hochschule mannheim

# Comparison of compression tools for biological data and analysis of possible improvements

Gabriel Eichelkraut

Bachelor Thesis

for the acquisition of the academic degree Bachelor of Science (B.Sc.)

Course of Studies: Computer Science

Department of Computer Science

University of Applied Sciences Mannheim

01.12.22

Tutors

Prof. Elena Fimmel, Hochschule Mannheim

TBD

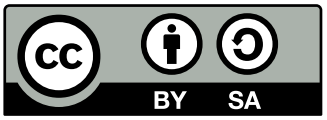
**Eichelkraut, Gabriel**:
Comparison of compression tools for biological data and analysis of possible improvements
/ Gabriel Eichelkraut. –
Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2022. 13 pages.

**Eichelkraut, Gabriel**:
Vergleich von Kompressionswerkzeugen für biologische Daten und Analyse von Verbesserungsmöglichkeiten
/ Gabriel Eichelkraut. –
Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2022. 13 Seiten.

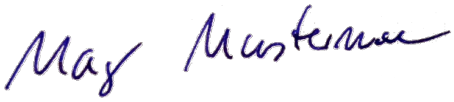
## Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.






Mannheim, 01.12.22



Gabriel Eichelkraut

# Abstract

***Comparison of compression tools for biological data and analysis of possible improvements***

TBD.

***Vergleich von Kompressionswerkzeugen für biologische Daten und Analyse von Verbesserungsmöglichkeiten***

TBD.

# Contents

**1. Introduction** **1**

**2. Structure of Human Genetic Data** **3**

**3. File Types Used to Store DNA** **5**
3.1. **FASTQ!** (**FASTQ!**) . . . . . 7
3.2. Sequence Alignment Map . . . . . 7

**4. Compression aproaches** **9**
4.1. Dictionary coding . . . . . 9
4.1.1. The LZ Family . . . . . 10
4.1.2. Huffman coding . . . . . 10
4.2. Implementations . . . . . 11

**5. Feasibillity Analysis for New Algorithm Considering Stochastic Organisation of Genomes** **12**
5.1. Pool of Tools . . . . . 13
5.2. Installation . . . . . 13
5.3. Alteration of Code to Determine Runtime . . . . . 13
5.4. Execution . . . . . 13
5.5. Data analysis . . . . . 13

**List of Abbreviations** **iv**

**List of Tables** **v**

**List of Figures** **vi**

**Listings** **vii**

**Bibliography** **viii**

**Index** **ix**

**A. Erster Anhang: Lange Tabelle** **ix**

# Chapter 1

# Introduction

Understanding how things in our cosmos work, was and still is a pleasure the human being always wants to fullfill. Getting insights into the rawest form of organic live is possible through storing and studying information, embeded in genetic codes. Since live is complex, there is a lot of information, which requires a lot of memory. With compression tools, the problem of storing information got restricted. Compressed data requires less space and therefore less time to be tranported over networks. This advantage is scaleable and since genetic information needs a lot of storage, even in a compressed state, improvements are welcomed. Since this field is, compared to others, like computer theorie and compression approchaes, relatively new, there is much to discover and new findings are not unusual. From some of this findings, new tools can be developed. They optimally increase two factors: the speed at which data is compressed and the compresseion ratio, meaning the difference between uncompressed and compressed data.

New discoveries in the universal rules of stochastical organisation of genomes might provide a base for new algoriths and therefore new tools or an improvement of existing ones for genome compression. The aim of this work is to analyze the current state of the art for probabilistic compression tools and their algorithms, and ultimately determine whether mentioned discoveries are already used. `might be thrown out due to time limitations` -> If this is not the case, there will be an analysation of how and where this new approach could be imprelented and if it would improve compression methods.

To reach a common ground, the first pages will give the reader a quick overview on the structure of human DNA. There will also be an superficial explanation for some basic terms, used in biology and computer science. The knowledge basis of this work is formed by describing differences in file formats, used to store genome data. In addition to this, a section relevant for compression will follow. This will go through the state of the art in coding theory.
In order to meassure an improvement, first a baseline must be set. Therefore the efficiency and effecitity of suiteable state of the art tools will be meassured. To be as precise as possible, the main part of this work focuses on setting up an environment, picking input data, installing and executing tools and finaly meassuring and documenting the results.
With this information, a static code analysis of mentioned tools follows. This will show where a possible new algorithm or an improvement to an existing one could be implemented. Running a proof of concept implementation under the same conditions and comparing runtime and compression ratio to the defined baseline shows the potential of the new approach for compression with probability algorithms.

# Chapter 2

# Structure of Human Genetic Data

To strengthen the understanding of how and where biological information is stored, this section starts with a quick and general rundown on the structure of any living organism.
All living organisms, like plants and animals, are made of cells (a human body can consist out of several trillion cells) Bianconi et al. 2013. A cell in itself is a living organism; The smalles one possible. It consists out of two layers from which the inner one is called nucleus. The nucleus contains chromosomes and those chromosomes hold the genetic information in form of Deoxyribonucleic Acid (DNA).

DNA is often seen in the form of a double helix. A double helix consists, as the name suggestes, of two single helix.

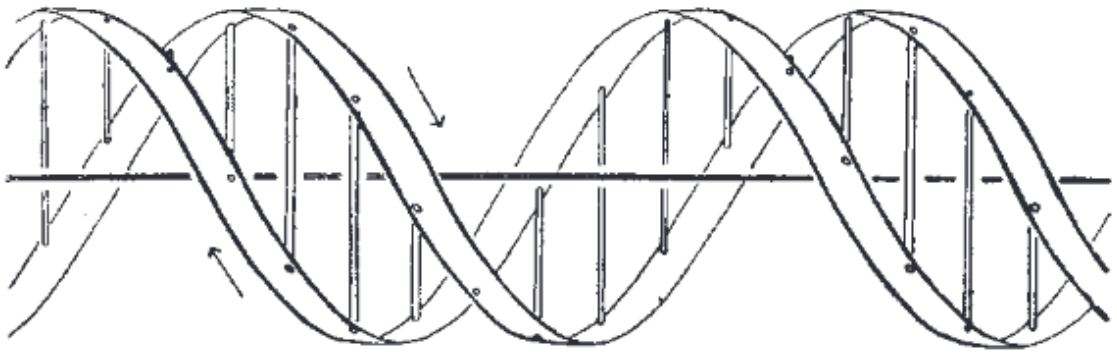


**Figure 2.1.:** A purely diagrammatic figure of the components DNA is made of. The smaller, inner rods symbolize nucleotide links and the outer ribbons the phosphate-suggar chains WATSON and CRICK 1953.

Each of them consists of two main components: the Suggar Phosphat backbone, which is not relevant for this work and the Bases. The arrangement of Bases rep-

resents the Information stored in the DNA. A base is an organic molecule, they are called Nucleotides WATSON and CRICK 1953.

For this work, nucleotides are the most important parts of the DNA. A Nucleotide can occour in one of four forms: it can be either adenine, thymine, guanine or cytosine. Each of them got a Counterpart with which a bond can be established: adenine can bond with thymine, guanine can bond with cytosine.
From the perspective of an computer scientist: The content of one helix must be stored, to persist the full information. In more practical terms: The nucleotides of only one (entire) helix needs to be stored physically, to save the information of the whole DNA because the other half can be determined by "inverting" the stored one. An example would show the counterpart for e.g.: `adenine, guanine, adenine` chain which would be a chain of `thymine, cytosine, thymine`. For the sake of simplicity, one does not write out the full name of each nucleotide, but only its initiat. So the example would change to `AGA` in one Helix, `TCT` in the other.
This representation ist commonly used to store DNA digitally. Depending on the sequencing procedure and other factors, more information is stored and therefore more characters are required but for now 'A', 'C', 'G' and 'T' should be the only concern.

# Chapter 3

# File Types Used to Store DNA

As described in previous chapters DNA can be represented by a string with the buildingblocks A,T,G and C. Using a common fileformat for saving text would be impractical because the ammount of characters or symbols in the used alphabet, defines how many bits are used to store each single symbol.
The **ascii!** (**ascii!**) table is a characterset, registered in 1975 and to this day still in use to encode texts digitally. For the purpose of communication bigger charactersets replaced **ascii!**. It is still used in situations where storage is short. Storing a single *A* with **ascii!** encoding, requires 8 bit ( excluding magic bytes and the bytes used to mark End of File (EOF)) . Since there are at least $2^8$ or 128 displayable symbols. The buildingblocks of DNA require a minimum of four letters, so two bits are needed In most tools, more than four symbols are used. This is due to the complexity in sequencing DNA. It is not 100% preceice, so additional symbols are used to mark nucelotides that could not or could only partly get determined. Further a so called quality score is used to indicate the certainty, for each single nucleotide, that is was sequenced correctly as what was stored.
More common everyday-usage text encodings like unicode require 16 bits per letter. So settling with **ascii!** has improvement capabilitie but is, on the other side, more efficient than using bulkier alternatives like unicode.

Several people and groups have developed different fileformats to store genomes. Unfortunally for this work, there is no defined standard filetype or set of filetypes, therefore one has to gather information by themselve. In order to not go beyond scope, this work will focus only on fileformats that fullfill following criteria:

- the format has reputation, either through a scientific paper, that prooved its superiority to other relevant tools or through a broad ussage of the format.
- the format should no specialize on only one type of DNA.
- the format mainly stores nucleotide seuqences and does not neccesarily include International Union of Pure and Applied Chemistry (IUPAC) codes besides A, C, G and T (Johnson 2010).
- the format is open source. Otherwise improvements can not be tested, without buying the software and/or requesting permission to disassemble and reverse engineer the software or parts of it.
- the compression methode used in the format is based on probabilities.

Information on available formats where gathered through various Internet platforms (Flicek 2022; UCSC - University of California 2022; *Global Alliance for Genomics and Health* 2022). Some common fileformats found:

- BED
- CRAM
- FASTA
- **FASTQ!**
- GFF
- SAM/Binary Alignment Map (BAM)
- twoBit
- VCF

Since methods to store this kind of Data are still in development, there are many more filetypes. The few, mentioned above are used by different organisations and are backed by scientific papers.
Considering the first criteria, by searching through anonymously accesable **ftp!** (**ftp!**) servers, only two formats are used commonly: FASTA or its extension **FASTQ!** and the BAM Format.

## 3.1. FASTQ!

Is a text base format for storing sequenced data. It saves nucleotides as letters and in extend to that, the quality values are saved. **FASTQ!** files are split into multiples of four, each four lines contain the informations for one sequence. The exact structure of **FASTQ!** format is as follows: `Line 1: Sequence identifier aka. Title, starting with an @ and an optional description.`
`Line 2: The seuqence consisting of nucleoids, symbolized by A, T, G and C.`
`Line 3: A '+' that functions as a seperator. Optionally followed by content of Line 1.`
`Line 4: quality line(s). consisting of letters and special characters in the ascii! scope.`

The quality values have no fixed type, to name a few there is the sanger format, the solexa format introduced by Solexa Inc., the Illumina and the QUAL format which is generated by the PHRED software. The quality value shows the estimated probability of error in the sequencing process. [...]

## 3.2. Sequence Alignment Map

Sequence Alignment Map (SAM) often seen in its compressed, binary representation BAM with the fileextension `.bam`, is part of the SAMtools package, a uitlity tool for processing SAM/BAM and CRAM files. The SAM/BAM file is a text based format delimited by TABs. It uses 7-bit US-ASCII, to be precise Charset ANSI X3.4-1968 as defined in RFC1345. The structure is more complex than the one in **FASTQ!** and described best, accompanied by an example:

```
Coor     12345678901234  5678901234567890123456789012345
ref      AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

+r001/1        TTAGATAAAGGATA*CTG
+r002         aaaAGATAA*GGATA
+r003       gcctaAGCTAA
+r004                      ATAGCT..............TCAGC
-r003                             ttagctTAGGC
-r001/2                                        CAGCGGCAT
```

**Figure 3.1.:** SAM/BAM file structure example

# Chapter 4

# Compression aproaches

The process of compressing data serves the goal to generate an output that is smaller than its input data. In many cases, like in gene compressing, the compression is idealy lossless. This means it is possible for every compressed data, to receive the full information that were available in the origin data, by decompressing it. Lossy compression on the other hand, might excludes parts of data in the compression process, in order to increase the compression rate. The excluded parts are typicaly not necessary to transmit the origin information. This works with certain audio and pictures files or network protocols that are used to transmit video/audio streams live. For DNA a lossless compression is needed. To be preceice a lossy compression is not possible, because there is no unnecessary data. Every nucleotide and its position is needed for the sequenced DNA to be complete. For lossless compression two mayor approaches are known: the dictionary coding and the entropy coding. Both are described in detail below.

## 4.1. Dictionary coding

Dictionary coding, as the name suggest, uses a dictionary to eliminate redundand occurences of strings. Strings are a chain of characters representing a full word or just a part of it. This is explained shortly for a better understanding of dictionary coding but is of no great relevance to the focus of this work: Looking at the string 'stationary' it might be smart to store 'station' and 'ary' as seperate dictionary enties. Which way is more efficient depents on the text that should get compressed.

The dictionary should only store strings that occour in the input data. Also storing a dictionary in addition to the (compressed) input data, would be a waste of resources. Therefore the dicitonary is made out of the input data. Each first occourence is left uncompressed. Every occurence of a string, after the first one, points to its first occurence. Since this 'pointer' needs less space than the string it points to, a decrease in the size is created.

### 4.1.1. The LZ Family

The computer scientist Abraham Lempel and the electrical engineere Jacob Ziv created multiple algorithms that are based on dictionary coding. They can be recognized by the substring `LZ` in its name, like `LZ77 and LZ78` which are short for Lempel Ziv 1977 and 1978. The number at the end indictates when the algorithm was published. Today LZ78 is widely used in unix compression solutions like gzip and bz2. Those tools are also used in compressing DNA.

Lempel Ziv 1977 (LZ77) basically works, by removing all repetition of a string or substring and replacing them with information where to find the first occurence and how long it is. Typically it is stored in two bytes, whereby more than one one byte can be used to point to the first occurence because usually less than one byte is required to store the length.

### 4.1.2. Huffman coding

The well known Huffman coding, is used in several Tools for genome compression. This subsection should give the reader a general impression how this algorithm works, without going into detail. To use Huffman coding one must first define an alphabet, in our case a four letter alphabet, containing `A, C, G and T` is sufficient. The basic structure is symbolized as a tree. With that, a few simple rules apply to the structure:

- every symbol of the alphabet is one leaf
- the right branch from every not is marked as a 1, the left one is marked as a 0

- every symbol got a weight, the weight is defined by the frequency the symbol occours in the input text
- the less weight a node has, the higher the probability is, that this node is read next in the symbol sequence

The process of compressing starts with the nodes with the lowest weight and buids up to the hightest. Each step adds nodes to a tree where the most left branch should be the shortest and the most right the longest. The most left branch ends with the symbol with the highest weight, therefore occours the most in the input data. Following one path results in the binary representation for one symbol. For an alphabet like the one described above, the binary representation encoded in ASCI is shown here `A -> 01000001, C -> 01000011, G -> 01010100, T -> 00001010`. An imaginary sequence, that has this distribution of characters `A -> 10, C -> 8, G -> 4, T -> 2`. From this information a weighting would be calculated for each character by dividing one by the characters occurence. With a corresponding tree, created from with the weights, the binary data for each symbol would change to this `A -> 0, C -> 11, T -> 100, G -> 101`. Besides the compressed data, the information contained in the tree msut be saved for the decompression process.

## 4.2. Implementations

### 4.2.1.

### 4.2.2.

### 4.2.3.

# Chapter 5

# Feasibillity Analysis for New Algorithm Considering Stochastic Organisation of Genomes

The first attempt to determine feasability of this project consists of setting basevalues, a further improvement can be meassured by. For this to be recreateable, a few specifications must be known:
CPU Core information ‘cat /proc/cpuinfo‘

Output for the last core:

processor : 15 vendor_id : AuthenticAMD cpu family : 23 model : 1 model name : AMD EPYC Processor (with IBPB) stepping : 2 microcode : 0x1000065 cpu MHz : 2400.000 cache size : 512 KB physical id : 15 siblings : 1 core id : 0 cpu cores : 1 apicid : 15 initial apicid : 15 fpu : yes fpu_exception : yes cpuid level : 13 wp : yes flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm rep_good nopl cpuid extd_apicid tsc_known_freq pni pclmulqdq ssse3 fma cx16 sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm cmp_legacy cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw topoext perfctr_core ssbd ibpb vmmcall fsgsbase tsc_adjust bmi1 avx2 smep bmi2 rdseed adx smap clflushopt sha_ni xsaveopt xsavec xgetbv1 xsaves clzero xsaveerptr virt_ssbd arat arch_capabilities bugs : sysret_ss_attrs null_seg spectre_v1 spectre_v2 spec_store_bypass bogomips : 4800.00 TLB size : 1024 4K

pages clflush size : 64 cache_alignment : 64 address sizes : 48 bits physical, 48 bits virtual power management:
Memory capacity (and more todo list): ‘dmidecode –type memory‘ || ‘dmidecode –type 17‘

## 5.1. Pool of Tools

For an initial test, a small pool of three tools was choosen.

- Samtools
- GeCo
- genie

Each of this tools comply with the criteria choosen in chapter 3.
To test each tool, the same set of data were used. The genome of a homo sapien id: GRCh38 were chosen due to its size TODO: find more exact criteria for testdata. The Testdata is available via an open FTP Server, hotsed by ensembl. Source:http://ftp.ensembl.org/pub/release-107/fasta/homo_sapiens/dna/
Testparameters that were focused on:

- Efficiency: **Duration** the Process had run for
- Effectivity: The difference in **Size** between input and compressed data
- todo: fehlerquote!

First was captured by TODO choose: - a linux tool to output the exact runtime (time <cmd>) - a alteration in the c code that outputs the time at start and end of the process runtime.

## 5.2. Installation

## 5.3. Alteration of Code to Determine Runtime

## 5.4. Execution

## 5.5. Data analysis

# List of Abbreviations

**DNA** Deoxyribonucleic Acid
**EOF** End of File
**IUPAC** International Union of Pure and Applied Chemistry
**LZ77** Lempel Ziv 1977
**SAM** Sequence Alignment Map
**BAM** Binary Alignment Map

# List of Tables

A.1. Tabelle mit ISO-Ländercodes . . . . . ix

# List of Figures

2.1. A purely diagrammatic figure of the components DNA is made of. The smaller, inner rods symbolize nucleotide links and the outer ribbons the phosphate-suggar chains WATSON and CRICK 1953. . 3

3.1. SAM/BAM file structure example . . . . . 8

# Listings

# Bibliography

Bianconi, Eva et al. (July 2013). “An estimation of the number of cells in the human body”. In: *Annals of Human Biology* 40.6, pp. 463–471. DOI: 10.3109/03014460.2013.807878.

Flicek, Paul (Oct. 24, 2022). *ENSEMBL Project*. URL: http://www.ensembl.org/.

*Global Alliance for Genomics and Health* (Oct. 10, 2022). URL: https://github.com/samtools/hts-specs..

Johnson, Andrew D. (Mar. 2010). “An extended IUPAC nomenclature code for polymorphic nucleic acids”. In: *Bioinformatics* 26.10, pp. 1386–1389. DOI: 10.1093/bioinformatics/btq098.

UCSC - University of California, Santa Cruz (Oct. 28, 2022). *UCSC Genome Browser*. URL: https://genome.ucsc.edu/ (visited on 10/28/2022).

WATSON, J. D. and F. H. C. CRICK (Apr. 1953). “Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid”. In: *Nature* 171.4356, pp. 737–738. DOI: 10.1038/171737a0.

# Appendix A

# Erster Anhang: Lange Tabelle

Hier ein Beispiel für einen Anhang. Der Anhang kann genauso in Kapitel und Unterkapitel unterteilt werden, wie die anderen Teile der Arbeit auch.

**Table A.1.:** Lange Tabelle mit ISO-Ländercodes

| Country | A 2 | A 3 | Number |
|---|---|---|---|
| AFGHANISTAN | AF | AFG | 004 |
| ALBANIA | AL | ALB | 008 |
| ALGERIA | DZ | DZA | 012 |
| AMERICAN SAMOA | AS | ASM | 016 |
| ANDORRA | AD | AND | 020 |
| ANGOLA | AO | AGO | 024 |
| ANGUILLA | AI | AIA | 660 |
| ANTARCTICA | AQ | ATA | 010 |
| ANTIGUA AND BARBUDA | AG | ATG | 028 |
| ARGENTINA | AR | ARG | 032 |
| ARMENIA | AM | ARM | 051 |
| ARUBA | AW | ABW | 533 |
| AUSTRALIA | AU | AUS | 036 |
| AUSTRIA | AT | AUT | 040 |
| AZERBAIJAN | AZ | AZE | 031 |
| BAHAMAS | BS | BHS | 044 |
| BAHRAIN | BH | BHR | 048 |
| BANGLADESH | BD | BGD | 050 |
| BARBADOS | BB | BRB | 052 |
| BELARUS | BY | BLR | 112 |
| BELGIUM | BE | BEL | 056 |
| BELIZE | BZ | BLZ | 084 |
| BENIN | BJ | BEN | 204 |
| BERMUDA | BM | BMU | 060 |
| BHUTAN | BT | BTN | 064 |
| BOLIVIA | BO | BOL | 068 |
| BOSNIA AND HERZEGOWINA | BA | BIH | 070 |
| BOTSWANA | BW | BWA | 072 |
| BOUVET ISLAND | BV | BVT | 074 |
| BRAZIL | BR | BRA | 076 |
| BRITISH INDIAN OCEAN TERRITORY | IO | IOT | 086 |
| BRUNEI DARUSSALAM | BN | BRN | 096 |
| BULGARIA | BG | BGR | 100 |
| BURKINA FASO | BF | BFA | 854 |
| BURUNDI | BI | BDI | 108 |
| CAMBODIA | KH | KHM | 116 |
| CAMEROON | CM | CMR | 120 |

| | | | |
|---|---|---|---|
| CANADA | CA | CAN | 124 |
| CAPE VERDE | CV | CPV | 132 |
| CAYMAN ISLANDS | KY | CYM | 136 |
| CENTRAL AFRICAN REPUBLIC | CF | CAF | 140 |
| CHAD | TD | TCD | 148 |
| CHILE | CL | CHL | 152 |
| CHINA | CN | CHN | 156 |
| CHRISTMAS ISLAND | CX | CXR | 162 |
| COCOS (KEELING) ISLANDS | CC | CCK | 166 |
| COLOMBIA | CO | COL | 170 |
| COMOROS | KM | COM | 174 |
| CONGO | CG | COG | 178 |
| COOK ISLANDS | CK | COK | 184 |
| COSTA RICA | CR | CRI | 188 |
| COTE D'IVOIRE | CI | CIV | 384 |
| CROATIA (local name: Hrvatska) | HR | HRV | 191 |
| CUBA | CU | CUB | 192 |
| CYPRUS | CY | CYP | 196 |
| CZECH REPUBLIC | CZ | CZE | 203 |
| DENMARK | DK | DNK | 208 |
| DJIBOUTI | DJ | DJI | 262 |
| DOMINICA | DM | DMA | 212 |
| DOMINICAN REPUBLIC | DO | DOM | 214 |
| EAST TIMOR | TP | TMP | 626 |
| ECUADOR | EC | ECU | 218 |
| EGYPT | EG | EGY | 818 |
| EL SALVADOR | SV | SLV | 222 |
| EQUATORIAL GUINEA | GQ | GNQ | 226 |
| ERITREA | ER | ERI | 232 |
| ESTONIA | EE | EST | 233 |
| ETHIOPIA | ET | ETH | 210 |
| FALKLAND ISLANDS (MALVINAS) | FK | FLK | 238 |
| FAROE ISLANDS | FO | FRO | 234 |
| FIJI | FJ | FJI | 242 |

*Beachten Sie, dass die Tabelle manchmal erst nach dreimaligem Lauf durch LaTeX richtig angezeigt wird.*