new, context dependent concept language models. We will do this in several different ways, starting with bigrams in this chapter. As we shall see, context dependent bigram modelling can bring improvement, but there is one thing we have to bear in mind, viz. that short sentences are a distinct feature of the system we are working with. That is, people will respond most of the time with the items that were asked for by the system. This means we will have concept sequences with very few items most of the time: 80% of the sequences consist of one or two concepts. And as can be seen in the tables of chapter 5, the average sequence length is 1.5 (without sentence end or begin). This means that context dependent bigram modelling might not bring as much as it would in systems which use longer sequences of items, because there is no real ordering constraint in a sentence with only one concept, apart from the fact that this concept follows the sentence beginning. We should keep this in mind when we look at the results for this kind of modelling.

6.2 Bigram Estimation

A common way to estimate the probabilities of certain events (i.e., in our case, uni- or bigrams) is to use Maximum Likelihood Estimation (MLE) $p(x) = c(x) / N$, where $c(x)$ is the number of times x is encountered in a set with N samples. But in MLE a zero probability is assigned to all unseen events, which, in our case, can be over 99% of all possible events. This would entail that these events are completely excluded from recognition, which is not a desirable situation. We use 76 concepts in our model (appendix I), that means there are 76·76=5776 bigrams possible, of which in the worst case only six are seen! To handle this, among other techniques, discounting models [10] have been developed. In our case, we use an absolute discounting model of which the parameters are established with the leaving-one-out method. We will discuss this in the next paragraphs.

6.2.1 Absolute Discounting

To make sure we do not end up with zero-probabilities, we need some kind of smoothing, for which we use absolute discounting in a form as described in [15]. First we need some definitions. Let N be the total sample size. Let $k = 1...K$ be the different events classes (e.g. a particular bigram). Let $N(k)$ be their sample counts, that is how often a certain observation is made, and let $P(k)$ be the corresponding probability of an observation k. Let $q(k)$ be the probabilities of an observation k in a less specific model, for instance a uniform distribution or a unigram model.

The general model for the discounting method is then established as follows. First, we define a discounting function $d: k \rightarrow d(k)$, which is to be subtracted from every sample count $N(k)$ and which determines the influence of the 'backing off' model $q(k)$ (i.e. the less specific model). We then have

$$P(k) = \frac{N(k) - d(k)}{N} + Q[d]q(k) \tag{6.1}$$

$Q[d]$ is the *discounted probability mass*, which depends on the discounting function d and is defined as:

$$Q[d] = \frac{1}{N} \sum_{k=1}^{K} d(k) \tag{6.2}$$

Assuming a model in which only sample counts $N(k) > 0$ are discounted by a constant value D, $0<D<1$, the discounted probability mass amounts to:

$$\frac{(K - n_0)D}{N} \tag{6.3}$$

with n_0 the number of event classes which occurred zero times. This is redistributed over all events $k = 1...K$, according to distribution $q(k)$:

$$P(k) = \frac{N(k) - D}{N} + D\frac{K - n_0}{N} q(k) \quad for\ N(k) > 0 \tag{6.4}$$

and

$$P(k) = D\frac{K - n_0}{N} q(k) \quad for\ N(k) = 0 \tag{6.5}$$

The extension to conditional probabilities is straightforward, and will not be dealt with here. Instead, we refer to [15] where a detailed discussion can be found.

6.2.2 Leaving-one-out

The unknown parameters (e.g. D) in the discounting models can be determined automatically with the leaving-one-out method. With this method, the training set is divided into a set with N-1 samples, and a set with one sample (the one left out). The relative frequencies are now estimated from the N-1 samples and the parameters are estimated with the one left out. This process is repeated N times, so all N partitions are considered. The basic advantage is that there is no need for dividing the training set into a training part and a cross-validation part. This is an efficient way of using the material, which is especially attracting when there is not very much data to work with.

To express the dependencies on the counts $N(k)$ and the general distribution $q(k)$ we use the notation $P(k) = P[N(k);q(k)]$. The training set consists of a sequence of observations k_n, n=1 ... N. The leaving-one-out log-likelihood L can be written as:

$$L = \sum_{n=1}^{N} \log P[N(k_n) - 1; q(k)] = \sum_{k=1}^{K} N(k) \log P[N(k) - 1; q(k)] \tag{6.6}$$

Optimising the model means maximising the log-likelihood (cf. perplexity in section 4.3). Now, let us define n_x as the number of event classes which occurred in the training set x times, and b as:

$$b = \frac{D \cdot n_0}{K} \quad < 1 \tag{6.7}$$

Using some math, we can transform (6.6) into [15]:

$$L = const(b) + n_1 \log b + \sum_{r>1} r n_r \log(r - 1 - b) \tag{6.9}$$

$$b \leq b_0 = \frac{n_1}{n_1 + 2n_2} < 1 \tag{6.10}$$

Thus an approximate solution is derived for the unknown parameter b.

6.3 Results

For testing these new contexts and their respective language models we had to use our method of parallel input, as was explained in chapter 3: The system was brought in its original state, but the search (i.e. N-gram search) through the concept graph is done with the context dependent language model. When we train new bigram models on the separate contexts, we get the following, relative improvement of 6.6% on the attribute error rate:

Bigram	AER	CER[1]
Context Independent	34.83%	17.28%
Context Dependent	32.54%	15.93%
Rel. Improvement	**6.6%**	**7.8%**

Table 6.1: Attribute error rates of context independent and context dependent bigram models.

With unigram language models we have a relative improvement of 7.1% (table 6.2). We computed these unigrams for reasons of comparison only, as they are obviously inferior to the bigrams: The context dependent unigram performs worse than the context independent bigram. The reason why there is a greater improvement for the unigram models than for the bigram models, is probably because a unigram model profits more from the fact that it is trained on different contexts. A bigram model already has some 'context-sensitivity', and therefore it profits less from specific context models.

Unigram	AER	CER
Context Independent	39.22%	22.96%
Context Dependent	36.45%	21.12%
Rel. Improvement	**7.1%**	**8.0%**

Table 6.2: Attribute error rates of context independent and context dependent unigram models.

In table 6.3A we can see the error rate distribution over the five contexts (plus the 'rest' context) for the bigram model(s). In table 6.3B we have the error rates for the unigram model(s). Before we discuss these results however, we need to make some things clear. The exact reasons for any gains or losses in error rates can only be revealed by an in-depth analysis of the dialogues, the graphs and the kind of errors that were made. For this, there was no time within the period this thesis had to be completed. This means that we cannot make any exact statements as to the nature of the errors. What follows is a brief analysis.

[1] Remember: We optimise on the attribute error rate, *not* on the concept error rate (because we optimise system performance). If we were to optimise on the CER, we could achieve CER = 15.75%, which is a relative improvement of 8.9%.

	Independent Bigram			Context Dependent Bigrams		Relative Improvement	
Context	AER	95% Int	CER	AER	CER	AER	CER
0	30.1%	±3.6%	18.2%	29.4%	17.2%	**2.2%**	5.2%
I	44.8%	±2.7%	17.7%	41.3%	16.3%	**7.6%**	8.0%
II	36.2%	±3.1%	20.7%	33.9%	18.9%	**6.2%**	8.7%
III	18.8%	±4.6%	9.5%	16.9%	9.5%	**10.0%**	0%
IV	6.5%	±4.7%	6.7%	7.3%	5.7%	**-12.6%**	14.4%
V	6.9%	±3.8%	5.1%	4.3%	3.7%	**37.5%**	28.6%

Table 6.3A: Error rates of the bigram model(s) split according to the different contexts.

	Independent Unigram			Context Dependent Unigrams		Relative Improvement	
Context	AER	95% Int	CER	AER	CER	AER	CER
0	36.8%	±3.8%	24.8%	32.9%	20.5%	**10.6%**	17.2%
I	48.5%	±2.7%	24.2%	46.0%	23.0%	**5.1%**	5.1%
II	39.6%	±3.2%	24.2%	37.5%	24.2%	**5.3%**	0%
III	22.6%	±5.0%	15.1%	20.3%	12.7%	**10.0%**	15.8%
IV	13.8%	±4.5%	12.4%	6.5%	3.8%	**53.0%**	69.2%
V	9.5%	±5.2%	8.0%	7.8%	5.8%	**18.1%**	3.4%

Table 6.3B: Error rates of the unigram model(s) split according to the different contexts.

6.3.1 Discussion of the Different Contexts

Contexts I, II and III

Contexts I, II and III show an improvement of the bigram modelling of 7.6%, 6.2% and 10.0% respectively (not taking the error margin into account). Context I and II show an even greater improvement than the unigrams. This is not surprising, as these contexts represent the "where to where" and "when" situations, which will often bring more than one concept in the user's response, in contrast to contexts III, IV, V and 0 which will more often result in only one concept (<yes>, <no> or <time>). So the bigrams will be able to use their sequential constraints better, which results in better performance.

Context V

The improvement of 37.5% of context V is somewhat remarkable, as the answers consists mainly of the concept <no> (context V is the "Would you like me to repeat the information?" situation). It even outperforms the unigram model for this context. We have to take into account however that we have an error margin of about ±3.8% absolute on the error rate. We are talking about five attribute errors on a total of 116 attributes. One error more or less can make a huge relative difference. If, for instance, we were to have four errors, which would yield an AER of 3.4%, we would get a relative improvement of 50% instead of 37.5%. Therefore, the only remark we can make is that there seems to be an improvement, but any quantitative statements do not seem to be justified.

With respect to the actual system performance, the gain is quite useful, because it is very annoying if the system understands the answer to the 'repeat' question wrongly. If it were to understand <yes> instead of <no> the user would get the train table information a second

time. He would probably hang up at this point, but it is also possible he wants information about a different connection (e.g. the return trip), which means he has to 'sit through'[2] the entire information repetition. This will probably not be received well.

Context IV

The same remarks for context V concerning the quantitative statements are also valid for context IV. At first sight, it seems quite alarming, an increase in attribute error rate of 12.6%, while the concept error rate decreases 14.4%. But when we look at these errors more closely, we can see the reasons. First of all, we have nine errors on a total of 123 attributes, so one error more or less will make a big difference. Secondly, there is one sentence in which two concepts are spoken, with a total of one attribute[3]. Only one concept is understood (plus a filler), but it is not a correct one. Now, this wrongly understood concept has three attributes. So, what do we end up with? With one deletion (of the spoken attribute) and three insertions (of the understood attributes), which makes a total of four errors! And because we are working with such small numbers, this means a 'dramatic' increase in AER (and a moderate increase in CER, viz. two: one deletion and one substitution).

Context 0

Most striking about context 0, the 'rest' context, is the difference between the unigram and bigram improvement, 10.6% and 2.2% respectively. The reason for this is unclear. Context 0 consists for a big part of the single concept sequences <yes> (19%), <date> (10%) and <no> (8%). One reason for the observed difference could be that the unigram model profits more from the fact that it is trained on a specific context which contains mainly one-concept sequences. We see a similar difference with context IV, but contexts III and V seem to contradict this. But, as mentioned before, quantitative judgements are too uncertain to make any exact comparisons.

6.3.2 Combining the One-Concept Contexts

A logical step after analysing the error rates per context, is creating one model for the short, one-concept sequences. This will contain the short answers as there are <yes> and <no>, but this did not bring any improvement. It did, however, give us an indication that even the small contexts are estimated fairly well.

6.3.3 Optimising the Empirical Parameters

SUSI allows the setting of different empirical parameters. One of them was already mentioned in chapter 2, the filler penalty, but there are three more parameters which can be altered. We will list them here, as well as the filler penalty for completeness:

1. *Filler penalty* (FP): A penalty which is added to every filler arc and which is time proportional to express the decreasing likelihood of long sequences of filler words.
2. *Concept language model factor* (CLF): This is a multiplicative factor with which the language model scores are multiplied during the search. With this factor the influence of the language model can be adjusted: When set to zero, no language model is used.
3. *Concept word penalty* (CWP): A constant which is added to each concept score. This influences the length of the sentences which are found, because longer sequences of concepts

[2] Up to now, the on-line system offers no barge-in.

[3] One of the concepts is <wollen_Aussage>, a concept which is not used to derive the meaning of an utterance, but merely aids the concept extraction process (see also 2.3.4). Therefore, it has no attributes.

become decreasingly likely when a positive constant is added to each of them. When set to infinity, the whole sentence will be covered by a filler.

4. *Rule probability factor* (RPF): The scores of the grammar rules get multiplied with this factor.

When we add these parameters to eqs. (2.5) and (2.6), we get the following result[4] for a concept sequence **C**, given an acoustic vector **O**, and |**C**| the number of concepts (filler penalty not included):

$$P(\mathbf{C}|\mathbf{O}) = \max_{\mathbf{C}} \left\{ \frac{\exp(\mathrm{CWP} \cdot |\mathbf{C}|) \cdot P(\mathbf{C})^{\mathrm{CLF}} \cdot P(\mathbf{O}|\mathbf{C})}{P(\mathbf{O})} \right\} \tag{6.11}$$

and

$$P(\mathbf{O}|\mathbf{C}) \approx \max_{\mathbf{W}} \left\{ P(\mathbf{W}|\mathbf{C})^{\mathrm{RPF}} \cdot P(\mathbf{O}|\mathbf{W}) \right\} \tag{6.12}$$

The values of these parameters in the above experiments were: FP=0.06, CLF=0.49, CWP=0.05, RPF=0.43. These values were established through previous optimisations using the *conjugate gradient* method [9]. Optimising on the context models did not result in an improvement. An important remark to be made at this point is, that this conjugate gradient method is not guaranteed to find the global optimum: Different start values of the parameters resulted in different end values (with up to 0.5% absolute difference in error rates).

6.3.4 Linear Interpolation of the Language Models

We combined the context dependent and context independent language models by a linear interpolation according to the following equation:

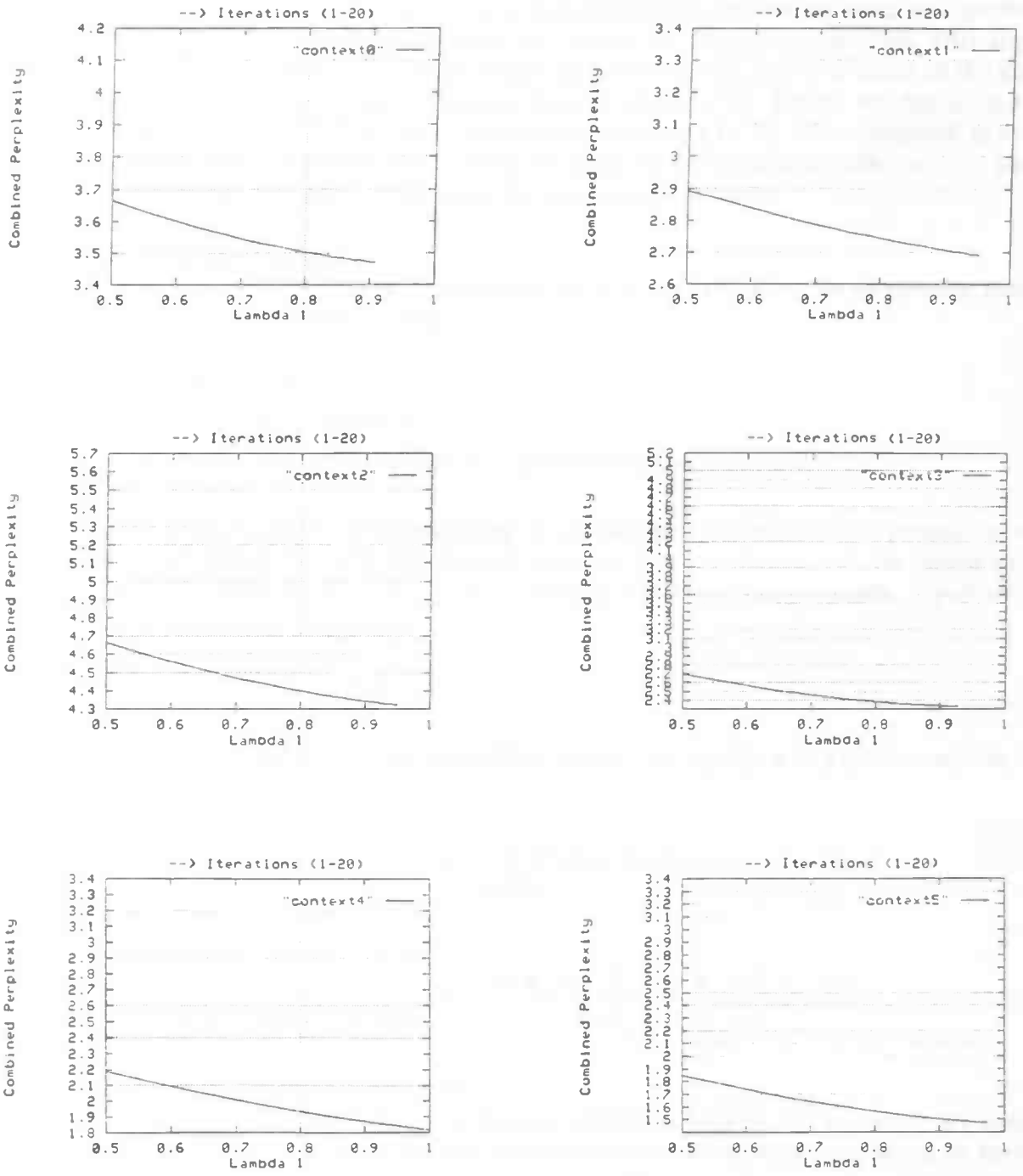
$$\mathrm{LM} = \lambda_1 \cdot \mathrm{LM}_{\mathrm{dependent}} + \lambda_2 \cdot \mathrm{LM}_{\mathrm{independent}} \quad \text{with} \quad \lambda_1 = 1 - \lambda_2 \tag{6.13}$$

This equation was estimated and optimised on the test set perplexity using the EM-algorithm [7], and was meant as a quick test for the robustness of the context dependent models. The results showed, that the context dependent models were estimated well, as λ_1 turns out to be close to 1. Figures 6.1a through 6.1f show the combined perplexities for the different contexts with the independent model, for 20 iterations of the EM algorithm. See Appendix V for the values of the parameters for all contexts.

6.4 Conclusions

Context dependent bigram modelling seems to be an effective way to improve performance in a relatively easy manner, although quantitative statements are only partly possible at this stage. More research needs to be done as to the exact nature of the errors that are still being made, as well as to the optimal context definitions. Due to the nature of the particular application we are working with, we only have short sequences of concepts. Applications which have to deal with longer sequences of elements might benefit even more from this kind of modelling, and it would be interesting to investigate this.

[4] Note that multiplying by x in the log-domain is equal to raising to the power x in the linear domain, and adding x in the log-domain is multiplying by x in the linear domain.



Figures 6.1a-f: Combined perplexities using the context dependent and context independent language models. The upper bound of the figure is the perplexity using the independent model, the lower bound is the perplexity using the dependent model.

7. Concept Set Probability Estimation

7.1 Introduction

In this chapter we will estimate context dependent language models using an algorithm for automatically finding stochastic models for a given set of random variables. This algorithm was developed by Helmut Lucke at Philips Research Aachen, and it is based on the idea of representing probabilistic (in)dependencies through graphs [17]. Before we discuss these techniques in 7.3, we will look at some probabilistic reasoning in 7.2. This paragraph is of a somewhat more technical nature and is mostly intended for the interested reader, so it is possible to skip it and still be able to understand the following paragraphs.

7.2 Probabilistic Reasoning

The following discussion in mainly based on [11] and [17]. First, let us state the three basic axioms of probability theory [17]:

$$0 \leq P(A) \leq 1$$

$$P(\text{Sure proposition}) = 1$$

$$P(A \text{ or } B) = P(A) + P(B) \text{ if A and B are mutually exclusive.}$$

A *probabilistic model* is "an encoding of probabilistic information that permits us to compute the probability of every well-formed sentence S in accordance with the above axioms" (p. 30, [17]). Now, let $\mathbf{X} = (X_1, \ldots, X_N)$, with X_i a random binary variable. A probabilistic model can determine things like

$$P(X_2 = 1, X_5 = 0 | X_3 = 0, X_4 = 1) \tag{7.1}$$

i.e., given that $X_3 = 0$ and $X_4 = 1$, the probability that $X_2 = 1$ and $X_5 = 0$. This can be done if we know $P(\mathbf{X})$, as

$$P(X_2 = 1, X_5 = 0, X_3 = 0, X_4 = 1) = \sum_{\text{all other } x_i} P(X_1 = x_1, X_2 = 1, \ldots, X_N = x_N) \tag{7.2}$$

and

$$P(X_2 = 1, X_5 = 0 | X_3 = 0, X_4 = 1) = \frac{P(X_2 = 1, X_5 = 0, X_3 = 0, X_4 = 1)}{P(X_3 = 0, X_4 = 1)} \tag{7.3}$$

following the rule of conditional probability.

However, $P(\mathbf{X})$ is very difficult to estimate this way, because not all events are seen often enough (or not at all) to allow reliable estimation (when using binary variables, we have 2^N parameters, with often only one constraint, viz. $\sum p = 1$, so we end up estimating $2^N - 1 \approx 2^N$ free parameters). In order to decrease the number of free parameters, we need more structure by e.g. assuming independencies. For instance, let $N = 3$. We say X_2 and X_3 are conditionally independent given X_1, if

$$P(X_2, X_3 | X_1) = P(X_2 | X_1) \cdot P(X_3 | X_1) \tag{7.4}$$

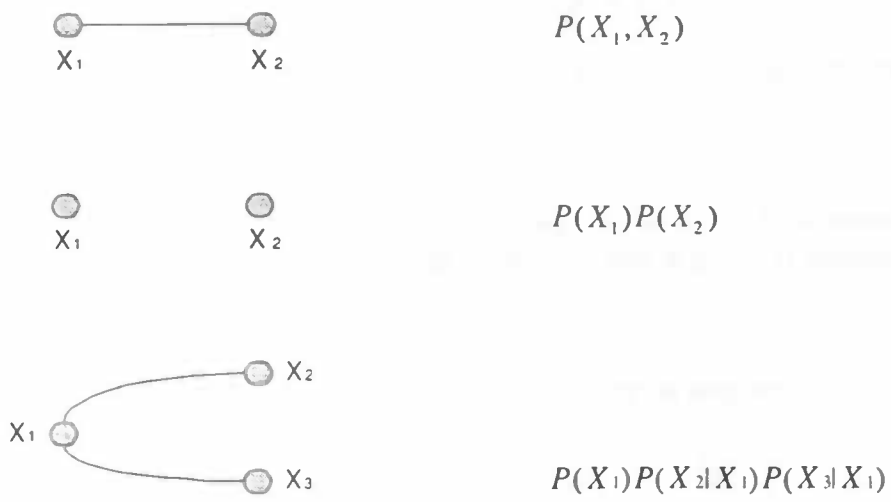
that means that

$$P(X_1, X_2, X_3) = P(X_1) \cdot P(X_2 | X_1) \cdot P(X_3 | X_1) \tag{7.5}$$

Thus, instead of estimating 2^3-1 = 7 parameters, which are required to estimate $P(X_1, X_2, X_3)$, we now need to estimate $(2^1 - 1) + 2 \cdot (2^1 - 1) + 2 \cdot (2^1 - 1) = 5$ parameters for the right hand side of eq. (7.5). This means more accurate estimations with a given amount of data. For instance, we will see later that the number of free parameters for context I will decrease from 2^N to 189, with $N = 22$.

7.3 Using Graphs for Specifying Conditional Dependencies

This discussion is mainly based on [17]. Graphs are a means for specifying conditional dependencies between variables. By specifying some independencies, graph theory can be used to derive others. See figure 7.1 for some examples of modelling probabilistic dependencies with graphs.



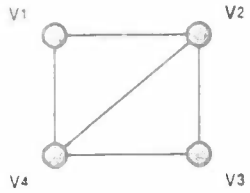
$P(X_1, X_2)$

$P(X_1)P(X_2)$

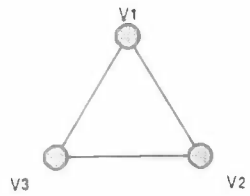
$P(X_1)P(X_2|X_1)P(X_3|X_1)$

Figure 7.1: Examples of graphs which represent probabilistic dependencies.

For useful forms of the joint probabilities $P(\mathbf{X})$, only *chordal graphs* are used. A graph is chordal when all *cycles* of length greater than 3 have a chord. A cycle is a path from $v_1 \ldots v_N$, where $v_N = v_1$, and has length (N-1). A chord in a cycle is an edge that joins two non-consecutive vertices of that cycle. An example will make this clearer, see figure 7.2.



Chordal graph, cycle of length 4, chord v_2v_4.



Also chordal, because there is no cycle > 3.

Figure 7.2: Chordal graphs

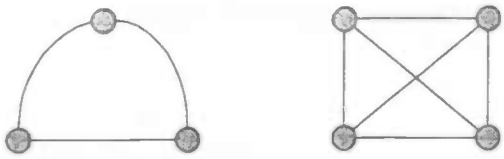


Figure 7.3: Complete graphs

A *complete graph* (figure 7.3) is a graph where all pairs of vertices are mutually adjacent to each other, and a *clique* is a maximal complete subgraph (figure 7.4). Now, let G be a graph and K_G the set of its cliques. We will consider trees that have cliques of G as vertices. The definition of these cliques trees (figure 7.5) is as follows:

A clique tree is a tree on the cliques of G that satisfies the clique tree intersection property.

Clique tree intersection property:

For every pair of distinct cliques K_1, $K_2 \in K_G$, the set $K_1 \cap K_2$ is contained in every path connecting K_1 and K_2 in the tree. It can be proven that a graph is chordal if and only if it has a clique tree.

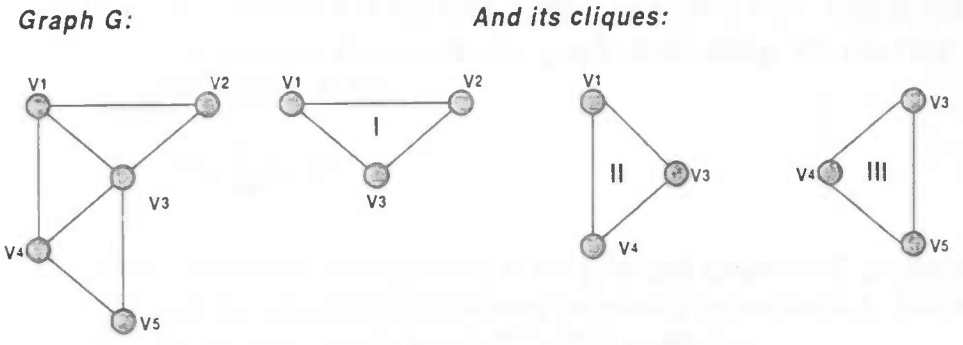


Figure 7.4: A graph G and its cliques.

Figure 7.5: The clique tree, from the cliques of figure 7.4.

The goal of all this graph theory is to come to an algebraic form of probability distributions corresponding to chordal graphs. So, let G be a graph with vertices $X_1,...,X_N$, representing the dependencies of the probability distribution $P(\mathbf{X})$, with $\mathbf{X} = X_1,...,X_N$. Let $T = (K_G, E_T)$ be a clique tree of G, where K_G is the clique set of G, and E_T is the edge set of T. Then

$$P(\mathbf{X}) = \frac{\prod_{K \in K_G} P(K)}{\prod_{(K_1, K_2) \in E_T} P(K_1 \cap K_2)} \tag{7.6}$$

where $P(K)$ denotes $P(X_{i,1},...,X_{i,k})$ if clique K contains vertices $X_{i,1},...,X_{i,k}$. This can be rewritten to:

$$P(\mathbf{X}) = P(K_1)\prod_{i=2}^{n}\frac{P(K_i)}{P(K_i \cap K_{j(i)})} \tag{7.7}$$

with $K_{j(i)}$ as the unique parent of K_i, $j(i) < i$.

This equation is useful because the factors are joint probability distributions of *subsets* of **X**, and can be estimated independently of each other by only observing the variables in these subsets. That is why these models are called *decomposable*[1]. The estimation of the probabilities can be done with e.g. a Maximum Likelihood Estimator, which would simply count the number of times an event occurs in the training data (see also 6.2).

7.4 Description of the Edge Pruning Algorithm

So, our goal is to find a stochastic model for a distribution $P(\mathbf{X})$ of a random vector **X**, of which we know nothing more than a set of N observations. We will search through possible graphs describing $P(\mathbf{X})$, looking for the one that is most compatible with the data.

First the data is divided into a training set T and a validation set V. Given a graph G and a training set T we can estimate the parameters of $P_G(\mathbf{X})$ by using eq. (7.7) for the events we have seen, and for the unseen events we use the *Add-One* estimate: $P(e) = (c(e)+1)/(N+K)$, where K is the total number of classes (=2^M with M binary variables). We will use P_G in stead of P to denote the graph dependency. We can then calculate the likelihood of the validation set V. Let

$$L_G(V) = \log\sum_{x\in V} P_G(x) \tag{7.8}$$

be the logarithm of this likelihood. This quantity is the principal measure of 'goodness' of our stochastic model, see also the discussion concerning perplexity in section 4.3. The algorithm tries to optimise eq. (7.8), by changing G, keeping T and V constant.:

1. Create a complete graph G;
2. For all edges in G find the edge e, with $G\backslash\{e\}$ being chordal, that maximises the gain in likelihood obtained by deleting that edge, i.e.

$$gain(e) = \frac{1}{|V|}(L_{G\backslash\{e\}}(V) - L_G(V)) \tag{7.9}$$

3. If the gain is below a certain threshold S, then stop, else remove e and continue.

This is called the *Edge Pruning Algorithm* (EPA).

[1] An interesting thing to notice is that the breaking down of joint probabilities into clique probabilities has its counterpart in human inference. Humans are bad at estimating $P(X_1,....X_N)$, especially when N becomes somewhat large. They therefore "conceptualize causal relationships by creating hierarchies of small clusters of variables [i.e., cliques (ED)], and the interactions among the factors in each cluster are categorized into prestored, prototypical structures, each requiring about N parameters", (p. 50, [4]).

7.5 Using the EPA

We will now describe how we use the EPA and what results we obtain in our effort to create new concept language models.

7.5.1 Creating a Concept Vector

Before we can use the algorithm, we have to transform the concepts that are found in a sentence into a vector representation. We do this by assigning a number to each concept which will denote its position in a binary vector **B**, the value 1 indicating the presence of the specific concept[2]. However, due to trainability problems and computational limitations we have to constrain the number of elements of **B** to 22 (leaving some room for future alterations, see chapter 8). We do this by clustering the concepts which are hardly ever used to the more frequently used concepts, e.g. <Pfaeffikon_stat_dest> to <destination> (for the exact clustering, see Appendix VI). We then create the bit vectors by extracting the concepts from the transcriptions of the six (five plus the 'rest') different contexts from the training corpus, thus ending up with six sets of bit vectors of which we want to estimate the probability distributions.

7.5.2 Clique Extraction

To estimate the probabilities, we first have to extract the cliques by using the EPA. In figure 7.6 the gain as a function of the number of deleted edges is plotted, in figure 7.7 the total gain is plotted. As is clear from the figures, a suitable threshold T for ending computation is $T=0$; after this point, the gain starts to decrease. In Appendix VII a graphic representation is given of the cliques that were found by the EPA for context I. We next construct a clique tree and compute

$$\forall i, \quad \frac{P(K_i)}{P(K_i \cap K_{j(i)})} \tag{7.10}$$

which means we can compute

$$P(\mathbf{X}) = P(K_1) \cdot \prod_{i=2}^{n} \frac{P(K_i)}{P(K_i \cap K_{j(i)})} \tag{7.11}$$

for all possible vectors **X**.

[2] So only the actual presence of a concept is noted. Neither its position, nor its count (in the sentence) are represented.

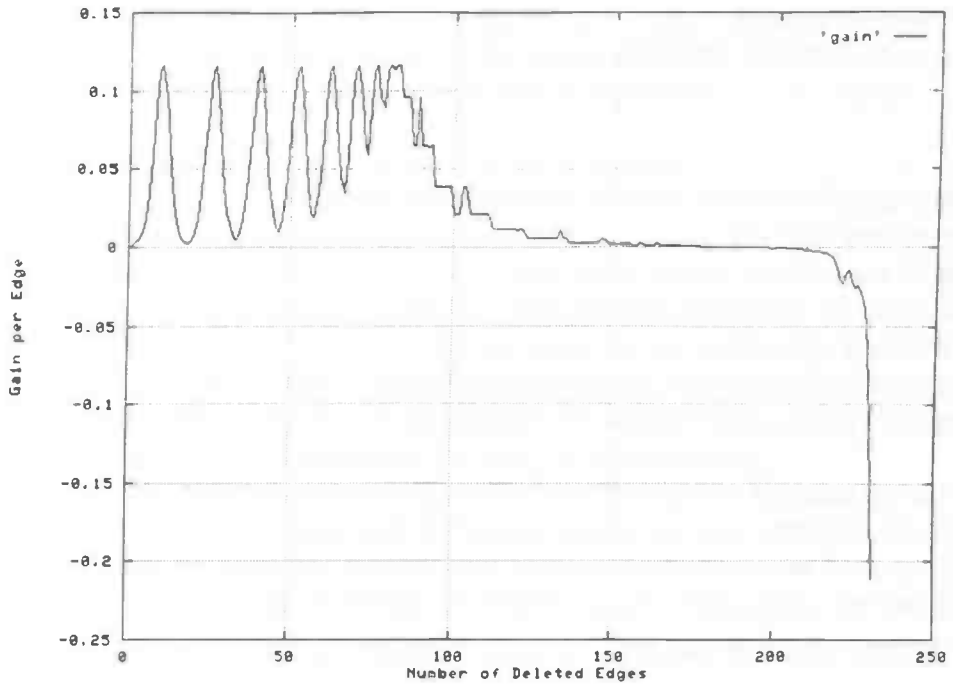


Figure 7.6: The gain per deleted edge.

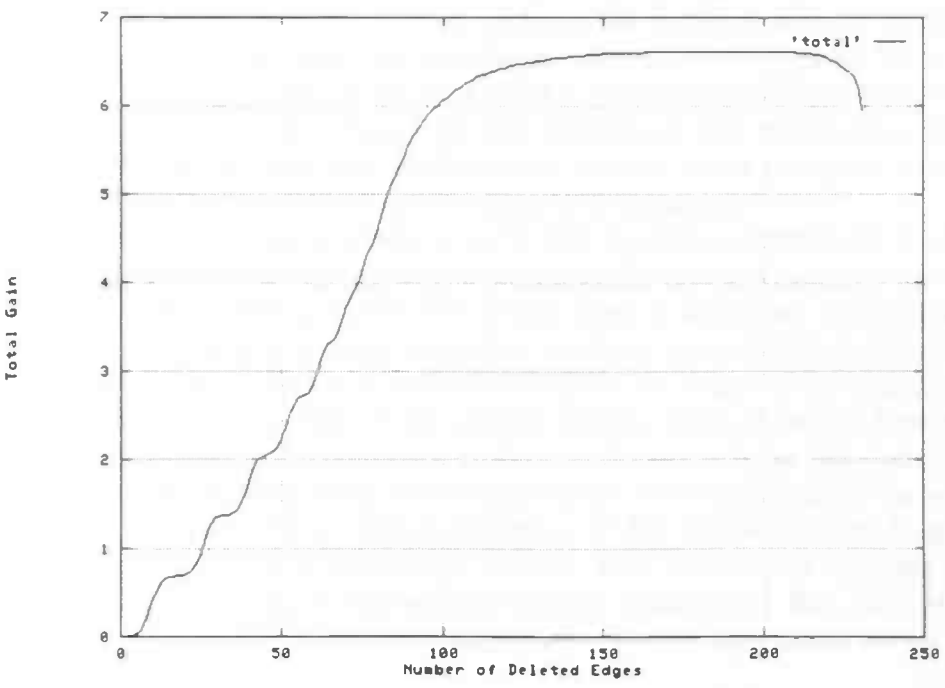


Figure 7.7: Total gain as a function of the number of deleted edges.

7.6 Results

We will now discuss the results of using the new context dependent models, which we will call *graph based*. But before we get to the actual numbers, we need to say more about using the N-best algorithm.

7.6.1 Using N-Best

In order to apply our new graph based models we first need to compute a path through the concept graph. This gives us a concept sequence $\mathbf{C} = c_1,...,c_k$. We then use our graph based models to estimate a probability for $\{c_1,...,c_k\}$. The score of this concept set is added to the acoustic and derivation scores of the path (i.e., the concept sequence). Because the acoustic and derivation scoring still has errors, we do not want the best path only, we want the N-best paths, so we can search for the best combination of acoustic, derivation and concept set scores (if the acoustic/derivation score did not have errors, we would have a perfect recognition, and therefore no need for the N best sentences, but only for the first best).

In practice, we do not need to compute all the N best paths. Because the list of paths is sorted with respect to their acoustic and derivation scores, we can terminate the search if the acoustic/derivation score becomes higher (that means worse) than the best combination of acoustic/derivation and concept set scores: if $P(\mathbf{O}|\mathbf{C}_{current}) < P(\mathbf{C}_{best}|\mathbf{O})$, we can stop searching ($P(\mathbf{O}|\mathbf{C}_{current})$ = the acoustic/derivation probability). This can be easily verified by looking at eqs. (2.5) and (2.6).

There is, however, a problem: We are not guaranteed to find the optimal solution (i.e. the optimal combination of acoustic/derivation and language model scores) within N paths. As can be seen in figures 7.8a and 7.8b, in 93% of the cases the best score is found within the first five paths that are computed. We have set N=100, as a reasonable compromise between speed and accuracy. But, when we set N=1000, we still find optimal solutions and figure 7.9 suggests there will still be more solutions after 1000 paths. Let us clarify this a little bit more.

Let S_1 be the acoustic/derivation score for concept sequence (path) $\mathbf{P}_1$. Let C_1 be the concept set score of $\mathbf{P}_1$, than we have a total score $T_1 = S_1 + C_1$. Now, if within N best paths, there is no acoustic/derivation score $S_x > T_1$ (that means the break criterion is not satisfied), we cannot be sure we have found the optimal score. There will remain the possibility that there is a certain $T_{opt} = S_{opt} + C_{opt}$, $T_{opt} < T_1$, with $S_{opt} > S_1$ and $C_{opt} < C_1$.

To see this, we will check the number of times the breaking criterion is actually met. With N = 100, in 95% of the turns the criterion is satisfied, which leaves 5% to be found 'somewhere' further down the line. If we set N = 1000, we find that 99% of the time the criterion is used, which means we have found the best solution in 99% of the cases. These numbers were established on the (relatively small) test set, so they are by no means a guarantee we will always find 99% of the correct solutions within 1000 iterations.

A solution for this problem would be to let the algorithm compute all possible paths. However, it is not written for this purpose and consequently too slow, as there are sometimes hundreds of thousands of paths possible. We would need an algorithm specially written for this purpose, but we do not have such an algorithm and there was not enough time to write it.

As is obvious now, it would be unfair to compare the results of the bigram method of the previous chapter with the results of the current models, because for the first method we have an optimal search, and for the second we do not! That is why we have recomputed the error rates for the bigram models, using the N-best search. Now the impact of the non-optimal search becomes clear, as we find a relative difference of 4%: 32.54% for the N-gram search method, against 33.95% for the N-best search method. This way we can compare the models with respect to their capabilities of representing context information. But, as is obvious, the graph based models will not be of practical importance for this application unless one of the two following things is true:

- They perform better than the bigram models combined with N-gram search, or
- They perform better than the bigrams combined with N-best search and we have a suitable search algorithm for the graph based models, so we can top the 32.54% of the bigrams.

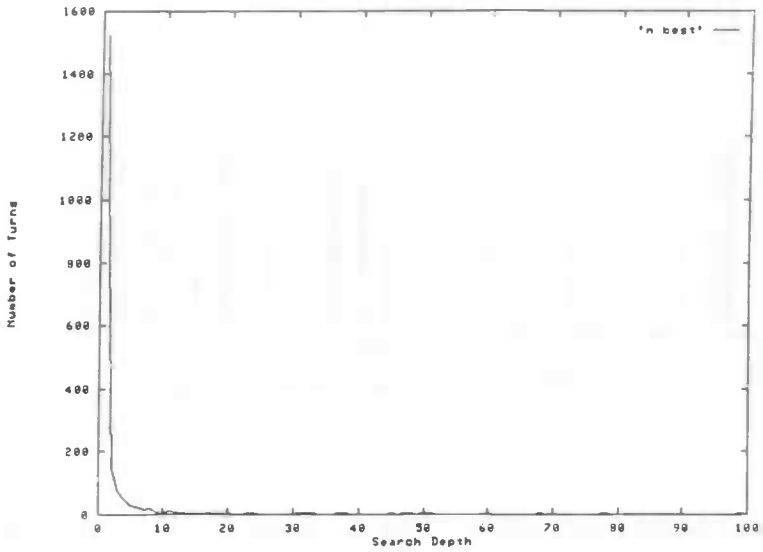


Figure 7.8a: The distribution of the search depth N using $N = 100$.

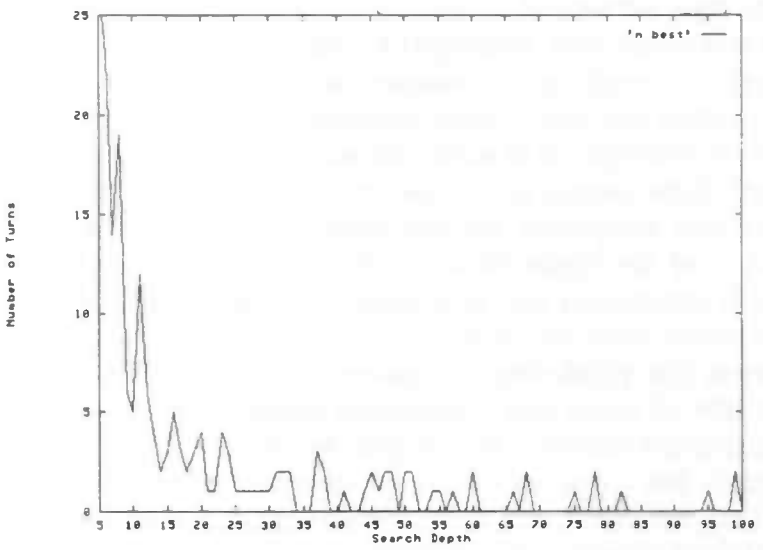


Figure 7.8b: The distribution of the search depth N, but on a different scale.

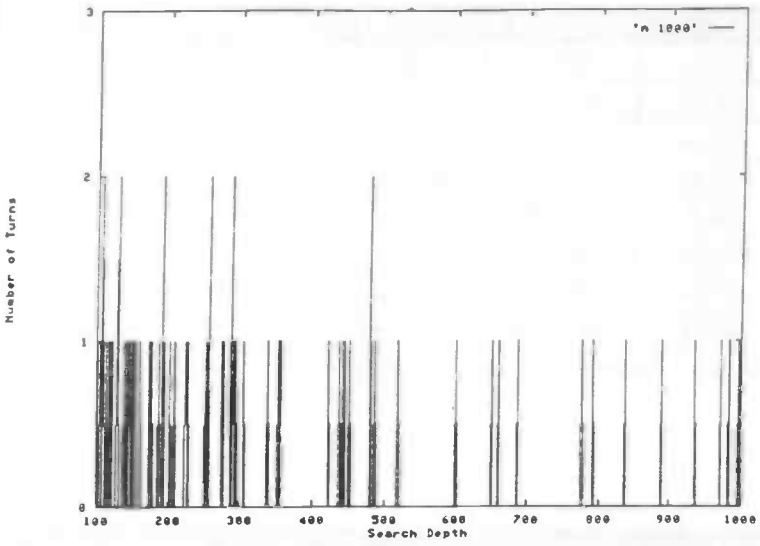


Figure 7.9: N-best search with $N = 1000$.

7.6.2 The Numbers

The error rates are shown in tables 7.1 and 7.2. The graph based results are given in both tables, for ease of inspection. No 95% confidence intervals are given, as they are similar to the ones we have seen in chapter 6. The results were established using a new parameter, the Graph Based Language Model factor (GLM), which is similar to the Concept Language Model factor we use for the bigrams (chapter 6). The established values were GLM=0.56 and RPF=0.49.

As is clear from the results, the graph based models perform worse than the bigram models, and only a little better than the unigrams (note that the N-best search did not have an averse effect on the unigrams). Again, we stress the necessity to take the numbers cautiously for reasons explained before. Still, the reasons for this rather poor performance are not clear. A first guess would be that the EPA did not succeed in finding the right (in)dependencies. However, inspection of the cliques of context I (Appendix VII) shows that all the concepts that seem to be important for context I are concentrated in one clique (the biggest). It contains <origin>, <destination>, <origin_and_destination>, <yes>, <no> and <bitte>. So this does not seem to be a problem. On the other hand, there also seem to be dependencies that are not at all relevant. E.g., <bitte> is not relevant, it is however in this clique, which means we have to estimate another extra parameter. This suggests that the assumption that we can determine probabilistic (in)dependencies for this probability model might not be correct, or that our clustering is not right. Another possible explanation is that estimations of such joint probabilities do not bring any advantage because of the relatively short sentences: 60% of the sentences contain only one concept, so the concept set probability will amount to the probability of one single concept, the unigram probability. This could be why the graph based models only perform slightly better than the unigrams: It estimates the unigram probability a little better (because it needs to be computed for the clique only), and when there are more than one concept in the sentence, the graph based models can profit from the fact that they can estimate the joint probability of these concepts better than the unigram models.

	Graph Based Model		Dependent Bigram		Difference	
	AER	CER	AER	CER	AER	CER
0	33.2%	20.2%	32.4%	18.8%	-2.4%	-6.9%
I	45.0%	22.4%	42.3%	18.9%	-6.0%	-15.3%
II	38.7%	28.1%	36.3%	21.1%	-6.2%	-24.9%
III	17.3%	11.9%	16.2%	9.5%	-6.4%	-20.0%
IV	4.1%	1.9%	4.9%	3.8%	+20%	+53%
V	7.8%	7.3%	4.3%	3.7%	-45%	-49%
Total	**36.00%**	**21.60%**	**33.95%**	**17.91%**	**-5.7%**	**-17%**

Table 7.1: Results of the graph based models compared to the bigrams, using N-best search.

	Graph Based Model		Dependent Unigram		Difference	
	AER	CER	AER	CER	AER	CER
0	33.2%	20.2%	32.7%	20.7%	-1.5%	+2.5%
I	45.0%	22.4%	46.0%	23.2%	+2.2%	+3.6%
II	38.7%	28.1%	37.6%	23.9%	+2.8%	-15%
III	17.3%	11.9%	19.9%	11.9%	+15%	0%
IV	4.1%	1.9%	6.5%	3.8%	+59%	100%
V	7.8%	7.3%	7.8%	5.8%	0%	-20.5%
Total	**36.00%**	**21.60%**	**36.39%**	**21.15%**	**1.1%**	**-2.1%**

Table 7.2: Results of the graph based models compared to the unigrams, using N-best search.

7.7 Combining the Graph Based and Bigram Models

Because the graph based models lack any concept order information, we would like to add this to them. The way we will do this, is to compute a bigram probability for the ordered sequence of concepts $C = c_1,...,c_k$, which is delivered by the N-best search, given the concept set $S_C = \{c_1,...,c_k\}$:

$$P(c_1,\ldots,c_k \mid \{c_1,\ldots,c_k\}) = \frac{\prod_{i=1}^{k+1} P(c_i \mid c_{i-1})}{\Phi_{c_1,\ldots,c_k}}, \qquad (7.12)$$

with $c_0 = c_{k+1}$ = sentence end / begin

Where $\Phi_{c_1,\ldots,c_k}$ is a normalisation factor. If we set $\Phi_{c_1,\ldots,c_k} = 1$ we do not normalise with respect to the concept set, an approach we will pursue first. After this, we will see what normalisation with respect to the concept set brings.

7.7.1 No Normalisation

With $\Phi_{c_1,\ldots,c_k} = 1$ we simply add the bigram score to the acoustic/derivation score and graph based model score. We do this by first finding the concept sequence, compute its set probability, then compute its bigram probability, and then we combine them with the acoustic/derivation probability.

The results show an increase in performance from 36.00% (graph based) to 34.77% (the combined model), a relative improvement of 3.4%. Though this seems quite a nice improvement, when we compare it to the bigram model alone (table 7.1), we see it performs worse: The bigram model has an AER of 33.95%. Here we can also see the problem with the optimisation algorithm. It did not find GLM = 0[3], though that would mean no effect from the graph based model, and only the bigram model would have effect, thus creating the configuration of table 7.1.

However, why does this combined model not perform better than the bigram model alone? The reason for this might be that because the EPA assigned too high probabilities for non-relevant combinations of concepts, the bigram rescoring can not alter this anymore. This can also be a reason why the bigram modelling is better in general: It only uses local constraints, as the graph based models use constraints on the sentence level, which might not always be correct or relevant.

7.7.2 Normalisation

The theoretic correct way to go, is to re-normalise the bigram probability for the concept sequence with respect to all possible sequences, given that set of concepts. That means we set:

$$\Phi_{c_1,\ldots,c_k} = \sum_{\pi \in S_k} \prod_{i=1}^{k+1} P(c_{\pi(i)} | c_{\pi(i-1)}) \tag{7.13}$$

This means, in words, we compute the bigram probabilities for every possible ordering of the concept set. This means we now have again $\sum P(c_1,\ldots c_k | \{c_1,\ldots c_k\}) = 1$.

Because the number of possible orderings increases very rapidly ($k!$), we created a fall-back procedure for $k > 7$. This means that when we have eight concepts or more, we no longer compute the normalised bigram probability in combination with the graph based model, but we only use the score as it is computed by the N-gram search. This means we influence the scores in a positive manner, but because the fall back procedure is used in only 1.5% of the cases, there is no noticeable effect.

The results of the re-normalisation are rather disappointing: The normalised version even performs worse than the not normalised version, viz. 35.61% (not normalised: 34.77%). The reason for the poor performance of the not-normalised bigram addition was explained in the previous section, the same argument is valid here. Because of wrongly determining the set of concepts, the bigram can not influence the understanding positively. This goes double for the normalised bigram, as it is normalised with respect to the (perhaps wrongly determined) set of concepts.

7.8 Conclusions

As we have seen, the graph based models do not perform well. Besides the problems with the non-optimal search, we can conclude that the estimation of the concept sets can not compete with the bigram modelling. This could be due to the relatively small sequence lengths, because of which a set estimation comes down to computing unigram probabilities. Another reason could be that the EPA did not succeed in finding the right (in)dependencies, although the cliques do seem to capture some of the relations pretty well, as the figures concerning the gain also suggest. This suggests that the assumption that we can determine rele-

[3] And, of course, the corresponding values for the other parameters.

vant probabilistic independencies might be wrong for this system, which may be due to system characteristics or to our clustering, but probably both.

We can therefore conclude that with respect to this particular application, and probably with respect to similar applications as well, that this method is not useful. This does not mean, however, this method cannot be used for systems where there are longer sequences involved, or where different, more relevant (in)dependencies exist, although this would have to be investigated first.

8. Establishing Different Contexts Automatically

8.1 Introduction

In order to check our 'expert' context definitions and to see whether it is possible to fully automate the context selection, we will look at ways for automatically defining contexts. We will use two different methods. The first one resembles the way we worked in chapters 5 and 6: We compute context dependent bigram models, only this time the contexts are not defined manually, but automatically by a clustering algorithm. The second method makes use of the graph based models we discussed in the previous chapter. We will use graph dependencies to estimate probabilities, only this time we will not use contexts in the sense of chapter 5; we will use the system state to estimate a conditional probability for a concept set, given that system state: $P(\{c_1,...,c_k\}|$ system state). We will keep talking about context dependent models, even though we are talking about conditional probabilities, because we can interpret $P(B|A)$ as the probability of B in a context A (p. 17, [17]). So, every system state defines a separate context.

8.2 Automatic Clustering

For clustering of the system states, we use an algorithm developed by Jochen Peters at Philips Research Aachen. It works as follows [18]. Let us define T_j as the texts which are grouped into clusters $Clus_i$ (a *text* being all turns belonging to a certain system state, see below). The vocabulary's different entries are denoted as v_k, and their counts within T_j are written as n_k^j, within $Clus_i$ as N_k^i, and within the whole corpus as N_k . To determine the best cluster to which a text is to be assigned, the *Mutual Information I* is used:

$$I = \sum_{k,i} P(v_k, Clus_i) \log\left(\frac{P(v_k, Clus_i)}{P(v_k) \cdot P(Clus_i)}\right) \tag{8.1}$$

Because I is uniquely defined for any cluster segmentation $\{Clus_i\}$, the algorithm can maximise this quantity (given a fixed number of clusters). Starting from some initial segmentation, each turn is shifted from its present cluster into that cluster which results in the highest increase of I. The algorithm terminates when single shifts would cause a decrease in the mutual information.

Using Mutual Information is also the approach for adaptation of probabilities that is chosen in [22], though there it is used on a word level and no defining of contexts is done.

8.2.1 Our Setup

We divide the training corpus according to the different system states, i.e. we put all concept sequences 'resulting' from a specific system state into one text, after concatenating the concepts on the sentence level: Each concept sequence is one 'word' now. This is possible, because there are only 560 different sequences in the training corpus. We thus established 242 (number of different system states) texts.

We performed clusterings[1] for four and for six clusters, the latter are listed in Appendix VIII. Because we have to anticipate concept sequences we have not seen in the training corpus, we have to create a fall-back procedure. Because this does not happen very often (0.6% of the time an unseen sequence is encountered in the test set), we use the simplest strategy possible, viz. we just use the first cluster.

We then establish language models according to the method described in chapter 6 (context dependent bigram modelling).

[1] Thanks to Jochen Peters for actually running the algorithm for us.

8.2.2 Results

Both the clustering using four clusters and the clustering using six clusters perform better than our manual clustering, the second one being the best. We have relative improvements of 8.4% and 9.7% respectively (tables 8.1a and 8.1b).

	AER	CER
Indep. Bigram	34.83%	17.28%
Auto Four	31.91%	15.84%
Improvement	**8.4%**	**8.3%**

Table 8.1a: Results of automatic clustering using four clusters.

	AER	CER
Indep. Bigram	34.83%	17.28%
Auto Six	31.46%	15.56%
Improvement	**9.7%**	**10,0%**

Table 8.1b: Results of automatic clustering using six clusters.

Cluster	No. of Turns	%	Av. No. of Concepts per Turn	No. of Attributes
0	4078	29	2.0	7486
1	2625	19	1.1	3176
2	3664	26	1.7	7587
3	2137	15	1.2	2577
4	699	5	1.3	2119
5	813	6	1.2	893
Total	**14016**	**100**	**1.6**	**23838**

Table 8.2: Distribution of the different clusters on the training corpus.

Cluster	No. of Turns	%	Av. No. of Concepts per Turn	No. of Attributes
0	604	31	1.9	1085
1	364	18	1.0	395
2	511	26	1.6	1085
3	285	14	1.1	331
4	99	5	1.5	310
5	123	6	1.1	119
Total	**1986**	**100**	**1.5**	**3325**

Table 8.3: Distribution of the different clusters on the test corpus.

	Test		Training	
Cluster	AER	CER	AER	CER
0	24.5%	4.4%	23.5%	3.5%
1	2.8%	1.5%	6.2%	3.2%
2	14.8%	4.5%	14.4%	4.7%
3	4.2%	1.1%	6.3%	2.0%
4	7.7%	4.4%	7.4%	4.4%
5	32.8%	6.2%	37.9%	11.5%

Table 8.4: Concept graph error rates for the different clusters.

	Independent Bigram			Automatic Six			Relative Improvement	
Cluster	AER	95% Int	CER	AER	95% Int	CER	AER	CER
0 (I)	45.5%	±3.0%	17.0%	42.3%	±3.0%	15.7%	7.0%	8.2%
1 (IV)	8.6%	±2.9%	8.9%	8.4%	±2.9%	8.4%	2.3%	5.6%
2 (II)	38.9%	±2.9%	22.3%	35.0%	±2.9%	19.8%	10.0%	11.2%
3 (V)	16.6%	±4.1%	9.2%	15.1%	±4.0%	8.9%	9.0%	3.3%
4 (III)	18.4%	±4.5%	10.8%	16.1%	±4.0%	9.5%	12.5%	12.0%
5 (I)	82.4%	±7.2%	35.0%	63.9%	±9.0%	28.3%	22.3%	19.1%

Table 8.5: Error rates for the automatically found clusters.

Because the clustering using six clusters seems to be the better one, we will now briefly discuss the clusters[2] of this six-cluster clustering. The characteristics of these clusters for the training set and the test set are given in tables 8.2 and 8.3, respectively, and in table 8.4 the concept graph error rates for the different clusters are listed (remember that the AER values are an upper bound, see section 4.4.2). Finally, table 8.5 shows the established error rates for these clusters. These, we will discuss next.

The best way to describe the newly found clusters, is to compare the manually found contexts with them, because the 'semantics' of those contexts is given in chapter 5 (and Appendix IV).

- Context II is completely contained in cluster 2;
- Context III is completely contained in cluster 4;
- Context IV is completely contained in cluster 1;
- Context V is completely contained in cluster 3 (Trivial: context V consists of only one system state);
- Context I is partly contained in cluster 0, where the "from where to where" situations are concerned, and partly in cluster 5, where the "Where to ... from A" and "from where to B" situations are concerned.

As can be seen in table 8.5, the error rates vary greatly over the different clusters. The error rates of Clusters 0 through 4 are to be expected (as compared to the manual contexts, cf. table 6.3A), but cluster 5 attracts the attention (and perhaps cluster 0 as well) because of its high error rate. One of the reason for this is that there are many station names to be found in cluster 5 (and cluster 0), which are often badly recognised by the speech recognition module (remember the high word graph error rates in section 4.4.1). However, as before, more research into the exact reasons for the error rates is necessary. This would reach beyond the goal of this chapters, which is to check whether it is possible to automate the context selection procedure and what the results of this are.

8.3 Using the EPA for Estimating Conditional Concept Set Probabilities

One way of looking at conditional probabilities of the form $P(B|A)$ is to say we define a probability $P(B)$ for a certain context A (p. 17, [17]). This is exactly what we will do next: Given a system state **S**, we estimate the concept set probability $P_S(\{c_1,...,c_k\})=P(\{c_1,...,c_k\}|\,\mathbf{S})$. For this we will use the EPA again, only this time the system state will also be encoded in the bit vectors we use to train the graph based models (see 7.5.1). As was mentioned before, we are limited to 32 positions in the bit vector for reasons of trainability. Up to now, we have

[2] We will keep talking about "clusters" instead of "contexts" to prevent confusion with the manually selected contexts.

used 22 of those, but now we also have to encode the system state. This means we have to decrease the number of concept clusters, and that we have to cluster the system states as well (see Appendix IX for the clustering). This way we end up with 26 bits (because of the clustering method we have chosen). We realise this is not the way to go if we really want a fully automatic procedure for estimating context dependent concept set probabilities, but we thought this method to be interesting enough to make some compromises.

For estimating the probabilities, we use an *ad-hoc* method which produces better results than the Add-One estimation method used in chapter 7. Our method consists of a relative frequency computation, which assigns a small constant probability $P(u)$ to every unseen event u, so that

$$\sum_{\text{All unseen events}} P(u) = \frac{1/N}{2} = \frac{1}{2N} \text{ , with } N \text{ the corpus size.} \qquad (8.2)$$

In words: The total probability of all unseen events is half the probability of seeing an event only once (the smallest possible frequency). Probabilistically speaking, this is of course no longer correct, as $\sum P > 1$ now, but the error is so small, we can ignore it.

8.3.1 Results

The results of the conditional model are shown in table 8.6. They show no significant difference compared to the graph based models of chapter 7. The problem is that we had to cluster everything due to trainability problems, so we cannot make any definitive statements as to the power of these conditional models. This would have to be investigated further, using greater data sets and an improved graph construction algorithm replacing EPA.

Graph Based Models	AER	CER
Conditional	35.34%	21.78%
Predefined Contexts	36.00%	21.60%

Table 8.6: Results of the conditional model.

8.4 Conclusions

As it seems, automatic clustering using Mutual Information is an easy, relatively simple way for selecting contexts, which can improve understanding considerably. This automatic clustering is to be preferred to the manual clustering, as it can be fully automated and it produces better results.. As the conditional graph based models are concerned, we cannot make any conclusive statements. Too many concessions have been necessary. It seems however, that for this kind of application, the method is not well suited; bigram modelling performs better and is easier to realise.

9. Previous Results

9.1 Introduction

We will now briefly discuss some previous results concerning context dependent concept language modelling in the Philips Automatic Inquiry System. These were established by Bruno Manzoni, within the framework of his master's thesis research [13] at Philips Research, the first six months of 1995. In this thesis, he describes his experiments with context dependent trigrams, and with, what he has called, a *context dependent separator* model. The first method is similar to our bigram approach, only Manzoni has used a different context selection and he used trigrams instead of bigrams. The second method also requires a predefined context selection, but it does not require the division of the training corpus into smaller corpora, in order to train the specific models. Instead, context dependent sentence separators are used. These are context dependent sentence beginnings and endings (cf. *word triggering*, [20]) This means we keep the same context division, but better estimations because the corpus stays intact.

Before we report the results of the experiments, we first need to state the differences between the setups that are used, because Manzoni's setup differs quite strongly from our setup. This makes it very difficult to compare results, but we will give it a try anyway.

9.2 Manzoni's Setup

Manzoni used 20,000 turns (we: 14,016) for his training set, and 10,000 turns (we: 1,986) for his test set. His word graphs came from different versions of the system, and he says about this that "some sentences are totally unsuitable in dialogue context" (p. 38, [13]). These data were of a different nature than ours, as they were obtained through the prototype system, while our data is from the on-line Swiss system. That means that Manzoni's material probably contains less 'real life' situations, and more testing situations, in which calls are performed by researchers. This may also be the reason that, even with the older system, he had a base attribute error rate of 34.84%, which is similar to ours. Manzoni did not report anything concerning the perplexity of the corpora he used.

His average sequence length was 2.5. We assume this was including sentence ending, which makes the length identical to ours. 27 concepts were used (Appendix X), which are roughly comparable to the 27 most frequently used concepts in our setup.

Finally, there are differences in the methods for determining contexts and in the selection of the contexts. These we will discuss in the next paragraph.

9.3 Context Dependent Trigram Modelling

The method we use for our context dependent bigram modelling is similar to the method Manzoni used for his trigrams, but not identical. First of all, he worked on the level of the individual sentences, and not on the dialogues. He used the textual format of the system prompt to determine the context. Secondly, he worked with a different context definition than we do. He has specified 11 contexts, which are listed in Appendix XI.

Using this method, Manzoni reports a relative improvement of 5.2% (table 9.1).

9.3.1 Reproducing the Results

The results of our bigram modelling are discussed extensively in chapter 6, so we will not go into them any further. Because Manzoni selected his contexts by hand, we will only use our manually selected contexts as well. We can also train trigrams with our own method, which perform about as well as the bigrams: A relative improvement of 5.6% (table 9.1). So, it does not seem that trigram modelling is better than bigram modelling, in this particular system. Which is by no means surprising, as we have an average sequence length of 1.5. This

means that with bigrams we already cover the entire sentence most of the time, so a trigram would basically do the same.

There is however one remark to be made. We were not able to use a 'real' trigram search, because the algorithm Manzoni developed for this, does not work in our system. Instead, we used what is called a *poor man's trigram* search. This is a routine that performs a bigram search, but uses the scores from a trigram. So the results need to be looked at cautiously.

	Manzoni (Research Corpus)		Drenth (Swiss Corpus)				
Context	Trigram		Trigram[1]			Bigram	
	AER	95% Int	AER	95% Int	CER	AER	CER
Independent	34.84%	±0.8%	34.29%	±1.6%	16.29%	34.83%	17.10%
Dependent	33.02%	±0.8%	32.36%	±1.6%	15.93%	32.54%	15.93%
Improvement	**5.2%**		**5.6%**		**2.2%**	**6.6%**	**7.8%**

Table 9.1: Results of context dependent tri- and bigram modelling.

9.4 Context Dependent Sentence Separators

Instead of dividing the corpus into smaller, context dependent corpora, one can also insert a context dependent sentence begin and end into each sentence of the corpus. Manzoni used the same separator for the end and begin, which, as he reports, produces some marginal errors. This could be changed by not using the same separator for sentence end and begin (we will use different end and begin symbols in our experiments). A small example will clarify this method, see example 9.1.

Context independent:
<origin> <destination> **<sentence_separator>**
Context dependent separators:
<Sent_begin_I> <origin> <destination> **<Sent_end_I>**

Example 9.1: Context dependent separators for context I.

Because the concept sequences are so short, using the separator model is basically the same as using the context dependent tri- or bigrams, because the separators 'have scope' over the entire sentence. For different applications, with long(er) sequences of items, this method does not seem to be very useful.

Manzoni reports a relative improvement of 8.8% using this context dependent trigram separator model (table 9.2). This means an improvement of error rate of 69% compared to the context dependent trigrams. He ascribes that to the better estimations he has established, because there was no longer the need to divide the corpus into smaller sub-corpora.

9.4.1 Reproducing the Results

We will create both a bigram and a trigram separator model. For the trigram, we have the same search problem as mentioned above, so we use a poor man's trigram search. Instead of one separator, we use two. The results of our investigations are shown in table 9.2.

Using the separator model, we establish a relative improvement of 6.0% and 4.8% on the bigram and trigram respectively. This means we do not find any significant differences between this method and the 'division' method. This might be because we do not seem to

[1] That is, *poor man's* trigram.

have trainability problems with the different contexts in the first place, as the interpolation of the language models (in 6.3.4) already suggested. But, as we have seen in the previous chapter, the differences in relative improvement may also be due to context selection. Manzoni had 11 contexts, which is considerably more than we do. One remarkable result is the small relative improvement of the CER with the bigram model. The reason for this is unclear, but we have to keep in mind that we optimise on the AERs, not on the CERs.

	Manzoni (Research Corpus)	**Drenth (Swiss Corpus)**			
	Trigram	Trigram		Bigram	
Context	AER	AER	CER	AER	CER
Independent	34.84%	34.29%	16.29%	34.83%	17.10%
Separator Model	31.79%	32.66%	15.90%	32.75%	16.71%
Improvement	**8.8%**	**4.8%**	**2.4%**	**6.0%**	**2.3%**

Table 9.2: Results for the context dependent separator models.

9.5 Conclusions

As was to be expected, there is not much to gain by using context dependent *trigram* models instead of bigrams. As we have mentioned already very often, this is probably due to the short concept sequences. The separator model does not seem to bring anything either. We were not able to reproduce Manzoni's results, which may be because of the differences in the two data sets and systems. Another reason could very well be the differences in context selection, because we have seen the effects of different selections in previous chapters.

The separator model seems to be of limited usefulness. As soon as the length of the concept sequences becomes greater, the context dependency effect will decrease. And because the difficulty to create separator models is about the same as creating independent models (as we did in chapters 6 and 8), the latter method is to be preferred.

10. Conclusions

This thesis was concerned with investigating the possibilities of using context information in speech understanding. We have used several different methods, as there are bigram and trigram modelling, concept set probability estimation, separator modelling and combinations of these models. We have also looked into different ways for modelling contexts, viz. manual clustering, automatic clustering, and using conditional models.

The results are very promising. We established a relative improvement of about 7% on the manual clustering, and about 10% using the automatically selected contexts. This is quite remarkable given the circumstances, as there are the high word graph error rates and concept graph error rates (chapter 4).

Especially interesting was that the automatic selection of contexts was better than the manual, 'expert' selection. Of course we have to keep the error margins in mind, but it still seems that this relatively easy method can bring considerable improvement. However, more research is necessary to determine the optimal cluster configuration, e.g. the number of clusters or the clustering criterion (in our case, mutual information).

We could not perform all experiments to our complete satisfaction. The Edge Pruning Algorithm was not suited for our specific system, which means that the obtained results are not conclusive and that more research will be necessary. Furthermore, we have seen that the statistical significance of the improvements is a difficult issue. No real quantitative statements are allowed, and to confirm the results, we need larger sets of data and perform the experiments again.

In conclusion, we can state that context dependent language modelling, and especially context dependent N-grams, seems to be a relatively easy way to establish an improved performance of the speech understanding component of the Philips System, and probably for similar systems as well.

Appendix I: The Concepts

The following 76 concepts are used in our speech understanding system. The station specific origin and destination names are *ad-hoc* measures for solving problematic understandings. Most of the concepts speak for themselves as to what their meaning is. It would take an in-depth explanation of the implemented system to explain the meaning of all the concepts, so we will leave it at this.

1. @ (*sentence separator*)
2. Wassen_station_dest
3. Wassen_station_orig
4. Villars_station_dest
5. Villars_station_orig
6. Uetikon_station_dest
7. Uetikon_station_orig
8. Pfaeffikon_station_dest
9. Pfaeffikon_station_orig
10. Rueti_station_dest
11. Rueti_station_orig
12. Oberriet_station_dest
13. Oberriet_station_orig
14. Muri_station_dest
15. Muri_station_orig
16. Moutier_station_dest
17. Moutier_station_orig
18. Leimbach_station_dest
19. Leimbach_station_orig
20. Langwies_station_dest
21. Langwies_station_orig
22. Kuessnacht_station_dest
23. Kuessnacht_station_orig
24. Hasle_station_dest
25. Hasle_station_orig
26. Erlenbach_station_dest
27. Erlenbach_station_orig
28. Dornach_station_dest
29. Dornach_station_orig
30. Corcelles_station_dest
31. Corcelles_station_orig
32. Chur_station_dest
33. Chur_station_orig
34. Buchs_station_dest
35. Buchs_station_orig
36. Au_station_dest
37. Au_station_orig
38. Altstaetten_station_dest
39. Altstaetten_station_orig
40. Affoltern_station_dest
41. Affoltern_station_orig
42. simple_date
43. negative_time
44. negative_date
45. simple_destination
46. negative_destination
47. simple_origin
48. negative_origin
49. meaningful_filler
50. alle_Verbindungen
51. Gegenrichtung
52. no
53. yes
54. date
55. time_and_date_sofort
56. time_and_date
57. train_time
58. day_time
59. complete_time
60. arrival_time_p
61. destination_and_arrival_time_p
62. lp_origin_and_destination
63. origin_and_destination
64. destination
65. origin
66. umsteigen
67. Zugbestimmung
68. bitte
69. kommen
70. wollen_Aussage
71. koennen_Frage
72. kommen_Frage
73. brauchen_Frage
74. wollen_Frage
75. wann_Frage
76. guten_Tag

Appendix II: Meaning of the Hexadecimal Numbers in the System State

There are eight different numbers possible. They signify what the system knows about a particular slot. There are 17 of these slots:

1. ankommen_mode: Not used in current system.
2. weekday: Not used in current system.
3. date: Speaks for itself.
4. start_time: ibid.
5. end_time: ibid.
6. exact_time_flag: Set when an exact time is given.
7. arrival_time_p: Set when the time given is an arrival time.
8. origin: Speaks for itself.
9. destination: ibid.
10. train_time: Signifies expressions as "the next", "the first", "the last" etc.
11. yes_no: If a confirmation or negation is found.
12. gegenrichtung: When the user also wants to know the connection the other way around.
13. alle_Verbindungen: When the user wants to know other connections.
14. dummy_origin: Used to temporarily save the "original" origin, so it can be used for "gegenrichtung" (english: return trip).
15. dummy_destination: ibid.
16. dummy_endstation: Also used internally.
17. last_start_time: The last possible connection.

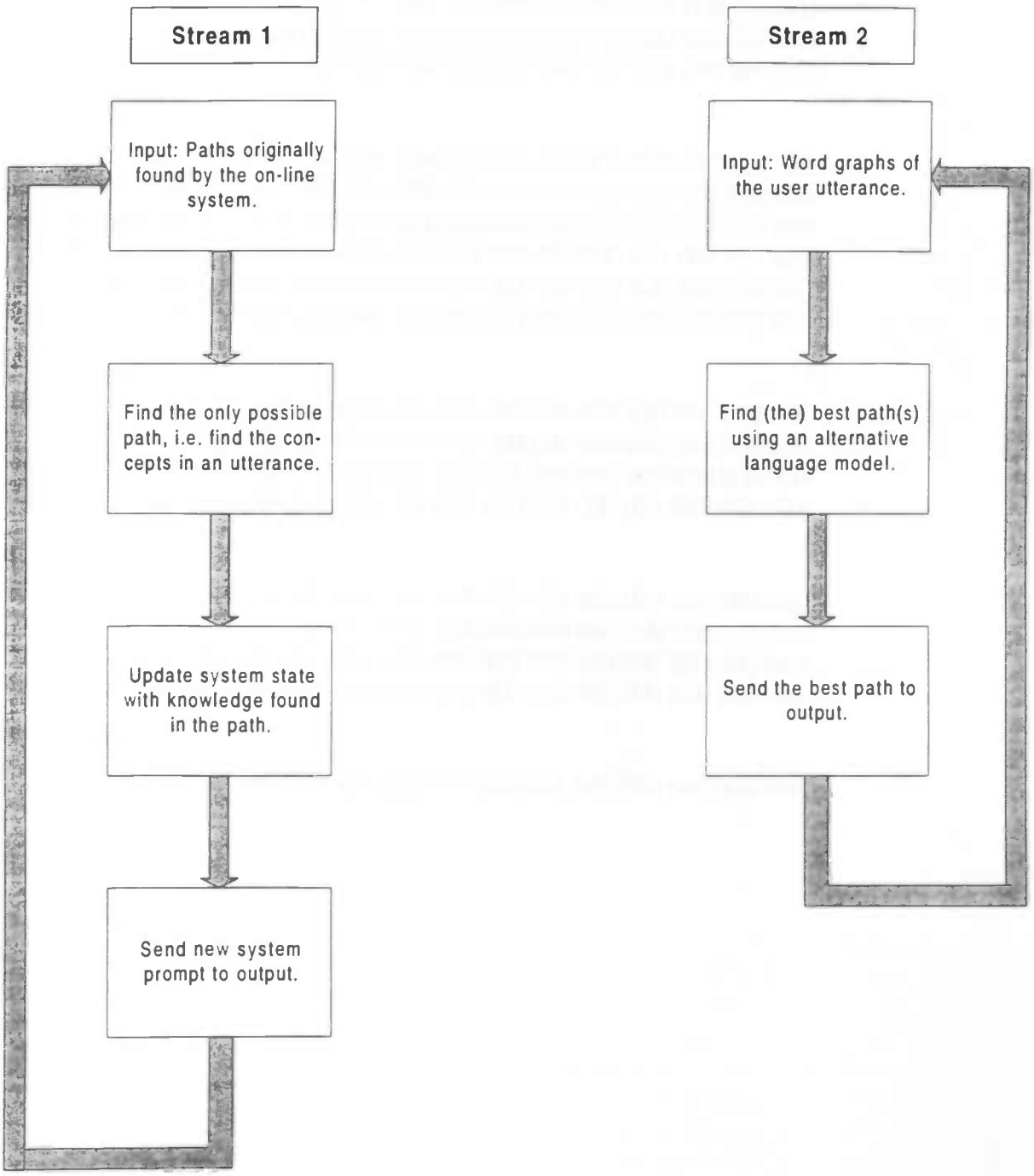
The numbers:

- 001 = SLOT_CLOSED: The slot cannot be altered anymore, its value is set.
- 002 = SLOT_VERIFIED: The slot has been verified and can be closed.
- 004 = VERIFICATION_QUESTION: The value of the concepts found will be verified: "So you want to travel from A to B?"
- 008 = ALWAYS_QUESTION: When a specification of an expression is wanted: "Please state the exact day you want to travel."
- 010 = COUPLES_QUESTION: Combines expressions into one expression, for example, combining the end time and the exact time flag.
- 020 = AMBIGUOUS_QUESTION: This results from a question which resolves ambiguous values: "Do you want to go from Aachen Main Station or Aachen West Station?"
- 040 = UNIQUE_QUESTION: Asks for a more specific statement from the user: "On which day of Easter do you want to travel?"
- 080 = EXTENSION_QUESTION: When the system wants a certain piece of information: "From where to where do you want to travel?"

Appendix III: Description of the Parallel Input Mechanism

The basic idea is as follows. We have gathered the output from the original system in concept graph format (see figure 3.1). So, when we feed the system this concept graph, it will produce exactly the same concept graph (because there is no other possibility) and exactly the same system state. The internal knowledge is thus exactly recreated.

Once a system state is established, we feed the system the original word graph, which is then processed with the alternative concept language model. This word graph is not used for updating the knowledge, as would be the normal procedure. So, at any time we have two concept graphs: One with only one possible path, and one with more possible paths, out of which the best is chosen with the alternative language model. A graphical representation will make things easier:



Appendix IV: System States of Manual Clustering

Our manual context selection consists of the following clustering of system states.

Context I:
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 001 001
000 000 000 000 000 000 000 080 080 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 080 080 000 001 000 000 000 000 000 000
000 000 000 004 004 001 000 080 080 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 004 080 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 004 080 000 001 000 000 000 000 000 000
000 000 000 000 000 000 000 080 004 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 001 044 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 001 044 000 001 000 000 000 000 000 000
000 000 000 000 000 000 000 003 044 000 000 000 000 000 000 000 000

Context II:
000 000 080 080 080 000 000 004 004 000 000 000 000 000 000 000 000
000 000 080 080 080 000 000 004 003 000 000 000 000 000 000 000 000
000 000 080 080 080 000 000 003 004 000 000 000 000 000 000 000 000
000 000 080 080 080 000 000 004 004 000 000 000 000 001 001 001 001
000 000 080 080 080 000 000 004 003 000 001 000 000 000 000 000 000
000 000 080 080 080 000 000 004 004 000 001 000 000 000 000 000 000

Context III:
000 000 004 080 080 000 004 003 003 000 000 000 000 000 000 000 000
000 000 004 080 080 000 000 003 003 000 000 000 000 000 000 000 000
000 000 004 080 080 000 000 003 003 000 001 000 000 000 000 000 000
000 000 003 080 080 000 000 004 004 000 000 000 000 001 001 001 001

Context IV:
000 000 003 004 004 001 003 003 003 000 000 000 000 000 000 000 000
000 000 004 004 004 001 000 003 003 004 000 000 000 000 000 000 000
000 000 004 004 004 001 000 004 004 000 000 000 000 000 000 000 000
000 000 003 004 004 001 000 003 003 000 000 000 000 000 000 000 000

Context V:
000 000 003 003 003 003 003 003 003 003 000 003 000 000 000 000 000

Appendix V: Perplexities of Interpolation Models

The perplexities of the different contexts, using the combined concept language models, for 20 iterations of the EM-algorithm. LM-File 0 is the context dependent model, LM-File 1 is the independent model.

Context 0
LM-File 0: Perplexity: 3.460
LM-File 1: Perplexity: 4.229
Iteration 0: 0.5000 0.5000 Perpl: 3.665
Iteration 1: 0.5489 0.4511 Perpl: 3.632
Iteration 2: 0.5939 0.4061 Perpl: 3.603
Iteration 3: 0.6345 0.3655 Perpl: 3.579
Iteration 4: 0.6709 0.3291 Perpl: 3.560
Iteration 5: 0.7031 0.2969 Perpl: 3.543
Iteration 6: 0.7316 0.2684 Perpl: 3.530
Iteration 7: 0.7567 0.2433 Perpl: 3.519
Iteration 8: 0.7787 0.2213 Perpl: 3.510
Iteration 9: 0.7981 0.2019 Perpl: 3.503
Iteration 10: 0.8151 0.1849 Perpl: 3.497
Iteration 11: 0.8301 0.1699 Perpl: 3.492
Iteration 12: 0.8434 0.1566 Perpl: 3.487
Iteration 13: 0.8551 0.1449 Perpl: 3.484
Iteration 14: 0.8656 0.1344 Perpl: 3.481
Iteration 15: 0.8749 0.1251 Perpl: 3.478
Iteration 16: 0.8832 0.1168 Perpl: 3.476
Iteration 17: 0.8907 0.1093 Perpl: 3.474
Iteration 18: 0.8974 0.1026 Perpl: 3.473
Iteration 19: 0.9035 0.0965 Perpl: 3.471
Iteration 20: 0.9090 0.0910 Perpl: 3.470

Context I
LM-File 0: Perplexity: 2.677
LM-File 1: Perplexity: 3.338
Iteration 0: 0.5000 0.5000 Perpl: 2.892
Iteration 1: 0.5520 0.4480 Perpl: 2.862
Iteration 2: 0.6005 0.3995 Perpl: 2.835
Iteration 3: 0.6450 0.3550 Perpl: 2.813
Iteration 4: 0.6854 0.3146 Perpl: 2.793
Iteration 5: 0.7216 0.2784 Perpl: 2.777
Iteration 6: 0.7538 0.2462 Perpl: 2.762
Iteration 7: 0.7824 0.2176 Perpl: 2.751
Iteration 8: 0.8076 0.1924 Perpl: 2.741
Iteration 9: 0.8298 0.1702 Perpl: 2.732
Iteration 10: 0.8493 0.1507 Perpl: 2.725
Iteration 11: 0.8665 0.1335 Perpl: 2.719
Iteration 12: 0.8816 0.1184 Perpl: 2.713
Iteration 13: 0.8949 0.1051 Perpl: 2.709
Iteration 14: 0.9066 0.0934 Perpl: 2.705
Iteration 15: 0.9169 0.0831 Perpl: 2.701
Iteration 16: 0.9260 0.0740 Perpl: 2.699
Iteration 17: 0.9341 0.0659 Perpl: 2.696
Iteration 18: 0.9412 0.0588 Perpl: 2.694
Iteration 19: 0.9475 0.0525 Perpl: 2.692
Iteration 20: 0.9531 0.0469 Perpl: 2.690

Context II
LM-File 0: Perplexity: 4.305
LM-File 1: Perplexity: 5.639
Iteration 0: 0.5000 0.5000 Perpl: 4.663
Iteration 1: 0.5617 0.4383 Perpl: 4.596
Iteration 2: 0.6168 0.3832 Perpl: 4.542
Iteration 3: 0.6653 0.3347 Perpl: 4.499
Iteration 4: 0.7073 0.2927 Perpl: 4.464
Iteration 5: 0.7435 0.2565 Perpl: 4.437
Iteration 6: 0.7745 0.2255 Perpl: 4.415
Iteration 7: 0.8012 0.1988 Perpl: 4.398
Iteration 8: 0.8241 0.1759 Perpl: 4.384
Iteration 9: 0.8438 0.1562 Perpl: 4.373
Iteration 10: 0.8608 0.1392 Perpl: 4.363
Iteration 11: 0.8756 0.1244 Perpl: 4.355
Iteration 12: 0.8884 0.1116 Perpl: 4.349
Iteration 13: 0.8996 0.1004 Perpl: 4.344
Iteration 14: 0.9094 0.0906 Perpl: 4.339
Iteration 15: 0.9181 0.0819 Perpl: 4.335
Iteration 16: 0.9257 0.0743 Perpl: 4.332
Iteration 17: 0.9325 0.0675 Perpl: 4.329
Iteration 18: 0.9386 0.0614 Perpl: 4.326
Iteration 19: 0.9439 0.0561 Perpl: 4.324
Iteration 20: 0.9488 0.0512 Perpl: 4.322

Context III
LM-File 0: Perplexity: 2.411
LM-File 1: Perplexity: 5.129
Iteration 0: 0.5000 0.5000 Perpl: 2.692
Iteration 1: 0.6423 0.3577 Perpl: 2.511
Iteration 2: 0.7381 0.2619 Perpl: 2.423
Iteration 3: 0.8004 0.1996 Perpl: 2.380
Iteration 4: 0.8410 0.1590 Perpl: 2.358
Iteration 5: 0.8678 0.1322 Perpl: 2.347
Iteration 6: 0.8857 0.1143 Perpl: 2.341
Iteration 7: 0.8980 0.1020 Perpl: 2.338
Iteration 8: 0.9065 0.0935 Perpl: 2.336
Iteration 9: 0.9125 0.0875 Perpl: 2.335
Iteration 10: 0.9167 0.0833 Perpl: 2.335
Iteration 11: 0.9197 0.0803 Perpl: 2.335

Iteration 12: 0.9219 0.0781 Perpl: 2.334
Iteration 13: 0.9235 0.0765 Perpl: 2.334
Iteration 14: 0.9246 0.0754 Perpl: 2.334
Iteration 15: 0.9254 0.0746 Perpl: 2.334
Iteration 16: 0.9260 0.0740 Perpl: 2.334
Iteration 17: 0.9265 0.0735 Perpl: 2.334
Iteration 18: 0.9268 0.0732 Perpl: 2.334
Iteration 19: 0.9270 0.0730 Perpl: 2.334
Iteration 20: 0.9272 0.0728 Perpl: 2.334

Context IV
LM-File 0: Perplexity: 1.816
LM-File 1: Perplexity: 3.348
Iteration 0: 0.5000 0.5000 Perpl: 2.191
Iteration 1: 0.6265 0.3735 Perpl: 2.066
Iteration 2: 0.7263 0.2737 Perpl: 1.985
Iteration 3: 0.8014 0.1986 Perpl: 1.932
Iteration 4: 0.8564 0.1436 Perpl: 1.897
Iteration 5: 0.8964 0.1036 Perpl: 1.873
Iteration 6: 0.9253 0.0747 Perpl: 1.856
Iteration 7: 0.9462 0.0538 Perpl: 1.845
Iteration 8: 0.9612 0.0388 Perpl: 1.836
Iteration 9: 0.9720 0.0280 Perpl: 1.831
Iteration 10: 0.9798 0.0202 Perpl: 1.827
Iteration 11: 0.9854 0.0146 Perpl: 1.824
Iteration 12: 0.9895 0.0105 Perpl: 1.822
Iteration 13: 0.9924 0.0076 Perpl: 1.820
Iteration 14: 0.9945 0.0055 Perpl: 1.819
Iteration 15: 0.9961 0.0039 Perpl: 1.818
Iteration 16: 0.9972 0.0028 Perpl: 1.818
Iteration 17: 0.9979 0.0021 Perpl: 1.817
Iteration 18: 0.9985 0.0015 Perpl: 1.817
Iteration 19: 0.9989 0.0011 Perpl: 1.817
Iteration 20: 0.9992 0.0008 Perpl: 1.817

Context V
LM-File 0: Perplexity: 1.529
LM-File 1: Perplexity: 3.303
Iteration 0: 0.5000 0.5000 Perpl: 1.849
Iteration 1: 0.6707 0.3293 Perpl: 1.666
Iteration 2: 0.7906 0.2094 Perpl: 1.569
Iteration 3: 0.8664 0.1336 Perpl: 1.519
Iteration 4: 0.9118 0.0882 Perpl: 1.495
Iteration 5: 0.9383 0.0617 Perpl: 1.483
Iteration 6: 0.9535 0.0465 Perpl: 1.478
Iteration 7: 0.9623 0.0377 Perpl: 1.475
Iteration 8: 0.9672 0.0328 Perpl: 1.475
Iteration 9: 0.9701 0.0299 Perpl: 1.474
Iteration 10: 0.9717 0.0283 Perpl: 1.474
Iteration 11: 0.9726 0.0274 Perpl: 1.474
Iteration 12: 0.9731 0.0269 Perpl: 1.474
Iteration 13: 0.9734 0.0266 Perpl: 1.474
Iteration 14: 0.9736 0.0264 Perpl: 1.474
Iteration 15: 0.9737 0.0263 Perpl: 1.474
Iteration 16: 0.9738 0.0262 Perpl: 1.474
Iteration 17: 0.9738 0.0262 Perpl: 1.474
Iteration 18: 0.9738 0.0262 Perpl: 1.474
Iteration 19: 0.9738 0.0262 Perpl: 1.474
Iteration 20: 0.9738 0.0262 Perpl: 1.474

Appendix VI: Clustering of Concepts

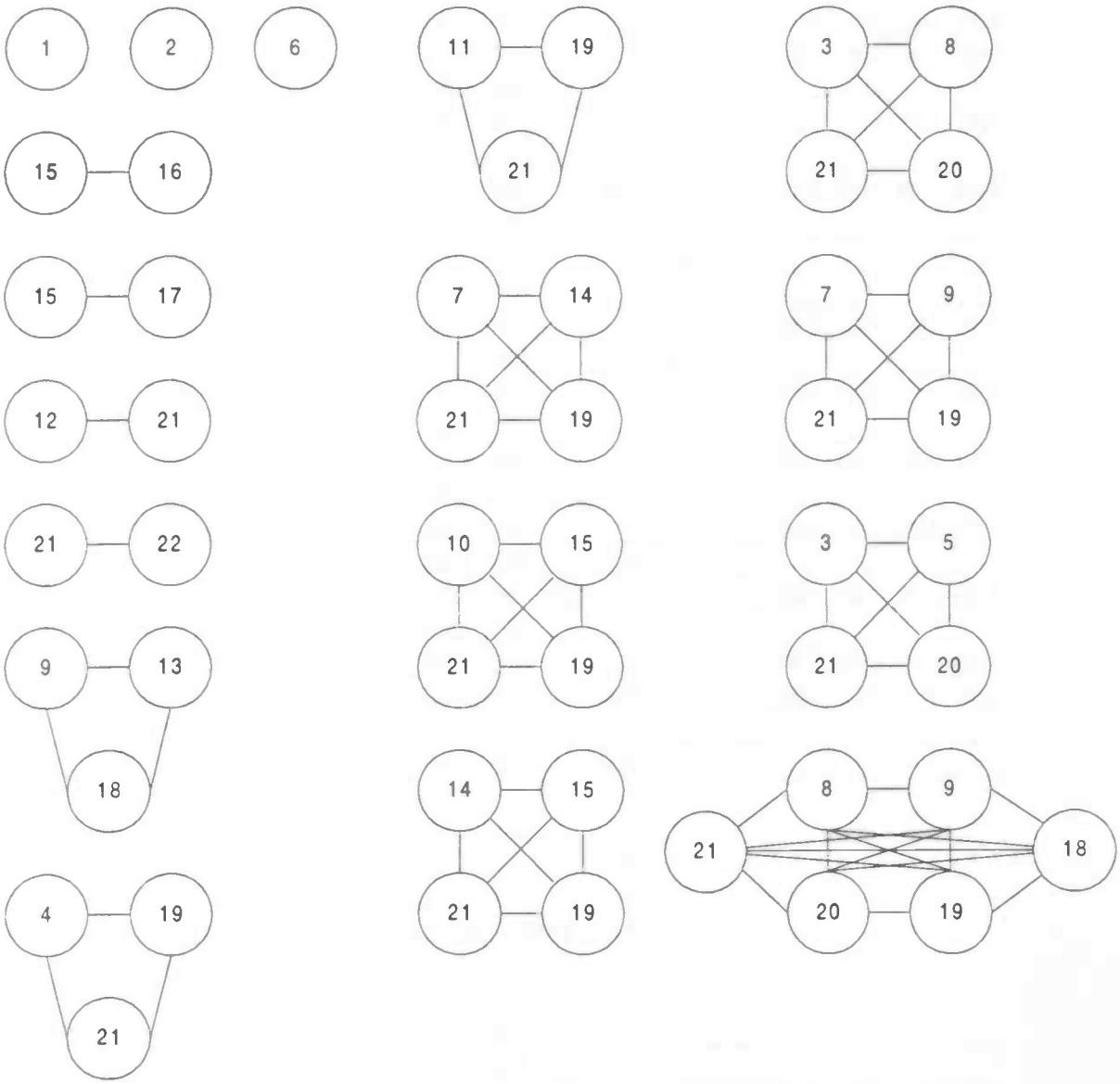
Due to computational limitations, we have to constrain ourselves to bit vectors of size 22. But we have 75 concepts (not counting sentence begin/end), so there is the need for some kind of clustering. We selected the concepts which were to be clustered manually, on the basis of whether they were relevant enough to keep or not. What follows is the clustering file, which we used as input for our clustering program. For readability, it is formatted into columns. All concepts preceded by '#' are clustered to the first previous concept without the '#'.

```
@negative_time
@negative_date
@negative_destination
@negative_origin
@meaningful_filler
@alle_Verbindungen
@Gegenrichtung
@no
@yes

@date
#@simple_date

@time_and_date_sofort
@time_and_date
@train_time
@day_time
@complete_time
@arrival_time_p
@destination_and_arrival_time_p

@origin_and_destination
#@lp_origin_and_destination

@destination
#@simple_destination
#@Wassen_station_dest
#@Villars_station_dest
#@Uetikon_station_dest
#@Pfaeffikon_station_dest
#@Rueti_station_dest
#@Oberriet_station_dest
#@Muri_station_dest
#@Moutier_station_dest
#@Leimbach_station_dest
#@Langwies_station_dest
#@Kuessnacht_station_dest
#@Hasle_station_dest
#@Erlenbach_station_dest
#@Dornach_station_dest
#@Corcelles_station_dest
#@Chur_station_dest
#@Buchs_station_dest
#@Au_station_dest
#@Altstaetten_station_dest
#@Affoltern_station_dest

@origin
#@simple_origin
#@Wassen_station_orig
#@Villars_station_orig
#@Uetikon_station_orig
#@Pfaeffikon_station_orig
#@Rueti_station_orig
#@Oberriet_station_orig
#@Muri_station_orig
#@Moutier_station_orig
#@Leimbach_station_orig
#@Langwies_station_orig
#@Kuessnacht_station_orig
#@Hasle_station_orig
#@Erlenbach_station_orig
#@Dornach_station_orig
#@Corcelles_station_orig
#@Chur_station_orig
#@Buchs_station_orig
#@Au_station_orig
#@Altstaetten_station_orig
#@Affoltern_station_orig

@bitte
#@kommen
#@wollen_Aussage
#@koennen_Frage
#@kommen_Frage
#@brauchen_Frage
#@wollen_Frage
#@wann_Frage
#@guten_Tag
#@umsteigen

@Zugbestimmung
```

Appendix VII: Graphic Representation of the Cliques found in Context I

The following is a graphic representation of the cliques, as they are found by the EPA for context I. It is interesting to notice that the concepts which seem to be relevant for this context, are also grouped together in the largest clique. Context I contains 189 parameters.

1. @negative_time
2. @negative_date
3. @negative_destination
4. @negative_origin
5. @meaningful_filler
6. @alle_Verbindungen
7. @Gegenrichtung
8. @no
9. @yes
10. @date
11. @time_and_date_sofort
12. @time_and_date
13. @train_time
14. @day_time
15. @complete_time
16. @arrival_time_p
17. @destination_and_arrival_time_p
18. @origin_and_destination
19. @destination
20. @origin
21. @bitte
22. @Zugbestimmung



Appendix VIII: Automatically Found Clusters

```
*==========================================================================
*
*  Gewuenschte Clusterzahl: 6 (= number of clusters)
*  Erreichter Wert der 'mutual information' I(X; Y): 1.03384
*  (= Established value of 'mutual information')
*==========================================================================

*---------------------- Cluster_0_ ----------------------  *
*                 26_Texte (= Texts)                      *
*                4058_Worte (= Words)                     *
*  Anteil an der 'mutual information':_0.279403           *
*---------------------------------------------------------  *

_____________________________040_____________________________
_____________________________040_____________001_001_001_001
_____________________________001_040_____001_____001_001_001_001
_____________________________003_080_____________001_001_001_001
_____________________________040_003_____________001_001_001_001
_____________________________080_080_____________________________
_____________________________080_080_____________001_001_001_001
_____________________________080_080_____001_____________________
_____________________________080_080_____001_____001_001_001_001
_____________________001_044_001_____________________________
_____________________004_080_080_____001_____________________
_____________004_004_001_____080_080_____________________________
_____________004_004_001_____080_080_____________001_001_001_001
_____________004_004_001_____080_080_____001_____________________
_____003_003_003_001_____001_040_003_____________001_001_001_001
_____004_____________004_080_____________________________
_____004_____________080_004_____________001_001_001_001
_____004_____________080_080_____________________________
_____004_____________080_080_____________001_001_001_001
_____004_____________080_080_____001_____________________
_____004_003_003_001_____080_080_003_____________001_001_001_001
_____004_004_004_001_____080_080_____________________________
_____004_004_004_001_____080_080_____________001_001_001_001
_____004_004_004_001_____080_080_____001_____001_001_001_001
_____004_004_004_001_____080_080_004_____________________________
_____00C_001_001_001_____003_003_____001_____________________

*---------------------- Cluster 1_ ----------------------  *
*                 48_Texte (= Texts)                      *
*                2625_Worte (= Words)                     *
*  Anteil an der 'mutual information':_0.204884           *
*---------------------------------------------------------  *

_____________________________022_____________________________
_____________________________024_____________________________
_____________________________024_____________001_001_001_001
```

_____024_____001_____
_____024_001_____001_001_001_001
_____024_003_____001_001_001_001
_____001_001_001_____001_024_____
_____001_001_001_001_____024_001_____
_____003_003_003_001_____004_003_003_____001_001_001_001
_____003_003_003_001_____004_004_____
_____003_003_003_003_003_003_003_003_003_003_____001_001
_____003_003_003_003_003_003_003_003_003_003_____001_001_001_001
_____003_004_004_001_____003_003_____
_____003_004_004_001_____003_003_____001_001_001_001
_____003_004_004_001_____003_003_____001_____
_____003_004_004_001_____003_003_____001_____001_001_001_001
_____003_004_004_001_____003_003_004_____
_____003_004_004_001_003_003_003_____
_____003_004_004_001_004_003_003_____
_____003_004_004_001_004_003_003_____001_001_001_001
_____003_004_004_001_004_003_003_____001_____
_____004_003_003_001_____003_003_____
_____004_003_003_001_____003_003_____001_001_001_001
_____004_003_003_001_____003_003_____001_____
_____004_003_003_001_____003_003_003_____
_____004_003_003_001_____003_003_004_001_____
_____004_003_003_001_003_003_003_____
_____004_003_003_001_003_003_003_____001_001_001_001
_____004_003_003_001_004_003_003_____
_____004_004_004_001_____003_003_____
_____004_004_004_001_____003_003_____001_001_001_001
_____004_004_004_001_____003_003_____001_____
_____004_004_004_001_____003_003_____001_____001_001_001_001
_____004_004_004_001_____003_003_004_____
_____004_004_004_001_____003_003_004_____001_001_001_001
_____004_004_004_001_____003_003_004_001_____
_____004_004_004_001_____003_004_____001_____
_____004_004_004_001_____004_004_____
_____004_004_004_001_____004_004_____001_____
_____004_004_004_001_____004_004_____001_____001_001_001_001
_____004_004_004_001_004_003_003_____
_____004_004_004_001_004_003_003_____001_____
_____004_004_004_001_004_004_004_____
_____024_____003_003_____
_____024_001_001_001_____003_003_____001_001_001_001
_____024_003_003_001_____003_003_____
_____024_003_003_001_____003_003_____001_001_001_001
_____024_003_003_001_____003_003_____001_____

```
*---------- Cluster 2_ ----------  *
*          47_Texte (= Texts)      *
*          3684_Worte (= Words)    *
* Anteil an der 'mutual information':_0.203582 *
*----------------------------------*

_____022_____001_____
_____003_080_080_____003_003_____001_001_001_001
_____004_080_080_____003_003_____001_____001_001_001_001
_____004_080_080_____003_003_003_____
_____008_____003_003_____
_____008_____003_003_____001_001_001_001
_____00C_____001_001_____
_____00C_____001_001_____001_____
_____00C_____003_003_____
_____00C_001_001_001_____003_003_____
_____00C_003_003_001_____003_003_____
_____00C_003_003_001_____003_003_____001_001_001_001
_____00C_003_003_001_____003_003_____001_____001_001_001_001
_____024_001_001_001_____003_003_____
_____044_____003_003_____
_____080_003_003_001_____003_004_____
_____080_003_003_001_____004_004_____
_____080_003_003_001_____004_004_____001_001_001_001
_____080_003_003_001_003_004_003_____001_001_001_001
_____080_004_004_001_____004_001_____001_001_001_001
_____080_004_004_001_____003_003_____
_____080_004_004_001_____003_003_____001_001_001_001
_____080_004_004_001_____003_003_____001_____
_____080_004_004_001_____003_003_004_____
_____080_004_004_001_____004_004_____
_____080_004_004_001_____004_004_____001_001_001_001
_____080_004_004_001_____004_004_____001_____
_____080_004_004_001_004_003_003_____
_____080_004_004_001_004_003_003_____001_001_001_001
_____080_004_004_001_004_003_003_____001_____
_____080_004_004_001_004_004_004_____
_____080_080_080_____003_003_____
_____080_080_080_____003_004_____
_____080_080_080_____003_004_____001_001_001_001
_____080_080_080_____003_004_____001_____
_____080_080_080_____003_004_____001_____001_001_001_001
_____080_080_080_____004_003_____
_____080_080_080_____004_003_____001_001_001_001
_____080_080_080_____004_003_____001_____
_____080_080_080_____004_004_____
_____080_080_080_____004_004_____001_001_001_001
_____080_080_080_____004_004_____001_____
_____080_080_080_____004_004_____001_____001_001_001_001
_____080_080_080_____004_004_____001_____
_____080_080_080_____004_004_____001_____001_001_001_001
```

_____080_080_080_____004_003_003_____
_____080_080_080_____004_004_004_____

*---------- Cluster 3_ ---------- *
* 25_Texte (= Texts) *
* 2137_Worte (= Words) *
* Anteil an der 'mutual information':_0.14603 *
*---------- *

_____001_001
_____001_001_001_001
_____001_024_____
_____001_024_____001_001_001_001
_____001_024_____001_____
_____024_____
_____024_001_____
_____024_001_____001_____
_____024_003_____
_____080_080_____001_____001_001_001_001
_____001_001_001_001_____001_024_____
_____001_001_001_001_____001_024_001_001_____001_001_001_001
_____003_003_003_____003_003_003_003_____003_____
_____003_003_003_001_____003_003_____
_____003_003_003_001_____003_003_____001_001_001_001
_____003_003_003_001_____003_003_____001_____001_001_001_001
_____003_003_003_001_____004_004_____001_001_001_001
_____003_003_003_001_003_003_003_____
_____003_003_003_001_003_003_003_____001_____
_____003_003_003_003_003_003_003_003_____003_____
_____003_003_003_003_003_003_003_003_____003_____001_001_001_001
_____004_004_004_001_____004_004_____001_001_001_001
_____004_004_004_001_____004_004_004_____
_____024_____001_003_003_____
_____080_004_004_001_____004_____001_001_001_001

*---------- Cluster 4_ ---------- *
* 28_Texte (= Texts) *
* 699_Worte (= Words) *
* Anteil an der 'mutual information':_0.100838 *
*---------- *

_____001_044_____001_001_____
_____024_____001_____
_____080_003_____001_____
_____080_004_____001_____
_____004_080_004_____001_____
_____003_003_001_____003_044_____
_____001_____044_001_____001_____
_____003_003_003_001_____003_003_____001_____
_____003_003_003_001_____044_001_003_____
_____003_080_080_____003_003_____001_____

_____003_080_080_____004_003_____
_____003_080_080_____004_004_____
_____003_080_080_____004_004_____001_001_001_001
_____004_____004_080_____001_001_001_001
_____004_080_080_____003_003_____
_____004_080_080_____003_003_____001_001_001_001
_____004_080_080_____003_003_____001_____
_____004_080_080_____004_004_____
_____004_080_080_____004_004_____001_____
_____004_080_080_____003_003_003_____001_____
_____004_080_080_____004_003_003_____
_____004_080_080_____004_003_003_____001_____
_____080_004_004_001_____003_003_____001_____001_001_001_001
_____080_004_004_001_____004_004_____001_____001_001_001_001
_____080_080_080_____003_003_____001_____
_____080_080_080_____003_003_____001_____001_001_001_001
_____080_080_080_____004_003_____001_____001_001_001_001
_____080_080_080_____004_004_____001_001_____001_001_001_001

*---------- Cluster 5_ ---------- *
* 68_Texte (= Texts) *
* 813_Worte (= Words) *
* Anteil an der 'mutual information':_0.0991008 *
*---------- *

_____040_____001_____001_001_001_001
_____042_____001_____
_____044_____
_____044_____001_001_001_001
_____001_040_____
_____001_040_____001_001_001_001
_____001_040_____001_____
_____001_044_____
_____001_044_____001_001_001_001
_____001_044_____001_____
_____001_044_____001_____001_001_001_001
_____003_040_____
_____003_044_____
_____003_080_____
_____003_080_____001_____
_____004_080_____
_____004_080_____001_001_001_001
_____004_080_____001_____
_____004_080_____001_____001_001_001_001
_____040_____
_____040_001_____
_____040_001_____001_001_001_001
_____040_001_____001_____
_____044_____
_____044_001_____
_____044_001_____001_001_001_001

044_001_001
044_001_001
044_001_001_001_001_001_001_001
044_003
080_003
080_003_001
080_003_001_001_001_001_001
080_004
080_004_001_001_001_001
080_004_001
080_004_001_001_001_001_001
080_080_001_001
001_001_001_001_044
001_001_001_001_044_001
001_001_001_044
001_001_001_044_001
001_001_001_044_001_001
001_001_001_001_001_044
003_003_001_044
003_003_001_044_001
003_003_001_080_004
004_004_001_004_080
004_004_001_004_080_001_001_001_001
004_004_001_004_080_001
004_004_001_080_004
004_004_001_080_004_001_001_001_001
004_004_001_080_004_001_001_001_001
004_004_001_080_080_001
004_004_001_004_080_004
004_004_001_004_080_004_001_001_001_001
001_001_040
001_044
001_044_001
001_044_001_001_001_001_001
001_044_001_001_001_001_001_001
001_001_001_001_001_044
001_001_001_001_001_044_001_001_001_001
001_001_001_001_044_001
003_003_003_001_001_024_003_001_001_001_001
004_080_004
004_004_004_001_080_004_004_001_001_001_001
080_004_004_001_004_003

Appendix IX: Clustering of Concepts and System States for the Conditional Models

For the conditional models, we need to use a different clustering, because we need to represent everything with maximally 32 bits.

@negative_destination

@negative_origin
#@wann_Frage

@no
@yes

@date
#@simple_date

@time_and_date
#@time_and_date_sofort
#@train_time
#@negative_date
#@negative_time

@day_time
@complete_time

@arrival_time_p
#@destination_and_arrival_time_p

@origin_and_destination
@lp_origin_and_destination

@destination
#@simple_destination
#@Wassen_station_dest
#@Villars_station_dest
#@Uetikon_station_dest
#@Pfaeffikon_station_dest
#@Rueti_station_dest
#@Oberriet_station_dest
#@Muri_station_dest
#@Moutier_station_dest
#@Leimbach_station_dest
#@Langwies_station_dest
#@Kuessnacht_station_dest
#@Hasle_station_dest
#@Erlenbach_station_dest
#@Dornach_station_dest
#@Corcelles_station_dest
#@Chur_station_dest
#@Buchs_station_dest
#@Au_station_dest
#@Altstaetten_station_dest
#@Affoltern_station_dest

@origin
#@simple_origin
#@Wassen_station_orig
#@Villars_station_orig
#@Uetikon_station_orig
#@Pfaeffikon_station_orig
#@Rueti_station_orig
#@Oberriet_station_orig
#@Muri_station_orig
#@Moutier_station_orig
#@Leimbach_station_orig
#@Langwies_station_orig
#@Kuessnacht_station_orig
#@Hasle_station_orig
#@Erlenbach_station_orig
#@Dornach_station_orig
#@Corcelles_station_orig
#@Chur_station_orig
#@Buchs_station_orig
#@Au_station_orig
#@Altstaetten_station_orig
#@Affoltern_station_orig
#@Gegenrichtung
#@alle_Verbindungen

@guten_Tag
#@meaningful_filler
#@wollen_Frage
#@bitte
#@wollen_Aussage
#@koennen_Frage
#@kommen_Frage
#@brauchen_Frage
#@Zugbestimmung
#@umsteigen
#@kommen

; Slotvalues
; We only use three slot values: 003, 004 and 080. We model these with two bits, so we have one bitcombination left for all the other possible values of the slots. We have to do this, because we want to model six slots. This means we need twelf bits to represent the values of these slots. If we were to use three bits per value, we would be able to model the system state more exactly, but it would mean eighteen bits in total, and we already use fourteen bits to model the clusters, which makes a total of 32 bits, which in theory should work, but in practice we get trainability problems.

; File for determining the relevant slotvalues and slots
; Please note: order is important.
003
004
080
#WHICH_SLOTS
; These are the slotnumbers that are relevant.
3
4
5
8
9
11

Appendix X: Manzoni's Concepts

The following concepts were used by Manzoni (p. 36, [13]).

meaningful_filler
no
yes
bitte
kommen
umsteigen
Zugbestimmung
wollen_Aussage
koennen_Frage
wollen_Frage
kommen_Frage
brauchen_Frage
guten_Tag
wann_Frage
date
time_and_date_sofort
time_and_date
train_time
day_time
complete_time
arrival_time_p
lp_origin_and_destination
destination_and_arrival_time_p
destination
origin_and_destination
origin

Appendix XI: Manzoni's Contexts

Manzoni classified the system prompts according to the following division (p. 52, [12]). The translation of the German words are ours.

General time (wann = *when*)
Date (an welchem Tag = *what day*)
Day time (um wieviel Uhr = *at what time*)
Origin → Destination (von wo nach wo = *from where to where*)
Destination (wohin = *to where*)
Origin (von wo = *from where*)
Yes (final verification)
Yes → General time (andere Zeit probieren? = *try a different time?*)
Yes → General (eine andere Verbindung? = *a different connection?*)
General (was kann ich für Sie tun? = *how may I help you?*)
Repeat (können Sie wiederholen = *could you repeat?*)

"The arrow → means that the question asks for different information. The information asked for has a certain order." Manzoni probably means that the answer can have more than one concept, and that these have a certain order. For example, "Would you like another connection?" can have as answer "Yes. I would also like to go to.....". This is modelled explicitly by Manzoni by the "Yes → General " context.

References

[1] Aust, H. (1996), *Dialogue Modelling*, The Fourth European Summer School on Language and Speech Communication -- Dialogue Systems 1996, Budapest, Hungary

[2] Aust, H., Oerder, M. (1994), *Database Query Generation from Spoken Sentences*, IEEE Workshop on Interactive Voice Technology for Telecommunications Applications, pp. 141-144, Kyoto, Japan

[3] Aust, H., Oerder, M., Seide, F. and Steinbiss, V. (1995), *A Spoken Language Inquiry System for Automatic Train Timetable Information*, Speech Communication 17, pp. 249 - 262

[4] Baggia, P. (1996), *Evaluation of Spoken Dialogue Systems*, The Fourth European Summer School on Language and Speech Communication -- Dialogue Systems, Budapest, Hungary

[5] Bahl, L., Jelinek, F., and Mercer, R. (1983), *A Maximum Likelihood Approach to Continuous Speech Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 5, no. 2

[6] Bouma, G., Koeling, R., Nederhof, M.-J., and van Noord, G. (1996), *Grammatical Analysis in a Spoken Dialogue System*, Language and Cognition 5, Groningen

[7] Dempster, A.P., Laird, N.M., and Rubin, D.B. (1977), *Maximum Likelihood from Incomplete Data via the EM Algorithm*, Journal of the Royal Statistical Society 39, no. 1, pp. 1-38

[8] Fu, K.S. (1982), *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs

[9] Hestenes, M.R., Stiefel, E. (1952), *Methods of Conjugate Gradients for Solving Linear Systems*, Nat. Bur. Standards, J. of Res. 49, 409-436

[10] Katz, S. (1987), *Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer*, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 35, no. 3

[11] Lucke, H. (1995), *Graph Theoretic Tools for Probabilistic Reasoning*, Internal forum Philips Research

[12] Lucke, H. (1996), *The Edge Pruning Algorithm for Finding Decomposable Stochastic Models*, to be published in IEEE transactions on PAMI

[13] Manzoni, B. (1995), *The Philips System: Implementation of N-gram, A study on N-gram language models, A study on Linguistic Context Dependencies*, master's thesis, Eurecom '95

[14] Ney, H. (1984), *The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition*, IEEE Transactions on ASSP, Vol. 32, no. 2

[15] Ney, H., Essen, U., and Kneser, R. (1994), *On Structuring Probabilistic Dependences in Stochastic Language Modelling*, Computer Speech and Language 8, pp. 1 - 38

[16] Partee, B., Ter Meulen, A., Wall, R. (1993), *Mathematical Methods in Linguistics*, Kluwer Academic Publishers, Dordrecht

[17] Pearl, J. (1988), *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, 2nd printing, Morgan Kaufmann Publishers, Inc.

[18] Peters, J. (1996), *Document Clustering for the Improvement of Language Models*, ITG-Fachbericht 139, Sept. '96

[19] Rabiner, L. and Juang, B-H. (1993), *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs

[20] Rosenfeld, R. (1994), *Adaptive Statistical Language Modelling: A Maximum Entropy Approach*, Ph.D. Thesis, Carnegie Mellon University

[21] Sachs, L. (1984), *Angewandte Statistik: Anwendung Statistischer Methoden*, 6th edition, Springer-Verlag Berlin

[22] Sarukkai, R., and Ballard, D. (1996), *Improved Spontaneous Dialogue Recognition using Dialogue and Utterance Triggers by Adaptive Probability Boosting*, International Conference on Spoken Language Processing 1996, Philadelphia, USA, pp. 208 - 211

[23] Steinbiss, V., Ney, H., Aubert, X., Besling, S., Dugast, C., Essen, U., Geller, D., Haeb-Umbach, R., Kneser, R., Meier, H.-G., Oerder, M. and Tran, B.-H. (1995), *The Philips Research System For Continuous-Speech Recognition*, Philips Journal of Research, 49, pp. 317 - 352

[24] Tran, B.-H., Seide, F., and Steinbiss, V. (1996), *A Word Graph Based N-Best Search in Continuous Speech Recognition*, ICSLP 1996, Philadelphia, USA, pp. 2127-2130

Acknowledgements

This thesis was written in the period from June until December 1996, at Philips Research in Aachen, Germany. It is the final requirement for my master's degree of Cognitive Science and Engineering, at the University of Groningen, the Netherlands. This work has come into being with the help of many people, who I would like to thank here. First and most important, there is Bernhard Rüber of Philips Research, under whose day-to-day supervision I have worked. Without his valuable suggestions, this work would not have reached the level it has now. Then there is Tjeerd Andringa, my supervisor at Groningen University, who has made important remarks concerning the significance of the error rates I established.

Furthermore, I would like to thank Jochen Peters, for actually performing the automatic clustering for me; Frank Seide, for computing the word graph error rates of the different periods; Andreas Kellner, for help concerning the implementation of SUSI; Reinhard Kneser, for valuable suggestions with regard to the graph error rates; Enrique Marti del Omo, at Philips Dialogue Systems, Aachen, who helped me with the gathering of the data; and finally, I would like to thank Volker Steinbiss, head of the man-machine interfaces department of Philips, for letting me do my research in his group.