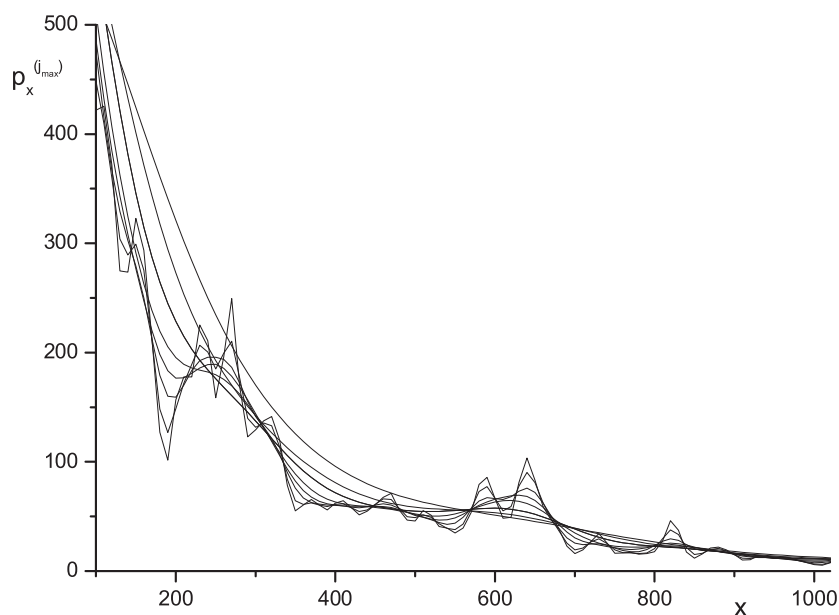


DRESDEN UNIVERSITY of TECHNOLOGY
DEPARTMENT of ELECTRICAL ENGINEERING
Laboratory of Acoustics and Speech Communication

Study Work
of
David Sündermann

Design and Development of General Symbol Statistics



author:	David Sündermann
date and place of birth:	August the 8 th , 1977 in Magdeburg
care:	Dipl.-Ing. Matthias Wolf
responsible university teacher:	Prof. Dr.-Ing. habil. Rüdiger Hoffmann
date of handing in:	March the 15 th , 2001

Contents

1	Foundations	1
1.1	MARKOV Chains	1
1.2	Implementation Strategies of Symbol Processing	3
1.2.1	Direct Data Access	3
1.2.2	Tree Structure	4
1.2.3	Sparse Tensors	8
1.2.4	An Example	12
1.2.5	Quantization and Domain Restriction	12
2	Fields of Application	15
2.1	Language Model	15
2.2	Musical Applications	18
2.3	Synthesis of Symbol Sequences	20
2.3.1	Synthesis on the Basis of Occurrence Probabilities	20
2.3.2	N-Gram and Multigram Synthesis	24
3	Experiments	29
3.1	The Computing Time Behaviour of the Compact Storage Algorithm	30
3.2	Determination of some Parameters of the <i>Verbmobil</i> Material . . .	33
3.2.1	Analysis	33
3.2.2	Parameter Extraction	34
3.3	Comparison of the Frequency Distributions of Notes in Musical Pieces of Different Epochs	35
3.4	Synthesis Experiments	37
3.4.1	Verification of the ZIPF Condition	37
3.4.2	Determination of the Average Number of Additions	38
3.4.3	Text synthesis based on n-grams	38
3.4.4	Text synthesis based on multigrams	39
3.4.5	Music synthesis	40
A	Appendix	41
A.1	German Manual of the Statistics Program Synther	41
	Bibliography	49

Chapter 1

Foundations

The investigation of accidental events with the help of statistic methods and probability based calculations has occupied crowds of mathematicians and other scientists for centuries. Already three hundred years ago the Swiss scholar JAKOB BERNOULLI created his *law of large numbers*, one of the most important models of the theory of probabilities. This classical principle is applicable to independent events, e.g. two throws of a dice or the birthdays of BILL CLINTON and GERHARD SCHRÖDER. However to reflect on dependent ones it is necessary to generalize BERNOULLI's scheme. These chains of dependent events, that were first studied in 1906 by ANDREI MARKOV, nowadays are used in various fields of mathematics and engineering sciences.

Besides it is sensible to think over the expression *event*, too. For instance chains of musical notes, of letters, words, measuring values and other generally called *symbols* can be investigated by using MARKOV's theory. This paper is to give you a short insight into symbol processing by means of the C++-program *synther*.

1.1 Markov Chains

First we look at a discrete process with the possible states

$$S_1, S_2, \dots, S_{x_{max}}.$$

At each moment

$$t_i = i\Delta t \quad \text{with} \quad i \in \mathbf{N}$$

they are able to change. The current assignment is to predict the states S^i for every moment t_i . In case they are completely independent (BERNOULLI's scheme), a possibility to solve this problem is to estimate the probabilities of the S_x for example by taking a huge number of random samples¹. For instance we

¹The connection between this estimated value and the real probability supplies the theorem of MOIVRE-LAPLACE, v. [Ma93].

look at a sample of twelve dice throws: (1, 4, 6, 6, 1, 2, 6, 2, 1, 2, 3, 4)². Now the probabilities $p(S_1), \dots, p(S_6)$ which belong to the possible states S_1, \dots, S_6 are to be estimated. An elegant kind of representation is to combine the $p(S_x) =: p_x$ in a vector:

$$\vec{p} := \begin{pmatrix} p_1 \\ \vdots \\ p_{x_{max}} \end{pmatrix} = \frac{1}{12} \begin{pmatrix} 3 \\ 3 \\ 1 \\ 2 \\ 0 \\ 3 \end{pmatrix}.$$

When we turn into the general symbol interpretation, each symbol embodies one state, the dependence on time may disappear, e.g. in case of handling word chains, therefore the discrete times are substituted by their indices to keep the order of the symbols S^0, S^1, S^2, \dots .

This statistic model that is based on occurrence probabilities is called **uni-gram model**.

However if the symbols are dependent, for instance the Boolean decisions if it's snowing New Year's Eve and New Year's Day, we have to deal with transition probabilities, in the simplest case with those between a symbol S_{x_1} and the following one S_{x_2} :

$$p((S^{i-1} = S_{x_1}) \wedge (S^i = S_{x_2})) =: p_{x_1, x_2}.$$

To determine these estimated probabilities we can act like described above. It is also useful to put the p_{x_1, x_2} into a compact structure that has to be a matrix in this case:

$$\underline{p} := \begin{pmatrix} p_{1,1} & \cdots & p_{1,x_{max}} \\ \vdots & & \vdots \\ p_{x_{max},1} & \cdots & p_{x_{max},x_{max}} \end{pmatrix}.$$

This kind of statistic models is known as **bigram model**.

If the influence of a symbol S^j on another one S^i with $i > j + 1$ is not negligible, an **n-gram model** should be used. The order n determines the degree of the dependence between the symbols. E.g. for $n = 3$ (**trigram**) the following transition probabilities are viewed:

$$p((S^{i-2} = S_{x_1}) \wedge (S^{i-1} = S_{x_2}) \wedge (S^i = S_{x_3})) =: p_{x_1, x_2, x_3}.$$

They can be embedded in a three-dimensional tensor \underline{P}_3 , like shown in the picture below:

²Each value symbolizes the number of pips of one throw.

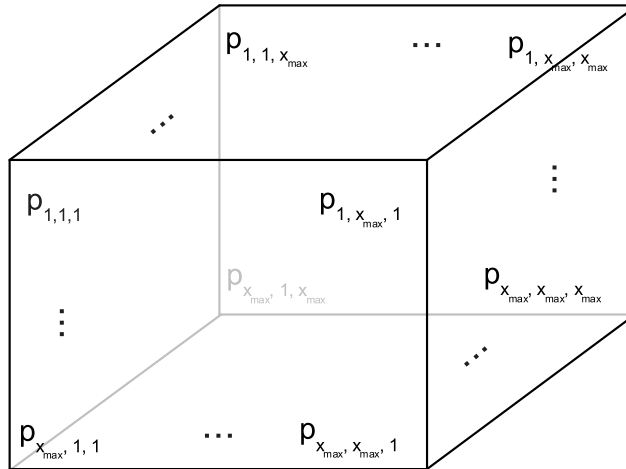


Figure 1.1: The tensor \underline{P}_3

In general case the representation is more difficult, n -dimensional tensors \underline{P}_n are developed and consist of x_{max}^n components.

1.2 Implementation Strategies of Symbol Processing

1.2.1 Direct Data Access

The reader might be surprised why we need implementation strategies in view of the fact that the foundations of n -gram modeling seem to be quite easy, but one of the biggest difficulties rests on the last sentence of the previous paragraph. The value of the components number x_{max}^n of the tensor \underline{P}_n may become very huge in dependence on the symbol number x_{max} and the order n . For instance we look at the trigram statistics of a newspaper that can include up to 5,000 different words, then we have to store $\sigma = x_{max}^n = 5,000^3 = 1.25 \cdot 10^{11}$ values³ (merely considered \underline{P}_3), we would need a memory capacity of 466 GB⁴ in case of using 32-bit numbers⁵. This example shows that we must think about efficient handling methods.

If a very limited domain is sufficient for the application, the statistic tensors need not be modified and can be stored in n -dimensional arrays. For example to inspect the transfer statistics of a binary cable with the help of a 10-gram we

³The quantity σ that is introduced here is equivalent to the number of values.

⁴1 GB = 2^{30} bytes

⁵In technical implementations it is advisable to use the absolute frequencies instead of the real probabilities, because many normalization steps can be saved.

only need a store of

$$\sigma = \sigma(\underline{P}_1) + \sigma(\underline{P}_2) + \cdots + \sigma(\underline{P}_{10}) = \sum_{k=1}^{10} 2^k = 2046,$$

$$\text{with } \underline{P}_1 := \overline{p} \text{ and } \underline{P}_2 := \underline{p}.$$

The decision if you should fall back on this not optimized variant of administration of statistic data depends on the available memory capacity and of course on the parameters x_{max} and n . Generally we have

$$\sigma_d = \sum_{k=1}^n x_{max}^k = x_{max} \frac{x_{max}^n - 1}{x_{max} - 1}. \quad (1.1)$$

The advantage of this simple method is the very fast access while reading as well as while storing data. We will see that in comparison with the other approaches this fact plays an important part.

1.2.2 Tree Structure

Especially to administer huge tensors with many zero elements there is an essentially economic store method: the data structure *tree* that is defined as follows (v. [Ad97]):

A *tree* is a graph with a finite number of nodes and directed, acyclic edges that must not be parallel.

- *One* node does not have a predecessor, it is called *root*.
- *All* nodes except for the root have exactly one predecessor.

Each element includes two values: an address α_{x_1, \dots, x_n} and the estimated probability p_{x_1, \dots, x_n} resp. the corresponding frequency. A general tree of the depth n is shown in the figure 1.2. It consists of the root and n layers that represent the tensors $\underline{P}_1, \underline{P}_2, \dots, \underline{P}_n$. The addresses α_{x_1, \dots, x_n} can be generally calculated as follows (figure 1.2 illustrates the equation):

$$\begin{aligned} \alpha_{x_1, \dots, x_n} &= x_{max} + x_{max}^2 + \cdots + x_{max}^{n-1} + \\ &\quad (x_1 - 1)x_{max}^{n-1} + (x_2 - 1)x_{max}^{n-2} + \cdots + (x_{n-1} - 1)x_{max} + x_n \\ &= \sum_{k=1}^{n-1} x_{max}^k + \sum_{k=1}^{n-1} (x_{n-k} - 1)x_{max}^k + x_n \\ &= \sum_{k=1}^{n-1} x_{n-k} \cdot x_{max}^k + x_n = \sum_{k=1}^n x_k \cdot x_{max}^{n-k} \end{aligned} \quad (1.2)$$

For $x_1 = x_2 = \cdots = x_n = x_{max}$ this equation passes over to (1.1) so that we have $\alpha_{max} = \sigma_d$.

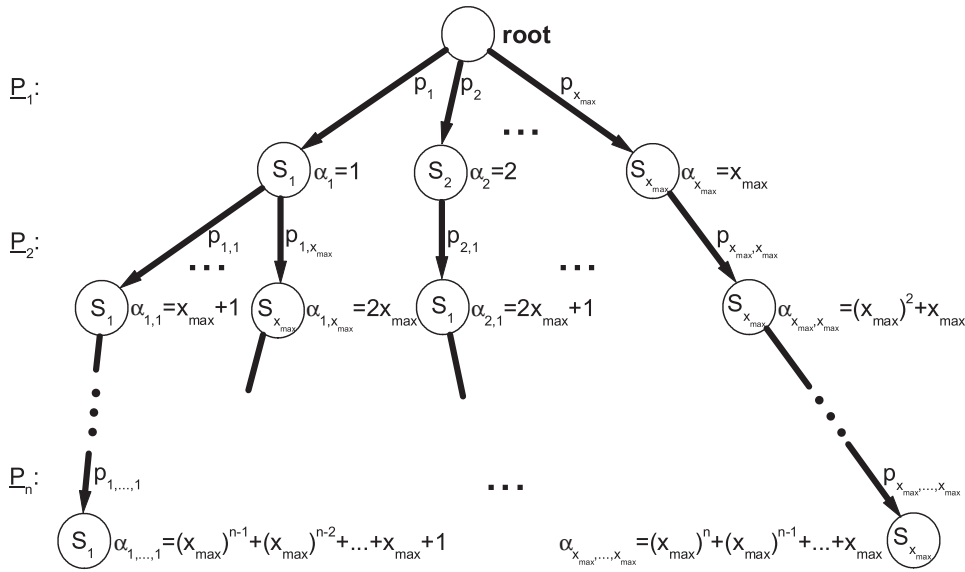


Figure 1.2: A general probability tree

Now we look at an example tree with the maximum order $n = 3$ and the $x_{max} = 4$ different symbols S_1, \dots, S_4 . The given sequence is $(S_3, S_1, S_4, S_3, S_4, S_2, S_4, S_2, S_4, S_4, S_1)$ or shortly $(3,1,4,3,4,2,4,4,1)$. The partial sequences up to the length $n = 3$ are formed and the addresses belonging to them are calculated:

type	sequence (x_1, \dots, x_n)	address α_{x_1, \dots, x_n}	frequency
unigram $(n = 1)$	(1)	1	2
	(2)	2	2
	(3)	3	2
	(4)	4	5
bigram $(n = 2)$	(1,4)	8	1
	(2,4)	12	2
	(3,1)	13	1
	(3,4)	16	1
	(4,1)	17	1
	(4,2)	18	2
	(4,3)	19	1
	(4,4)	20	1
trigram $(n = 3)$	(1,4,3)	35	1
	(2,4,2)	50	1
	(2,4,4)	52	1
	(3,1,4)	56	1
	(3,4,2)	66	1
	(4,2,4)	76	2
	(4,3,4)	80	1
	(4,4,1)	81	1

Table 1.1:
Example statistics

By means of these values we can construct the corresponding tree:

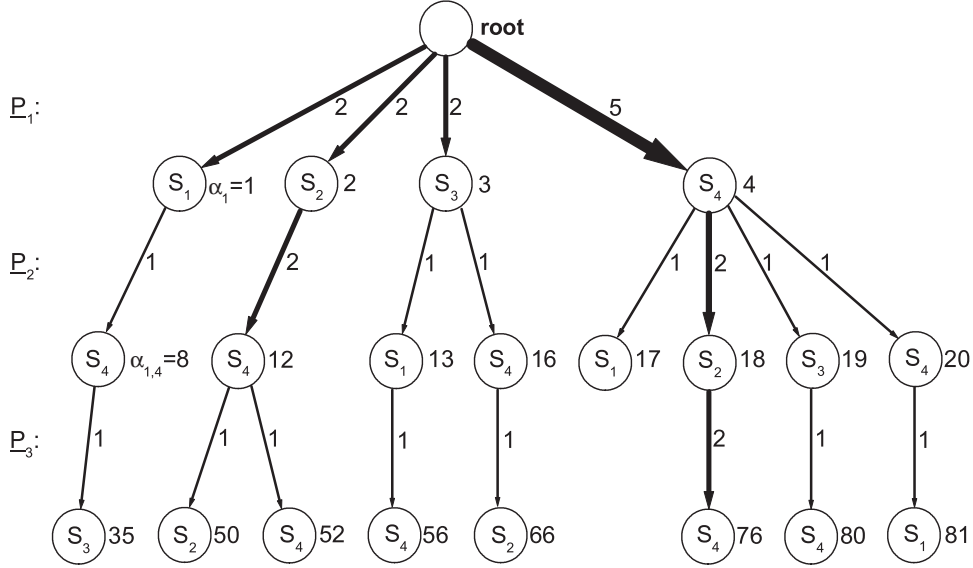


Figure 1.3: An example tree

Explanation of this figure: Inside the circles we can find the symbols, the numbers beside them are the α_{x_1, \dots, x_n} , those at the edges stand for the frequency they are taken (the thickness of the arrows reflects this value).

The simplest kind of implementation is to administer merely a table like 1.1 that only contains the addresses α_{x_1, \dots, x_n} and the corresponding frequencies (like already mentioned on p. 3 the probabilities usually are not stored themselves)⁶. But how to navigate in this list while setting up or accessing to it later, when it is used e.g. as a language model in a speech processing application?

An important condition for a very fast data access is that the entries of the table are in an ascending order concerning its addresses. To look for the frequency of an element (x_1, \dots, x_n) the simplest procedure is to compute α_{x_1, \dots, x_n} using (1.2) and to start the search in the middle of the current tensor table. If the address there is greater than α_{x_1, \dots, x_n} we resume the search in the first half of the list, otherwise in the other one. The same decision we make for the now interesting part of the table: We compare the α_{x_1, \dots, x_n} with the address in the middle of the current half and decide in which quarter we continue the search. This action is repeated either until we hit on the searched element or until we find two direct neighbors in the list so that one of their addresses is greater than α_{x_1, \dots, x_n} and the other one less⁷.

⁶This table we call $T(P_1, \dots, P_n)$. Its contents are all non-zero elements of the tensors P_1, \dots, P_n and the addresses α_{x_1, \dots, x_n} .

⁷The just described search algorithm is called *binary search*, v. [Ni00].

It depends on the context of our search how we terminate our action: During the training of our tree we have to increment the frequency of the current element by one, that means if we cannot find it in the table we must insert it in position, which makes special demands on the used data structure, e.g. in some cases linked pointers could be more flexible than two-dimensional arrays, because the insertion turns out very fast.

In applications where the data must be accessed merely for reading, the table structure becomes very advantageous as against the other implementations of the n-grams: When the searched element cannot be found, its frequency is zero, otherwise it can be read in the determined row of the list. This method is very economical of time, because between the number of entries in the table $\sigma_t = \sigma(T(\underline{P}_1, \dots, \underline{P}_n))$ and the number of searching steps $\frac{t_{search}}{t_{step}}$ exists the following connection:

$$\frac{t_{search}}{t_{step}} \leq \text{int}(\text{ld}(\sigma(T(\underline{P}_1, \dots, \underline{P}_n)))) + 1$$

$\text{int}(x)$ is the integer part of the real number x and $\text{ld}(x) = \log_2 x$.

For instance when we look at a tree with 1,000,000 nodes then we need less than or equal to

$$\text{int}(\text{ld}(1,000,000)) + 1 = 20$$

steps to find a special frequency in the list. Of course the numerical mathematics and the computer sciences make more efficient methods to detect the right node available, above all considering the fact that the number of elements that belong to the individual tensors are very different, thus we can generally expect

$$\sigma(\underline{P}_1) \ll \sigma(\underline{P}_2) \ll \dots \ll \sigma(\underline{P}_n) \approx \sum_{k=1}^n \sigma(\underline{P}_k) = \sigma_t.^8 \quad (1.3)$$

That means when we look for a node located in the tensors $\underline{P}_1, \dots, \underline{P}_{n-1}$ it is commendable not to start the search in the middle of the data base but in the middle of the interesting part. Anyway this algorithm is very fast and probably takes the most time to compute the α_{x_1, \dots, x_n} .

When we want to predict the necessary memory capacity of the list we must not only draw conclusions from the σ_t but also from the requirements of a single row. We have to store the frequency (in general a 32-bit number) as well as the α_{x_1, \dots, x_n} . The equations (1.1) and (1.2) supply the greatest possible address α_{max} that needs at least

$$\begin{aligned} \sigma_\alpha &= \text{int}(\text{ld}(\alpha_{max})) + 1 \\ &= \text{int} \left(\text{ld} \left(x_{max} \frac{x_{max}^n - 1}{x_{max} - 1} \right) \right) + 1 \\ &= \text{int}(\text{ld}(x_{max}) + \text{ld}(x_{max}^n - 1) - \text{ld}(x_{max} - 1)) + 1 \text{ [bit]}, \end{aligned}$$

⁸In contrast to (1.1) in this case the $\sigma(\underline{P}_n)$ comprise merely the NZEs.

and with the assumption $x_{max} \gg 1$

$$\begin{aligned}\sigma_\alpha &\approx \text{int}(\text{ld}(x_{max}) + n\text{ld}(x_{max}) - \text{ld}(x_{max})) + 1 \\ &\approx \text{int}(n\text{ld}(x_{max})) + 1 \text{ [bit]}.\end{aligned}\tag{1.4}$$

That means that the necessary store capacity for the addresses is directly proportional to the order n .

To return to the example of 1.2.1 (a trigram of a newspaper with 5,000 different words), we would need an address memory for each row of

$$\sigma_\alpha = \{\text{int}(3 \text{ld}(5,000)) + 1\} \text{bit} = 37 \text{ bit},$$

this could be represented by three 16-bit numbers, so that each element of the table all in all needs 80 bit (32 bit for the frequency and 48 bit for the addresses).

1.2.3 Sparse Tensors

In many scientific problems we stumble across matrices with a low density of non-zero elements. For instance while simulating a linear electric network of a high dimension we have to handle such objects. When the share of NZEs (Non-Zero Elements) is less than 5 per cent these matrices are called *sparse* (definition v. [Pö00]). In the previous paragraphs we have seen that while dealing with symbol statistics such sparse matrices come into being. Therefore it is advisable to enlist corresponding works of the numerical mathematics.

A quite efficient and very flexible method is the *compact storage algorithm* of RHEINBOLDT and MESZTENYI (v. [Rh80]) that is used by *synther* to organize the frequency data. It concerns a cyclic twice linked list that rests on the following principle: Each NZE is represented by three numbers in the list: its value (e.g. a probability) and its linked x- resp. y-coordinate. In the following the method of working is to be explained. The elements are also placed into a table that consists of two parts: an interface ($\in \mathbf{N}^{2 \times x_{max}}$) between the coordinates of the NZEs and the reference values which are used in the second list ($\in \mathbf{N}^{3 \times x_{max}}$) to identify the position of the elements. In the following we want to designate the interface list as org_2 and the NZE list as T_2 .

To begin with an empty¹⁰ matrix $\underline{P}_2 \in \mathbf{N}^{x_{max} \times x_{max}}$: The rows of org_2 are numbered from 1 to x_{max} . T_2 is still empty (there are no elements to enter).

	org_2				T_2
index	1	2	...	x_{max}	$x_{max} + 1$
frequency					
x	1	2	...	x_{max}	
y	1	2	...	x_{max}	

Table 1.2:
Compact storage algorithm:
empty matrix

⁹If $x_{max} = 2^\nu$ with $\nu \in \mathbf{N}$ the second addend 1 becomes 0.

¹⁰matrix of zeros

When we want to add an element with the coordinates (x_k, y_k) we must write its value p_{x_k, y_k} into the frequency row of T_2 . Now we have to replace the number with the index x_k of org_2 's x-row by the index of T_2 's new column. On the other hand the old x-value of org_2 must be put into the x-row of the new column of T_2 . This procedure is carried out in the y-row, too:

	org_2						T_2			
index	1	2	...	x_k	...	y_k	...	x_{max}	$x_{max} + 1$	$x_{max} + 2$
frequency										
x	1	2	...	$x_{max} + 1$					p_{x_k, y_k}	
y	1	2	...			$x_{max} + 1$...	x_{max}		

Table 1.3:

Compact storage algorithm: matrix with one element

For each new element we proceed on the same principle.

However when we look at tensors of arbitrary dimensions we must extend the compact storage algorithm: Each further dimension needs a new row in the list. In contrast with the tree structure handled in 1.2.2 this method can be modified, so one need not know the number of different symbols x_{max} before the statistic analysis. This characteristic is very advantageous at the sight of the time-consuming action of a new calculation of all addresses of the tree list in case of exceeding the estimated x_{max} . The simple trick to realize this feature is to substitute the indices of org_n for their negative values. Besides we can start numbering the indices of T_n with 1. The simplest way to make clear this extended algorithm is by means of the example in 1.2.2: the development of unigram, bigram and trigram statistics on the basis of the given sequence $(S_3, S_1, S_4, S_3, S_4, S_2, S_4, S_2, S_4, S_4, S_1)$. When we want to put all these n-grams into one table it has a quite complex structure. The empty table (before we start the analysis) looks as follows:

type		org_n	T_n
	symbol		
	index	-1	1
unigram ($n = 1$)	frequency		
	x_1	-1	
bigram ($n = 2$)	frequency		
	x_1	-1	
	x_2	-1	
trigram ($n = 3$)	frequency		
	x_1	-1	
	x_2	-1	
	x_3	-1	

Table 1.4:

Extended compact storage algorithm: empty matrix

Now we start working out our example sequence: The first symbol (S_3) is automatically allocated to the index -1 , the second (S_1) to -2 and the third (S_4) to -3 . We get three unigram entries $\{(-1), (-2), (-3)\}$, two bigrams $\{(-1, -2), (-2, -3)\}$ and one trigram $\{(-1, -2, -3)\}$. Each of them occurs only once for the time being:

type		org_n			T_n			
		symbol index	S_3 -1	S_1 -2	S_4 -3	1	2	3
unigram	frequency				1	1	1	
($n=1$)	x_1	1 -1	2 -2	3 -3	-1	-2	-3	
bigram	frequency				1	1		
($n=2$)	x_1	1 -1	2 -2	-3	-1	-2		
	x_2		1 -2	2 -3	-2	-3		
trigram	frequency				1			
($n=3$)	x_1	1 -1	-2	-3	-1			
	x_2		1 -2	-3	-2			
	x_3		-1	-2 -3	-3			

Table 1.5:
Extended compact
storage algorithm:
three NZEs

With the help of the last symbol S_2 that corresponds with the index -4 we can transform the given symbol sequence into the index sequence $(-1, -2, -3, -1, -3, -4, -3, -4, -3, -3, -2)$. When we continue the algorithm until the end the table looks as follows:

type		org_n				T_n									
		S_3 -1	S_1 -2	S_4 -3	S_2 -4	1	2	3	4	5	6	7	8	9	(*)
unigram	frequency					2	2	5	2						
($n=1$)	x_1	1	2	3	4	-1	-2	-3	-4						(**)
bigram	frequency					1	1	1	1	2	2	1	1		
($n=2$)	x_1	4	2	8	6	-1	-2	-3	1	3	-4	5	7		
	x_2	3	8	7	5	-2	-3	-1	2	-4	4	6	1		
trigram	frequency					1	1	1	1	2	1	1	1		
($n=3$)	x_1	4	2	8	7	-1	-2	-3	1	3	-4	6	5		
	x_2	3	1	8	5	-2	-3	-1	2	-4	4	6	7		
	x_3	2	8	7	6	-3	-1	1	-4	3	4	5	-2		

Table 1.6: Extended compact storage algorithm: example sequence

We have seen that inserting a new element is quite simple in the aspect of memory administration because it is always written to the end of the existing list. However the navigation in the list turns out comparatively large-scaled. The search for a given symbol sequence is to be checked by means of table 1.6 and the example sequence (S_2, S_3, S_2) . To begin with we transfer it into $(x_1, x_2, x_3) = (-4, -1, -4)$ using the first row of the table. Now we look at org_3 for the substitute of x_1 and find the number 7. Consequently we go in T_3 to the position with the index 7. There we come across the triple $(x'_1, x'_2, x'_3) = (6, 6, 5)$. The next step is to inspect the equivalence of x_2 and x'_2 . For that purpose the reference x'_2 must be followed until we stumble on a value less than zero: $x'_2 = 6 \Rightarrow 4 \Rightarrow 2 \Rightarrow -3$. That means $x'_2 \Leftrightarrow -3 \neq x_2$, so the search must be continued at the T_3 's column with the index $x'_1 = 6$: $(x''_1, x''_2, x''_3) = (-4, 4, 4)$. From the previous step it is known that $x''_2 = 4$ is not equivalent to x_2 , therefore we would have to go on with the reference in x''_1 , but it is already about a negative number, thus the search can be abandoned here with the result that the frequency of the sequence (S_2, S_3, S_2) is zero.

The memory requirements of this storage algorithm are in almost each case greater than those of the corresponding tree structure. Table 1.6 illustrates the capacity need σ_s of the sparse tensors $\underline{P}_1, \dots, \underline{P}_n$:

$$\begin{aligned} \sigma_s &= \overbrace{\sum_{k=2}^n k \cdot x_{max}}^{org_k} + \overbrace{x_{max}}^{T_1} + \overbrace{\sum_{k=2}^n (k+1) \cdot \sigma(\underline{P}_k)}^{T_k} \\ &= \frac{x_{max}}{2} (n^2 + n) + \sum_{k=2}^n (k+1) \cdot \sigma(\underline{P}_k) \end{aligned} \quad (1.5)$$

The rows of the table 1.6 that are marked with (*) resp. (**) need not be stored because they always satisfy the same structure:

type		org_2			T_2			
	index	-1	...	$-x_{max}$	1	...	$\sigma(\underline{P}_n)$	(*)
unigram ($n = 1$)	x_1	1	...	x_{max}	-1	...	$-x_{max}$	(**)

Table 1.7:

Extended compact storage algorithm: not stored rows

Unfortunately the computing time behaviour of the compact storage algorithm is not as efficient as the tree model because in worst case we must search the complete list in order to find the frequency of a given symbol. Determining a general expression for the necessary number of searching steps is difficult considering the interlaced search method and highly depends on the statistic characteristics of the basis material. Some experimental results of the computing time investigations can be found in paragraph 3.1.

1.2.4 An Example

By means of a practical example the differences of the memory efficiency of the three implementation strategies that we became acquainted with are to be compared. For that purpose a part of the text material (CDs 1 to 5) of the *Verbmobil*¹¹ project which contains scheduling appointments of spontaneous speech serves as an example. The basis material comprises 177,781 words. The result of the analysis (maximum order $n = 3$) was:

n	$\sigma(\underline{P}_n)$
1	4,936 = x_{max}
2	44,995
3	100,043

Table 1.8:
Example statistics on the basis
of the *Verbmobil* material

Considering that the most used symbol (*ich*) occurs 6,235 times the data type that represents the frequency can be restricted to 16-bit, so in the first place σ_d for the direct data access is computable with (1.1).

To determine the needed store capacity for the tree structure first we must think about the maximum address (v. 1.2.2) resp. its logarithmic representation σ_α defined in (1.4). x_{max} is estimated at less than 2^{16} (cf. above), therefore σ_α becomes 48 bit.

Also the memory requirements of the compact storage algorithm could be minimized by adjusting the frequency data type to the given basis material. However the example realization that is implemented in *synther* generally stores all frequencies as 32-bit numbers.

type	needed memory	[MB]
direct data access	$\sigma_d \cdot 16\text{bit} =$	229,426.473
tree structure	$\sigma_t(16\text{bit} + \sigma_\alpha) =$	1.144
compact storage algorithm	$\sigma_s \cdot 32\text{bit} =$	2.154

Table 1.9:
Required memory capacity of the different implementation strategies

1.2.5 Quantization and Domain Restriction

Independent of the used implementation strategy it is advantageous to limit the number of different symbols. An effective method is to quantize the values that are represented by the symbols. Above all that works while investigating digitized physical quantities like voltage (e.g. sampled sounds), temperature or time (v. 3.1). The quantization degree q ($q \in \mathbf{Q}^{12}$, $q \geq 1$) determines, "how many"

¹¹Verbmobil is a long-term interdisciplinary Language Technology project (1993-2000), for further information v. [DF00].

¹² \mathbf{Q} is the set of rational numbers.

neighbored values are combined. Non-integer values of q can be realized by grouping different numbers of symbols: E.g. $q \approx \pi = 3.14159\dots$ can be realized by using group sizes of $q_1 = \text{int}(q) = 3$ and $q_2 = \text{int}(q) + 1 = 4$ symbols. The question is now how many groups of the respective size have to be used, $x_{max,1}$ is the number of the q_1 -groups and $x_{max,2}$ that of the q_2 -groups. The quantization degree we strive for we are able to express with

$$q = \frac{x_{max}}{x_{max,1} + x_{max,2}},$$

besides the above balance can be consulted:

$$\begin{aligned} x_{max} &= q_1 x_{max,1} + q_2 x_{max,2} \\ &= \text{int}(q) x_{max,1} + (\text{int}(q) + 1) x_{max,2}. \end{aligned}$$

After solving the system of these both equations we get

$$\begin{aligned} x_{max,1} &= x_{max} \left(\frac{\text{int}(q) + 1}{q} - 1 \right), \\ x_{max,2} &= x_{max} \left(1 - \frac{\text{int}(q)}{q} \right). \end{aligned}$$

In general $x_{max,1}$ and $x_{max,2}$ are also rational numbers, in an application they have to be rounded off. For instance when we want to quantize a set of 1,000 different symbols using the mentioned quantization degree $q \approx \pi$ we should split it into 272 groups with $q_1 = 3$ and 46 with $q_2 = 4$.

The following algorithm assigns automatically a certain number of symbols to a group:

```

parameters: q, xMax;

combine int(q) symbols;
x=int(q);
counter=1;

while (x<xMax)
{
  if (x/counter<q)
  {
    combine int(q)+1 symbols;
    x=x+int(q)+1;
  }
  else

```

```

    {
        combine int(q) symbols;
        x=x+int(q);
    }
    counter++;
}

```

In case $q \approx \pi$ this algorithm produces group sizes of the order (3, 4, 3, 3, 3, 3, 3, 3, 4, 3, ...).

The general effect of quantization on the needed memory capacity is to be shown on the basis of the direct data access. The ratio of the required memory after to that before the quantization is computable with the help of (1.1):

$$\frac{\sigma_{d,q}}{\sigma_d} = \frac{\frac{x_{max}}{q} \cdot \frac{(\frac{x_{max}}{q})^{n-1}}{\frac{x_{max}}{q} - 1}}{x_{max} \frac{x_{max}^n - 1}{x_{max} - 1}} = \frac{x_{max} - 1}{x_{max} - q} \cdot \frac{(\frac{x_{max}}{q})^n - 1}{x_{max}^n - 1} .$$

For $x_{max} \gg 1$ this equation becomes

$$\frac{\sigma_{d,q}}{\sigma_d} \approx \frac{x_{max}}{x_{max} - q} \cdot (q^{-n} - x_{max}^{-n})$$

and for $x_{max} \gg q$ even

$$\frac{\sigma_{d,q}}{\sigma_d} \approx q^{-n} .$$

The following example emphasizes the enormous significance of this expression: We want to investigate the statistic features of a music recording. The available material has CD-quality that means above all 2^{16} different possible sample values. Our analysis is based on an n-gram model of the maximum order $n = 2$ (bigram). Using 32-bit numbers for storing the frequency values by direct data access we would need at least 16 GB (!). Quantizing the symbol number by converting this high fidelity music into 8-bit data¹³ reduces the required store to no more than 257 kB. It is true that the memory size that is occupied by the other implementation strategies generally decreases, too, but mostly less conspicuously.

When we deal with n-gram statistics of quantities that cannot be quantized (for instance the vocabulary of a language model) we should try to limit the set of symbols all the same. The simplest method is to restrict the domain: Instead of constructing a general translation system it is commendable e.g. only to consider scheduling appointments (like realized by the *Verbmobil* project [v. DF00]). Another possibility to reduce the number of different elements is to combine a set of synonymous resp. homogeneous symbols for instance numerals, surnames etc.

¹³Who thinks that 8-bit quantized music sounds like scratching should know that the famous *Microsoft Sound* has the following features: 22.05 kHz, 8 bit, mono.

Chapter 2

Fields of Application

2.1 Language Model

Stochastic language models that are described with the help of n-gram probabilities are investigated profoundly above all with regard to their use in speech recognition (e.g. vide [Be99] or [Sc95]). Technical speech recognition systems consist of several interacting modules which take on different tasks on different

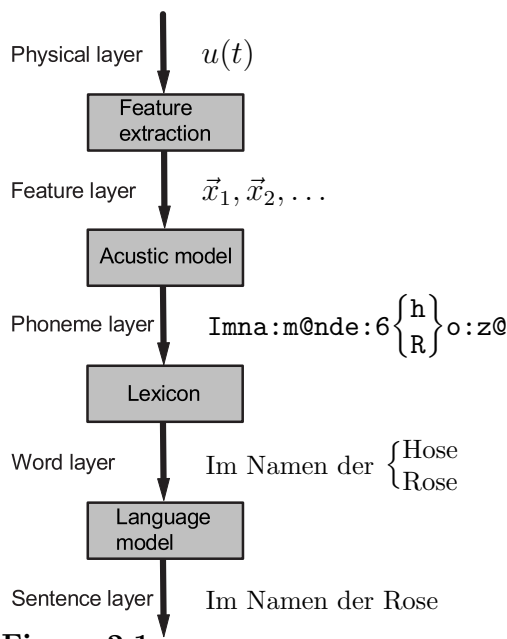


Figure 2.1: General structure of a speech recognizer with an example word sequence

layers. Figure 2.1 demonstrates which position of the processing chain of the recognizer the language model holds. Its fundamental assignment is to detect the likeliest way through the word hypothesis grid of the word layer. This grid \underline{S} consists of i_{max} symbol vectors with h_i different hypotheses:

$$\begin{aligned} \underline{S} &= (\vec{S}^1, \vec{S}^2, \dots, \vec{S}^{i_{max}}) \\ &= \left(\left(\begin{array}{c} S_1^1 \\ \vdots \\ S_{h_1}^1 \end{array} \right), \dots, \left(\begin{array}{c} S_1^{i_{max}} \\ \vdots \\ S_{h_{i_{max}}}^{i_{max}} \end{array} \right) \right). \end{aligned}$$

It is conspicuous that we generally have to compare all possible paths through the grid to determine the most probable one. The number of different paths is

$$n_w = \prod_{i=1}^{i_{max}} h_i. \quad (2.1)$$

Determining the likeliest way through the grid depends on the maximum order of the given n-grams. The probability of a given symbol sequence (one path of

the grid) we can estimate as follows:

$$p_{n,i_{max}}(S^1, S^2, \dots, S^{i_{max}}) = \prod_{k=1}^{n-1} p(S^k | S^{k-1}, \dots, S^1) \prod_{l=n}^{i_{max}} p(S^l | S^{l-1}, \dots, S^{l-n+1})$$

with $p(S^i | S^{i-1}, \dots, S^j) = p(S^i)$ for $i = j$.¹ (2.2)

Special cases of this equation for the orders $n = 1 \dots 3$ can be found in [Be99]:

n	$p_{n,i_{max}}(S^1, S^2, \dots, S^{i_{max}})$
1	$\prod_{i=1}^{i_{max}} p(S^i)$
2	$p(S^1) \prod_{i=2}^{i_{max}} p(S^i S^{i-1})$
3	$p(S^1) p(S^2 S^1) \prod_{i=3}^{i_{max}} p(S^i S^{i-1}, S^{i-2})$

Table 2.1:

likelihood of the sequence $(S^1, S^2, \dots, S^{i_{max}})$ in dependence on the maximum order n

According to (2.1) n_w can become an enormous number when we want to deal with long chains or many hypotheses. For this reason an efficient search algorithm² was developed that checks the path probabilities of all paths that end at the nodes (symbols) of the grid. Only the likeliest of them must be carried on. To explain the mechanism of searching the likeliest word sequence in a hypothesis grid we look at an example that is based on the *Verbmobil* text material that already was used in 1.2.4:

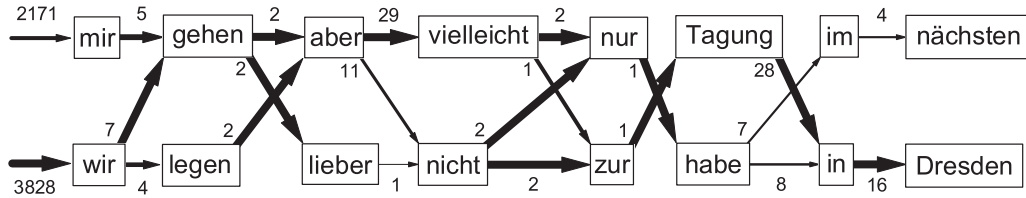


Figure 2.2: example hypothesis grid

The numbers at the arrows of this example grid $\underline{S} = (\vec{S}^1, \vec{S}^2, \dots, \vec{S}^8)$ with $h_i = 2$ ³ are the bigram transition frequencies except those that end at the symbols *mir* and *wir*, these are the occurrence frequencies of the possible start symbols of the sequence. The transition frequency of two succeeding hypotheses that are not connected is zero. To simplify matters we discuss the task merely for bigram statistics. To carry out the VITERBI Algorithm we need the formula of the sequence likelihood of table 2.1. It is true that we actually need the estimated

¹Occurrence and transition probabilities were discussed in 1.1.

²It is called VITERBI Algorithm, cf. [Fi00].

³Here is to be mentioned that generally the hypotheses at the output of a word recognizer are connected with different probabilities that must be taken into consideration. In our example we expect equal likelihood.

occurrence resp. transition probabilities of the examined symbols, but they only differ from the relative frequencies in being normalized to 1. Equation (2.2) makes plausible that the result of our calculation is proportional to the result of a calculation with estimated probabilities. Finally we need only the maximum likelihood, so that a proportionality constant does not disturb the decision.

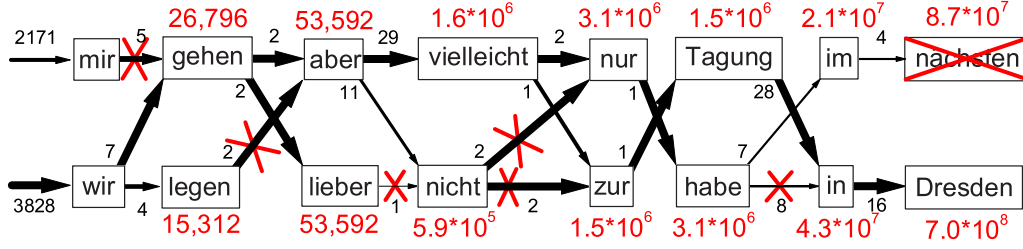


Figure 2.3: VITERBI Algorithm

To start the VITERBI Algorithm we go directly into the second column of the grid (\vec{S}^2) and compute all possible sequence probabilities of the order $i_{max} = 2$, that means:

- $p_{2,2}(mir, gehen) = \frac{2,172 \cdot 5}{c_2} = \frac{10,860}{c_2}$,
- $p_{2,2}(mir, legen) = 0$,
- $p_{2,2}(wir, gehen) = \frac{26,796}{c_2}$,
- $p_{2,2}(wir, legen) = \frac{15,312}{c_2}$.

The constants c_i are used to normalize the estimated values to 1⁴ because long symbol chains create huge numbers which perhaps exceed the range of the used data types. In our example we assume that $c_i = 1$. The next step is to check the arrows that end at the elements of \vec{S}^2 : Only the likeliest of each symbol survives, the others are crossed out. The calculated values of the surviving paths must be remembered, therefore in figure 2.3 they are noted down close to the words belonging to them using red digits. Now we change to $\vec{S}^3 = (aber, lieber)$ and compute the $p_{2,3}$ of all possible paths by multiplying the surviving $p_{2,2}$ with the transition probabilities between the elements of \vec{S}^2 and \vec{S}^3 :

- $p_{2,3}(\bullet^5, gehen, aber) = 26,796 \cdot 2 = 53,592$,
- $p_{2,3}(\bullet, gehen, lieber) = 53,592$,
- $p_{2,3}(\bullet, legen, aber) = 30,624$,
- $p_{2,3}(\bullet, legen, lieber) = 0$.

⁴or to another value, however in this case we must not designate $p_{n,i_{max}}$ as probability

⁵symbolizes the predecessor that is determined during the backtracking after all.

Now we must check these values and cross out so many arrows that always merely the likeliest path ends at the symbol *aber* resp. *lieber*. The $p_{2,3}$ values of the surviving arrows are wrote down close to the symbols using red digits. This procedure is continued until $\vec{S}^8 = (\text{nächsten}, \text{Dresden})$. The symbol that is the final element of the sequence with the greatest value $p_{2,8}$, *Dresden*, survives, the other one is crossed out. Now we can start the backtracking until \vec{S}^1 by taking only the surviving arrows. In this way we get the likeliest symbol sequence of our example: *wir gehen aber vielleicht zur Tagung in Dresden*.

2.2 Musical Applications

The very extensive apparatus of the music theory has been extended for about fifty years by introducing statistic methods. For example the *musique sérielle*, a composition technique that was developed by O. MESSIAEN and P. BOULEZ, investigates sequences of musical note parameters like pitch, duration, volume, timbre etc. Dependent on the current music style these sequences show big differences of their statistic characteristics. E.g. the dodekaphony (twelve tone technique) of A. SCHÖNBERG has almost equal estimated occurrence probabilities of all tones of an octave in contrast to the compositions of the baroque period (BACH, HÄNDEL, VIVALDI etc.) that hardly used chromatic⁶ transitions (cf. 3.3).

When we want to use the theory of n-grams (explained in chapter 1) to analyze musical structures at first we must think about the well-definition of a symbol in that case. The exclusive handling of tone sequences will not be sufficient in most cases, because general music processing is confronted with the following structures:

- tone sequences (melodies)
- simultaneous tones (chords)
- polyphonous structures (e.g. fugues)
- multi-timbral structures (several simultaneous instruments)

and besides has to consider special events like glissando, use of pedals, change of sound characteristics etc. Therefore we have to find a suitable representation form of the musical event. Another important aspect which must be taken into consideration is that the analyzed score should be machine-readable. The internationally standardized MIDI⁷-file format is the solution of both problems. Another invaluable advantage of MIDI-files is the presence of huge data banks of

⁶a sequence of semitones

⁷MIDI is the acronym for Musical Instrument Digital Interface.

serious music (v. e.g. [Mi01]) as well as light music (v. e.g. [Ge00]) in the internet. A general MIDI-event consists of three parts:

- the delta-time,
- a status byte and
- several data bytes.

The **delta-time** is the time difference between the current event and its predecessor. When it is equal to zero simultaneousness is forced. Notes are always described by two events: *note on* and *note off*. The **status byte** contains the event type identification and the MIDI-channel⁸ belonging to it. All other information like key velocity, pitch etc. is contained in the **data bytes**.

By means of this MIDI-event definition the demand for implementation of the complex musical structures listed above can be fulfilled; a polyphonous, mono-timbral sequence and its MIDI-event representation is demonstrated in figure 2.4.



index	delta-time ⁹	event-type (note)	pitch	12	0	off	Eb5	25	96	off	Eb5
1	0	on	G4	13	0	on	D4	26	0	on	F5
2	0	on	Bb4	14	0	on	G4	27	96	off	Ab4
3	192	off	Bb4	15	96	off	D4	28	0	on	D4
4	0	on	Eb5	16	0	on	Eb4	29	96	off	D4
5	96	off	Eb5	17	96	off	Eb4	30	0	off	F5
6	0	on	D5	18	0	off	G4	31	0	on	Eb4
7	96	off	G4	19	0	on	Ab4	32	0	on	A4
8	0	off	D5	20	192	off	Ab4	33	96	off	Eb4
9	0	on	Eb5	21	0	on	Ab4	34	0	on	F4
10	96	on	C4	22	0	on	F5	35	96	off	F4
11	96	off	C4	23	96	off	F5	36	0	off	A4
				24	0	on	Eb5				

Figure 2.4: J. S. BACH: The Well-Tempered Clavier, Book I No. 2 Fugue, bar 5

⁸MIDI-channels can be imagined like the channels of a mixer: Each of them can carry an own instrument with its individual features like sound effects, panorama and filter settings etc.

⁹A delta-time value of $384 = 3 \cdot 2^7$ corresponds to a quarter note. The prime factor 3 is used because the trisection plays an important part in the music (triplets).

Beside the application to the music theory for the purpose of investigation of music forms and styles, the knowledge about the statistic features of a given MIDI arrangement which was acquired with the help of n-gram modeling can be used for synthesis of new MIDI-event sequences, in plain language: for improvisation and composition.

2.3 Synthesis of Symbol Sequences

2.3.1 Synthesis on the Basis of Occurrence Probabilities

When a sequence is to be synthesized considering merely the occurrence probabilities p_x of the used symbols that are either examined by analyzing a set of given samples or are arbitrarily defined to achieve special results, we need a random numbers generator that emits uniformly distributed numbers $r \in [0, 1]$. Pictorially we split a line segment of the length 1 into x_{max} parts of the different lengths p_x , $x = 1 \dots x_{max}$, and interpret the value r as the distance between the left end of the line segment and a random point P_r in the direction to its right end. The index x_r of the part which includes P_r corresponds to the randomly determined symbol S_{x_r} of the sequence¹⁰.

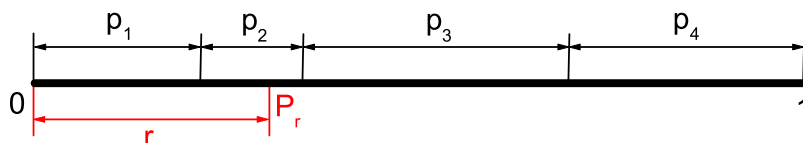


Figure 2.5: Determination of a symbol index (in this case $x_r = 2$) by using a random number r only considering the occurrence probabilities p_1, \dots, p_4

The just described procedure can be formally expressed:

$$x_r = \min_{x=1 \dots x_{max}} \left\{ x \mid \sum_{k=1}^x p_k \geq r \right\}. \quad (2.3)$$

At first sight this equation seems to be disproportionately complex but it directly represents a possible algorithmic implementation of the determination of the x_r :

```
given: p[1], ..., p[xMax];

r=rnd(); //random number between 0 and 1
sum=0;
counter=0;
```

¹⁰The mathematical foundation of this method is the concept of *geometric probability*, v. [Ve96].

```

while (sum<r)
{
    counter++;
    sum=sum+p[counter];
}
if (counter==0) counter++;

xR=counter; //symbol index

```

To estimate the average time (resp. the average number of additions μ_a) that this algorithm needs to compute one symbol index we should look back at figure 2.5: With the probability p_1 the point P_r falls into the first segment, one addition would be needed. P_r falls into the second segment with the probability p_2 , in this case two operations would be carried out, and so on. Therefore the expected average number of additions of our example is

$$\mu_{a,example} = p_1 \cdot 1 + p_2 \cdot 2 + p_3 \cdot 3 + p_4 \cdot 4 \text{ [flops]}^{11}.$$

So generally we have

$$\mu_a = \sum_{k=1}^{x_{max}} k \cdot p_k \text{ [flops]}. \quad (2.4)$$

Now we want to examine the occurrence probabilities p_x more exactly by means of the example *language modeling* (cf. 2.1). The ZIPF law (developed 1935 by the linguist G. K. ZIPF, cf. [We91]) says that the occurrence probabilities of any natural language can be approximated with the aid of the following formula:

$$p_x = \frac{A}{x}, \quad (2.5)$$

A is a constant that depends on the respective language, and the p_x must be put in order, so that

$$p_1 > p_2 > \dots > p_{x_{max}}. \quad (2.6)$$

For the present we do not know if this demand is fulfilled by the n-gram analysis algorithms that generally assign the symbol indices in chronological order of the input (v. the instance on p. 10). Therefore we look at example statistics with the symbol number $x_{max} = 4$ and the unigram probabilities p_x that fulfill ZIPF's law (2.5). Of course the sum of the p_x must be 1, generally expressed by the following equation:

$$\sum_{x=1}^{x_{max}} \frac{A}{x} = 1, \quad (2.7)$$

¹¹**floating point operations** is a customary unity of the computer-numerical mathematics. Modern processors need almost the same time for an addition and a multiplication (1 flop). Bit shift operations (e.g. in the algorithm above: `counter++`) are much faster. Therefore they are represented by the approximate value 0 flops.

applied to our example it becomes

$$A \cdot \left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4}\right) = 1 \implies A = \frac{12}{25}. \quad (2.8)$$

Now we want to predict the contents of a sequence of the length $l = 25$. On condition that the symbols are independent we first compute the p_x and the corresponding symbol frequencies that are $l \cdot p_x$:

x	p_x	frequency
1	0.48	12
2	0.24	6
3	0.16	4
4	0.12	3

Table 2.2:
Estimated symbol frequencies of a sequence

The only information we have are these frequencies, so we are bound to assume uniform distribution of the symbols:

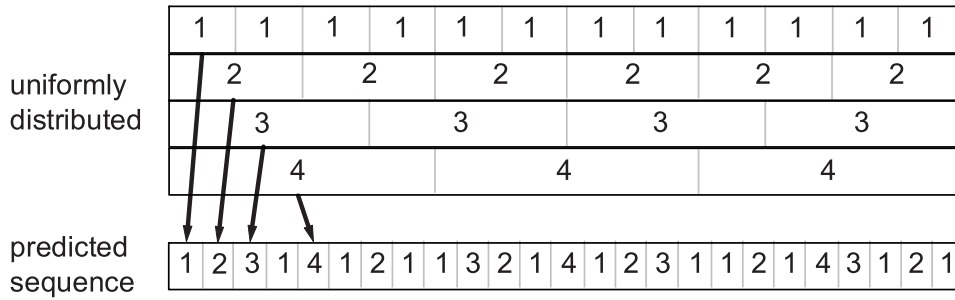


Figure 2.6: Prediction of a symbol sequence¹²

This figure makes clear why the symbols appear in descending order of their probabilities (always on condition of validity of the ZIPF law and uniform distribution of the symbols), therefore an analysis system would automatically correlate the symbols of such an input sequence with its internal indices representation, thus the demand (2.6) would be fulfilled¹³, and we are allowed to combine (2.4) and (2.5) as follows:

$$\mu_a \approx \sum_{k=1}^{x_{max}} k \cdot \frac{A}{k} = x_{max} \cdot A \text{ [flops]}. \quad (2.9)$$

Equation (2.7) shows the connection between the constants x_{max} and A (mostly x_{max} is given, A is sought for), but the general computation for large x_{max} (e.g.

¹²The decision between two "simultaneous" symbols, e.g. 3 and 1 on the third and fourth position of the sequence, was made randomly.

¹³By means of the already mentioned *Verbmobil* material in 3.4.1 is demonstrated that these considerations apply quite well to practical applications.

the *Verbmobil* material has $x_{max} = 4,936$, cf. p. 12) is not as simple as in the example calculation (2.8). With the help of the continuous function

$$f(x) = \frac{1}{1+x} \quad (2.10)$$

we can transform equation (2.7) into

$$\sum_{x=0}^{x_{max}-1} f(x) = \frac{1}{A} . \quad (2.11)$$

Now we are able to make use of the EULER-MACLAURIN summation formula (v. [Ma78]):

$$\sum_{x=0}^{x_{max}-1} f(x) = \int_0^{x_{max}-1} f(x)dx + \frac{f(0) + f(x_{max}-1)}{2} + S_{x_{max}-1} \quad (2.12)$$

with¹⁴

$$S_{x_{max}-1} = \sum_{p=1}^{p_{max}} \frac{B_{2p}}{(2p)!} \left(f^{(2p-1)}(x_{max}-1) - f^{(2p-1)}(0) \right) + R_{p_{max}} . \quad (2.13)$$

The derivatives of our function (2.10) are

$$f'(x) = -\frac{1}{(x+1)^2}, \quad f'''(x) = -\frac{6}{(x+1)^4}, \quad \dots$$

and generally

$$f^{(2p-1)} = -\frac{(2p-1)!}{(x+1)^{2p}} .$$

By means of this result we can transform (2.13) into

$$\begin{aligned} S_{x_{max}-1} &= \sum_{p=1}^{p_{max}} \frac{B_{2p}}{(2p)!} \left(-\frac{(2p-1)!}{x_{max}^{2p}} + (2p-1)! \right) + R_{p_{max}} \\ &= -\sum_{p=1}^{p_{max}} \frac{B_{2p}}{2p} x_{max}^{-2p} + \sum_{p=1}^{p_{max}} \frac{B_{2p}}{2p} + R_{p_{max}} \end{aligned}$$

With this term we can return to (2.12), and after calculating the indefinite integral of $f(x)$ we have

¹⁴ B_k are the BERNOULLI numbers; $p_{max} \in \{2, 3, \dots\}$, $R_{p_{max}}$ is a rest term.

$$\begin{aligned} \sum_{x=0}^{x_{max}-1} f(x) &= \left[\ln(1+x) \right]_{x=0}^{x_{max}-1} + \frac{1}{2}x_{max}^{-1} - \sum_{p=1}^{p_{max}} \frac{B_{2p}}{2p} x_{max}^{-2p} + \frac{1}{2} + \sum_{p=1}^{p_{max}} \frac{B_{2p}}{2p} + R_{p_{max}} \\ &\simeq \ln(x_{max}) + \frac{1}{2}x_{max}^{-1} - \sum_{p=1}^{p_{max}} \frac{B_{2p}}{2p} x_{max}^{-2p} + C \end{aligned}$$

with the EULER constant $C = 0,57721566\dots$. The sign " \simeq " means *almost equal to*: The relative deviation of the right side from the left is less than 10^{-6} for $x_{max} \geq 10$ and $p_{max} \in \{2, \dots, 12\}$. For instance with $p_{max} = 2$ this connection becomes¹⁵

$$\sum_{x=0}^{x_{max}-1} f(x) \simeq \ln(x_{max}) + \frac{1}{2}x_{max}^{-1} - \frac{1}{12}x_{max}^{-2} + \frac{1}{120}x_{max}^{-4} + C. \quad (2.14)$$

This result and the equation (2.11) are finally used to compute ZIPF's constant:

$$A \simeq \frac{1}{\ln(x_{max}) + \frac{1}{2}x_{max}^{-1} - \frac{1}{12}x_{max}^{-2} + \frac{1}{120}x_{max}^{-4} + C}.$$

For the *Verbmobil* material we get $A_{VM}(x_{max} = 4,936) \simeq 0.11011$ ¹⁶ and with (2.9) we estimate the average number of additions at $\mu_a \approx 544$.¹⁷ Here should be mentioned, that most implementation strategies need substantially more computing time to make available the probabilities p_x than the addition process consumes (v. 1.2.2 and 1.2.3), but the viewed synthesis algorithm accesses to the p_x as often as it carries out an addition, so that the numerical value μ_a does not decrease in significance.

2.3.2 N-Gram and Multigram Synthesis

When a symbol sequence is to be synthesized on the basis of n-grams with $n > 1$ using the statistic data of the symbol transitions that were trained by the algorithms explained in 1.2, we can act as follows:

- The first element of the chain x_r^1 is determined with the help of equation (2.3) and the procedure that was discussed in the previous paragraph:

$$x_r^1 = \min_{x=1\dots x_{max}} \left\{ x \mid \sum_{k=1}^x p_k \geq r^1 \right\}.$$

¹⁵The BERNOULLI numbers can be found for example in [Br96].

¹⁶SHANNON used the values $x_{max} = 12,366$ and $A_E = 0.1$ to approximate the word frequencies in the English language.

¹⁷Extensive synthesis experiments with the *Verbmobil* unigram data resulted in an actual value of $\mu_a \approx 422$. According to this deviation from the theoretical value it seems to be sufficient that A is approximated by $\frac{1}{\ln(x_{max})+C}$ if x_{max} is great enough.

- For the determination of the second element x_r^2 we must consider the transition probabilities from x_r^1 to any other symbol and make the decision randomly, similar to above:

$$x_r^2 = \min_{x=1\dots x_{max}} \left\{ x \mid \sum_{k=1}^x p_{x_r^1, k} \geq \frac{r^2}{p_{x_r^1}} \right\} , \quad (2.15)$$

in doing so we use the *multiplication theorem of probabilities*

$$p(A \wedge B) = p(A) \cdot p(B|A) .$$

- The i^{th} element of the sequence is produced by the general formula

$$x_r^i = \min_{x=1\dots x_{max}} \left\{ x \mid \sum_{k=1}^x p_{x_r^{i-n+1}, \dots, x_r^{i-1}, k} \geq \frac{r^i}{p_{x_r^{i-n+1}, \dots, x_r^{i-1}}} \right\} \text{ for } i > n . \quad (2.16)$$

In the nineties of the last century French scientists (FRÉDÉRIC BIMBOT and others) formed the notion *multigram* for an interpolated n-gram (v. [De96]). In plain language that means that the estimated conditional likelihood for the occurrence of an element within a symbol string does not merely depend on the n-gram probability of the maximum order n_{max} that rests on the $n_{max} - 1$ predecessors, but on all possible orders, unigram until n_{max} -gram, considering a given distribution. Each n-gram is correlated with a likelihood $\mathcal{L}(n)$, so that the multigram probability of a symbol with the index x^i is estimable consulting always the corresponding $n - 1$ antecedents:

$$\begin{aligned} p(x^i) &= \mathcal{L}(1)p_{x^i} + \mathcal{L}(2)\frac{p_{x^{i-1}, x^i}}{p_{x^{i-1}}} + \dots + \mathcal{L}(n_{max})\frac{p_{x^{i-n_{max}+1}, \dots, x^i}}{p_{x^{i-n_{max}+1}, \dots, x^{i-1}}} \\ &= \sum_{n=1}^{n_{max}} \mathcal{L}(n)\frac{p_{x^{i-n+1}, \dots, x^i}}{p_{x^{i-n+1}, \dots, x^{i-1}}} \text{ for } i \geq n_{max} , p_{x^i, \dots, x^j} = 1 \text{ for } i > j . \end{aligned} \quad (2.17)$$

The following table shows some examples for possible definitions of $\mathcal{L}(n)$:

definition	note
$\mathcal{L}_1(n) = \frac{1}{n_{max}}$	uniform distribution
$\mathcal{L}_2(n) = \begin{cases} 1 & \text{for } n=m \\ 0 & \text{else} \end{cases}$	simple m -gram
$\mathcal{L}_3(n) = \frac{1}{\sum_{m=1}^{n_{max}} \frac{n}{m}}$	stronger weight of less orders
$\mathcal{L}_4(n) = \frac{1}{\sum_{m=1}^{n_{max}} \frac{n_{max}-n+1}{m}}$	stronger weight of greater orders

Table 2.3: Example definitions of $\mathcal{L}(n)$

Now we want to illustrate this theoretical treatise, the estimation of the occurrence probabilities of the x_1, \dots, x_{max} presupposing a given predecessor sequence, by means of the following instance:

- The predicted sequence from 2.3.1 (v. figure 2.6) is to be the basis of conventional n-gram statistics with the maximum order $n_{max} = 3$
- The given predecessors are $x^{i-2} = 1$ and $x^{i-1} = 3$, so that we get the following statistic values in addition to table 2.2:

(x^{i-1}, x^i)	frequency	$\frac{p_{x^{i-1}, x^i}}{p_{x^{i-1}}}$
(3, 1)	3	0.75
(3, 2)	1	0.25
(3, 3)	0	0
(3, 4)	0	0

(x^{i-2}, x^{i-1}, x^i)	frequency	$\frac{p_{x^{i-2}, x^{i-1}, x^i}}{p_{x^{i-2}, x^{i-1}}}$
(1, 3, 1)	0	0
(1, 3, 2)	1	1
(1, 3, 3)	0	0
(1, 3, 4)	0	0

Table 2.4:
Multigram example:
Estimated conditional
transition probabilities

- As the multigram likelihood we choose the variant $\mathcal{L}_3(n)$ of the table 2.3, so we concretely have

n	$\mathcal{L}(n)$
1	$\frac{6}{11} = 0.\overline{54}$
2	$\frac{3}{11} = 0.\overline{27}$
3	$\frac{2}{11} = 0.\overline{18}$

Table 2.5:
Multigram example:
Concrete values for $\mathcal{L}(n)$

Now we use the equation (2.17) which is interpretable as a two-dimensional geometric probability, so that its nature can be illustrated as follows:

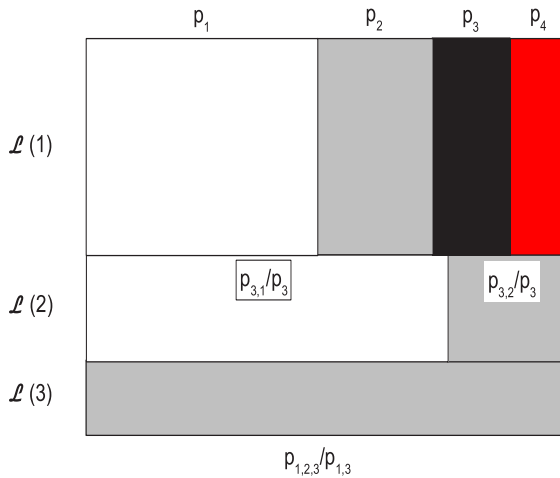


Figure 2.7:
Multigram example:
Graphic interpretation

E.g. $p(x_1)$ corresponds to the white part of the unit square area and is consequently calculable by adding the individual rectangular partial areas, and finally we get

x^i	$p(x^i)$	
1	$0.\overline{54} \cdot 0.48 + 0.\overline{27} \cdot 0.75 + 0.\overline{18} \cdot 0 = \mathbf{0.46}$	Table 2.6: Multigram example: Computation of the numeric values of the $p(x^i)$
2	$0.\overline{54} \cdot 0.24 + 0.\overline{27} \cdot 0.25 + 0.\overline{18} \cdot 1 = \mathbf{0.38}$	
3	$0.\overline{54} \cdot 0.16 + 0.\overline{27} \cdot 0 + 0.\overline{18} \cdot 0 = \mathbf{0.09}$	
4	$0.\overline{54} \cdot 0.12 + 0.\overline{27} \cdot 0 + 0.\overline{18} \cdot 0 = \mathbf{0.07}$	

These interpolated symbol probabilities can be used for language modeling, concretely for speech recognition (v. 2.1)¹⁸, as well as for synthesis of symbol chains. The determination of an element of the synthesis string can be carried out for $i \geq n_{max}$ with the help of the modified formula (2.3) and the likelihood $p(x^i)$ that is defined in (2.17):

$$x_r^i = \min_{x=1 \dots x_{max}} \left\{ x \left| \sum_{k=1}^x \left(\mathcal{L}(1)p_k + \sum_{n=2}^{n_{max}} \mathcal{L}(n) \frac{p_{x^i-n+1, \dots, x^i-1, k}}{p_{x^i-n+1, \dots, x^i-1}} \right) \geq r^i \right. \right\}. \quad (2.18)$$

This algorithm's emission of the value x_r generally needs $x_r \cdot n_{max}$ multiplications and accesses to likelihood values (v. 1.2), $x_r \cdot (n_{max} - 1)$ additions within the rows and x_r additions of these rows. We want to assume that $x_r \approx \mu_a \approx A \cdot x_{max}$ (cf. equation (2.9)), so we get the following approximation of the computation expenditure (considering footnote 11 of this chapter):

operation	expression	
flops	$2A \cdot x_{max} \cdot n_{max}$	Table 2.7: Expected computation expenditure of multigram synthesis
accesses to p_\bullet	$A \cdot x_{max} \cdot n_{max}$	

The low share of NZEs of the transition probability tensors $\underline{P}_n, n > 1$ that was especially discussed in 1.2.3 entails that most of the addends (actually the values $p_{x^i-n+1, \dots, x^i-1, k}$) of the equation (2.18) disappear¹⁹, thus a large part of the computing time could be saved if the concrete algorithm would not have to search for such elements²⁰. The average number of addends unequal to zero in the n^{th} layer is $\frac{\sigma(\underline{P}_n)}{\sigma(\underline{P}_{n-1})}$ with $\sigma(\underline{P}_0) = 1$, an instance for its progression by means of the mentioned analysis material of *Verbmobil* is shown below (also cf. table 1.8):

¹⁸The great advantage of multigrams is that on condition that the training material had the maximum order n_{max} word sequences of the length $l \leq n_{max}$ that never appeared in the training material are correlated with a small likelihood, thus they generally are recognizable; equation (2.2) would always assign the estimated likelihood zero to such strings.

¹⁹Already the example above (tables 2.4-2.6) illustrates this effect.

²⁰This is e.g. supported by the *compact storage algorithm*.

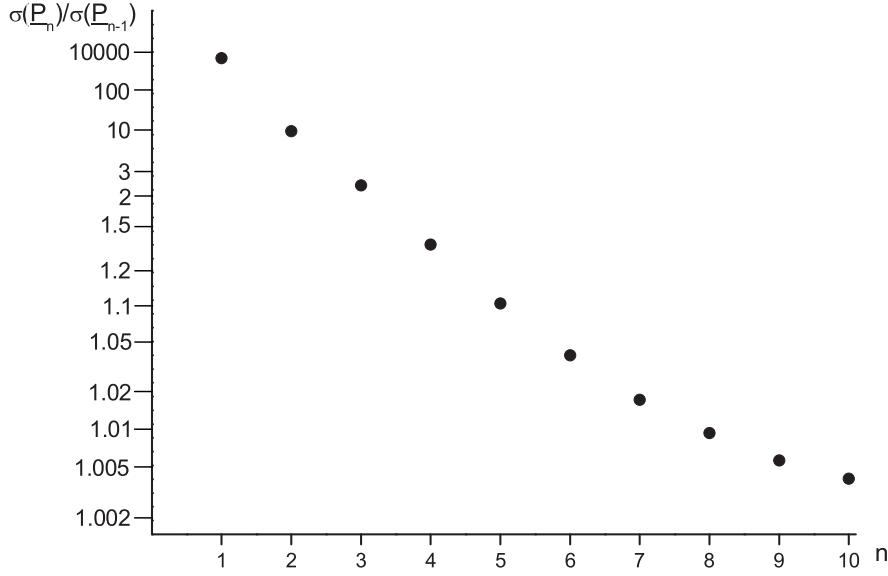


Figure 2.8: *Verbmobil*: Average number of different possible successors of a given sequence with $n - 1$ elements

Another possibility to accelerate the synthesis algorithm is the separation of the multigram- ($\mathcal{L}(n)$) and n-gram probabilities (P_n) and their treatment in different random processes:

$$n_r^i = \min_{n=1 \dots n_{max}} \left\{ n \mid \sum_{m=1}^n \mathcal{L}(m) \geq r_1^i \right\} ,$$

$$x_r^i = \min_{x=1 \dots x_{max}} \left\{ x \mid \sum_{k=1}^x \frac{p_{x^i-n_r^i+1, \dots, x^i-1, k}}{p_{x^i-n_r^i+1, \dots, x^i-1}} \geq r_2^i \right\} .$$

That means that first a horizontal stripe of the unit square according to figure 2.7 is randomly estimated (with the likelihood $\mathcal{L}(1)$ we get the first one $\Rightarrow n_r^i = 1$, with $\mathcal{L}(2)$ the second one $\Rightarrow n_r^i = 2$, and so on). Within the n_r^i th stripe we calculate the symbol index in the customary way. Therefore the final expression of $p(x^i)$ is the sum of the respective multigram probabilities $\mathcal{L}(n)$ multiplied by the conditional likelihood $\frac{p_{x^i-n+1, \dots, x^i-1, x^i}}{p_{x^i-n+1, \dots, x^i-1}}$, so equation (2.17) is fulfilled for this implementation method, too. The estimation of the computing time results in the following needed operations: less than n_{max} additions to determine n_r^i , and to calculate x_r^i e.g. $\mu_a \approx A \cdot x_{max}$ additions and accesses to the probabilities p_\bullet if $n_r^i = 1$ otherwise substantially less. So the expected calculation expenditure is generally less than

operation	expression
flops	$A \cdot x_{max} + n_{max}$
accesses to p_\bullet	$A \cdot x_{max}$

Table 2.8:

Expected computation expenditure of multigram synthesis using an efficient implementation strategy

Chapter 3

Experiments

By means of some example experiments the theoretical treatises of the previous chapters are to be illustrated. As already mentioned for this purpose the statistic program *synther*, that is developed by the author, is used; a detailed (German) manual can be found in the appendix. To duplicate easier the following steps, some system requirements are to be prearranged:

- The used operating system is *MicrosoftTM WindowsTM NT 4.0*.
- The CD-Rom drive is allocated the drive letter **E:**, and the compact disc that is enclosed with this work is put in.
- The program *synther* is installed by executing

```
E:\syntherz.exe
```

The *synther* path **C:\synther** should be accepted.

- `cd C:\synther`

In the following we want to use the input prompt `>` to mark *DOS* commands that are executed in the working directory **C:\synther**, for instance

```
>notepad dateien.txt
```

- We need the mathematics program *MatlabTM Version 5* to extract some additional information from the n-gram tensors. The m-files that are included in **C:\synther\matlab** must be copied into the **bin** path of the *Matlab* master directory to make available the needed functions.
- *Matlab* commands are distinguished from *DOS* commands by the introducing prompt `>>`, e.g.

```
>>plot(log(1:100))
```

3.1 The Computing Time Behaviour of the Compact Storage Algorithm

The Microsoft Sound, already mentioned in 1.2.5, is to be the basis material of trigram statistics:

```
>del quelle\*
>copy E:\quellen\microsoft\TheMicrosoftSound.wav quelle
>attrib -R quelle\*
```

The *Matlab* function `wav2txt.m` converts the contents of a wav-file into an ASCII-file that can be interpreted by *synther*:

```
>>wav2txt('TheMicrosoftSound');
```

The file list can be built as follows

```
>del quelle\*.wav
>del dateien.txt
>dir /b quelle >> dateien.txt
```

Now we want to carry out a trigram analysis of this sampled sound material (`-o 3` means order $n = 3$):

```
>synther -t -n -o 3
```

During *synther*'s work we are able to inspect its status by opening the analysis statistics file `C:\synther\statstic.txt` that contains a list of working time values in seconds that are written everytime hundred symbols are processed¹. That means we can estimate the maximum entry number of `statstic.txt` by dividing the sum of all symbols that are processed $s_{p,max}$ by 100. In the current case we have `TheMicrosoftSound.wav`'s file size of 135,876 bytes that is equivalent to the maximum symbol number $s_{p,max}$ ² deducting a file header of 872 bytes,

```
>>spMax=135004;
```

so the expected maximum number of entries of `statstic.txt` is 1,350. With the help of

```
>>tw=dlmread('C:\synther\statstic.txt','\n');
```

its contents are converted into the vector `tw` that represents the working time t_w in dependence on the number of processed data s_p . We get the percentage of processed data as follows:

¹The number of processed symbols is to be designated as s_p .

²in view of its features that are mentioned in 1.2.5

```
>>size(tw,2)*100/spMax*100
```

Now we are also able to display the time progression in dependence on the processed data:

```
>>plot(tw)
```

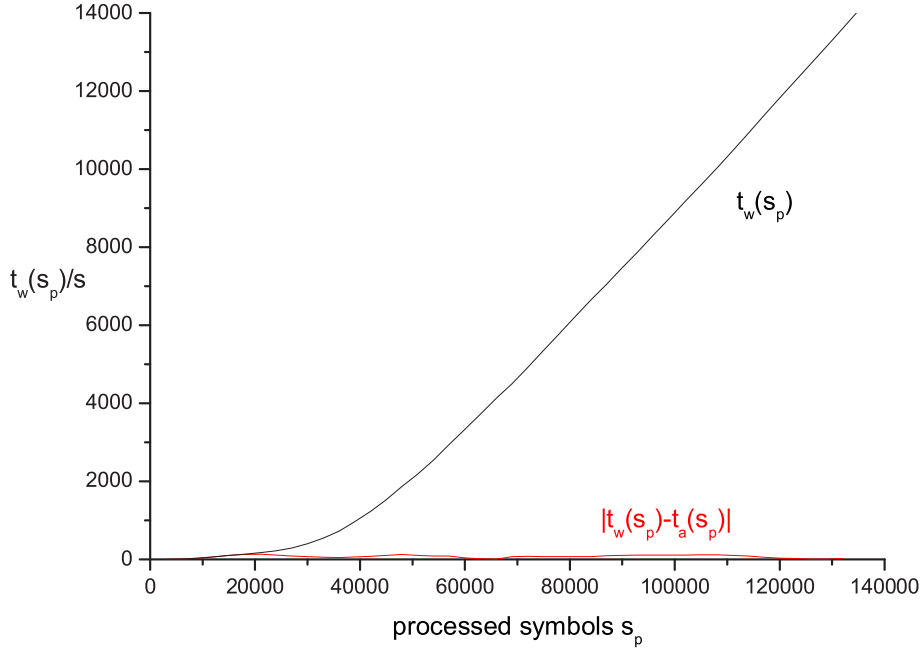


Figure 3.1: Working time behaviour of the compact storage algorithm

This progression especially results from the data handling of this algorithm: In 1.2.3 we have seen that new data sets are always written to the right end of the tensor table. The search for existing entries also starts at the right end, thus the searching act retards continuously. A very simple model is the following: In the beginning the algorithm works so fast that we can approximate its progression until $s_{p,1}$ by a zero function. Between $s_{p,1}$ and $s_{p,2}$ we take the increasing working time of inserting a new data set into account, the total time for processing $s_p - s_{p,1}$ symbols is of course the sum of these (linearly) ascending individual time components: Therefore we assume in this part a quadratic progression of $t_w(s_p)$. After reaching $s_{p,2}$ the tensors are saturated: Most entries already occurred, only their frequency values are increased, but the sizes of the tensors stay nearly the same, so we model $t_w(s_p)$ by a linear function. To sum it up we introduce the approximate function $t_a(s_p) \approx t_w(s_p)$:

$$t_a(s_p) = \begin{cases} 0 & \text{for } 0 \leq s_p \leq s_{p,1} \\ t_q(s_p) = \frac{1}{f_q}(s_p - s_{p,q})^2 & \text{for } s_{p,1} \leq s_p \leq s_{p,2} \\ t_l(s_p) = \frac{1}{f_l}(s_p - s_{p,l}) & \text{for } s_{p,2} \leq s_p \leq s_{p,max} \end{cases} \quad (3.1)$$

When we consider the parity of the following boundary values:

$$t_q(s_{p,1}) = t'_q(s_{p,1}) = 0, \quad t_q(s_{p,2}) = t_l(s_{p,2}) \quad \text{and} \quad t'_q(s_{p,2}) = t'_l(s_{p,2})$$

we can transform (3.1) into

$$t_a(s_p) = \begin{cases} 0 & \text{for } 0 \leq s_p \leq s_{p,1} \\ \frac{1}{f}(s_p - s_{p,1})^2 & \text{for } s_{p,1} \leq s_p \leq s_{p,2} \\ \frac{2}{f}(s_{p,2} - s_{p,1})(s_p - \frac{s_{p,1} + s_{p,2}}{2}) & \text{for } s_{p,2} \leq s_p \leq s_{p,max} \end{cases}$$

When we model the t_w curve of the above example

```
>>clear
>>tw=dlmread('E:\tensoren\TheMicrosoftSound\statstic.txt','\n');
```

by means of the constants $s_{p,1} = 13,000$, $s_{p,2} = 55,000$ and $f = 588$ kHz we get an approximation error $|t_w(s_p) - t_a(s_p)| < 150$ s:

```
>>sp1=13000; sp2=55000; f=588000;
>>sp1=sp1/100; sp2=sp2/100; f=f/100^2;
>>%time values were only written each hundredth processed symbol
>>sp=(1:size(tw,2));
>>tq=1/f*(sp-sp1).^2;
>>t1=2/f*(sp2-sp1)*(sp-(sp1+sp2)/2);
>>ta(1:sp1)=0;
>>ta(sp1:sp2)=tq(sp1:sp2);
>>ta(sp2:size(tl,2))=t1(sp2:size(tl,2));
>>max(abs(tw-ta))
```

The graph of the error function $|t_w(s_p) - t_a(s_p)|$ looks like displayed in figure 3.1:

```
>>plot(abs(tw-ta))
```

The free parameters $s_{p,1}$, $s_{p,2}$ and f strongly depend on the maximum analysis order n_{max} and the basis material characteristics, in particular x_{max} , $\sigma(\underline{P}_n)$ and the length of the processed source files³. The frequency f is directly proportional to the computer velocity.

³When the source file length $\gg n_{max}$ we generally must use a huge number of files to get enough statistic data, so we have essentially less $(n+1)$ -grams than n -grams. For instance the number of 10-grams of the mentioned *Verbmobil* material amounts to only 63% of the unigram number (v. 3.2).

3.2 Determination of some Parameters of the *Verbmobil* Material

The first step to determine parameters like x_{max} , $\sigma(\underline{P}_n)$ and others on the basis of the *Verbmobil* texts is of course an n-gram analysis of this material. In view of the huge number of processed symbols (177,781 words) it is intelligible that the computing time of 10-gram statistics may amount to several hours. Therefore paragraph 3.2.1 can be skipped by coping the analysis results from the CD, either with the maximum order 3:

```
>copy E:\tensoren\verbmobil\o3_8247f\* .
```

or maximum order 10:

```
>copy E:\tensoren\verbmobil\o10_8247f\* .4
```

respectively

```
>copy E:\tensoren\verbmobil\o10_1f\* .5
```

```
>attrib -R *
```

3.2.1 Analysis

The 8,247 *Verbmobil* source files are taken

```
>del quelle\*
>copy E:\quellen\verbmobil\text_8247f\* quelle
>attrib -R quelle\*
```

and the source file list is written

```
>del dateien.txt
>dir /b quelle >> dateien.txt
```

Now we can start the analysis of the order $n = 10$

```
>synther -t -n -o 106
```

⁴There are no **org**-files in this directory, so some functions could not work.

⁵The basis material of the analysis data of this directory was only one big ASCII-file that included 8,247 individual texts, thus the problems that were mentioned in footnote 3 disappear, however n-sequences that reached beyond the end of a text were counted still they did not occur in reality.

⁶The percentage of processed data can be checked like explained in paragraph 3.1 with `spMax=177781`

3.2.2 Parameter Extraction

- (a) **total number of all words (symbols)**

```
>>sum(Tx(1))
```

- (b) **total number of all sequences of the length n**

```
>>n=3;  
>>sum(Tx(n))
```

- (c) **number of different words (x_{max})**

```
>>size(Tx(1),1)
```

- (d) **number of different sequences of the length n ($\sigma(\underline{P}_n)$)**

```
>>size(Tx(n),1)
```

- (e) **occurrence frequency of the word *ich***

```
>synther -f ich
```

- (f) **occurrence frequency of any symbol**

If we investigate other symbols than words (for instance MIDI-notes) we can take a look at their intern ASCII-representation:

```
>notepad symbole.txt
```

Now we use the desired line and execute the above procedure (e.g. the MIDI-event *deltatime=96, note on, pitch=G4, key velocity=64, MIDI-channel=6*)

```
>synther -f zeit:96___kanal:6___note0n:67___vel:647
```

- (g) **frequency of the word sequence *alles klar***

(Other symbol sequences are handled analogously.)

```
>synther -f "alles klar"
```

⁷The connection between the note number (in this case 67) and the pitch is explained for example in [La97].

3.3 Comparison of the Frequency Distributions of Notes in Musical Pieces of Different Epochs

This application is to be demonstrated by means of the example of paragraph 2.2. We want to compare the *Piece for Piano* (op. 33a) of ARNOLD SCHÖNBERG with the already used BACH fugue (cf. figure 2.4). The analysis of the MIDI-source files is prepared like described in 3.2.1, merely two lines differ:

```
>copy E:\quellen\MIDI\schoenberg.mid quelle
```

respectively

```
>copy E:\quellen\MIDI\bach\fugue.mid quelle
```

and

```
>synther -t -n -m -o 1
```

With the help of command 3.2.2 (f) and the note numbers 21...108 (the key register of a piano) we can construct the following frequency distributions:

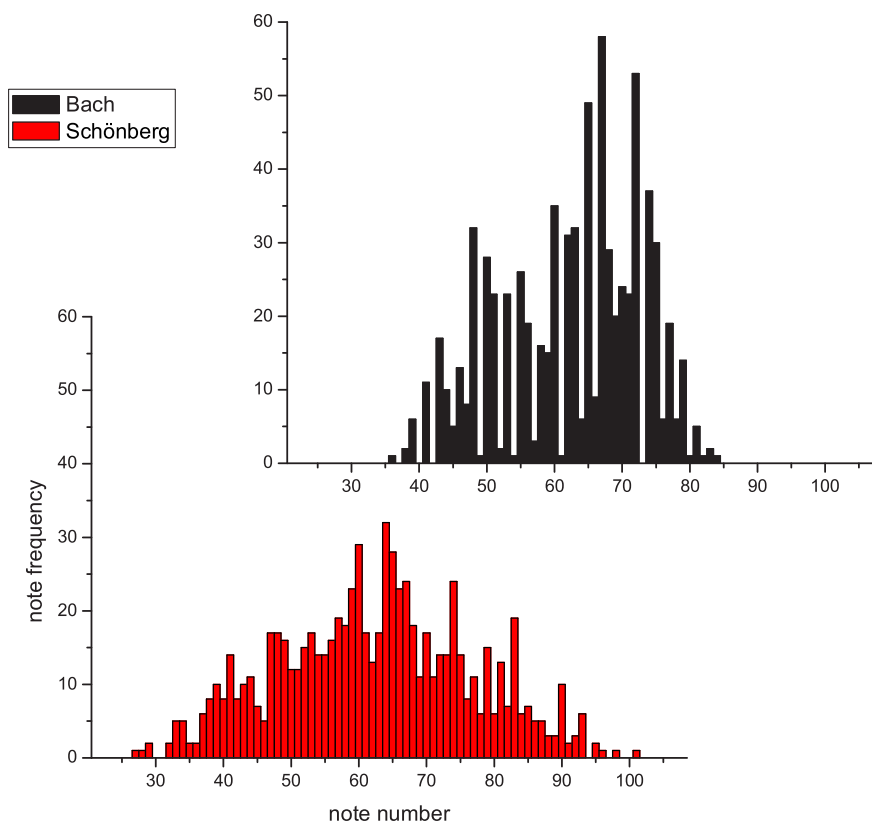


Figure 3.2: Note frequency distributions of BACH and SCHÖNBERG

To simplify matters these distributions can be loaded, too:

```
>>sb128=dlmread('E:\quellen\MIDI\schoenberg.txt','\n');
```

resp.

```
>>b128=dlmread('E:\quellen\MIDI\bach\fugue.txt','\n');
```

The comparability of these both pieces is only guaranteed if the number of played notes is similar. This requirement is met for the chosen pieces, we can simply check it by feeding

```
>>sum(sb128), sum(b128)
```

The declaration that was made at the beginning of paragraph 2.2 is to be tested now. The total frequencies of the twelve tones of the scale can be computed by

```
>>for I=1:12;  
    sb12(I)=sum(sb128(I:12:128));  
    b12(I)=sum(b128(I:12:128));  
end  
>>plot(sb12,'r'), hold, plot(b12,'k')
```

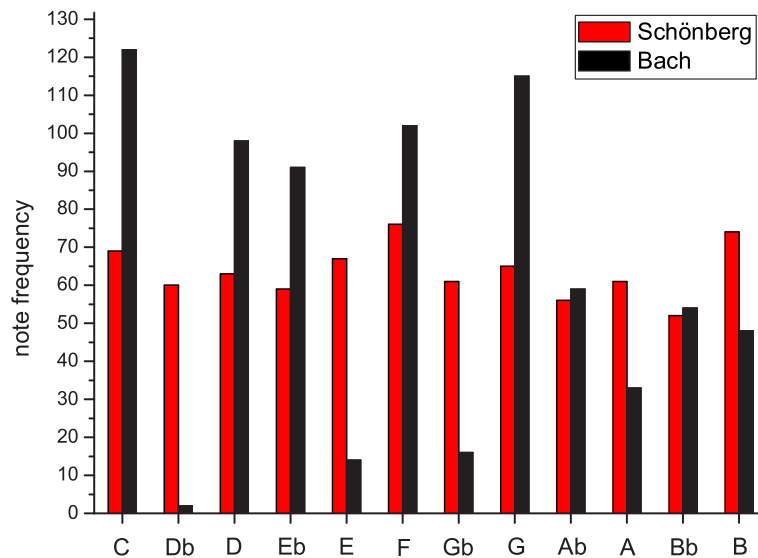


Figure 3.3: Note occurrence frequencies of SCHÖNBERG and BACH

By means of these results we can confirm the forecasted almost uniform distribution of SCHÖNBERG's example work. BACH's confinement to the fundamental key Eb major⁸ is reflected in the big frequencies of the scale's tones Eb, F, G, Ab, Bb, C and D.

⁸cf. the score example in figure 2.4

3.4 Synthesis Experiments

3.4.1 Verification of the Zipf condition

The determination of the average additions number assumed that the occurrence probabilities p_x shape a monotonically decreasing function in dependency on the symbol index x . Practical applications show that it is not totally right, but through the recursive arithmetical rule

$$p_x^{(0)} := p_x ,$$

$$p_x^{(j)} := \frac{1}{2}(p_x^{(j-1)} + p_{x+1}^{(j-1)}) \quad \text{for odd } x \text{ and } j \in \{1, 2, \dots, j_{max}\} \text{ and}$$

$$p_x^{(j)} := \frac{1}{2}(p_{x-1}^{(j)} + p_{x+1}^{(j)}) \quad \text{for even } x \text{ and } j \in \{1, 2, \dots, j_{max}\}$$

that is implemented in the *Matlab*-function `smooth.m`

```
>>jMax=100;  
>>px=smooth(Tx(1),jMax);9
```

the p_x -graph is smoothed a little and we will realize that its general progression is monotonically decreasing:

```
>>plot(px(1:1000));
```

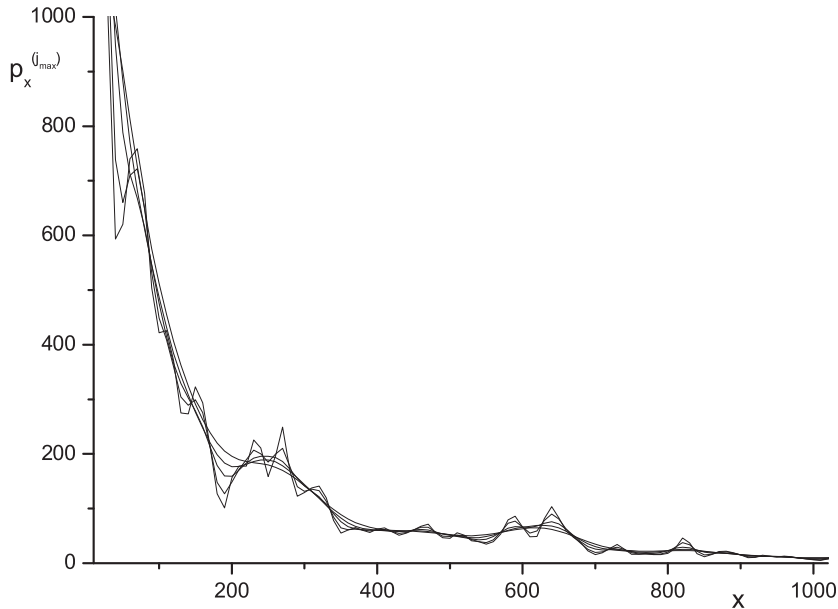


Figure 3.4: Recursive smoothing of p_x for different j_{max}

⁹This step requires the file preparations that are explained in 3.2(.1).

3.4.2 Determination of the Average Number of Additions¹⁰

For that purpose the *Matlab* function `synth1.m` that simulates the synthesis routine of *synther* was prepared. The arguments of `synth1.m` are the T_1 -tensor (v. 1.2.3) and the number of synthesis cycles (number of synthesized symbols). Of course the estimated values of μ_a rest on a GAUSSIAN distribution: Its standard deviation decreases when the cycles number rises. Therefore a huge value is chosen:

```
>>cycles=100000;  
>>synth1(Tx(1),cycles)
```

3.4.3 Text synthesis based on n-grams

The intern representation of any symbol is an ASCII-word (e.g. v. 3.2.2 (f)), therefore it is obvious that the manifold synthesis characteristics investigated in paragraph 2.3 for the present are illustrated by means of synthesized word chains (text sequences). As the basis tensors we use a *Verbmobil* analysis of the maximum order 10 and 17778 input words:

```
>copy E:\tensoren\verbmobil\o10_1f_10p\* .  
>attrib -R *
```

First we start a simple unigram synthesis on the basis of equation (2.3):

```
>synther -s -so 1 -z 100
```

The result could look as follows:

```
>notepad synth.txt
```

```
...wir          den  
ist            der  
wir           Montag  
siebenundvierzigste  ich  
ne            Dienstag  
eine         also  
dann        das  
und         auf  
Uhr        von  
wenn      w"uerde...
```

When the synthesis order is raised to $n = 3 > 1$ (2.15) and (2.16) are brought into action:

¹⁰cf. paragraph 2.3.1 and especially footnote 17 of chapter 2

```
>synther -s -so 3 -z 100
```

The resemblance to realistic scheduling dialogs is remarkable:

```
...an                bin
  <"ah>              ich
  sollen             schon
  wir               ausgeplant
  <"ah>             vom
  <"ahm>            vierundzwanzigsten
  von              bis
  da              zum
  aus            achtzehnten
  dann          November...
```

The naturalness of the emitted word sequence intensifies when we choose order $n = 10$:

```
...f"unfzehn        das
  Uhr              w"are
  w"urde           schon
  ich             mal
  vorschlagen     ein
  wenn           Termin
  wir            klar
  uns           ja
  da           okay
  treffen     also...
```

3.4.4 Text synthesis based on multigrams

Since the sensitive effects of the multigram synthesis are hardly detectable in view of the roughness of *Verbmobil*'s spontaneous speech it is advisable to use High German basis material for the next experiments, for instance FRANZ KAFKA's work *Die Verwandlung*:

```
>copy E:\tensoren\kafka\* .
>attrib -R *
```

An example 10-gram analysis demonstrates that syntheses of great orders have a tendency to copy the original:

Daraufhin ging der Herr tatsächlich sofort mit langen Schritten ins Vorzimmer; seine beiden Freunde hatten schon ein Weilchen lang mit ganz ruhigen Händen aufgehört und hüpften ihm jetzt geradezu nach, wie in Angst, Herr Samsa könnte vor ihnen ins Vorzimmer eintreten und die Verbindung mit ihrem Führer stören.

Now we want to check the effect of a multigram synthesis with the maximum order $n_{max} = 10$ (cf. table 2.3):

uniform distribution ($\mathcal{L}_1(n)$):

```
>synther -s -so 10 -z 100 -sa 1
```

Warum ging denn die Servietten und die nicht er in einer Berufskrankheit besser als infolge war es stören mußte, und die Mutter er schien durch sich neue war keine und Speisen dagegen schmeckten ihm verlassen haben.

stronger weight of less orders ($\mathcal{L}_3(n)$):

```
>synther -s -so 10 -z 100 -sa 3
```

Dabei wachte und quälten ihn. Die Trotzdem, unbeweglich auf blieb um Das ist die Ruhe wieder Möglichkeit dessen Verwirklichung nicht zu leisten; alles er Laufe ließ Mutter Nächte und Tage lang waren und Gregor gegessen hatten, entfalteteten die Servietten und offenbar überkroch nun sah der Stadt.

stronger weight of greater orders ($\mathcal{L}_4(n)$):

```
>synther -s -so 10 -z 100 -sa 4
```

Da öffnete sich die Tür des Schlafzimmers, und da sah auch die durch das viele Hungern verursachte Schwäche machten es ihm unmöglich, sich zu bewegen.

3.4.5 Music synthesis

Still the results of music synthesis are merely difficultly representable in writing, here is to be referred to the extensive MIDI source material on the CD (v. its table of contents E:\index.txt). The explanation of the several additional options (in particular the MIDI interface switch `-m`) can be found in the *synther* manual in the appendix. Also a whole string of synthesis results are available.

Appendix A

A.1 German Manual of the Statistics Program Synther

Synther Version 2.12 - Bedienungsanleitung
=====

Inhalt

1. Synther im WWW
2. Analyse
 - 2.1. Voraussetzungen
 - 2.2. Analyseparameter
 - 2.2.1. MIDI-Analyseparameter
 - 2.3. Datenmanagement
3. Synthese
 - 3.1. Allgemeines
 - 3.2. Syntheseparameter
4. Parameter
5. Fehler
 - 5.1. Dokumentierte Fehlermeldungen
 - 5.2. Weitere Fehlerursachen
 - 5.2.1. Fehler bei der Analyse
 - 5.2.2. Fehler bei der Synthese
 - 5.3. Tipps

1. Synther im WWW

Aktuelle Synther-Version:

<http://www.synther.de/download/software/syntherz.exe>

Aktuelle Bedienungsanleitung:

<http://www.synther.de/sy/index.html>

Studienarbeit von David Sündermann (Synthers theoretischer Background):

<http://www.synther.de/sy/studienarbeit.ps>

2. Analyse

2.1. Voraussetzungen

Die zu analysierenden Dateien müssen Sie in das Unterverzeichnis *quelle* des Pfades, in dem sich *synther(.exe)* befindet, kopieren. Nun müssen in die Datei *dateien.txt* die Namen aller zu analysierenden Dateien geschrieben werden, getrennt durch Zeilenumbruch, nach dem letzten Namen ebenfalls ein Zeilenumbruch. Der Inhalt von *dateien.txt* kann beispielsweise wie folgt aussehen:

```
"example1.mid
example2.mid
example3.mid
"
```

(ohne Anführungszeichen)

Sollte es sich um eine große Menge von Quelldateien handeln, kann diese Namensliste auch automatisch erstellt werden: z.B. mit Hilfe des Befehls *ls quelle >> dateien.txt* (UNIX) oder *dir /b quelle >> dateien.txt* (DOS) im Synther-Verzeichnis.

2.2. Analyseparameter

Um die Analyse durchzuführen, muss *synther* mit dem Parameter *-t* (Tensoren bearbeiten) gestartet werden. Sollte es sich jedoch um die erste derartige Analyse handeln, sind also noch keine Tensoren vorhanden, muss zusätzlich die Option *-n* (wie neue Tensoren) verwendet werden:

```
synther -t -n
```

Nun wird die Analyse mit ihren Standardwerten durchgeführt, die jedoch ebenfalls mittels Optionen beeinflusst werden können:

-o o Mit der natürlichen positiven Zahl *o* bestimmen Sie die maximale Tensorenordnung der Analyse, die später als höchstmögliche Syntheseordnung entscheidenden Einfluss auf die Eigenschaften der Synthese hat. Der Standardwert für *o* ist 10.

z.B. *synther -t -n -o*

`-a` bewirkt eine ausführlichere Bildschirmausgabe, beispielweise wird der Name der z.Z. bearbeiteten Datei eingeblendet. Ist dieser Schalter nicht aktiv, gibt Synther nach jedem hundertsten bearbeiteten Symbol Zeitwerte am Bildschirm aus, die, in eine Datei umgeleitet, z.B. mittels Matlab weiterverarbeitet werden können.

`-f f` gibt die Häufigkeit der Symbolfolge `f` am Bildschirm aus. Zu beachten ist, dass sie in Anführungszeichen geschrieben werden muss. Wenn nur nach der Häufigkeit eines einzelnen Wortes gesucht werden soll, können die Anführungszeichen entfallen.

z.B. `synther -f "ich liebe dich"`
oder `synther -f Rockmusik`

2.2.1. MIDI-Analyseparameter Das MIDI-Interface wird mit der Option `-m` eingeschaltet. Nun werden alle Quelldateien als MIDI-Dateien interpretiert:

`synther -t -n -m`

Die MIDI-Schnittstelle wiederum verfügt ebenfalls über einige Optionen, die im Folgenden erläutert werden sollen:

`-v` Ist dieser Schalter nicht aktiv, wird die Velocity (=Anschlaggeschwindigkeit) auf den Wert `v=64` eingestellt, was einer starken Symbol- und somit Datenreduktion entspricht (sonst existieren 128 verschiedene Anschlagwerte).

`-cm` bedeutet, dass *Channel Messages* (oder MIDI-Controller) wie z.B. Fußpedale berücksichtigt werden sollen, was zwar i.A. die Datenmenge erhöht, jedoch manchmal obligatorisch ist, da sonst rhythmische Probleme beim Synthesergebnis vorliegen können.

`-c` legt den MIDI-Channel aller eingehenden Daten auf 1, was zwar den Datenumfang reduziert, jedoch multitimbrales Komponieren (=mit mehreren gleichzeitig erklingenden Instrumenten) unmöglich macht.

`-p` schaltet die Interpretation von *Pitch-Wheel-Informationen* ein

`-pc` ermöglicht die Berücksichtigung von *Program-Change-Meldungen*

`-q q` aktiviert die Zeitquantisierung, und zwar auf jedes ganzzahlige Vielfache von `q`, wodurch sich die Symbolvielfalt drastisch reduziert lässt. Ist `-q` nicht im Einsatz, entspräche das einer Verwendung von `-q 1`, was jedoch einer MIDI-Standard-bedingten Zeitquantisierung auf Vielfache von $1/384$ einer Viertelnote entspricht, was z.B. bei einem Tempo von 140 bpm einer Auflösung von $1/384/140*60 \text{ s} = 1/2^{7/7} \text{ s} = 1,12 \text{ ms}$ gleichkommt.

2.3. Datenmanagement

Wenn mehrere Quellen analysiert werden sollen, ist es wichtig, die Tensoren in verschiedenen Verzeichnissen abzulegen. Da zu einer Datenbank mehr als nur die Tensordateien selbst gehören, ist es zweckmäßig, das komplette Verzeichnis inkl. *synther(.exe)* und anderer zum Datenaustausch nicht notwendigen Dateien zu transferieren, weil sie meist nur einen kleinen Teil der Speicherkapazität ausmachen. So könnte eine vernünftige Verzeichnisstruktur wie folgt aussehen:

```
synther/Texte/Zeitungsartikel/*.*
synther/Texte/Roman/*.*
synther/Texte/Gedichte/*.*
...
synther/MIDI/Chopin/*.*
synther/MIDI/Bach/*.*
synther/MIDI/Beatles/*.*
...
synther/Klaenge/Strassenlaerm/*.*
...
```

Da es in bestimmten Fällen jedoch von Interesse sein kann, nur die informationstragenden Daten zu übermitteln, folgt nun eine Liste der benötigten Dateien:

```
org1.bin, org2.bin usw.
T1.bin, T2.bin usw.
symbole.txt
MIDI.bin (im Falle von -m)
```

3. Synthese

3.1. Allgemeines

Die Synthese erfolgt standardmäßig mit dem Befehl *synther -s*, ihre Eigenschaften können jedoch durch die unter 3.2. aufgeführten Parameter verfeinert werden. Nach erfolgter Synthese findet man das Ergebnis in der Datei *synth.txt* und im Falle, dass die MIDI-Schnittstelle verwendet wurde (*synther -s -m*), in *synth.mid*. Nach jedem synthetisierten Symbol wird sein Index am Bildschirm ausgegeben.

3.2. Syntheseparameter

-s gibt Synther die Anweisung, dass es nun als Synthetisator zu agieren hat.

-so so steht für die Syntheseordnung, den wichtigsten Parameter bei der Synthese. Sie stellt einen Grad für die Konfusion, gleichzeitig auch für die Originaltreue dar. Sie ist eine natürliche Zahl, die der Relation

0<so<=o genügen muss; o ist die Ordnung der Analyse (s. 2.2.). Die Standardeinstellung ist so=2

Bsp.: *synther -s -so 3*

-sa sa (Synthese Art) beeinflusst ebenfalls entscheidend das Synthesergebnis:

- sa=0, sa=2 heißt, dass nur Markovquellen der Ordnung so bei der Synthese berücksichtigt werden,
- sa=1 bezieht alle Ordnungen zwischen 1 und so gleichberechtigt mit ein,
- sa=3 benutzt zwar alle Ordnungen, jedoch mit stärkerer Wichtung der niedrigeren Ordnungen,
- sa=4 wie sa=3, jedoch werden die höheren Ordnungen stärker gewichtet.

-z z ist die Zahl der Synthesezyklen, bestimmt also, wie viele Symbole der Ausgabestrom enthalten soll. z entspricht z.B. der Anzahl der Wörter in einem Text oder ca. dem Doppelten der Noten eines Musikstückes (-m), der Grund liegt darin, dass zur Darstellung einer Note vom MIDI-Standard zwei Symbole verwendet werden: eines für den Notenanfang und eines für das Ende. Voreingestellt ist z=30.

Bsp.: *synther -s -z 300*

-m aktiviert das MIDI-Interface für die Synthese.

-tmp tmp bestimmt das Tempo des synthetisierten MIDI-Titels in bpm. tmp ist dabei eine natürliche Zahl > 0.

4. Parameter

Hier finden Sie noch einmal eine Schnellübersicht der *synther* zu übergebenen Parameter. Nähere Informationen finden Sie in den Kapiteln Analyseparameter (2.2.) und Syntheseparameter (3.2.).

- a ausführliche Bildschirmausgabe
- c Alle MIDI-Events auf Kanal 1 umleiten
- cm Channel Messages akzeptieren
- f f Gibt die Häufigkeit der Wortfolge f aus
- h Gibt einen Hilfeverweis am Bildschirm aus
- help s. -h
- m MIDI-Schnittstelle aktivieren
- n neue Tensoren erzeugen
- o o Analyseordnung
- p Pitch Wheel Informationen akzeptieren
- pc Program Change Meldungen akzeptieren
- q q Zeitquantisierung
- s Synthese durchführen

-*sa sa* Syntheseart
-*so so* Syntheseordnung
-*t* Analyse durchführen
-*tmp tmp* Tempo in bpm
-*v* beliebige Anschlaggeschwindigkeit akzeptieren
-*z z* Synthesezyklen
-? s. -*h*

5. Fehler

5.1. Dokumentierte Fehlermeldungen

F1: Achtung! Auf Grund der Einstellungen sind Synthesefehler moeglich!

Diese seltene Meldung taucht dann auf, wenn der Synthetisator in einen Synthesezweig gerät, der sich als Sackgasse entpuppt. Führen Sie die Synthese nochmals durch! Sollte der Fehler wiederholt auftauchen, wird geraten, entweder das Analysematerial zu vergrößern, oder die Zahl der Zyklen (-*z*) zu verringern.

F2: Fehler an Position xxx

Diese Meldung erhält man bei Verwendung von MIDI-Dateien eines exotischen Standards. Bitte notieren Sie sich die Position *xxx* und die betroffene Datei (Option -*a* einschalten) und kontaktieren Sie suendermann@synther.de, damit der Fehler schnellstmöglich behoben werden kann! Um eine umfangreiche Datei zunächst nach Fehlern zu untersuchen, eignet sich

z.B. `synther -t -n -m -o 1 -c -q 100000 -a ,`

eine laufzeitoptimierte Analyse. Das Verfahren ist ebenfalls für Dateienstapel anwendbar, da die Bearbeitungszeit je Symbol kaum noch zunimmt. Vor einer sehr zeitaufwendigen Analyse empfiehlt es sich, den o.g. Befehl auszuführen, um eventuelle Fehler bereits im Voraus zu detektieren.

5.2. Weitere Fehlerursachen

5.2.1. Weitere Fehler bei der Analyse

Symptom:

Nach der Begrüßung reagiert Synther nicht mehr

mögliche Ursachen:

1. Es wurde das MIDI-Interface aktiviert, obwohl in *dateien.txt* keine MIDI-Dateien angegeben wurden.

Symptom:

Gleich nach der Begrüßung bricht Synther mit einer Fehlermeldung ab

mögliche Ursachen:

1. Die in *dateien.txt* aufgelisteten Dateien befinden sich nicht im Verzeichnis *quelle*.

2. Die Analyseordnung wurde zu hoch eingestellt. Da vom Betriebssystem doppelt so viele Dateien geöffnet werden müssen, wie die Ordnung, hängt die Maximalordnung von den Systemeinstellungen ab. Bei Windows z.B. kann die Zahl erhöht werden, indem in der Datei *config.sys* bzw. *config.nt* der Parameter *files* auf die benötigte Größe eingestellt wird.
3. Im Syntherverzeichnis befinden sich schreibgeschützte Dateien, auf die Synther nicht zugreifen kann. Ändern Sie in diesem Falle die Dateiattribute entsprechender Dateien!

5.2.2. Weitere Fehler bei der Synthese

Symptom:

Gleich nach der Begrüßung bricht Synther mit einer Fehlermeldung ab

mögliche Ursachen:

s. 5.2.1

Symptom:

Nach x Durchläufen bricht Synther mit einer Fehlermeldung ab

mögliche Ursachen:

1. s.a. 5.1.

2. Es wurde eine geringere Analyseordnung (x) als Syntheseordnung verwendet

Symptom:

Nach Durchlaufen aller Zyklen bricht Synther mit einer Fehlermeldung ab

mögliche Ursachen:

1. Die Analyseordnung wurde zu hoch eingestellt (s. 5.2.1).

5.3. Tipps

Wenn Analyse und Synthese zwar funktionieren, das Ergebnis jedoch nicht zufriedenstellend ist, finden Sie vielleicht in diesem Abschnitt einen passenden Verbesserungsvorschlag

Das MIDI-Synthesestück ist unrrhythmisch.

U.U. ist die Zeitquantisierung q während der Analyse zu hoch gewählt worden. Wahrscheinlicher ist jedoch, dass das Basismaterial nichthörbare MIDI-Events wie Channel Messages, Pitch Wheel o.Ä. enthielt, die während der Analyse ignoriert wurden. Kontrollieren Sie mit Hilfe eines Sequenzers das Ausgangsmaterial und schalten Sie ggf. eine der Optionen *-cm*, *-p* oder *-pc* zu!

Obwohl das Ausgangsmaterial gleichzeitig erklingende Instrumente enthielt, spielt im Syntheseprodukt stets nur eines.

MIDI-Dateien enthalten Musikstücke in Partiturform. Die Stimmen der einzelnen Instrumente können entweder parallel oder seriell abgespeichert werden. Da Synther die MIDI-Events stets parallel interpretiert, muss die

Struktur der Dateien ggf. mit einem Sequenzer angepasst werden. Um die Dateistruktur zu überprüfen, tragen Sie einfach wie gehabt das MIDI-Stück in *dateien.txt* ein und führen danach den Befehl

```
synther -t -n -m -a
```

aus. Sobald die erste Rechenzeitmeldung am Bildschirm ausgegeben wird, können Sie Synther abbrechen, z.B. mit <Ctrl>+<C>. Nun wurden die MIDI-Daten in eine auch für den Laien verständliche ASCII-Form konvertiert, die im Syntherverzeichnis in der Datei *quelle.txt* zu lesen ist. Wenn die Kanalnummern der dort aufgeführten Noten nur seriell auftauchen, also z.B. 100 Noten auf Kanal 1, dann 120 Noten auf Kanal 2 usw., ist die Struktur ungeeignet. Besser ist eine beinahe zufällige Abfolge der Kanäle.

Bibliography

[Ka95] KAFKA, FRANZ: Die Verwandlung. Diogenes Verlag, Zürich 1995

Mathematical Foundations

- [Br96] BRONSTEIN; SEMENDJAJEW: Taschenbuch der Mathematik. B. G. Teubner, Leipzig 1996
- [Ma78] v. MANGOLDT; KNOPP: Einführung in die höhere Mathematik. S.Hirzel Verlag, Leipzig 1978
- [Ma93] Formelsammlung zur Wahrscheinlichkeitsrechnung und Mathematischen Statistik. TU Dresden Abteilung Mathematik 1993
- [Pö00] PÖNISCH, G.: Computerorientierte numerische Mathematik, Script zur Vorlesung 2000
- [Rh80] RHEINBOLDT, W. C.; MESZTENYI C. K.: On a data structure for adaptive finite element mesh refinements. ACM Trans. Math. Software 1980
- [Ve96] VETTERS, KLAUS: Formeln und Fakten. B. G. Teubner, Leipzig 1996

Informational Engineering and Informatics

- [Ad97] ADAMSKI, I; RÜDIGER, L: Algorithmen und Datenstrukturen. TU Dresden Institut für Softwaretechnik 1997
- [Fi00] FINGER, ADOLF: Praktikum Fehlerreduktionssysteme. TU Dresden Institut für Nachrichtentechnik 2000
- [Ho98] HOFFMANN, RÜDIGER: Signalanalyse und -erkennung. Springer-Verlag, Berlin/Heidelberg 1998
- [Ni00] <http://hissa.nist.gov/dads/> (National Institute of Standards and Technology)

[We91] WELSH, DOMINIC: Codes und Kryptographie. VCH, Weinheim 1991

Language Processing

[Be99] BECCHETTI, CLAUDIO; RICOTTI, LUCIO PRINA: Speech Recognition, Theory and C++ Implementation. John Wiley & Sons Ltd., West Sussex 1999

[De96] DELIGNE, SABINE; YVON, FRANÇOIS; BIMBOT, FRÉDÉRIC: Introducing Statistical Dependencies and Structural Constraints in Variable-Length Sequence Models. ENST - Dept. Signal & Dept. Informatique, Paris 1996

[Df00] <http://verbmobil.dfki.de> (Deutsches Forschungszentrum für künstliche Intelligenz)

[Sc95] SCHUKAT-TALAMAZZINI: Automatische Spracherkennung. Friedrich Vieweg und Sohn, Braunschweig/Wiesbaden 1995

Music and MIDI

[Cz89] CZEISZPERGER, MICHAEL S.: MIDI file specification. The Ohio State University 1989

[Ge00] <http://www.geocities.com/Powerman5k86/MiDiPaGe/index.html>

[Hi87] HIRSCH, FERDINAND: Das große Wörterbuch der Musik. Verlag Neue Musik, Berlin 1987

[La97] LANGE, JÜRGEN: Das MIDI-Datenformat. Hochschule für Musik Dresden Studio für Elektronische Musik 1997

[Mi01] <http://www.midiworld.com>