Universität Bielefeld

Technische Fakultät
AG Praktische Informatik

Index-based algorithms for motif search and their integration in a system for differential genome analysis

Dissertation

Michael Beckstette

Index-based algorithms for motif search and their integration in a system for differential genome analysis

Zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
der Universität Bielefeld
vorgelegte
Dissertation

von
Michael Beckstette

Bielefeld, im Juni 2007

Group for Practical Computer Science
Faculty of Technology
Bielefeld University
D-33594 Bielefeld
Germany

mbeckste@TechFak.Uni-Bielefeld.DE

Genehmigte Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
der Technischen Fakultät der Universität Bielefeld.
Vorgelegt am 27.06.2007 von Michael Beckstette,
verteidigt und genehmigt am 14.12.2007.

Gutachter:
Prof. Dr. Robert Giegerich, Universität Bielefeld
Prof. Dr. Stefan Kurtz, Universität Hamburg

Gedruckt auf alterungsbeständigem Papier nach ISO 9706.

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore, all progress depends on the unreasonable man.

Georg Bernard Shaw, Irish playwright.

Acknowledgments

I thank my supervisor, Robert Giegerich, for providing me with an interesting research topic and for his scientifically support throughout my time in the Practical Computer Science group. I really enjoyed the uncomplicated atmosphere on M3 and being part of this group.

I thank Stefan Kurtz for arising my interest in sequence analysis and in suffix based index structures in particular, and for many stimulating discussions and a productive collaboration throughout the last years. I further appreciate his willingness to appraise this thesis.

In particular, I am thankful to Robert Homann for productive discussions and being a great colleague over the years, and for tolerating smoking in the office. His excellent programming skills and deep algorithmic knowledge were of inestimable value in the development of the *PoSSuM* software distribution.

Special thanks go to Alexander Sczyrba for the daily coffee breaks, the joint work on the *XenDB* project, and discussions about the question whether senior pole vaulting or handball is the more ambitious sport. Unfortunately he is still wrong in this point.

I appreciate the financial support from Intervet Innovation GmbH for the Genlight project.

Finally, I thank my parents and family for their support during my studies in Bielefeld, and Anja for her patience with me, and for motivating me to continue working towards my Ph.D. You became an invaluable part of my life.

List of own publications

Peer-reviewed publications

- **Michael Beckstette**, Robert Homann, Robert Giegerich, and Stefan Kurtz.
  Fast index based algorithms and software for matching position specific scoring matrices. In *BMC Bioinformatics* 2006, 7:389. (acquired "Highly accessed" designation from publisher).

- Alexander Sczyrba*, **Michael Beckstette***, Ali H. Brivanlou, Robert Giegerich, and Curtis R. Altmann.
  XenDB: Full Length cDNA Prediction and Cross Species Mapping in Xenopus laevis. In *BMC Genomics* 2005, 6:123 (BMC Genomics Research Highlight September 2005)

- **Michael Beckstette**, Jens Mailänder, Richard Marhöfer, Alexander Sczyrba, Enno Ohlebusch, Robert Giegerich, and Paul M. Selzer.
  Genlight: An Interactive System for High-throughput Sequence Analysis and Comparative Genomics. In *Journal of Integrative Bioinformatics* (JIB) 8, (1), 2004. This article also appeared in the Yearbook Bioinformatics 2004, ISBN 3-88579-127-7, pages 79-94.

- Alexander Sczyrba, **Michael Beckstette**, Robert Giegerich, and Curtis R. Altmann.
  Identification of 10,500 Xenopus laevis Full Length Clones through EST Clustering and Sequence Analysis. In *Proceedings of the German Conference on Bioinformatics* GCB, 2004; GI-Edition, Lecture Notes in Informatics, P-53, ISBN 3-88579-382-2.

- **Michael Beckstette**, Dirk Strothmann, Robert Homann, Robert Giegerich and Stefan Kurtz.
  PoSSuMsearch: Fast and Sensitive Matching of Position Specific Scoring Matrices using Enhanced Suffix Arrays. In *Proceedings of the German Conference on Bioinformatics* GCB, 2004; GI-Edition, Lecture Notes in Informatics, P-53, pages 53-64, ISBN 3-88579-382-2.

- **Michael Beckstette**, Alexander Sczyrba, and Paul M. Selzer.
  Genlight: An Interactive System for High-throughput Sequence Analysis and Comparative Genomics. In *Proceedings of the German Conference on Bioinformatics* GCB, 2004; GI-Edition, Lecture Notes in Informatics, P-53, pages 179-186, ISBN 3-88579-382-2.

- Calin O. Marian, Stefano J. Bordoli, Marion Goltz, Rachel A. Santarella, Leisa P. Jackson, Olga Danilevskaya, **Michael Beckstette**, Robert Meeley, and Hank W. Bass.
  The Maize Single myb histone 1 Gene, Smh1, Belongs to a Novel Gene Family and Encodes a Protein That Binds Telomere DNA Repeats in Vitro. In *Plant Physiol.* 133: 1336-1350; 2003.

*) joint first authors

Other publications

- Y. He, S. Qin, X.M. Pan, **M. Beckstette**, and R. Giegerich
  Using protein secondary structure predicition to build structural template. In *CASP6 method papers*, 2004
- P. Martinez, K. Malde, **M. Beckstette**, and J. Baguna
  The origin of bilateral animals. A multigene approach. In *Comparative Biochemistry and Physiology. Abstracts of the Society for Experimental Biology Annual Main Meeting.* 141: S119-S120, 2005

Contents

List of Figures

1 Introduction

1.1 The continuing challenge of biosequence analyis

Just about 50 years ago, Watson and Crick discovered with their pioneering work the double helix structure of DNA [WC53], and only about 30 years ago, with the bacteriophage *MS2*, the first genome of an organism was sequenced [FCD+76]. In these 50 years, several scientific findings revolutionized our understanding of evolution and life, and new research disciplines like molecular genetics and computational biology were constituted. In particular, research results from these two interacting disciplines led to substantial scientific advances in the last decades. Table 1.1 on the following page gives a time line of some of these major milestones and findings.

Computational biology generated new algorithms to address and solve biological problems. Among the most prominent ones are database search methods that allow for the comparison of nucleic or amino acid sequences with provision for evolutionary events like mutations, insertions and deletions. With the availability of such methods, the field of comparative sequence analysis evolved to the probably most successful and expanding discipline in computational biology. It became a key discipline for the discovery and understanding of molecular mechanisms necessary for the machinery of an organism [RYW+00, EPC+00]. The foundations of this discipline go back to the early 1970's, when it was discovered [Fit70] that conservations in the nucleic acid sequence of genes, and accordingly in the amino acid sequence of proteins, lead to a conserved secondary and tertiary structure, and thus to a conserved (similar) function. Founded on this observation, the comparison of sequences of molecules allows to deduce knowledge from one or several known sequence(s) to a new, uncharacterized sequence if the nucleic or amino acid sequence of the molecules is conserved. This finding has not only become the groundwork for all of today's pairwise sequence comparison methods [NW70, SW81, LP85, AGM+90, AMS+97] commonly used for searching large sequence databases, but also for several motif and domain databases that contain motif descriptors of conserved (parts) of sequences, like regular expresions [NWB98], position specific scoring matrices [GME87], or hidden Markov models [Edd98], and their model specific search routines. Such collections of diagnostic signatures [WCF+98, HSL+04, HSW03, FMSB+06], which often describe functionally relevant parts of a molecule, like protein domains, transcription factor binding sites in DNA, or catalytic active sites, have become an invaluable part for homology based annotation and classification of nucleic or amino acid sequences into functionally related groups or families.

Responsible for the abiding success of comparative sequence analysis were not only algorithmic contributions, but also the progress in genome sequencing that generates an ever increasing amount of sequence data available for comparative studies. This astonishing progress is reflected in the increasing number of genomes sequenced in the last years. To give an example, the *Genomes OnLine Database (GOLD)* [LTHK06] lists not less than 2120 fully sequenced genomes by April 2007, with

Year	Event
1953	Discovery of the structure of the DNA double helix [WC53]
1958	Discovery of the semi-conservative replication of DNA [MS58]
1965	The first theory of molecular evolution; the Molecular clock concept [ZP65]
1965	Atlas of Protein Sequences, the first protein database [DECS65]
1966	Encryption of the genetic code is completed; first codon decrypted in 1961 [MN61]
1970	Needleman-Wunsch algorithm for global protein sequence alignment [NW70]
1972	Development of recombinant DNA technology, which permits isolation of defined fragments of DNA [CCBH73]
1975	Sanger DNA sequencing [SC75]
1976	Complete genome sequence of bacteriophage MS2 (3569bp) [FCD$^+$76]
1977	Maxim-Gilbert DNA sequencing [MG77]
1981	Smith-Waterman algorithm for local protein sequence alignment [SW81]
1981	Human mitochondrial genome sequenced [ABB$^+$81]
1981	The concept of a sequence motif [Doo81]
1982	Phage λ genome sequenced [SCH$^+$82]
1982	First public GenBank release containing 606 sequences
1985	*FASTP*/*FASTN* sequence similarity search algorithms invented [LP85]
1987	First profile search algorithms [GME87]
1990	Introduction of the *BLAST* program (version 1) for fast sequence similarity searching [AGM$^+$90]
1993	Protein modeling with hidden Markov models [HKB$^+$93, KMSH94]
1995	First bacterial genomes (*Haemophilus influenzae* and *Mycoplasma genitalium*) completely sequenced [FAW$^+$95, FGW$^+$95]
1996	First archeal genome completely sequenced (*Methanococcus jannaschii*) [BWO$^+$96]
1996	First eukaryotic genome completely sequenced (*Sacharomyces cerevisae*) [GBB$^+$96]
1997	Introduction of gapped *BLAST* and *PsiBLAST* [AMS$^+$97]
1998	The first genome of a multicelluar organism is sequenced (*Caenorhabditis elegans*) [Con98]
1999	The genome sequence of *Drosophila melanogaster* is sequenced [ACH$^+$00]
2001	The draft sequence of the human genome becomes available [Con01]
2005	GenBank exceeds 100 gigabases
2005	454 Life Sciences announces massively parallel, high-throughput pyrosequencing approach [MEA$^+$05]

Table 1.1: A brief time line of milestones in genomics and computational biology.

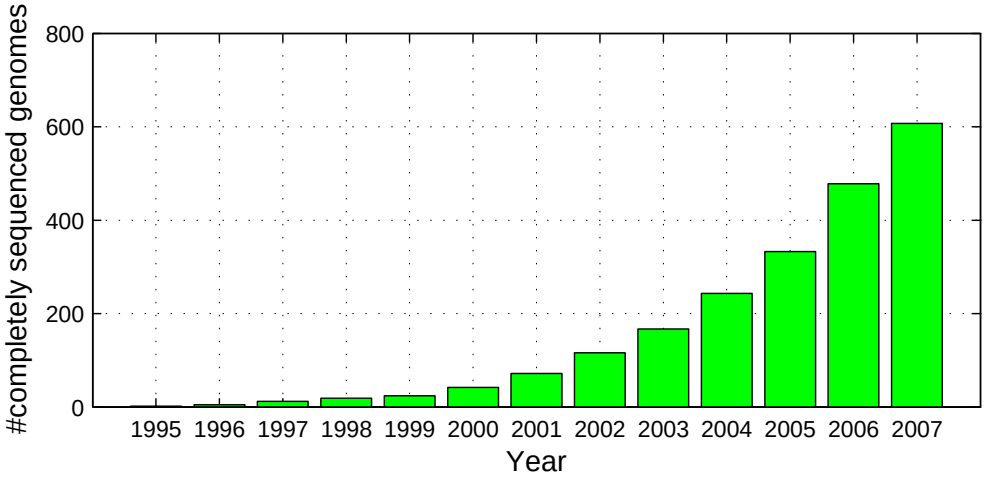


Figure 1.1: Number of completely sequenced genomes per year. For 2007, the data acquisition period is January through April. Data source: *Genomes OnLine Database* (`http://www.genomesonline.org`).

607 completed only in 2007 (see Figure 1.1). In addition, major nucleotide sequence databases like GenBank continue to grow at an exponential rate, with a doubling of their number of bases approximately every 18 months [BKML+07]. Due to the dispersal of new high-throughput sequencing technologies [MEA+05], which reduce the amount of time necessary to fully sequence the genome of some species from years to days, these numbers will increase with an even faster rate in the future. Another corollary of this technological progress in genome sequencing, however, is the fortification of the gap between data generation and data analysis already observed today. More precisely, several of today's widely used sequence analysis programs, e.g., pairwise sequence comparison methods for database searching, like *BLAST* [AMS+97] and *FASTA* [LP85] or search tools for conserved sequence motifs, like *MATCH* [KGR+03], *EMATRIX* [WNB00], and the search tools from the *HMMer* package [Edd98], are more and more faced with difficulties in processing these large amounts of sequence data in reasonable time. Often this problem is alleviated by a massive increase of the applied compute resources, like large cluster systems, though for some computational intensive methods their application on complete genomes remains challenging even then. This problem is mainly founded in the fact that most of today's widely used sequence analysis methods show a running time that is at least linear in the size of the search space (i.e., length of the processed sequences) and hence their running times suffer from the exponentially growing sequence space. Consequently, there is a strong need for new, efficient algorithms capable to handle tomorrows's large amounts of sequence data. Ideally one is interested in algorithms that show a running time that is independent of the size of the searched sequence space.

A possible solution to this dilemma is offered by indexing of sequences with full text index data structures like suffix trees [Wei73] or the more space efficient (enhanced) suffix arrays [MM90, AKO04]. These data structures can be built in linear time and space from the sequences to be searched with several algorithms [Wei73, McC76, Ukk95, KS03, KSPP03, KA03] and allow for very efficient access to subwords. Hence they can be used to efficiently solve many problems in sequence

analysis. Although it is folk knowledge since the middle of the 1980's, that there are "myriads" of possible applications of such data structures [Apo85], in particular in the analysis of huge amounts of sequence data, they barely found their way in today's widely used sequence analysis programs.

One of the main goals of this thesis is the utilization of full text index data structures for the compute intensive searching with position specific scoring matrices (PSSMs for short), a well known motif model with a variety of applications in sequence analysis [GME87]. For this purpose, we developed and implemented new index-based algorithms for searching with PSSMs, which clearly outperform existing methods in terms of running time. We also demonstrate how index based PSSM searching in combination with a fragment chaining approach can be used for efficient protein family classification, and for speeding up computation intensive database searching with hidden Markov models. With the *PoSSuM* software distribution, we also provide implementations of the presented algorithms in form of a flexible command line tool.

We further integrated *PoSSuMsearch* as a database search method in our integrated high-throughput sequence analysis system Genlight, which is also a contribution of this work. Genlight offers an interactive, biologist compatible, and user friendly environment for a variety of large-scale sequence analysis tasks with a special focus on (differential) comparative genome analyses. It employs a set oriented operational model, that allows to reuse generated results, and to perform complete analysis workflows in an interactive way. The system integrates several widely used sequence analysis methods and databases in a common environment, and is capable to perform analyses on a complete genome or proteome scale by employing a distributed client server approach, even for non index-based analysis methods. We demonstrate the practical usability of Genlight with different case studies in which the system was used and which lead to substantial new scientific findings.

1.2 Structure of this thesis

In this thesis, we present new efficient index-based algorithms for searching with PSSMs in large sequence sets, and their integration into an interactive system capable for large-scale differential comparative genome analyses.

In the following Chapter, we start with some introductory and motivating remarks on sequence motifs and motif finding. We describe different modeling concepts for sequence motifs and consensi. This includes regular expression based motif descriptors, PSSMs and their construction principles, Gribskov profiles, and profile hidden Markov models. We discuss in detail strengths and weaknesses of the different modeling concepts.

In Chapter 3, we make several new algorithmic contributions to the field of searching with PSSMs. With algorithm *ESAsearch* and its variants we present new non-heuristic, index-based algorithms for searching with PSSMs that achieve sublinear running time in the expected case and linear running time in the worst case under certain assumptions. The variants include a version achieving improved running time by operating on sequences recoded with a reduced alphabet, as well as a version to determine the k best matching substrings for a PSSM efficiently, without a concrete threshold specification. In various benchmark experiments for nucleotide as well as amino acid sequences, we evaluate the performance of *ESAsearch* and its variants and compare our algorithms with the best previous methods in terms of running time. We also address the problem of non-

comparable PSSM-scores by developing a method based on dynamic programming that allows for the efficient computation of a matrix similarity threshold for a PSSM, given an E-value or p-value. In contrast to other methods, our algorithm, called *LazyDistrib*, employs lazy evaluation of the dynamic programming matrix leading to superior running times. We further describe the *PoSSuM* software distribution implementing our algorithms.

In Chapter 4, we introduce the concept of PSSM family models to increase the power of database searches with PSSMs. We combine algorithm *ESAsearch* with an efficient fragment chaining algorithm to search with PSSM family models and evaluate its performance for accurate protein family classification. Therefore, we compare our approach with a *state of the art* hidden Markov model based method and measure the classification performance for different evaluation scenarios in terms of sensitivity and specificity. We further demonstrate the capabilities of PSSM family models to act as efficient pre-filters allowing to speedup database searching with the compute intensive hidden Markov models, as is implemented in the *hmmsearch* program, dramatically.

In Chapter 5, we describe the interactive high-throughput sequence analysis system Genlight. We provide an in-depth report of the overall architecture, the different parts of the system, and elucidate the system's functionalities, including an overview of the integrated analysis methods and databases. We further demonstrate the practical useability of our system with three case studies in which Genlight was used, published in [MBG$^+$03, BMM$^+$04, SBB$^+$].

In the last Chapter 6, we conclude with a review of the achieved results and discuss potential future developments and extensions.

We round up this thesis with an appendix. It contains, in particular, a detailed manual for the programs included in the *PoSSuM* software distribution.

2 Modeling concepts for sequence motifs and consensi

2.1 Basic definitions and nomenclature

We start with some basic definitions and notations used throughout this thesis. Some definitions may be omitted here as they are introduced later where they are needed.

Definition 1 An *alphabet* $\mathcal{A} = \{a_0, a_1, ..., a_k\}$ is a finite, non empty set. The elements of $\mathcal{A}$ are characters.

Definition 2 A *sequence* or string S of length n over an alphabet $\mathcal{A}$ is the concatenation of n characters of the alphabet. In particular ε denotes the empty string/sequence. By $\mathcal{A}^n$ we denote the set of sequences of length $n > 0$ over $\mathcal{A}$. The set of all possible sequences over an alphabet $\mathcal{A}$ including the empty sequence ε is denoted by $\mathcal{A}^*$. It holds: $S \in \mathcal{A}^n$ and $\mathcal{A}^n \subset \mathcal{A}^*$. $\mathcal{A}^* := \bigcup_{i \geq 0} \mathcal{A}^i$ with $\mathcal{A}^0 := \{\varepsilon\}$ and $\mathcal{A}^{i+1} := \{aw \mid a \in \mathcal{A}, w \in \mathcal{A}^i\}$. The set of non empty sequences over $\mathcal{A}$ is denoted with $\mathcal{A}^+ = \mathcal{A}^* \backslash \{\varepsilon\}$. We write S as a sequence of symbols

$$S = s_1 s_2 s_3 ... s_n$$

Here $s_i \in \mathcal{A}$ is the i-th character of the sequence. We denote the i-th character also by $S[i]$. The *length* of a sequence or string, denoted by $|S|$, is the number of characters in S.

Definition 3 If $S = uvw$ for some (possibly empty) strings u,v,$w \in \mathcal{A}^*$, then

- u is a *prefix* of S,
- v is a *subword* of S, and
- w is a *suffix* of S.

Definition 4 Let $S \in \mathcal{A}^*$ be a sequence, then we denote the set of subwords of S of length m by $\mathsf{words}_m(S) := \{w \in \mathcal{A}^m \mid w \text{ is a } \textit{subword} \text{ of } S\}$.

2.2 Motifs, domains, and sequence families

While the number of different, naturally occurring proteins is huge, most of them can be grouped into a limited number of families on the basis of similarities in their sequences. Proteins belonging

to a particular family generally share functional attributes and in most cases are derived from an evolutionary common ancestor. Throughout this work we use the term *family* for related nucleic acid or protein sequences whose relationship extends over the entire molecule. This relationship may be evolutionary, structural and/or functional. Examples for well known sequence families are the bacterial 16s rRNAs, which build together with proteins the ribosomal complex and are involved in its enzymatic activity, or the protein family of cytochrome C molecules. Cytochrome C is a highly conserved protein across the spectrum of species, found in plants, animals, and many unicellular organisms. The molecule has been studied for the glimpse it gives into evolutionary biology. E.g., both chickens and turkeys have the identical molecule (amino acid for amino acid) within their mitochondria, whereas ducks possess molecules differing by one amino acid. Similarly, both humans and chimpanzees have the identical molecule, while rhesus monkeys possess mitochondria differing by one amino acid. Cytochrome C is involved in manifold reactions and pathways inside the cell. It can catalyze several reactions such as hydroxylation and aromatic oxidation, and shows peroxidase activity by oxidation of various electron donors. It is also an intermediate in apoptosis, a controlled form of cell death used to kill cells in the process of development or in response to infection or DNA damage.

It is apparent, when studying protein sequence families, that some regions have been more conserved than others during evolution and in some cases the sequence of an unknown protein is too distantly related to any known protein to detect its resemblance by overall sequence alignment, but it can be identified by occurrences of conserved modules or particular residue types in its sequence. We call such modules *domains*. They can be generally described as a family of subsequences occurring in different contexts. In case of amino acid sequences, a domain may be defined as units of sequence conservation or as units that independently fold into the same 3D structure. When analyzing proteins, domains are omnipresent building blocks. Prominent examples of proteins that contain several domains with different functionality as building blocks detectable by conservations on the amino acid sequence level are the major subunit of bacterial DNA polymerase I, also known as *Klenow* fragment. The *Klenow* fragment, which can be isolated by proteolysis from the DNA polymerase I holoenzyme, consists of two domains, one with DNA dependent polymerization functionality and one with 3'-5' exonuclease activity for proofreading during DNA replication. See Figure 2.1 for an example of this domain structure. The *Klenow* fragment has a wide range of applications in molecular biology, like the synthesis of double-stranded DNA from single-stranded templates or the production of blunt ends in double-stranded DNA molecules by digesting away protruding 3' overhangs with its 3'-5' exonuclease activity. Another example for typical multi domain proteins are RNA polymerase II molecules, which catalyze the DNA dependent polymerization of RNA during transcription. Since the revolutionary discovery of the structure of the yeast RNA polymerase II [CBK01][1] it is known, that RNA polymerases are more complex, multi chain molecules with a distinct quartery structure and single chains build up from multiple domains (see Figure 2.2).

For short subsequences of high sequence similarity (within a sequence family) we use the term *motif*. These can be small protein domains, transcription factor binding sites in DNA, or the catalytic active site of a family of enzymes. See Figure 2.2 for an example. These regions are generally important for the function of the molecule like its binding properties or enzymatic activity and/or

[1] Very recently, in 2006, this discovery was honored with the award of the noble price in chemistry for Roger D. Kornberg.

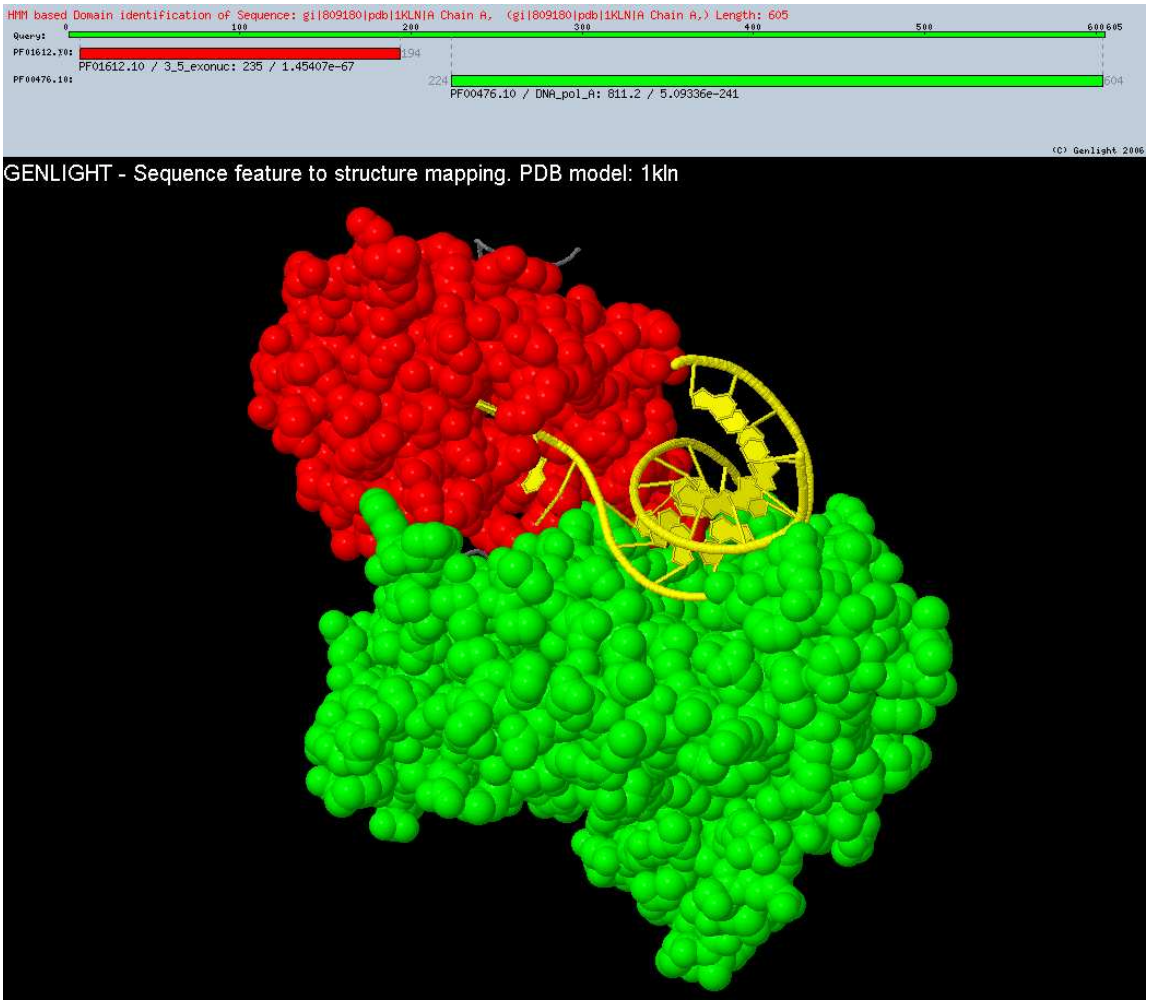


Figure 2.1: The two domains of the major subunit of bacterial DNA polymerase I. Domain structure as detected by sequence conservation (top) and the corresponding 3D structure (bottom) (complex with duplex DNA). The amino-terminal domain (red) has 3'-5' exonuclease activity for proofreading during replication whereas the carboxy-terminal domain (green) is responsible for accurate replication of DNA. This Figure was generated using sequence analysis and sequence feature to structure mapping capabilities of the Genlight system [BMM+04].

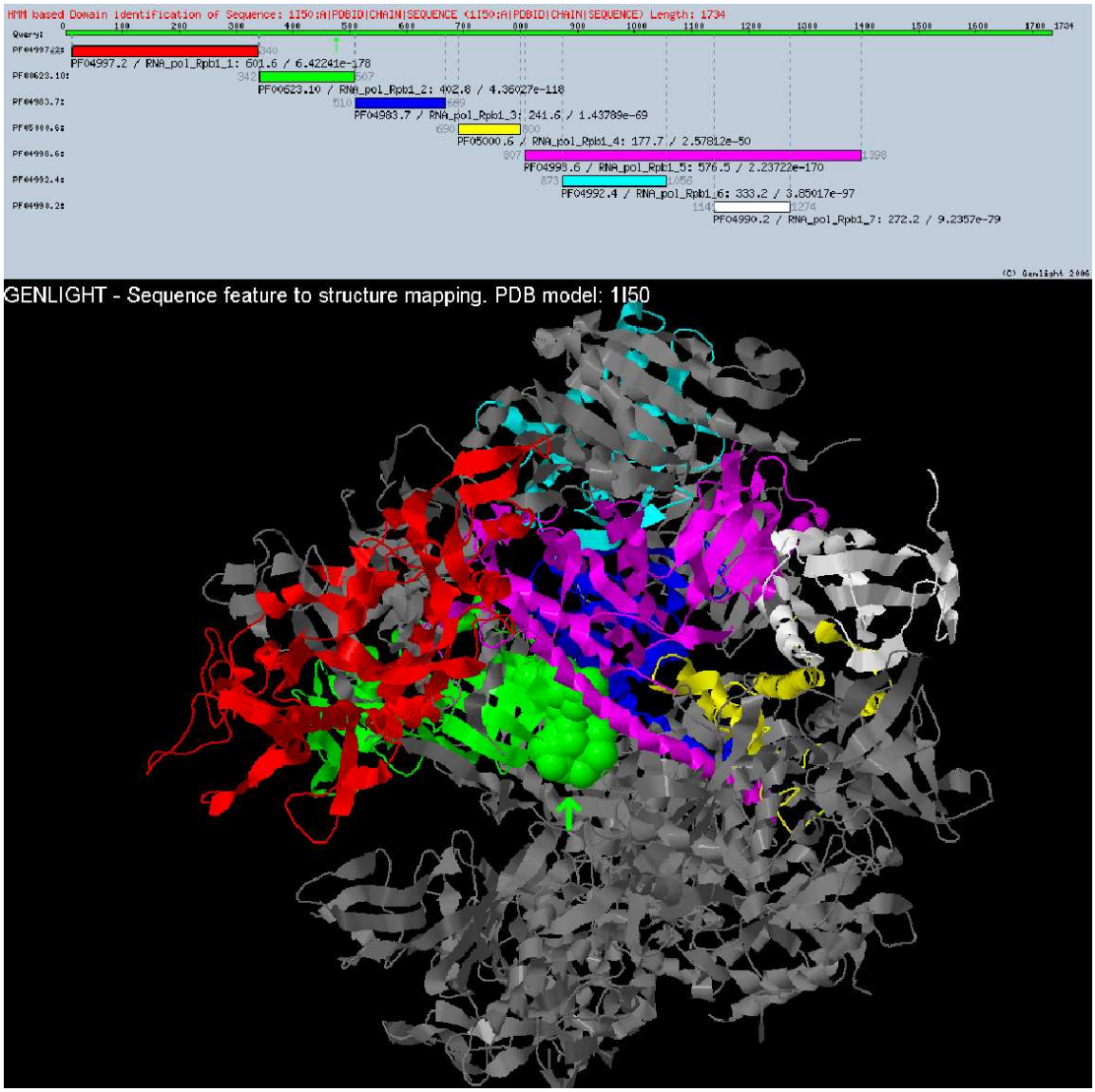


Figure 2.2: Domain structure of the α-chain of RNA polymerase II determined by analysis of the amino acid sequence (top) and domain information mapped on crystallographic 3D structure [CBK01] (bottom). The catalytic active site (marked with green arrow) is located in the center of the molecule at residue positions 478-485 in the second (green marked) domain. It contains the conserved sequence motif NADFDGD (Asn-Ala-Asp-Phe-Asp-Gly-Asp) [ZMD+96].

for the maintenance of its three-dimensional structure. E.g., transcription factor binding sites are small conserved regions typically found upstream and close to the transcription start site of a gene. Through binding of a transcription factor, which is a protein, specific for the binding site, the expression of that gene is regulated by activating or inhibiting the transcription machinery. Such *motifs* can be identified by analyzing the constant and variable properties of groups of similar (sub)sequences. In case of proteins this often allows to derive a diagnostic signature for a family or domain. This *motif* then distinguishes family members from all other unrelated proteins[2].

The use of diagnostic sequence motifs to classify nucleic acid or amino acid sequences into functionally related groups/families and hence predict their function(s), has a long history in the analysis of bio-molecular sequences and is an essential and commonly used technique today. We motivate the importance of sequence motifs and their subsequently described motif descriptor models with a citation from the mid-1980s of R.F. Doolittle, a well known expert in protein sequence analysis

> *"There are many short sequences that are often (but not always) diagnostics of certain binding properties or active sites. These can be set into small subcollections and searched against your sequence [Doo86]."*

When dealing with sequence motifs, one basically faces two problems. The first, briefly described in the following section, is the initial detection of a yet unknown motif in a set of given sequences and the second is its representation with an adequate motif descriptor model that profiles the instances of the motif in the set of sequences.

2.3 Motif finding

One scenario in which the problem of motif finding arises, is the discovery of binding sites of regulatory elements like transcription factors. Consider a set of upstream regions of genes, putatively co-regulated by a common transcription factor. Such genes can be determined from a microarray experiment by selecting genes with a common expression pattern under the same conditions. Then, all their upstream sequences should contain a common binding site for the transcription factor, which has to be identified. Alternatively, if the transcription factor is already known, a popular, applicable, experimental technique to confirm motif binding and determine protein-DNA interaction is chromatin immunoprecipitation (ChIP). ChIP is also applicable on a large scale with its high-throughput variant, called ChiP on chip [IHS+01, RRW+00]. In a ChIP experiment, DNA with bound transcription factors is broken up into various small parts by shearing. With the help of an antibody, specific to the putatively responsible transcription factor, antibody-transcription factor-DNA complexes are precipitated. After washing out of the antibody and the transcription factor, the selected small pieces of DNA can be amplified with PCR for subsequent sequencing. We end up with a set of sequences containing the common binding site motif of the transcription factor. However, ChIP is only an option if the precise transcription factor is known and a specific antibody for it is available.

[2] Although in literature such signatures are also called patterns, in this thesis we use the term *pattern* only for regular expression like motif descriptors, like the ones described in section 2.4.

Through either microarray or ChIP based methods, we obtain a set of DNA sequences, which we have reason to believe, respond to the same transcription factor. The problem of motif-finding is to find the regulatory elements that these DNA sequences have in common. In other words, we want to find subsequences, that are significantly over-represented in our set of sequences. More generally, the problem of finding a motif can be abstracted to a search problem, that takes as input a set of sequences with some kind of commonality, like predicted similar function, or structure, or a common context like upstream regions of putatively co-regulated genes. Then, the output consists of a set of relatively short subsequences of the input sequences and their description with a motif model. In case of regulatory DNA motifs, these subsequences are typically 8-15bp long and although they often have a constant size, since a constant-size transcription factor must bind to the motif, they are highly variable. Consequently it is not sufficient to find an exact substring of some length common to all sequences under consideration.

Computationally, the motif-finding problem may be viewed as one of multiple local alignment [HS99, FHSW04]. Given a group of sequences that share a common biological property, multiple local alignment methods attempt to locate and align similar subsequences, which may confer this property. That is, given protein or DNA sequences, locate a region (i.e., a substring) of fixed length from each sequence such that a score determined from the set of regions is optimized.

Beyond the discovery of transcription factor binding sites, there are numerous other applications for motif finding. E.g., one could search for functional motifs at exon-intron boundaries, in 3'-untranslated regions of localized RNAs, in 5'-untranslated regions of translationally regulated RNAs or to find domains and motifs in sets of protein sequences.

Generally, we can formulate the motif finding problem as follows. Given a set of sequences $L = \{S_1, S_2, \ldots, S_l\}$ with $S_i \in \mathcal{A}^*$ for all $1 \leq i \leq l$, and a word length $k > 0$. For the sake of simplicity, we assume here that a reasonable k is already known and hence take k as fixed, although in practice this is often not the case. Let w^* be a word with $|w^*| = k$, that has the *best* match to the set of sequences L. We define *best* based on the Hamming distance, although other distances are also possible. Let $d(w, S_i)$ denote the minimum Hamming distance between w and any word of $\mathsf{words}_k(S_i)$. Further, we define the total distance between w and the set of sequences L as

$$D(w, L) := \sum_{i=1}^{l} d(w, S_i). \tag{2.1}$$

Then the optimal solution to the motif finding problem is to find a $w^* \in \mathcal{A}^k$ such that $D(w^*, L)$ is minimized. Here the word w^* in combination with the distance measure D defines a motif (descriptor) along with the set of its instances $\{w_1, w_2, \ldots, w_l \mid w_i \in \mathsf{words}_k(S_i) \wedge \nexists\, w^{*\prime} \in \mathcal{A}^k, w^{*\prime} \neq w^* : D(w^{*\prime}, L) < D(w^*, L)\}$. In general a motif defines a set of words and can be derived from a set of words, and it is noted in form of a motif descriptor.

The most obvious method to solve the motif finding problem would be simply to search exhaustively through the set of all possible $w \in \mathcal{A}^k$, to find the best match. Unfortunately this exhaustive search is very expensive and often not feasible for problem sizes occurring in practice. Observe, that there are $|\mathcal{A}|^k$ words of length k. Taking the total length of all sequences as $n = \sum_{i=1}^{l} |S_i|$, the running time of the sketched algorithm is $\mathcal{O}\left(k \cdot n \cdot |\mathcal{A}|^k\right)$, as each w must be checked against all $\mathcal{O}(n)$ words in L and each check takes $\mathcal{O}(k)$ time. If we can assure, that the sequences in L are error free, the running time can be reduced to $\mathcal{O}\left(k \cdot n^2\right)$ by checking only the $\mathcal{O}(n)$ words that actually occur in

L instead of all $|\mathcal{A}|^k$ possible words of length k. However, this can lead to an overlooking of the true w^*, since it may be the case that the best, or true, w^* is one that is very close to a number of words occurring in L, but not exactly equal to any of them.

In practice, the motif finding problem is usually either reduced to an enumeration and verification problem or to a multiple alignment problem. Either class of problems has been shown to be NP-Hard [WJ94, Bra94]. Therefore numerous different algorithms have been proposed, employing various heuristics or ad hoc constraints to discover motifs efficiently [RHEC98, YTI+98, Kei02]. These methods can be subdivided into two broad categories based on the two major algorithmic paradigms for motif finding. These are

- **Combinatorial approaches**. Programs like *Consensus* [HHS90] or *Pratt* [JCH95] belong to this category.

- **Probabilistic approaches**. To this category belong methods based on Expectation Maximization (EM) [LR90], like *MEME* [BE95a, BE95b] and methods based on Gibbs sampling techniques [LAB+93, NLL95], like *AlignACE* [HETC00] or *MotifSampler* [TLM+01].

Since a detailed description of all the different variants of these paradigms is not in the scope of this thesis, we only give a brief overview over the most widely used tools (see Table 2.1) and refer the reader in particular to [TLB+05, LT06]. These articles describe, compare and evaluate in terms of prediction accuracy in detail different computational approaches for the prediction of regulatory elements in nucleotide sequences. The described algorithmic ideas in these articles are in most cases also applicable to amino acid sequences.

2.4 Regular expressions as motif descriptors

Once a motif has been derived from a set of related sequences, it must be described with some kind of motif descriptor. A basic way to describe a sequence motif and historically one of the oldest approximate pattern models in sequence analysis uses regular expressions. A regular expression, often called a *pattern*, is an expression that describes a set of strings. They are usually used to give a concise description of a set, without having to list all its elements. Motif descriptors in form of regular expressions are used to describe amino acid (see Figure 2.4) as well as nucleotide motifs. A well known example for a nucleotide motif describable with a regular expression is the TATA-box found in the promotor region of many prokaryotic genes. The TATA-, often also called Pribnow-box, is a conserved cis-regulatory element. It is the binding site of either transcription factors or histones (binding of a transcription factor blocks binding of a histone and vice versa) and is involved in the process of transcription by RNA polymerase. It has the consensus DNA sequence $5' - \texttt{TATAAT} - 3'$ but can vary slightly. E.g., `TAAT`, `TATAT`, and `TAAAT` can also be found. The set containing the four strings `TAAT`, `TATAT`, `TATAAT`, and `TAAAT` can be described by the pattern `TAT?AA?T` or alternatively, it is said that the pattern matches each of the four strings. Here the '?' indicates that there is zero or one occurence of the preceeding expression. In most formalisms describing regular expressions, the following operations for their construction are provided:

Program	Operating principle	Algorithmic paradigm
AlignAce [HETC00]	Gibbs sampling algorithm that returns a series of motifs as PSSMs that are overrepresented in the input set.	Probabilistic
Consensus [HS99]	Models motifs using PSSMs, searching for the matrix with maximum information content.	Combinatorial
MEME [BE95b]	Optimizes the expectation value of a statistic related to the information content of the motif.	Probabilistic
MotifSampler [TLM$^+$01]	Matrix-based, motif finding algorithm that extends Gibbs sampling by modeling the background with a higher order Markov model.	Probabilistic
Oligo/dyad-analysis [HRCV00]	Detects overrepresented oligonucleotides and spaced motifs with dyad-analysis.	Probabilistic
Pratt [JCH95]	Identifies conserved motifs in a set of unaligned protein sequences. The method guarantees to find the highest scoring motif in a user defined motif class, according to a defined fitness measure.	Combinatorial
PROTOMAT [HH91]	Detects series of overrepresented motifs in form of ungapped blocks in amino acid sequences by employing a combinatorial algorithm[SAC90] as well as a modification of Lawrence's Gibbs sampler[LAB$^+$93].	Combinatorial/ Probabilistic
QuickScore [RD04]	Based on an exhaustive searching algorithm that estimates probabilities of rare or frequent words in genomic sequences.	Combinatorial
YMF [ST03]	Uses an exhaustive search algorithm to find motifs with the greatest z-score. Motifs are formulated as sequences over the IUPAC alphabet	Combinatorial.

Table 2.1: Widely used motif finders and their operating principles.

- **concatenation**: A centered dot ($\cdot$) or minus (-) concatenates two regular expressions. In practice the concatenation operator is often not explicitly written, that is $\mathtt{T} - \mathtt{T} = \mathtt{T} \cdot \mathtt{T} = \mathtt{TT}$.

- **alternation**: A vertical bar separates alternatives. For example, `TAA | TTA` matches `TAA` or `TTA`, which can commonly be shortened to `T(A | T)A`.

- **grouping**: Parentheses are used to define the scope and precedence of the operators. E.g., `TAA | TTA` and `T(A | T)A` are different patterns, but they both describe the same set of strings.

- **quantification**: A quantifier after a character or group specifies how often that preceding expression is allowed to occur. The most common quantifiers are ?, +, and *. The question mark indicates that there is zero or one occurrence of the previous expression. The plus sign indicates that there is at least one occurrence of the preceding expression and the asterisk, also called *Kleene* operator, indicates there are zero, one or any number of occurrences of the preceding expression.

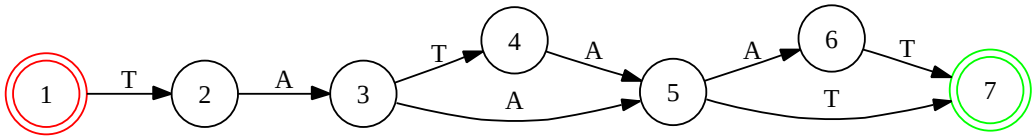


Figure 2.3: Deterministic finite state automaton (DFA) for regular expression `TAT?AA?T`, describing the TATA-box motif found in many gene promotors. Here state 1 is the start state and state 7 is the only accepting state.

These rules can be combined to form arbitrarily complex expressions, which again are regular expressions. The set of strings matched by a regular expression R is also called the semantic or language of R, denoted by $\mathcal{L}(R)$. Generally, we define the syntax of regular expressions and their semantic as follows.

Definition 5 Let $\mathcal{A}$ be an alphabet. A *regular expression* and its associated *language* (semantic) $\mathcal{L}$ over $\mathcal{A}$ is defined as follows.

- ϵ is a regular expression with $\mathcal{L}(\epsilon) := \{\epsilon\}$.
- a is a regular expression for any $a \in \mathcal{A}$ with $\mathcal{L}(a) := \{a\}$.
- If α and β are regular expressions, then $\alpha \cdot \beta$ is a regular expression with $\mathcal{L}(\alpha \cdot \beta) := \{uv | u \in \mathcal{L}(\alpha), v \in \mathcal{L}(\beta)\}$.
- If α and β are regular expressions, then $\alpha \mid \beta$ is a regular expression with $\mathcal{L}(\alpha \mid \beta) := \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$.
- If α is a regular expression, then (α) is a regular expression with $\mathcal{L}((\alpha)) := \mathcal{L}(\alpha)$.
- If α is a regular expression, then α^* is a regular expression with $\mathcal{L}(a^*) := \bigcup_{i \geq 0} \mathcal{L}(\alpha^i)$, where $\mathcal{L}(\alpha^0) := \{\epsilon\}$ and $\mathcal{L}(\alpha^{i+1}) := \mathcal{L}(\alpha \cdot \alpha^i)$.

In the Chomsky hierarchy of formal languages, the class of languages describable by a regular expression is called type-3 language, or regular language. It is a subset of type-2 context free languages, type-1 context sensitive languages and type-0 unrestricted languages. Note that, regular languages are exactly all languages that can be decided by a (deterministic and non-deterministic) finite state automaton (DFA and NDFA for short).

A DFA takes in a string of input symbols. For each input symbol it will then transition to a state given by following a transition function. While transitioning from state to state, symbols are accepted or rejected. When all symbols are accepted and the current state is an accepting state, the string is accepted. We define a DFA according to the following definition.

Definition 6 *A deterministic finite state automaton (DFA)* is a 5-tuple, $(S, \mathcal{A}, \delta, s_0, A)$ consisting of

1. a finite set of states S,

2. an alphabet $\mathcal{A}$,
3. a transition function $\delta : S \times \mathcal{A} \to S$,
4. a start state $s_0 \in S$,
5. a set of accepting states $A \subseteq S$.

Let $M = (S, \mathcal{A}, \delta, s_0, A)$ be a DFA and $T = t_0 t_1 \dots t_n$ be a string over $\mathcal{A}$. M accepts or matches T if and only if a sequence of states $r_0, r_1, \dots r_n$ exists in S satisfying the following conditions:

1. $r_0 = s_0$
2. $r_{i+1} = \delta(r_i, t_i)$, for $i \in [0, n-1]$
3. $r_n \in A$.

That is, for matching a pattern described by a regular expression we must construct the corresponding DFA and process the sequence to be searched with the DFA. See Figure 2.3 for an example of a DFA recognizing a simple sequence motif described by a regular expression.

In practice, different regular expression matching engines and tools utilizing them, like *grep*, *awk*, or *Perl*, use different flavors of regular expressions with varying syntaxes and in some cases different semantics. An effort of standardization was undertaken by defining a *POSIX*[3] specification for regular expressions. POSIX distills the various common flavors into just two classes, *Basic Regular Expressions* (BREs), and *Extended Regular Expressions* (EREs). Fully POSIX-compliant tools use one or both of the flavors.

2.4.1 Consensus strings

For the description of the consensus of a set of sequences, e.g. a multiple alignment of related sequences, sometimes simple strings over extended alphabets are used. These alphabets contain special letters, representing character classes and allow to describe equivocalities in a column of the alignment. These letters may describe subsets of the alphabet, defined by a common property, e.g. polarity or hydrophobicity in case of amino acids. See Table 2.2 for an example of a standardized extended nucleotide alphabet. Obviously, these consensus strings are regular expression, since any character class can be written as a sequence of alternations between its members.

2.4.2 Prosite patterns: Regular expressions for protein family assignment

A well known collection of diagnostic sequence patterns for protein family assignment is the PROSITE database [HBB+06]. It contains in its latest release Rel 19.29 1331 documented patterns. PROSITE patterns are manually curated and are derived with expert knowledge about groups or families of sequences from multiple alignments. In particular, attention is drawn to the residues and regions thought or proved to be important to the biological function of that group of proteins, since they are often highly specific and hence discriminative descriptors. These biologically significant regions are generally:

[3] POSIX = **P**ortable **O**perating **S**ystem **I**nterface for Uni**x**

Symbol	Semantic	Description
R	$A \mid G$	purine
Y	$C \mid T$	pyrimidine
W	$A \mid T$	weak hydrogen bond
S	$G \mid C$	strong hydrogen bond
M	$A \mid C$	amino group
K	$G \mid T$	keto group
H	$A \mid C \mid T$	not guanine
B	$G \mid C \mid T$	not adenine
V	$G \mid A \mid C$	not thymine
D	$G \mid A \mid T$	not cytosine
N	$G \mid A \mid T \mid C$	any

Table 2.2: IUPAC extended nucleotide alphabet

- Enzyme catalytic sites.
- Prosthetic group attachment sites (heme, pyridoxal-phosphate, biotin, etc.).
- Amino acids involved in binding a metal ion.
- Cysteines involved in disulphide bonds, since they are involved in and important for secondary structure formation.
- Regions involved in binding a molecule (ADP/ATP, GDP/GTP, calcium, DNA, etc.) or another protein.

If a pattern common to all sequences under consideration is found, e.g. with one of the motif finders presented in Table 2.1, it is screened versus the SwissProt protein database, to make sure, that it matches all other known members of this family and only this and hence makes a good discriminative descriptor for the protein family under consideration.

PROSITE patterns are formulated as limited regular expressions, which represent a subset of the class of full regular expressions. In contrast to full regular expressions, they contain no *Kleene* operator, and alternations are only defined between strings of length 1 (single characters). These limitations have almost no negative effect on their ability to describe biological sequence patterns adequately, but allow the construction of easier matching engines and the use of fast bit-vector algorithms, like SHIFT-AND [WM92] and/or SHIFT-OR [BYG89]. Even fast approximate matching of these limited regular expressions (e.g. matching of regular expressions allowing errors) is possible [Mye99].

The syntax of Prosite patterns

PROSITE uses the standard IUPAC one-letter code to represent the amino acids and established widely accepted conventions for the notation of regular expression based patterns in computational biology. The employed syntax, which is different from the standard POSIX notation for regular expressions, is given in the Table 2.3.

`-N-P-Q-` `-N-X-Q-`	Each element in a pattern is separated from its neighbor by a `-`. The symbol `X` is used for a position where any amino acid is accepted
`N-[AST]-Q-`	Ambiguities are indicated by listing the acceptable amino acids for a given position between square brackets, i.e. in this example Alanine, Serine and Threonine.
`N-{MP}-Q-`	Curly brackets indicate residues that are not accepted in this position i.e. not Methionine or Proline.
`N-A(2,3)-Q-`	Repetition of an element of the pattern can be indicated by following that element with a numerical value or a numerical range between parenthesis, i.e. in this example `N-A-A-Q` and `N-A-A-A-Q`.
`N-A(2,3)-Q>`	If a pattern is restricted to either the amino- or carboxy-terminal end of a sequence, that pattern either starts with `<` or respectively ends with `>`.
`.`	A period ends the pattern.

Table 2.3: Syntax of PROSITE patterns

The pattern describing the TATA-box element, as given before, can be written in PROSITE syntax as `T-A-T(0,1)-A-A(0,1)-T`. Beyond the pattern definition, a PROSITE entry contains additional annotation information about the sequence family characterized by the pattern, like a listing of already known members of the family, active site position and many more. Figure A.1 on page 198 gives a concrete example of a PROSITE entry.

Although motif descriptors based on regular expressions are quite successful for protein function prediction and family assignment [HB01], there are a number of protein families as well as functional or structural domains that cannot be accurately detected using this kind of motif descriptors due to their extreme sequence divergence. In such cases alternative, more flexible techniques are used to build a model that describes a family of related sequences adequately. One of these modeling concepts are position specific scoring matrices which are often better suited for motif description in heterogeneous protein families than regular expressions.

2.5 Position specific scoring matrices

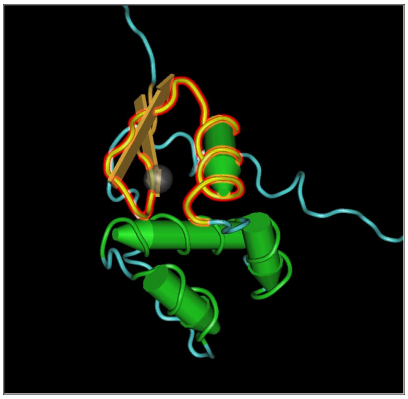
Position specific scoring matrices (PSSMs), often also called position-weight matrices (PWMs), probabilistic patterns, or profiles, have a long history in sequence analysis (see [GME87]). They are successfully used in nucleotide as well as in amino acid sequence analysis as approximate motif models, e.g. for the representation of transcription factor binding sites (TFBSs for short) or conserved regions of proteins. In particular for modeling of short conserved regulatory motifs in DNA, like TFBSs, PSSMs are the method of choice. This can also be seen in a comparison of 13 computational tools for TFBS prediction described in [TLB+05], where the majority of tools uses PSSMs to describe the predicted motifs.

The primary intuition of a PSSM is that a multiple alignment of related sequences, which is normally the building material for a PSSM, can reveal position-specific amino acid or nucleotide propensities. If these information is properly deployed it should increase the sensitivity in a database search for recognition of distant homologs. Many studies have shown that database searches using PSSMs as queries are more effective at identifying distant protein relationships than are searches that use the

(A)

ABC3G_LAGLA/285-305	Cfs..CaekVaeflqenpHvnl..H
ABRU_DROME/546-567	Cpk..CgkiYrsahtlrtHledk.H
ACE1_TRIRE/402-424	CrepgCtkeFkrpcdltkHekt..H
ACE2_SCHPO/475-495	Cdl..CkagFvrhhdlkrHlri..H
ACE2_YEAST/605-627	ClypnCnkvFkrrynirsHiqt..H
ADNP_HUMAN/514-535	Cpy..CrstFndvekmaaHmrmv.H
ADNP_MOUSE/233-254	Cpy..CrstFndvekmaaHmrmv.H
ADNP_RAT/234-255	Cpy..CrstFndvekmaaHmrmv.H
ADR1_YEAST/106-126	Cev..CtraFarqehlkrHyrs..H
AEF1_DROME/270-290	Cvi..CkkqFrqsstlnnHiki..H
AEF1_DROME/242-262	Cnf..CpkhFrqlstlanHvki..H
AEGA_ECOLI/15-37	Cha..CeiaCvmahndeqHvlsqhH
AIOL_HUMAN/148-168	Cnq..CgasFtqkgnllrHikl..H

(B)



(C)

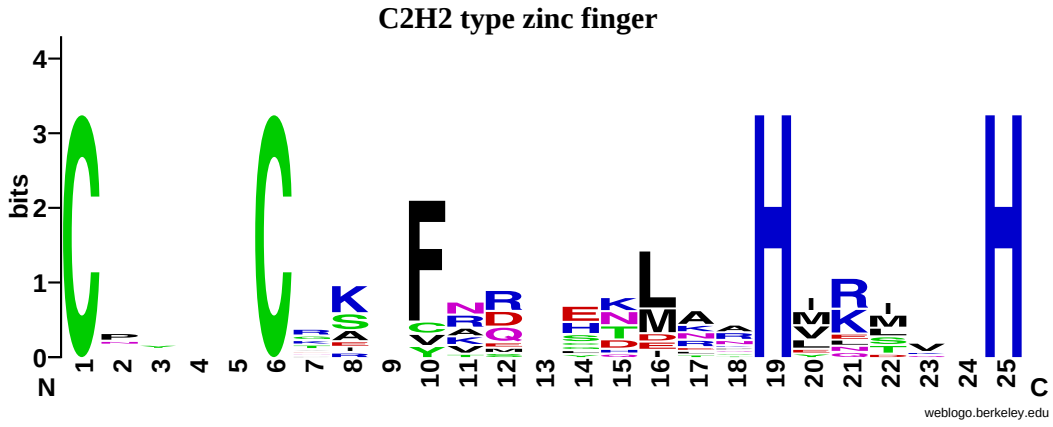


Figure 2.4: Multiple alignment of C2H2 type zinc finger domain sequences (A) and three dimensional structure of C2H2 domain of the mouse protein Argenine N-Methyltransferase 3 (B). Yellow marked part of structure corresponds to part of sequence shown in the multiple alignment. As apparent in the sequence logo (C)[CHCB04], this domain family can be described by the characteristic pattern `C-x(2,4)-C-x(3)-[LIVMFYWC]-x(8)-H-x(3,5)-H` (PROSITE Accession: PS00028). Zinc finger domains have been found in numerous nucleic acid binding proteins and for several members of this family zinc-dependent DNA or RNA binding properties could be demonstrated, experimentally. The two conserved cysteines and histidines of the C2H2 type at the extremities of the domain are involved in the tetrahedral coordination of a zinc atom, when binding to DNA or RNA.

information of a single sequence like standard pairwise sequence comparison methods. In sequence analysis with PSSMs, the basis for comparison is not a general purpose substitution matrix like BLOSUM [HH89], as in traditional pairwise sequence comparison [AGM+90], but also structural information implicit in the multiple alignment of related sequences used for PSSM construction. Unlike the pairwise alignment methods, intuitively, PSSMs use the fact that certain positions in a family of related sequences are more conserved than other positions, due to their conformational or structural relevance, and allow substitutions less readily in these conserved positions. The PSSM model incorporates this position specific information and allows, if used in a database search scenario, increased sensitivity and specificity, compared to pairwise sequence alignment methods that use position independent general purpose scoring matrices like BLOSUM62 or PAM120. One popular program that makes use of PSSMs for database searching is the Position-Specific Iterated BLAST (PSI-BLAST) program [AMS+97]. PSI-BLAST computes a PSSM from a set of homologous sequences and iteratively scans the database with the derived PSSM as a scoring matrix. In each iteration the PSSM is recomputed based on the set of found sequences. A complementary approach is used in the IMPALA software package [SWP+99]. IMPALA compares a single query sequence with a database of PSI-BLAST generated PSSMs using a variant of the Smith-Waterman algorithm [SW81] to compute an optimal alignment between a PSSM and a sequence.

In protein sequence analysis PSSMs often help to model and to identify structurally or functionally important regions within a family of proteins, such as catalytic sites, substrate binding sites and intermolecular interaction sites. Those regions are assumed to have a highly conserved tertiary structure to be biologically functional. Certain types of structural information, however, are not generally captured by PSSM search methods and recent publications describe approaches to include available structural information explicitly. The Structure-Based Alignment Tool (SALTO) [KTP+04] for example aligns protein query sequences to PSSMs derived from NCBI's conserved domain database (CDD) [MBADS+05]. The algorithm uses additional rules to compute only alignments that are consistent with the conserved regions of domain alignments from the CDD. A different more visual approach to include structural information is used in the program 3MATRIX [BLB03]. 3MATRIX combines sequence information determined by sequence searches with PSSMs and maps these on structural models obtained from protein structure databases, like PDB (see Figure 2.5 for an example). This allows to link sequence attributes like residue conservation with structural attributes, e.g. solvent accessibility. Here again, the underlying idea is that conserved sequence motifs can be seen as structural elements that may have the same local tertiary structure in whatever protein they are found in. Hence linking three dimensional information from crystallographic experiments to PSSMs may provide new insights in the potential functional or structural contributions of residues. In [BLB03] the authors state that the structural environment of conserved residues described by a PSSM allows one to better target them for further experiments, such as mutagenesis or drug design.

2.5.1 From alignment blocks to PSSMs

The prerequisite for the construction of a PSSM is the discovery of a motif shared by several or all members inside a family of related sequences. As already described in section 2.3, this is a basic, but challenging task. A special type of motif often described in literature is the *block* or *alignment block*. An alignment block is (a part of) a gapless local multiple sequence alignment (see Figure 2.6 for an

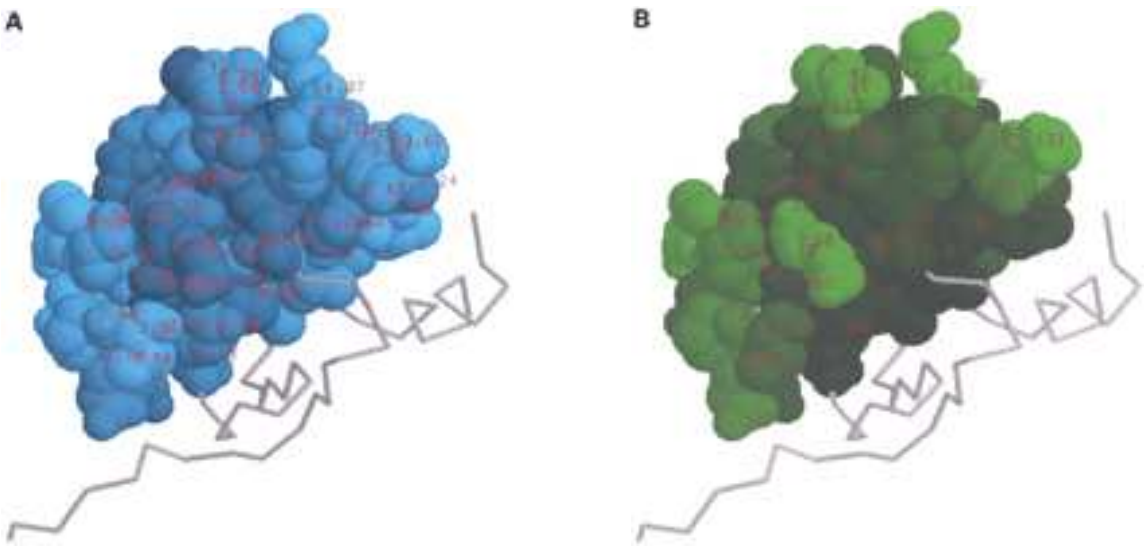


Figure 2.5: 3MATRIX visualization for a sequence motif described by a PSSM representing a glycine-rich cytoskeletal associating protein (CAP-Gly) domain (InterPro ID: IPB000938) found in the PDB structure 1LPL, a CAP-Gly protein from *Caenorhabditis elegans*. Different shades of blue in the left model (A) visualize different degrees of sequence conservation. In the right model (B) amino acids are colored with a shade of green determined by the relative solvent accessible surface area of the amino acid at each motif position. The most solvent-exposed amino acids are also the most highly conserved in this motif, which is an expected result for a motif known to represent a protein-protein interaction domain. Figure taken from [BLB03].

example). The proper determination of characteristic, family specific alignment blocks, representing functionally important and thus conserved regions of nucleotide or amino acid sequences, is the first step in the PSSM construction process. Blocks can be carved out from fully-ungapped regions of gap containing multiple global alignments constructed with standard multiple alignment programs, like ClustalW [HTG$^+$94], DCA [SMD98, Sto98] or T-Coffee [NHH00] or generated by using local sequence alignment methods such as BLAST. Also some of the methods presented in section 2.3 on page 11 for finding a localized region of sequence similarity in a set of sequences without first having to produce an alignment can be used. An example for such *ab initio* motif discovery methods is the CONSENSUS program [HS95] for the prediction of regulatory motifs. Other programs like the PROTOMAT system [HH91] even allow the fully automated detection of ordered, characteristic sets of alignment blocks from a family of related amino acid sequences. High quality multiple alignments, usable to derive useful alignment blocks, as well as already derived PSSMs for database searching are publicly available in several collections and databases. The following enumeration gives a brief overview about the largest and most popular collections:

- the *BLOCKS* database [HGPH00, HP99]
  Contains groups of short multiple alignment blocks of conserved, characteristic protein sequences, e.g. catalytic domains and receptor sites;

- the *eBLOCKS* database [SLSB05, Gal05]
  is a database of ungapped alignments of highly conserved regions among a protein family or superfamily. eBLOCKS is automatically generated from all against all PSI-Blast searches in the SwissProt database;

- the *Pfam* (ProteinFamilies) database [BBD$^+$00]
  Large collection of manually curated high quality, and automatically generated multiple alignments of protein families and inferred hidden Markov models;

- the *TIGRFAMs* protein family database
  Manually curated multiple sequence alignments of protein families;

- the *PRINTS* database [AMG$^+$06, ACF$^+$00]
  Database containing protein fingerprints in form of (ordered) sets of gapless multiple alignments and derived PSSMs;

- the *PROSITE* database [HBFB99]
  Collection of PSSMs and regular expressions of characteristic protein sequence motifs;

- the *HAMAP* protein family database [GMR$^+$03]
  HAMAP is a collection of multiple alignments of orthologous microbial protein families, generated manually by expert curators. They are used for the high-quality automatic annotation of microbial proteomes in the UniProtKB/SwissProt protein knowledge base;

- the *PRODOM* database [CGK99]
  Database containing multiple alignments of protein domain families;

- the commercial *TRANSFAC* database [WCF$^+$98]
  on eukaryotic cis-acting regulatory DNA elements and trans-acting factors, containing information on transcription factors, regulated genes, regulatory sites and binding sites of transcription factors modeled as PSSMs;

- the *JASPAR* database [SAE$^+$04]
  is a freely available, high quality transcription factor binding profile database. It is a non-redundant and curated collection of transcription factor DNA-binding preferences of multi cellular organisms, modeled as position specific scoring matrices. All models in this database are derived from published collections of experimentally defined transcription factor binding sites.

We now render the concept of *alignment blocks*, *PSSMs* and their relationship more precisely.

Definition 7 An *Alignment block* A of length m is a set of l sequences of length m over alphabet $\mathcal{A}$. A is represented as an $l \times m$ matrix with elements $a_{i,j} \in \mathcal{A}$ and $1 \leq i \leq l \wedge 1 \leq j \leq m$, with $a_{i,j}$ be the j-th character of the i-th sequence.

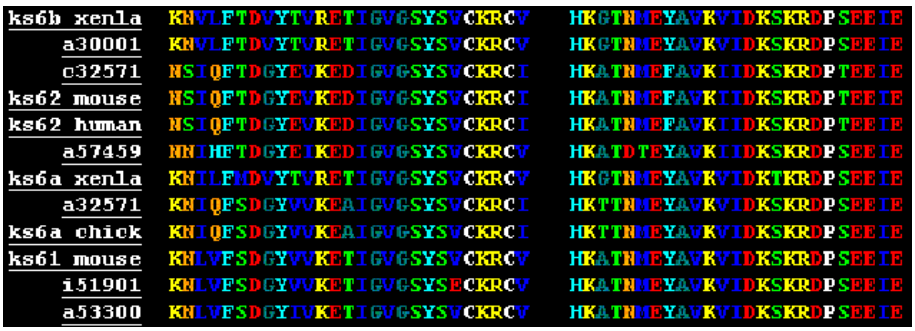


Figure 2.6: Alignment block representing a part of the multifunctional calcium/calmodulin-dependent protein kinase II (CaMKII), a kinase enriched in synapses. CaM kinases belong to the family of Ser/Thr protein kinases and have an extremely wide tissue distribution.

See Figure 2.6 for an example of an alignment block. A PSSM is an abstraction of an alignment block of related sequences (see Figure 2.7 for an example) and can be defined as follows.

Definition 8 A *PSSM* M is a function $M : [0, m-1] \times \mathcal{A} \to \mathbb{R}$, where m is the length of M and $\mathcal{A}$ is a finite alphabet. In case of matrices over the nucleotide alphabet we define the reverse complement $\overline{M}$ of the PSSM M as $\overline{M}(i, a) = M(m-1-i, \overline{a})$ for all $i \in [0, m-1]$ and $a \in \mathcal{A} = \{A, C, G, T\}$, where $\overline{a}$ is the Watson Crick complement of nucleotide a. That is $\overline{A} = T$, $\overline{C} = G$, $\overline{G} = C$, and $\overline{T} = A$.

Usually M and $\overline{M}$ are represented by an $m \times |\mathcal{A}|$ matrix, as shown in Table 2.4.

Large collections of curated, high quality alignment blocks allow the identification and function assignment of a sequence by comparing it against every alignment block. The first step for such a comparison is the conversion of each block into a PSSM. Several methods for this conversion have been developed [GME87, TAK94, TAK94, HH96, SKB+96, WNB99] over the last two decades. The basic task when computing a PSSM from a given alignment block is to estimate the probabilities of each amino acid or nucleotide appearing at each position of the alignment block and convert them into a scoring system that discriminates best between true positive and true negative members of the family represented by the alignment block. The scores in each row of a PSSM can be derived in a number of ways, but are naturally based on the frequency distribution of the characters observed in that position of the block, such that a nucleotide or amino acid that occurs more frequently receives a higher score.

In the following paragraphs we describe some widely used PSSM construction methods and illustrate the basic principles and underlying ideas. Since PSSMs are particularly used as motif descriptors for protein families, most of the construction methods below were originally developed for amino acid sequences.

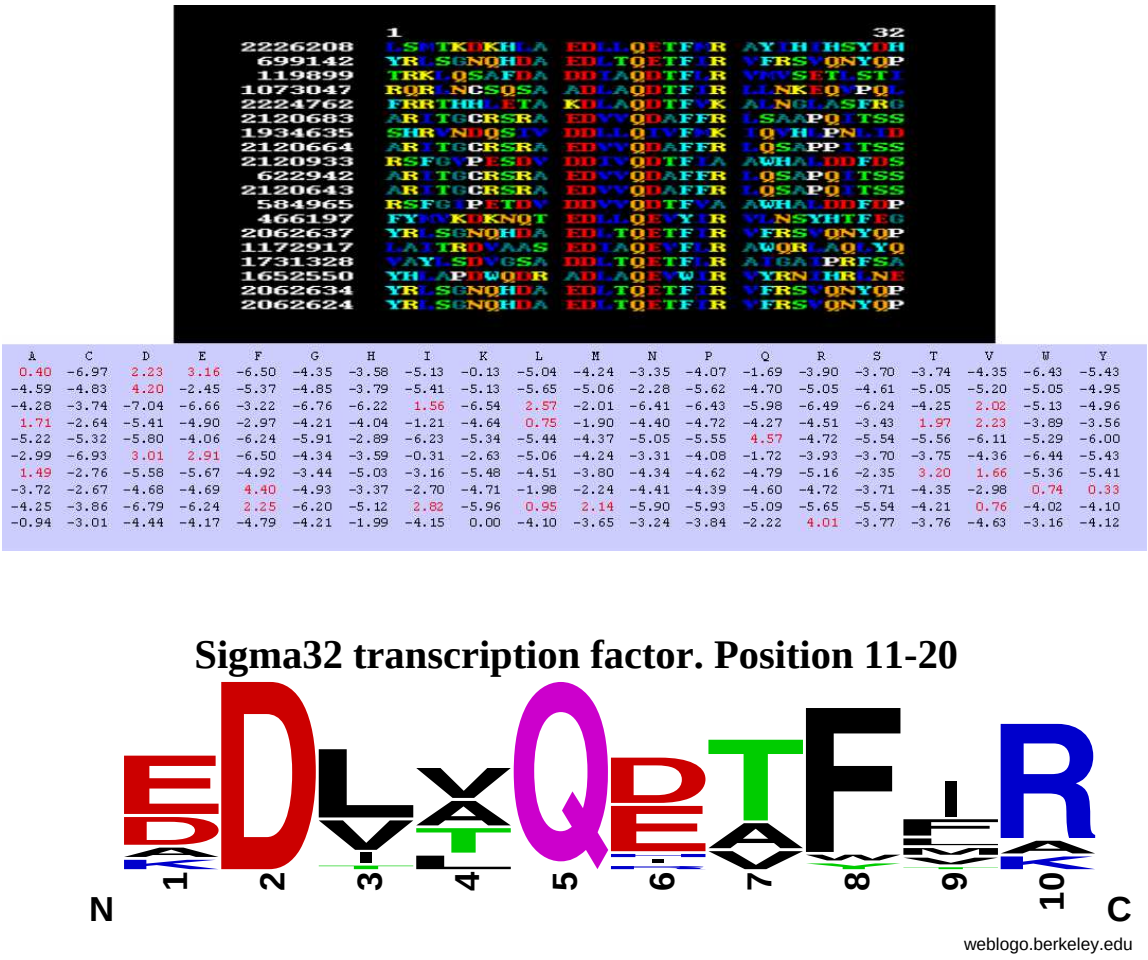


Figure 2.7: Multiple alignment of σ_{32} transcription factor sequences (top), corresponding PSSM (center) and sequence logo based on relative frequencies (bottom) (representing block from position 11 until position 20). Observe that high scoring values in the PSSM correspond to highly conserved residues in the multiple alignment and larger character symbols in the sequence logo.

(A)denin	(C)ytosin	(G)uanin	(T)hymin	(A)denin	(C)ytosin	(G)uanin	(T)hymin
28.50	256.54	85.51	28.50	47.70	47.70	28.62	9.54
28.62	47.70	47.70	9.54	106.40	148.95	21.27	21.28
45.54	45.54	45.54	500.92	41.19	32.95	8.24	32.95
320.83	0.00	71.29	106.94	96.46	41.34	13.78	41.34
47.29	15.76	15.76	31.53	31.53	15.76	15.76	47.29
41.34	13.78	41.34	96.46	106.94	71.29	0.00	320.83
32.95	8.24	32.95	41.19	500.92	45.54	45.54	45.54
21.28	21.27	148.95	106.40	9.54	47.70	47.70	28.62
9.54	28.62	47.70	47.70	28.50	85.51	256.54	28.50

Table 2.4: A 9×4 PSSM M (left matrix) and its reverse complement $\overline{M}$ (right matrix) of length $m = 9$ over the nucleotide alphabet $\mathcal{A} = \{A, C, G, T\}$, describing a transcription factor binding site motif found in the promotor sequences of HOX A3 genes. HOX (homeobox) genes and their regulation play a major role in developmental proliferation of cells. Example taken from the TRANSFAC database.

2.5.2 Sequence weighting procedures

Families of sequences are almost always highly biased and this correlation should not be ignored when aligning if it is sufficiently extensive. A typical protein family in sequence databases is a highly non-random sample of sequences where organisms with a long-term research tradition like *E.coli* or *D.melanogaster*, pathogens with medical impact like *H.pylori*, or economically relevant organisms like *S.cerevisae* are heavily over-represented irrespective of their evolutionary role. Beside such issues based on selection, statistical correlation between similar sequences may arise from their common evolutionary origin or as a result of similar functional requirements.

For the construction of PSSMs which subsequently will be used as a motif descriptor in database searches, it is common to find a group of sequences with a certain amount of diversity when constructing an alignment block. Some members of this group will be nearly identical, whereas others may be as little as 20% identical, when aligned. Obviously, each of the nearly identical sequences contributes much less information than each of the 20% identical sequences.

To compensate for over-representation among multiply aligned sequences which would lead to PSSMs that inadequately model the underlying sequence families and that would overspecialize to the over-represented sequences, various concepts of sequence weighting procedures have been developed. These methods give the *outlier* sequences, those that do not belong to the highly similar, over-represented group, additional importance in the calculation of the PSSM values.

For the weighting, numerical coefficients (sequence weights) are associated with each sequence to denote the degree of independence of this sequence from the others in the multiple alignment. For example, low weights are given to sequences that are redundant and high weights to sequences that are diverged. In its most drastic form, additional similar sequences are discarded from the set of sequences to be aligned, i.e. they achieve a zero weight, and only highly different sequences remain [NLLL97, HSHA92]. Other techniques use the full sequence information and can be roughly

categorized in two classes, evolutionary tree-based and sequence distance-based methods. Tree based approaches assume that the sequences in the set under consideration have a common evolutionary origin and are a result of divergent evolution and that an evolutionary tree can be constructed from sequence or additional information [THG94].

The distance-based methods [HH94, LXB94] avoid the problems of tree topology and even do not require that the sequences are related at all. Sequence weights are calculated from a residue similarity matrix, like PAM or BLOSUM or from the amino acid type diversity observed in each column of the multiple alignment. For example in the method proposed in [HH94], for each position in a sequence, a weight inversely proportional to the number of different amino acids in the column and the number of times the amino acid of interest appears in the column, is computed. The weight of a sequence is then the average of the weights in all positions, normalized to sum up to 1.

The approaches for sequence weighting described so far have in common, that a single sequence weight is assigned to a sequence. Implicitly this means, that the evolutionary rate of residue changes is believed to be equal in all positions of the sequence. In [SER+99] the authors introduce a sequence distance-based approach that incorporates position-specific sequence weights. They describe that their method based on position-specific independent counts, produces PSSMs, that are in many cases more powerful in detecting members of protein fold families, than e.g. PSI-BLAST derived PSSMs [AMS+97]. Although many different sequence weighting procedures are described in literature, it remains to be difficult to identify a single, best weighting method, since the choice of the weighting method depends both on what the resulting PSSM will be used for and the particular group of sequences being modeled.

Methods that determine a PSSM from a set of aligned sequences basically face two difficulties. First, the problem of interdependence between sequences in the underlying alignment block (biased data), and second, the problem how to derive a PSSM from an alignment block adequately, especially when the number of aligned sequences is small. The first problem can be addressed with sequence weighting procedures. Principles and approaches for the second problem will be presented in the following. For the methods to be described, we assume that the sequences in the alignment block are already weighted to compensate selection biases and potential redundancies.

2.5.3 Basic PSSM construction principles

The first step when generating a PSSM is to determine a matrix based on of absolute residue frequencies. This is also fundamental for more complex PSSM generation methods. In literature such counting matrices are often also called *profiles*, position frequency matrices (PFMs for short) or simply *count matrices*. Although in literature some authors distinguish between counting matrices, profiles, and PSSMs we use the generic term PSSM for all these types of matrices throughout this work, since for the matching algorithms described in the next chapter, it is irrelevant how the matrix values are determined and what their semantics are. However, we will see that some matrix value determination principles are better suited for scoring in a database search scenario than others.

In a PSSM based on residue counts, each element in the matrix contains the number of occurrences of a certain residue at a specific position in the alignment block. More precisely: Let A be an

$$A: \begin{bmatrix} 1 & 2 & 3 \\ G & T & A \\ A & A & A \\ T & C & C \\ G & G & G \\ C & C & T \end{bmatrix} \quad M_A = \begin{bmatrix} 1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 1 \\ 2 & 1 & 1 & 1 \end{bmatrix}$$

Figure 2.8: Let A be an alignment block of 5 sequences of length 3 over the nucleotide alphabet $\mathcal{A} = \{A, C, G, T\}$. The PSSM M with $m_{i,a} = \texttt{count}_A(i, a)$ based on absolute frequencies frequency corresponding to A is shown on the right.

alignment block of l sequences of length m over alphabet $\mathcal{A}$, denoted by $S_1, \ldots, S_l$, and $\delta(i, a, k),\ i \in [0, m-1],\ a \in \mathcal{A},\ k \in [1, l]$ a Kronecker symbol, such that

$$\delta(i, a, k) = \begin{cases} 1, & \texttt{if } S_k[i] = a, \\ 0, & \texttt{otherwise}. \end{cases} \tag{2.2}$$

Now let $\texttt{count}_A$ be a function $[0, m-1] \times \mathcal{A} \to \mathbb{N}$ defined as

$$\texttt{count}_A(i, a) := \sum_{k=1}^{l} \delta(i, a, k) = |\{S \in A \mid S[i] = a\}|$$

Hence a PSSM M based on absolute frequencies (counts) is a $m \times |\mathcal{A}|$ matrix with

$$m_{i,a} = \texttt{count}_A(i, a) \tag{2.3}$$

See Figure 2.8 for an example of an alignment block over the nucleotide alphabet and the corresponding PSSM based on absolute frequencies.

Absolute frequencies are easily converted to relative frequencies. That is, each element of the PSSM contains the fraction of the total number of occurrences of a residue at a specific position in the alignment block and the number of aligned sequences. Let l be the number of aligned sequences in the alignment block, then a PSSM based on relative frequencies is a $m \times |\mathcal{A}|$ matrix M with

$$m_{i,a} = \frac{\texttt{count}_A(i, a)}{l} \tag{2.4}$$

In a database searching scenario, where a PSSM is used as a descriptor for a family of related sequences, PSSMs based on residue counts or relative frequencies are not appropriate. Counts or relative frequencies are an imperfect representation of a column in the alignment block since they do not take the background residue frequencies into account. For an example, reconsider the alignment block and derived PSSM given in Figure 2.8. Here $m_{1,G} = m_{2,C} = m_{3,A}$ holds and consequently, when using this PSSM in a database search, a guanine in the first position achieves the same score as a cytosine in the second and an adenine in the third position, although it is probably more likely to see a cytosine just by chance than a guanine, since cytosine occurs more often in the sequences to be searched than guanine. Further, the sequences included in the alignment block are an incomplete sample of the full set of related sequences and hence the derived counts or frequencies can be misleading and deviate significantly from the frequencies of the whole family. This leads to

the basic problem of how to convert observed counts into true frequencies or scores, adequately. The set of observed counts is finite and almost always contains zero counts for one or more amino acids or nucleotides. However, zero frequencies are undesirable, because they may exclude true but unusual members of a given family. Therefore, some methodology have to be used to estimate the true frequencies at a position in the sequence based on the observed counts in the corresponding column in the alignment block.

2.5.4 PSSMs based on odds ratios

Instead of using residue counts or relative frequencies one ideally would like to create an odds ratio based score for each column in the alignment block. Residue counts can be converted to odds ratios of probabilities that are expected to be observed [BH87, SH90]. Let $q_{i,a}$ be the unknown probability for residue a occurring in column i of the alignment block and p_a be the expected frequency of a in a random sequence, which can be estimated from the overall (background) frequency of residue a in a large sequence database. As the number of sequences l in the alignment block A grows, estimates of $q_{i,a}$ should converge to the relative frequencies $\frac{\texttt{count}_A(i,a)}{l}$. Thus, for sufficiently large values of l, we can estimate the odds ratio of residue a appearing in column i of the alignment block as

$$\frac{q_{i,a}}{p_a} \approx \frac{\texttt{count}_A(i,a)}{l \cdot p_a} \tag{2.5}$$

In [HWB90] these odds ratios are directly used as score values in a PSSM. That is, a PSSM based on odd ratios is a $m \times |\mathcal{A}|$ matrix M with

$$m_{i,a} = \frac{q_{i,a}}{p_a}. \tag{2.6}$$

To achieve a more convenient additive scoring system, some methods use log-odds ratios instead of simple odds ratios [BH87, LAB$^+$93]. In these methods a PSSM based on log-odd ratios is defined as a $m \times |\mathcal{A}|$ matrix M with

$$m_{i,a} = \log\left(\frac{q_{i,a}}{p_a}\right). \tag{2.7}$$

In this case the PSSM values $m_{i,a}$ are the log of the ratio of two probabilities - the probability that symbol a occurs at position i in the family described by the PSSM and the probability that a occurs at position i just by chance. p_a is often also called the probability of the *null model*, or background probability, since it expresses how likely it is to see symbol a by chance. Although log-odd ratios provide a simple, additive scoring system that maximizes selectivity for observed residues, log-odd ratio based PSSMs have some drawbacks:

- **Residue similarity problem:** They do not take conservative replacements of residues into account and hence may fail to detect distantly related members of the family.

- **Incomplete sample problem:** There are often not enough sequences included in the alignment block A, making A an incomplete sample of the full set of related sequences. In practice, the number of sequences needed to accurately estimate the expected amino acids at each position in a protein is often larger than the number typically available in an alignment block

for most protein families. Moreover, it is prevalently the case that the available data is skewed towards one or more subfamilies of the protein being modeled, such that a large fraction of the sequences are highly redundant and minor variants of each other. In such a case $\frac{\texttt{count}_A(i,a)}{l}$ is not an adequate estimate for $q_{i,a}$. This problem is known as *overfitting* and a variety of approaches commonly known as *regularization* have been developed to deal with it.

- **Zero count problem:** For a specific column $i \in [0, m-1]$, $\texttt{count}_A(i,a) = 0$ often holds for some $a \in \mathcal{A}$, especially for relatively small values of l. This converts to odd ratios of zero. A zero count might indicate that the residue cannot occur in position i, or, which is much more likely in most cases, it is the consequence of insufficient knowledge about the true instances of the model. That is, not enough truly related sequences are included in the alignment block A. Additionally, in either case a technical problem arises with zero counts. $\texttt{count}_A(i,a) = 0$ would lead to undefined values in the PSSM if scores based on logarithms are used, since $\log(0)$ is undefined.

2.5.5 Average score methods

PSSMs based on simple odds ratios or log-odds ratios do not take similarities between certain characters of the alphabet into account, for example a substitution of amino acid leucin by a chemically similar one like isoleucin. PSSMs based on score averaging methods as firstly introduced by Gribskov [GME87] and successfully used in [TAK94] address this issue by weighting residues with a similarity score based on their biochemical properties. The entries of such a PSSM are calculated by averaging scores from a substitution matrix like PAM or BLOSUM. An average is taken of all pairwise scores obtained from the used substitution matrix for an aligned residue and each of the residues seen in the column under consideration. Unobserved residues receive scores based on their presumed association with the observed residues.

For a given $|\mathcal{A}| \times |\mathcal{A}|$ substitution matrix S, a PSSM M based on the average score method according to [GME87] is a matrix with

$$m_{i,a} = \sum_{b \in \mathcal{A}} w_{i,b} s_{a,b}, \tag{2.8}$$

where $s_{a,b}$ denotes the similarity matrix score for residue a replacing residue b and $w_{i,b}$ is a weight for the appearance of residue b in column i of the alignment block. For a simple average weighting $w_{i,b}$ can be determined as

$$w_{i,b} = \frac{\texttt{count}_A(i,b)}{l}, \tag{2.9}$$

and for logarithmic weighting as

$$w_{i,b} = \log\left(\frac{\texttt{count}_A(i,b)}{l}\right) \tag{2.10}$$

while setting $\texttt{count}_A(i,b) = 1$ for any residue not appearing in column i. Consequently Gribskov's average PSSMs address the residue similarity problem and can deal with zero counts.

The major criticism on Equation (2.8) is, that it is purely heuristic and does not rely on any statistical model of (protein) sequence family evolution. The notion of amino acid substitution

matrices implicitly accepts that the mutation probabilities or in other words the evolutionary rates are identical at every position of the protein family, an assumption which is somehow opposite to the basic idea of a position specific scoring matrix.

Observe that the odds ratio score given in Equation (2.6) can also be interpreted as an average score that uses a simple substitution matrix S with

$$s_{a,b} = \begin{cases} \frac{1}{p_a}, & \texttt{if } a = b, \\ 0, & \texttt{otherwise}. \end{cases} \tag{2.11}$$

Another approach to derive a PSSM based on average scores is described in [Alt91]. In this contribution, the authors showed that any substitution matrix has a log-odds score interpretation. That is, substitution scores can be interpreted as scaled log-odds with an implicit set of amino acid pair substitution probabilities $q_{a,b}$. More precisely,

$$s_{a,b} = \frac{1}{\lambda} \log \left(\frac{q_{a,b}}{p_a p_b} \right). \tag{2.12}$$

Here $\frac{1}{\lambda}$ is a scaling factor and p_a and p_b denote the background probabilities of residues a and b. Thus Equation (2.8) can be rewritten as

$$m_{i,a} = \frac{1}{\lambda} \sum_{b \in \mathcal{A}} w_{i,b} \log \left(\frac{q_{a,b}}{p_a p_b} \right). \tag{2.13}$$

and we notice that the average score is a weighted average of log-odds ratios. To explicitly retain a log-odd interpretation, it can be considered to weight each odds ratio before taking the log [HH96] as

$$m_{i,a} = \log \left(\sum_{b \in \mathcal{A}} w_{i,b} \frac{q_{a,b}}{p_a p_b} \right). \tag{2.14}$$

A potential drawback of averaging methods is that they do not take the number of sequences in the alignment block into account. When there are only few sequences and the actual distributions of residues in a certain column are uncertain, they make sensitive PSSMs. However, with an increasing number of sequences, averaging substitution values reduce PSSM specificity [HH96]. The average PSSM method also does not adequately emphasize positions that are highly conserved. Consider, for example, a residue that is absolutely conserved in every sequence in a family of 100 sequences. Such a position is required, often participating in critical structures or functions such as the active site of an enzyme. Using simple average weighting (Equation (2.9)), in a PSSM based on Gribskov's average score method such a column would result in a row of values identical to the corresponding row for the conserved residue in the substitution matrix S (see Equation (2.8)). This inability to properly model highly conserved residues was the motivation for a further refinement. In [GV96] Gribskov and Veretnik introduced a new approach for the computation of a PSSM from a given alignment block, called *evolutionary profiles*. One of their basic ideas is to take into account that the amount of conservation among protein sequences varies widely from position to position. Thus any position in a sequence should be allowed to evolve at its own evolutionary rate. This implies modeling different positions in a sequence using different substitution matrices, each corresponding to a rate of change at different evolutionary distances.

2.5.6 Explicit log-odd score methods

Another widely used method to deviate a PSSM from a given alignment block is introduced in [HH96]. This method, widely simply known as *Henikoff's method*, is, in contrast to Gribskov's average score method, explicitly based on log-odd scores. That is $m_{i,a} = \log\left(\frac{q_{i,a}}{p_a}\right)$ with $q_{i,a}$ estimated as

$$q_{i,a} \approx \frac{l}{l+B_i}\frac{\mathtt{count}_A(i,a)}{l} + \frac{B_i}{l+B_i}\frac{b_{i,a}}{B_i}. \tag{2.15}$$

Before we describe this formula in more detail, reconsider that odd ratios equal to zero, caused by zero residue frequencies $\frac{\mathtt{count}_A(i,a)}{l}$ used for the estimation of $q_{i,a}$, have to be prevented, when using log-odd scores. In Gribskov's average score method with logarithmic weighting (see Equations (2.8) and (2.10)) this was addressed by simply setting $\mathtt{count}_A(i,a) = 1$ for any residue not appearing in column i. Alternatively, zero residue frequencies can be avoided by adding some kind of hypothetical sequences to the alignment block. For each column i in the alignment block, this involves adding *pseudo-counts* to the observed counts $\mathtt{count}_A(i,a)$ based on some belief about the actual, incompletely observed, distribution of residues in that column. This means that, even if a given amino acid does not appear in a column of the alignment block, it is given a fake count. Fake counts are also added for the amino acids which appear in the column, e.g. we add one to each count. This is the simplest pseudo-count method also known as Laplace's rule. When probabilities are calculated, the fake counts are treated exactly like real observed counts. Thus, for simple Laplace pseudo-counts, $q_{i,a}$ can be estimated as

$$q_{i,a} \approx \frac{\mathtt{count}_A(i,a)+1}{l+1} \tag{2.16}$$

for any $a \in \mathcal{A}$. Though this is a very simple approach to avoid the zero count problem it has some disadvantages and does not perform well in practice. In the following we will discuss more sophisticated methods to choose the pseudo-count values B_i and $b_{i,a}$.

In Henikoff's method (Equation (2.15)), $b_{i,a} > 0$ is the number of pseudo counts added to the observed count of residue a in column i, and $B_i = \sum_{a\in\mathcal{A}} b_{i,a}$ the total number of pseudo counts added in column i. Both $\frac{\mathtt{count}_A(i,a)}{l}$ and $\frac{b_{i,a}}{B_i}$ are estimates for the probability $q_{i,a}$ of residue a appearing in column i and Equation (2.15) expresses a weighted average between them. The relative sizes of l and B_i balance, whether the observed counts dominate, when l is large with respect to B_i or the pseudo-counts when B_i is large with respect to l. With the usage of pseudo-counts it is guaranteed that $q_{i,a} > 0$ holds for any $i \in [0, m-1]$ and $a \in \mathcal{A}$. Consequently, well defined PSSM scores can be computed as log-odd ratios according to Equation (2.7).

Another PSSM construction method explicitly based on log-odd scores is described in [LAB+93]. Here $q_{i,a}$ values are estimated as

$$q_{i,a} \approx \frac{\mathtt{count}_A(i,a) + B_i \cdot p_a}{l+B_i}. \tag{2.17}$$

That is, pseudo counts B_i are added to the observed counts $\mathtt{count}_A(i,a)$ in proportion to the background probability p_a for some residue a. This has the appealing feature, that $q_{i,a}$ is approximately equal to the background probability p_a if only a few sequences are available, i.e. all the real counts are very small compared to B_i. At the other extreme, where many sequences are available, the

effect of the added pseudo-counts becomes insignificant and $q_{i,a}$ is essentially equal to the relative frequency $\frac{\texttt{count}_A(i,a)}{l}$.

Although, with adding pseudo-counts to observed counts the zero count problem and to some extent the overfitting problem, since pseudo-counts can be used as regularizers, are addressed in general, it remains open how to choose an adequate number of pseudo-counts.

Determination of pseudo counts

Several different methods for calculating pseudo counts have been proposed over the last decade. In [LAB+93] $b_{i,a}$ is simply taken to be proportional to the overall frequency of residue a in a sequence. That is $b_{i,a} = B_i \cdot p_a$. A major criticism of this method is, that it does not take possible constraints imposed by residues observed in a column into account. For example if a certain residue a is observed, then the pseudo-count for residues b with high substitution probabilities $p_{b,a}$ (like leucine-isoleucine mutations) should be higher than the background frequency p_a would imply and the pseudo-count for residues with low substitution probabilities should be less. This drawback was the motivation for an improvement introduced in [TAK94]. More precisely, let $p_{a,b}$ be the probability that residue b is substituted by residue a, then the pseudo-count $b_{i,a}$ can be calculated based on residue substitution probabilities as

$$b_{i,a} = B_i \sum_{b \in \mathcal{A}} p_{b,a}. \tag{2.18}$$

This approach takes residue similarities into account by adding substitution probabilities, but the residues actually observed in a certain column are neglected. To take the observed residues into account, the authors of [TAK94] propose to calculate $b_{i,a}$ as

$$b_{i,a} = B_i \cdot \sum_{b \in \mathcal{A}} \frac{\texttt{count}_A(i,b)}{l} \frac{p_{b,a}}{\sum_{b \in \mathcal{A}} p_{b,a}}. \tag{2.19}$$

A similar method using similarity scores instead of substitution probabilities was proposed in [Cla94].

Selecting the total number of pseudo-counts

So far, we described approaches for determination of pseudo counts $b_{i,a}$ that are added to the observed count of residue a in column i of the alignment block, given the total number of pseudo counts B_i. It remains open, how to adequately choose B_i. In literature, B_i is often estimated to be some function of the number of sequences in the alignment block, independent of i. For example, in [LAB+93, TAK94] the authors chose $B_i \approx \sqrt{l}$ based on empirical estimations. In [HH96] the authors report that this choice is not ideal, especially when the number of sequences in the alignment block is small, since the number of pseudo-counts can never exceed the number of counts. They propose to compute position-based pseudo counts B_i for each column $i \in [0, m-1]$, instead of using the same number of pseudo-counts for all columns of the alignment block. The basis for their computation of B_i is to take residue diversity into account. That is, a conserved column in the alignment block requires fewer total pseudo counts than a diverse column. Let R_i be the number of observed different residues in column i. Then the position specific number of pseudo counts B_i is computed as

$$B_i = \gamma R_i, \tag{2.20}$$

where γ is an empirically determined positive number. Since $1 \leq R_i \leq |\mathcal{A}|$ holds, it follows that:

$$\gamma \leq \gamma \cdot R_i \leq \min\{\gamma \cdot l, \gamma \cdot |\mathcal{A}|\} \tag{2.21}$$

Using position based pseudo-counts, Equation (2.15) can be rewritten as

$$q_{i,a} \approx \frac{l}{l + \gamma R_i} \frac{\mathtt{count}_A(i,a)}{l} + \frac{\gamma R_i}{l + \gamma R_i} \frac{b_{i,a}}{\gamma R_i}. \tag{2.22}$$

Hence for position based pseudo-counts, pseudo-counts dominate observed counts, if $l < \gamma \cdot |\mathcal{A}|$ holds. For $l > \gamma \cdot |\mathcal{A}|$ observed counts always dominate, regardless of R_i. As a consequence Equation (2.15) tends to $\frac{\mathtt{count}_A(i,a)}{l}$ for larger values of l, as required. For a highly conserved column in the alignment block, that is $R_i = 1$, observed counts dominate, if $l > \gamma$.

Pseudo-counts based on Dirichlet mixtures

Another sophisticated method to compute position specific pseudo-counts, similar in their general form to the substitution probability method (see Equation (2.19)), is known as Dirichlet-mixtures. The mixtures are created by statistical analysis of the distribution of amino acids at particular positions in a large number of proteins. Rather than using pairwise residue substitution data, probabilities $q_{i,a}$ are derived from mixtures of Dirichlet densities computed using prior information [BHK$^+$93, SKB$^+$96]. Here a Dirichlet density is a probability density over all possible combinations of amino acids appearing in a given position. It gives high probability to certain distributions and low probability to others. For example, a particular Dirichlet density may give high probability to conserved distributions where a single amino acid predominates over all others. Another possibility is a density where high probability is given to amino acids with a common identifying feature, such as the subgroup of hydrophobic amino acids. Now, the idea is to incorporate such prior information about residue distributions that typically occur in columns of multiple alignments into the process of building a statistical model. In [SKB$^+$96] the authors present a method to condense the information in databases of multiple alignments into a mixture of Dirichlet densities over amino acid distributions and to combine this prior information with the observed amino acid counts $\mathtt{count}_A(i,a), a \in \mathcal{A}$, to form more effective estimates of the expected distributions.

2.5.7 Construction of amino acid PSSMs in the BLOCKS database

PSSMs used in the BLOCKS database searching application *BLIMPS* [HHAP95] are based on log-odd scores (see Equation (2.7)). Probabilities $q_{i,a}$ are estimated according to Equation (2.15) and pseudo-counts $b_{i,a}$ based on substitution probabilities are computed using Equation (2.19). Position based pseudo-counts B_i are determined using Equation (2.20). Hence a PSSM derived from an alignment block A of length m of protein sequences, as used by the *BLIMPS* program, is a $m \times 20$ matrix M with

$$m_{i,a} = \log\left(\frac{q_{i,a}}{p_a}\right) \tag{2.23}$$

with

$$q_{i,a} = \frac{l}{l+\gamma\cdot R_i}\cdot\frac{\mathtt{count}_A(i,a)}{l} + \frac{\gamma\cdot R_i}{l+\gamma\cdot R_i}\cdot\frac{\gamma\cdot R_i\sum_{b\in\mathcal{A}}\frac{\mathtt{count}_A(i,b)\cdot p_{b,a}}{l\cdot\sum_{c\in\mathcal{A}}p_{c,a}}}{\gamma\cdot R_i} \tag{2.24}$$

$$= \frac{\mathtt{count}_A(i,a)}{l+\gamma\cdot R_i} + \frac{\gamma\cdot R_i}{l+\gamma\cdot R_i}\cdot\sum_{b\in\mathcal{A}}\frac{\mathtt{count}_A(i,b)\cdot p_{b,a}}{l\cdot\sum_{c\in\mathcal{A}}p_{c,a}} \tag{2.25}$$

2.5.8 Wu's minimal risk scoring matrices

In [WNB99] a minimal-risk method for the estimation of frequencies of amino acids at a conserved position in a protein family is introduced. The method finds the optimal weighting between a set of observed amino acid counts and a set of pseudo-frequencies which represent prior information about the frequencies, by computing the optimal number of pseudo-counts to add. Optimality is defined by a criterion called *risk*, which is the expected distance between the estimated frequencies and true population frequencies, determined from the background distribution of amino acids or from applying a substitution matrix to the observed data. The optimal weighting is computed by minimizing the risk, measured by either a squared-error or relative-entropy metric. The resulting frequency estimates are then used to estimate the probabilities $q_{i,a}$ and minimal-risk PSSMs are constructed based on log-odd scores, like the PSSMs used by *BLIMPS* described above (see Equation (2.23)). The method is implemented in the program *eMatrix-maker* [4]. Furthermore several databases exist containing PSSMs constructed with the minimal-risk method, e.g. the databases *eBLOCKS* [SLSB05] and *eSIGNAL* [5].

In the following, we describe the main ideas of minimal-risk PSSMs more precisely. Let $\vec{\hat{f}}_i$, $i \in [0, m-1]$ be a vector of dimension $|\mathcal{A}|$, with $\hat{f}_{i,a} = m_{i,a}$ and $m_{i,a}$ defined according to Equation (2.4), that denotes the observed frequencies of symbols $a \in \mathcal{A}$ in column i of the alignment block. These frequencies are generated by some unknown true population frequencies $\vec{f}_i$, which should be estimated by $\vec{f}_i^*$ as well as possible. Wu and coworkers propose to estimate an optimal $f_{i,a}^*$ as a weighted sum of the observed frequencies and pseudo-frequencies,

$$f_{i,a}^*(\beta_i, \lambda_{i,a}) = (1-\beta_i)\hat{f}_{i,a} + \beta_i\lambda_{i,a}. \tag{2.26}$$

The β_i values are weights and $\lambda_{i,a}$ denote the pseudo-frequency of symbol a in column i of the alignment block. The method allows to use background frequencies, as well as substitution pseudo-frequencies for the determination of $\lambda_{i,a}$. That is

$$\lambda_{i,a} = \begin{cases} f(a), & \text{[Background frequencies]} \\ \sum_{b\in\mathcal{A}} S_{a,b}\cdot\hat{f}_{i,a} & \text{[Substitution frequencies].} \end{cases} \tag{2.27}$$

Here $f(a)$ denotes the background frequency for symbol a and S is a $|\mathcal{A}| \times |\mathcal{A}|$ substitution matrix containing residue similarity information, e.g. a PAM or BLOSUM matrix. Then, $S_{a,b}$ represents the conditional probability of seeing amino acid a given amino acid b. To solve Equation (2.26) we have to compute a weight β, such that $\vec{f}_i^*$ approximates $\vec{f}_i$ as well as possible. To estimate an

[4] `http://brutlag.stanford.edu/ematrix-maker`
[5] `http://brutlag.stanford.edu/esignal/`

optimal weight β, denoted by β^*, a criterion of optimality called *risk* is defined as the expected distance between estimated frequencies $\vec{f}_i^*$ and the yet unknown true frequencies $\vec{f}_i$. For given $\hat{\vec{f}}_i$ and $\vec{\lambda}_i$, we advocate to choose parameters that minimize the distance between $\vec{f}_i^*$ and $\vec{f}_i$. Distance computation can be performed using a squared error metric or based on relative entropy, leading to the following definitions of *risk* R:

$$R = E\left(|\vec{f}_i^* - \vec{f}_i|^2\right) = \begin{cases} \sum_{a\in\mathcal{A}} E(f_{i,a}^* - f_{i,a})^2 & \text{[Squared error]} \\ \sum_{a\in\mathcal{A}} E\left(f_{i,a}^* \log\left(\frac{f_{i,a}^*}{f_{i,a}}\right)\right) & \text{[Relative entropy].} \end{cases} \tag{2.28}$$

Evaluation experiments performed in [WNB99] showed that PSSMs using the squared error metric for frequency estimation perform better than PSSMs using the relative entropy metric. Hence we restrict in the following on the first to explain the method. For the squared error metric, Wu and coworkers showed that weight β_i^* is optimal, if the following two relationships between β_i^* and the true frequencies $\vec{f}_i$ hold.

$$\beta_i^* = \begin{cases} \frac{1-\sum_{a\in\mathcal{A}} f_{i,a}^2}{1-\sum_{a\in\mathcal{A}} f_{i,a}^2 + l\sum_{a\in\mathcal{A}}(f_{i,a}-f(a))^2} & \text{[Background frequencies]} \\ \frac{1-\sum_{a\in\mathcal{A}}(S_{a,a}f_{i,a}+f_{i,a}(f_{i,a}-s_{i,a}))}{1+\sum_{a\in\mathcal{A}}[(l-1)(f_{i,a}-s_{i,a})^2-2S_{a,a}f_{i,a}+\sum_{b\in\mathcal{A}} S_{a,b}^2 f_{i,b}]} & \text{[Substitution frequencies]} \end{cases} \tag{2.29}$$

where

$$s_{i,a} = \sum_{b\in\mathcal{A}} S_{a,b} f_{i,b} \tag{2.30}$$

For a detailed derivation of Equation (2.29) from Equation (2.28) see the Appendix in [WNB99]. The first case in Equation (2.29) describes the relationship when using background frequencies as pseudo frequencies $\lambda_{i,a}$, the second case when using substitution frequencies. With Equation (2.29) an initial estimate for the unknown frequencies $\vec{f}_i$ can be used to achieve a better estimate. More precisely, the initial estimate for $\vec{f}_i$ serves as a starting point to determine (1) a weight β^* using Equation (2.29) and (2) (more accurate) frequencies $\vec{f}_i^*$ by using β^* in Equation (2.26). This procedure may be applied iteratively, but when the number of sequences in the alignment block is small, an iterative approach can lead to progressive overfitting and poor estimates. As an initial estimate, Wu chose

$$f_{i,a} = \frac{\texttt{count}_A(i,a)}{l+B} + \frac{B\cdot\lambda_{i,a}}{l+B}, \quad a\in\mathcal{A}. \tag{2.31}$$

Here B denotes the total number of pseudo-counts to add and was chosen by Wu proportional to the number of sequences l as $B = \sqrt{l}$.

Finally the score values of a minimum risk PSSM are computed as log-odd scores from the frequency estimates as

$$m_{i,a} = \log\left(\frac{f_{i,a}^*}{f(a)}\right), \quad i\in[0,m-1],\ a\in\mathcal{A}. \tag{2.32}$$

2.5.9 Construction of nucleotide PSSMs in the TRANSFAC database

As a final example of a construction method for PSSMs from a given alignment block we describe the PSSM building process in the *TRANSFAC* database. In *TRANSFAC* the PSSMs representing transcription factor binding site motifs are generated based on weighted, relative frequencies ([KGR+03] and personal communication with A.Kel). Again, let $\hat{f}_{i,a} = \frac{\texttt{count}_A(i,a)}{l}$ be the observed

A	A	G	A	A	-	A
A	T	-	A	A	T	G
C	T	G	-	G	-	G
C	C	-	A	G	T	T
C	C	G	-	G	-	-

(A)denin	(C)ytosin	(G)uanin	(T)hymin	(-) gap
0.4	0.6	0	0	0
0.2	0.4	0	0.4	0
0	0	0.6	0	0.4
0.6	0	0	0	0.4
0.4	0	0.6	0	0
0	0	0	0.4	0.6
0.2	0	0.4	0.2	0.2

Figure 2.9: A multiple alignment of length seven of five nucleotide sequences (left) and a corresponding PSSM, containing relative frequencies and position specific gap costs (right)

relative frequency of nucleotide a to occur in column i. Then a TRANSFAC PSSM of length m is a $m \times |\{A, C, G, T\}|$ matrix M with

$$m_{i,a} = I(i) \cdot \hat{f}_{i,a}. \tag{2.33}$$

Here the observed relative frequency $\hat{f}_{i,a}$ is weighted with the information vector $I(i)$ defined as

$$I(i) := \sum_{a \in \Sigma} \hat{f}_{i,a} \cdot \ln(|\mathcal{A}| \cdot \hat{f}_{i,a}). \tag{2.34}$$

The information vector describes the conservation of the position i in a matrix. The intention here is, that the multiplication of the frequencies with the information vector should result in a higher acceptance of mismatches in less conserved regions, whereas mismatches in highly conserved regions are very much discouraged. In [KGR+03] the authors claim, that this leads to a better performance in recognition of transcription factor binding sites, compared with methods that do not use the information vector [KKMBW99].

2.6 Gribskov's profile model

Gribskov's profile model, introduced and described in [GME87] and [GLE90], extends the concept of PSSMs according to Definition 8 on page 23 by facilitating position-dependent penalties for the modeling of insertions and deletions. The underlying idea is, that insertions and deletions in multiple alignments of related sequences occur at different positions with different frequencies, depending of the variability or degree of conservation at these positions. Accordingly, position dependent insertion/deletion (gap) costs should be incorporated into the PSSM model. In Gribskov's profile model, an additional column in the matrix contains these information. See Figure 2.9 for an example of a PSSM with position specific gap costs derived from a gap-containing multiple sequence alignment. Observe, that in this model no difference is being made between insertion and deletion costs, since an insertion in one sequence can be viewed as a deletion in another.

To use a Gribskov profile in a database search, sequences are aligned to the profile using dynamic programming and the alignment is rated with a score. The general idea of the method is similar to the alignment of two sequences and can be extended to the comparison of two profiles [Got93].

In case of aligning a single sequence to a Gribskov profile, the profile is viewed as a string, where each row represents a character. The objective is to compute an optimal alignment of the string and the Gribskov profile where the score reflects how well the string fits the profile. We make this more precise now.

Let $M_\sqcup : [0, m-1] \times \mathcal{A}_\sqcup \to \mathbb{R}$, be a PSSM over a finite alphabet $\mathcal{A}_\sqcup = \mathcal{A} \cup \{\sqcup\}$ that includes a special gap symbol $\sqcup$. For an alignment of a sequence $S = s_0 \dots s_n$ and a PSSM with position specific gap costs $M_\sqcup$, we need a scoring function that should express the aberration of a character $c \in \mathcal{A}$ of S from the j-th row of $M_\sqcup$, $j \in [0, m-1]$. We assume that a pairwise scoring function $\sigma : \mathcal{A}_\sqcup \times \mathcal{A}_\sqcup \to \mathbb{R}$ for all characters in $\mathcal{A}_\sqcup$ exists. This can be based on normal PAM or BLOSUM scoring matrices. Assume that $M_\sqcup$ is based on relative frequencies, then we may choose the scoring function

$$score(c, i) = \sum_{c' \in \mathcal{A}_\sqcup} \sigma(c, c') M_\sqcup(c', i), i \in [0, m-1] \tag{2.35}$$

to model a position dependent scoring. This function performs a weighted comparison of a character c with the values of row i of $M_\sqcup$ and with the characters occurring in column i of the multiple alignment respectively. E.g., the score for matching character G to the second row of the profile given in Figure 2.9 is $score(G, 2) = 0.2 \cdot \sigma(G, A) + 0.4 \cdot \sigma(G, C) + 0.4 \cdot \sigma(G, T)$. The optimal alignment of the sequence S and the PSSM with position specific gap costs $M_\sqcup$ can now be computed by applying dynamic programming. We denote the optimal alignment of the prefix $s_0, \dots, s_i$ of S and the j-th row vector of $M_\sqcup, j \in [0, m-1]$ with $V(i, j)$. With the following recurrences for the dynamic programming matrix

$$\begin{aligned}
V(i, -1) &= \sum_{k \le i} \sigma(\sqcup, s_k) \cdot |\mathcal{A}| \cdot M_\sqcup(k, \sqcup) \\
V(-1, j) &= \sum_{k \le j} score(k, \sqcup) \\
V(i, j) &= \max \begin{cases} V(i-1, j-1) + score(j, s_i), \\ V(i-1, j) + \sigma(s_i, \sqcup) \cdot |\mathcal{A}| \cdot M_\sqcup(j, \sqcup), \\ V(i, j-1) + score(j, \sqcup) \end{cases}
\end{aligned}$$

the optimal alignment between a sequence S of length $n+1$ and a PSSM with position specific gap costs $M_\sqcup$ of length m can be calculated in $O(|\mathcal{A}_\sqcup| mn)$ time and $O(mn)$ space. Likewise for pairwise sequence alignment, this algorithm can be extended to an algorithm using an affine gap-cost model with the same time and space complexity.

In practice PSSMs with position specific gap costs are not prevalent. To our knowledge they are only used as (additional) motif descriptors in the PROSITE [HBFB99] and HAMAP [GMR$^+$03] databases. In most situations where it is necessary to include gap information, due to practical concerns, like the need to model longer or more variable parts of sequences, a different motif descriptor model like the subsequently described (profile) hidden Markov models are used.

2.7 Hidden Markov models

Originally developed and applied to problems in speech recognition in the late 1960's and early 1970's, hidden Markov models became very popular in bioinformatics in the late 1980's and early 1990's. Since then they have found many applications, e.g. gene prediction [BK97, Bur98], recognition of transmembrane domains in proteins [KLvHS01] or protein family classification. They are successfully used as sequence family models [BCD$^+$04, HSW03] to reflect how the sequences of the family relate by substitutions, insertions and deletions to the consensus sequence of the family. Since HMMs are general probabilistic models with a wide range of possible application and not limited to problems in bioinformatics, we start with a brief introduction of the underlying general theory, before focusing on a special type of HMMs often used in sequence analysis.

2.7.1 Foundations of hidden Markov model theory

A hidden Markov model (HMM for short) λ over an alphabet $\mathcal{A}$ describes a probability distribution over the set of finite words $w \in \mathcal{A}^*$. Let $\mathbb{P}[w|\lambda]$ be the probability of w under the model λ. We call $\mathbb{P}[w|\lambda]$ the production probability of λ for the sequence w. An HMM can be used to characterize a family of sequences by assigning a production probability to a sequence w screened versus the model λ, giving a measure of how likely it its, that w belongs to the family described by λ. If the production probability $\mathbb{P}[w|\lambda]$ is significant, w matches the model and can be seen as a new member of the sequence family described by this HMM.

Similar to a Markov Model, an HMM consists of a set of states $\{S_1, S_2, \ldots, S_N\}$ and transitions connecting states. Each state has a local probability distribution, the state transition probabilities, describing the probability of a certain state transition. Let s_t denote the state of an HMM λ at point t. State transition probabilities for N states can be defined by a $N \times N$ matrix A with

$$a_{i,j} = \mathbb{P}[s_t = S_j | s_{t-1} = S_i], i, j \in [1, N] \tag{2.36}$$

expressing the probability for a transition from state S_i at point $t-1$ to state S_j at point t. For the initialization of the stochastic process, we define starting probabilities π_i for each state S_i. The resulting vector $\vec{\pi}$ is defined as

$$\pi_i = \mathbb{P}[s_1 = S_i], i \in [1, N]. \tag{2.37}$$

The transition structure of a discrete HMM can be described as a directed graph with a node for each state, and an edge between two nodes if the corresponding state transition probability is non zero (see Figure 2.10). In contrast to a Markov model, in an HMM state transitions are not directly observable, they are hidden. Observable is a sequence of characters generated by a sequence of state transitions of the HMM. It is convenient to think of an HMM as a generative model that generates a sequence of characters from an output alphabet $\mathcal{A}_\epsilon := \mathcal{A} \cup \{\epsilon\}$, resulting in a sequence of observations $w = w_1, w_2, \ldots, w_T$, $w_i \in \mathcal{A}_\epsilon$ with probability $\mathbb{P}[w|\lambda]$. The process of state transitions evolves in some dimension, often time, though not necessarily. The model is parametrized with state transition probabilities governing the state at a time $t+1$, given that one knows the previous states at time t. Markov assumptions are used to truncate the dependency of having to know the entire history of states up to point t in order to assess the next state $t+1$ such that only one step back is

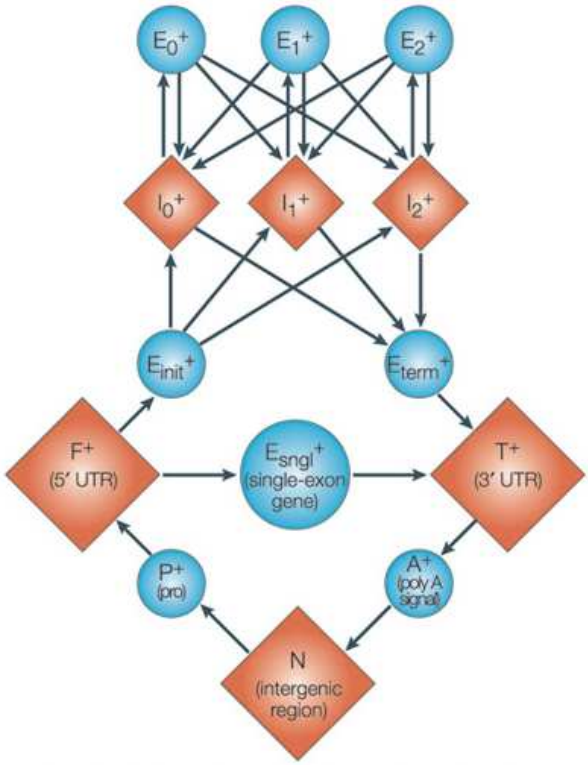


Figure 2.10: Different states and transitions in the Genscan hidden Markov model. Genscan is a gene-prediction algorithm that, like other HMMs, models the transition probabilities from one part (state) of a gene to another. Here each circle or square represents a functional unit (a state) of a gene on its forward strand (for example E_{init} is the 5' coding sequence (CDS) and E_{term} is the 3' CDS, and the arrows represent the transition probability from one state to another). Figure adopted from Genscan manual.

required. A pass through the HMM continues from state to state according to the state transition probabilities[6]. For each transition an HMM generates a character from the output alphabet $\mathcal{A}_\epsilon$ with a certain state dependent probability, the symbol emission probability $\mathbb{P}[o_t = w_k | s_t = S_j]$, $w_k \in \mathcal{A}_\epsilon$, resulting in a stream of emitted symbols (observations), as the process passes through the states. If we have a finite alphabet of output symbols $\mathcal{A}_\epsilon$ and thus discrete symbol emission probabilities, they can be described by an $N \times |\mathcal{A}_\epsilon|$ matrix B with

$$b_{j,k} = \mathbb{P}[o_t = w_k | s_t = S_j], w_k \in \mathcal{A}_\epsilon, 1 \leq j \leq N, 1 \leq k \leq |\mathcal{A}_\epsilon|$$

$\mathbb{P}[o_t = w_k | s_t = S_j]$ denotes the probability of generating symbol $w_k \in \mathcal{A}_\epsilon$ in state S_j at time t. A state without a symbol emission probability distribution is called a silent state. Observe that this is no restriction to the general concept of HMMs, in which a state has no special type and each state is a symbol emitting state, since a silent state S_j can be seen as a symbol emitting state, emitting the empty string ϵ with symbol emissions probability $\mathbb{P}[o_t = \epsilon | s_t = S_j] = 1$ and $\mathbb{P}[o_t = c | s_t = S_j] = 0$

[6]Sometimes special start- and end states are used to define a start and end point for a pass through the model.

for all other characters $c \in \mathcal{A}$. An HMM λ with discrete probability distributions is well defined by the triple

$$\lambda = (\vec{\pi}, A, B). \tag{2.38}$$

When using HMMs there are the following three basic problems of interest:

1. **The Evaluation Problem:** Sometimes also called the likelihood problem. Given a HMM λ and a sequence of observations $w = w_1, w_2, \dots w_T$, $w_i \in \mathcal{A}_\epsilon$, what is the production probability $\mathbb{P}[w|\lambda]$ that w is generated by λ?

2. **The Decoding Problem:** Given model λ and a sequence of observations $w = w_1, w_2, \dots, w_T$, $w_i \in \mathcal{A}_\epsilon$, what is the most likely state sequence $q = q_1, q_2, \dots, q_T$ with $q_i \in \{S_1, S_2, \dots, S_N\}$ across the model that generates the observed sequence w?

3. **The Learning Problem:** Given model λ and a sequence of observations $w = w_1, w_2, \dots, w_T$, $w_i \in \mathcal{A}_\epsilon$, how should the model parameters $(\vec{\pi}, A, B)$ be adjusted in order to maximize $\mathbb{P}[w|\lambda]$?

To compute the production probability $\mathbb{P}[w|\lambda]$, we have to take all state sequences/paths $q = q_1, q_2, \dots, q_T$ through λ into account that produce the sequence of observations $w = w_1, w_2, \dots w_T$, $w_i \in \mathcal{A}_\epsilon$, and compute and add their probabilities. We denote the set of paths through λ producing w by Q_w and write $\mathbb{P}[w|\lambda]$ as

$$\mathbb{P}[w|\lambda] = \sum_{q \in Q_w} \mathbb{P}[w, q|\lambda] = \sum_{q \in Q_w} \pi_{q_1} b_{q_1,w_1} \cdot a_{q_1,q_2} \cdot b_{q_2,w_2} \cdot a_{q_2,q_3} \cdot \ldots \cdot a_{q_{T-1},q_T} \cdot b_{q_T,w_T}. \tag{2.39}$$

Obviously, the number of paths increases exponentially with the length of the sequence of observations and a straightforward calculation of $\mathbb{P}[w|\lambda]$ leads to an algorithm, solving the evaluation problem in $\mathcal{O}\left(2T \cdot N^T\right)$ time, where $2T$ is the cost of computing the probability for a single path and N^T is the number of paths of length T. It is apparent, that this approach is infeasible in practice, even for moderate values of T. A more efficient approach makes use of dynamic programming and calculates $\mathbb{P}[w|\lambda]$ in polynomial time. In particular this algorithm is known as the *Forward Algorithm*. We make this now more precise. We define the problem of computing the probability $\mathbb{P}[w|\lambda]$ in terms of prefixes of the observed sequence. Let $\alpha_{t,i} = \mathbb{P}[w_1, \dots, w_t, s_t = S_i|\lambda]$ denote the probability of observing the partial observation sequence $w_1, \dots, w_t$ and being in state S_i at point t. Then the following recurrences hold:

$$\begin{aligned}
\alpha_{1,j} &= \pi_j b_{j,1} \quad \texttt{for any } j \in [1, N] \\
\alpha_{t+1,j} &= \left(\sum_{i=1}^{N} \alpha_{t,i} a_{i,j}\right) b_{j,t+1} \quad \texttt{for any } t \in [1, T-1] \wedge j \in [1, N] \\
\mathbb{P}[w_1, \dots, w_T|\lambda] &= \sum_{i=1}^{N} \alpha_{T,i}
\end{aligned} \tag{2.40}$$

To determine $\mathbb{P}[w|\lambda]$ with the *Forward Algorithm* we calculate the values of $\mathcal{O}(N \cdot T)$ cells of the dynamic programming matrix, spending $\mathcal{O}(N)$ operations per cell. Hence the overall time complexity is $\mathcal{O}\left(N^2 T\right)$ and the space complexity is $\mathcal{O}(N \cdot T)$. In a similar way we can define a backward recursion calculating $\mathbb{P}[w|\lambda]$. This leads to the *Backward Algorithm*. Here we define the problem of

computing $\mathbb{P}[w|\lambda]$ in terms of suffixes of the observed sequence. Let $\beta_{t,i} = \mathbb{P}[w_{t+1}, \ldots, w_T, s_t = S_i|\lambda]$ be the probability to observe the sequence $w_{t+1}, \ldots, w_T$ and being in state S_i at point t. Then the following recurrences can be used to compute $\mathbb{P}[w|\lambda]$ efficiently:

$$\begin{aligned} \beta_{T,j} &= 1 \quad \texttt{for any } j \in [1, N] \\ \beta_{t,j} &= \sum_{i=1}^{N} \beta_{t+1,i} a_{j,i} b_{i,t+1} \quad \texttt{for any } t \in [T-1, 1] \wedge j \in [1, N] \\ \mathbb{P}[w_1, \ldots, w_T|\lambda] &= \sum_{i=1}^{N} \pi_i b_{i,1} \beta_{1,i} \end{aligned} \tag{2.41}$$

The complexity of the *Backward Algorithm* is again $\mathcal{O}\left(N^2 T\right)$ time and $\mathcal{O}\left(N \cdot T\right)$ space. Further on, we observe that with the definitions of α and β the following equation holds for arbitrary t.

$$\mathbb{P}[w_1, \ldots, w_T|\lambda] = \sum_{i=1}^{N} \alpha_{t,i} \beta_{t,i} \tag{2.42}$$

With the *Forward* or *Backward* algorithm, we can compute the probability $\mathbb{P}[w|\lambda]$ that a sequence of observations was produced by a given model λ and thus solving the evaluation problem. $\mathbb{P}[w|\lambda]$ can be rewritten in terms of a score or p-value and can be used in HMM based protein family classification to accept or not to accept the sequence of observations as a new member of the family that was used to build the model.

The HMM decoding problem can be solved with an algorithm known as the *Viterbi algorithm*, which again applies dynamic programming. It is similar to the *Forward algorithm* except that we do not sum over the predecessor states at point t, but taking the maximum.

Let $\epsilon_{t,j}$ denote the highest probability that the partial observation sequence $w_1, \ldots, w_t$ and state sequence ending in state $j \in [1, N]$ up to point t. Then the *Viterbi algorithm* can be defined by the following recurrences:

$$\begin{aligned} \epsilon_{1,j} &= \pi_j b_{j,1} \\ \epsilon_{t+1,j} &= \underset{i \in [1,N]}{\operatorname{argmax}} \{\epsilon_{t,i} a_{i,j}\} b_{j,t+1} \end{aligned} \tag{2.43}$$

If we store a pointer, pointing from $\epsilon_{t+1,j}$ back to the selected predecessor state $\epsilon_{t,i}$ which is the state for which $\epsilon_{t,i} \cdot a_{i,j},\ i \in [1, N]$, is maximal, we can calculate the most likely state sequence recursively, starting with ϵ_{T,j^*} and $j^* = \underset{1 \leq i \leq N}{\operatorname{argmax}} \{\epsilon_{T,i}\}$ and thus solving the decoding problem. Since we calculate the values of $\mathcal{O}\left(N \cdot T\right)$ cells of the DP matrix, spending $\mathcal{O}\left(N\right)$ operations per cell, the overall time complexity of the *Viterbi Algorithm* is $\mathcal{O}\left(N^2 T\right)$ and the space complexity is $\mathcal{O}\left(N \cdot T\right)$. Here N denotes the number of states and T is the length of the emitted sequence.

To solve the learning problem, we must find for a given HMM λ with already defined topology and observed data the model parameters $(\vec{\pi}, A, B)$ that maximize $\mathbb{P}[w|\lambda]$. There is no known optimal analytical way of doing this. However, there exist algorithms that iteratively re-estimate the model

from some arbitrary starting point which guarantee to find a local maximum. The most common one is the *Baum-Welch* or forward-backward algorithm [Rab90], which is a version of the general expectation maximization (EM) method often used in statistics. For an accurate estimation of the model parameters a lot of training data is needed, making the training of the model a critical and computationally expensive step. To give an example, consider an HMM over an alphabet containing 20 symbols, representing the 20 different naturally occurring amino acids. All emission probabilities of all 20 amino acids have to be estimated in all emitting states. Especially in profile HMMs (a type of an HMM with a special topology, described in the next section), in which each conserved position in the sequence is modeled by a different emitting state, the number of estimated parameters can be enormous. This phenomenon is related to overfitting, which occurs when there is not enough data to obtain good estimates for the model parameters, and consequently the model will not generalize adequately to new data.

2.7.2 Profile hidden Markov models

In particular successful in sequence analysis since the 1990's is a special type of HMMs called profile hidden Markov models (pHMMs for short). pHMMs were first introduced in [HKB$^+$93] and [KMSH94] and are simple types of hidden Markov models with a linear, left-to-right, repetitive structure of states (see Figure 2.11 for an example), well suited to model multiple alignments and probably the most popular application of hidden Markov models in computational biology. They have been proved to be a powerful method in biological sequence analysis, especially successful in performing sequence database searching and detecting remote homologies [Edd98, KBH98, MG02]. pHMMs are also common in speech recognition, where they are sometimes called *time-dependent* HMMs or *time-parametrized* HMMs.

The prefix "profile" is used because pHMMs are similar to and address the same problem as the formerly described PSSMs (often also called profiles). Likewise to PSSMs they are often derived from multiple alignments of related sequences and capture position-specific information about how conserved each column of the alignment is, and which residues are likely. pHMMs are general, statistical models for any system that can be represented as a succession of transitions between discrete states. As a model capturing the information of a protein family, the discrete states correspond to the successive columns of a protein multiple sequence alignment. Although, in principle, pHMMs can even be determined from unaligned sequences by successive rounds of optimization, in practice, protein pHMMs are built from curated multiple sequence alignments, like the ones collected in the PFAM [BBD$^+$00] or TIGRFAM [HSW03] databases. For the construction of a pHMM usable as a discriminative motif descriptor, we assume a given multiple alignment of a sequence family and a derived consensus sequence. In contrast to general HMMs, pHMMs have basically three types of states with associated special semantics: *match* (M), *insert* (I) and *delete* (D) states. *Match* and *insert* states are symbol emitting states whereas the *delete* state is a silent state. Each of these states models a position of the consensus sequence of the sequence family delineated by the multiple alignment, and describes how members of the family deviate from the consensus sequence at that position. More precisely:

- The *match* state models that the generated character has evolved from the position in the consensus sequence.

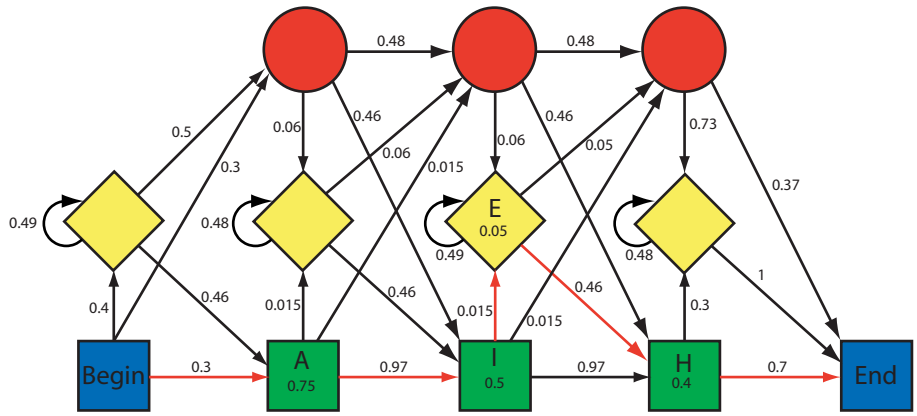


Figure 2.11: The transition structure of a pHMM consisting of repeated elements of *match* (green squares), *insert* (yellow diamonds) and silent *delete* (red circles) states. A pass through the pHMM starts in a special start-state and continues from state to state according to the state transition probabilities, until a special end-state is reached. State transition probabilities are given as numbers next to the directed arcs. Symbol emissions and their probabilities are given as letters and numbers inside the state symbols.

- The *insert* state models that the generated character has been inserted between two neighboring positions in the consensus sequence.
- The self-loop on the *insert* state models that several consecutive characters can be inserted between two positions of the consensus sequence.
- The *delete* state models that the position has been deleted from the consensus sequence.

A path through the model always starts from the *begin/start* state and ends with the *end* state. Likewise to general HMMs, on the path through the model, state transitions occur with a certain probability and in symbol emitting states, a symbol from the output alphabet is emitted with a certain probability. To give an example, let q be the state sequence of the red marked path in Figure 2.11. Then q generates the sequence `AIEH` with probability

$$\mathbb{P}[w = \texttt{AIEH}, q|\lambda] = 0.3 \cdot 0.75 \cdot 0.97 \cdot 0.5 \cdot 0.015 \cdot 0.05 \cdot 0.046 \cdot 0.4 \cdot 0.7 = 1.0541E^{-6}. \tag{2.44}$$

That is, we compute $\mathbb{P}[w = \texttt{AIEH}, q|\lambda]$ as the product of the state transition probabilities and the emission probabilities of the emitted symbol along the path through the model. Instead of multiplying probabilities, in practice, often the log-odd scores are summed up.

The central part of a pHMM is a sequence of match states, corresponding to columns in the multiple alignment. Each match state emits (aligns to) a single residue, with a probability score that is determined by the frequency that residues have been observed in the corresponding column of the multiple alignment. Each match state therefore has an assigned vector of $|\mathcal{A}|$ probabilities, describing a probability distribution of the symbols of $\mathcal{A}$. That is, in case of pHMMs, build from a multiple alignment of amino acid sequences 20 probabilities for scoring the 20 amino acids. Observe

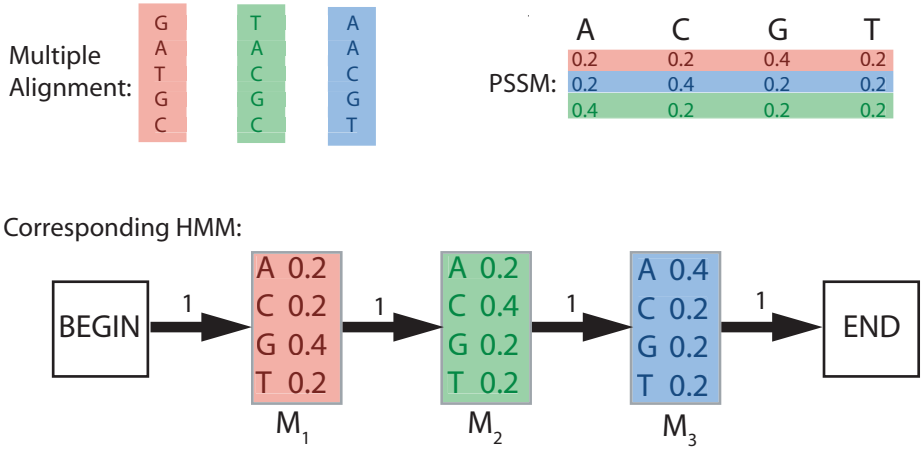


Figure 2.12: A PSSM based on probability values can be seen as a pHMM consisting of a linear sequence of match states with state transition probabilities of 1 between them. In this view, each match state corresponds to a column in the multiple alignment and hence a row in the PSSM. It emits a symbol from the output alphabet with a certain probability. That is, the symbol emission probability distribution of a match state corresponds to the distribution of scores or rather probabilities in a row of the PSSM.

that the meaning of this probability vector is similar to the meaning of a row vector in a PSSM. More over, if the PSSM contains probability values, then they are equivalent. Hence a PSSM is essentially equivalent to a pHMM composed only of match states (see Figure 2.12) and can be seen as a method that looks for ungapped alignments to a consensus of a multiple alignment[7]. If we extend this perception to PSSMs that include position specific gap costs, like the Gribskov PSSMs described in section 2.6, the position specific gap costs correspond to transition probabilities for moving to an *insert* or *delete* state.

The main difference between Gribskov's PSSMs with position specific gap costs and a pHMM is that the PSSM model requires the transition from a *match* state to an *insert* state and the transition from a *match* state to a *delete* state to have both the same probability. This is dispositional in the sense, that an insertion in one sequence can be seen as a deletion in another. In contrast to the basic PSSM model as defined in Definition 8, a pHMM is capable of modeling alignments including insertions and deletions (with the insert and delete states mentioned above), which allows the more adequate description of much longer and more variable parts of conserved sequences like complete conserved domains or complete sequences, rather than just a relatively small ungapped motif.

pHMM construction from a multiple alignment

For the construction process of a pHMM from a given multiple alignment two important decisions must be taken into account:

[7] In [Edd98] the author makes the distinction between pHMMs, which he calls profile models and motif HMMs which are built of linear sets of *match* states and are essentially PSSMs.

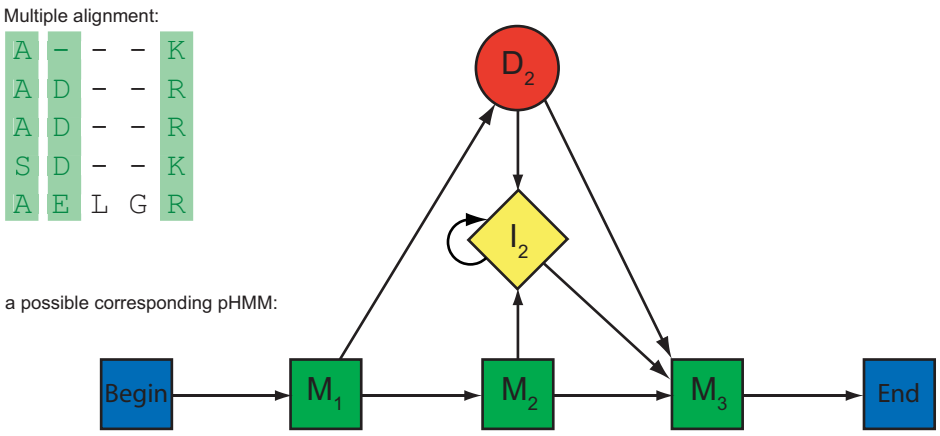


Figure 2.13: A possible pHMM for the given multiple alignment. The three *match* states M_1, M_2, M_3 correspond to the green marked columns in the alignment. The third and fourth column are treated as insertions between M_2 and M_3 and are modeled with the *insert* state I_2. The *delete* state D_2 allows to skip state M_2.

- **The topology and model length**. That is, we have to decide, which columns of the multiple alignment must be assigned to match states and which must be modeled with insertion states. A rule of thumb used in practice is to consider columns with more than half gap characters as highly variable regions that should be modeled with insertion states. See Figure 2.13 for an example.

- **The model parameters**. Reconsider that an HMM λ with a discrete probability distribution is well defined by the triple $(\vec{\pi}, A, B)$ (see Equation (2.38)). Initial probabilities $\vec{\pi}$, transition probabilities A, and emission probabilities B can be estimated from the multiple alignment. For this estimation, again pseudo-count methods are used to avoid problems caused by zero character frequencies and to adequately estimate character distributions.

In the following we describe the structure of a pHMM as introduced in [HKB+93] and [KMSH94]. Assume that the linear sequence of *match* states is defined e.g. by selecting those columns of the multiple alignment that contain less than half gap characters. The next step is to deal with insertions and deletions. Since insertions, i.e. portions of a sequence that do not match anything in the model, can potentially occur at any position, we add an *insert* state to each *match* state. Deletions, i.e. segments of the multiple alignment and *match* states that are not matched by a sequence scored with the model, are handled with *delete* states. We associate a *delete* state with each *match* state. This allows to skip *match* states. Additionally we add an *insert* state before the first *match* state to allow to skip prefixes of the sequence before entering the first *match* state. We call a group of *match*, *insert*, and *delete* states at the same consensus position in the alignment a *node*, and the model length is the number of nodes between the *begin* and the *end* state. Finally we end for the multiple alignment given in Figure 2.13 with the pHMM architecture given in Figure 2.11. Following the propposed architecture of [KMSH94] leads to generalized models, where in principle an insertion or deletion

can occur at any position in a sequence evaluated or generated with the pHMM. A slightly different pHMM topology often used in practice to model families of related sequences, is the so called *PLAN 7* architecture, developed by Eddy [Edd98] and implemented in the *HMMER* software package[8]. It is somewhat more complex, but much more flexible than the original pHMM architecture introduced by Krogh and coworkers in [KMSH94]. Unlike Krogh's pHMM architecture, PLAN 7 has no state transitions from *delete* to *insert* states and from *insert* to *delete* states. Additional special states in the PLAN 7 architecture even allow the construction of local alignments. Alignments can be local with respect to sequence (i.e. allowing a match to the model anywhere within a longer sequence), as well as with respect to the model (i.e. allowing fragments of the model to match the sequence). Multiple hit alignments, for instance to model repetitive protein domains, are also possible. For a detailed description of the PLAN7 architecture, see the *HMMER* manual.

Once the structure of the pHMM is determined, the model parameters, like transition and symbol emission probabilities have to be estimated from the multiple alignment. This is analogous to solving the formerly described learning problem. For the estimation of symbol emission probabilities from sample counts in the multiple alignment, the same or similar methods as the described methods for PSSM score estimation are used.

Observe that a pHMM defines a discrete probability distribution over the whole space of sequences or words from $\mathcal{A}^*$ respectively. Accordingly, the objective of the construction and training process is to control the shape of that distribution by associating the peaks of the function around members of the sequence family represented by the multiple alignment. That means, that the model, which describes the consensus sequence for the family, not the sequence of any particular member, should discriminate between true and false family members as well as possible. As already stated, there is no analytical, optimal way of doing this, but in practice, iterative methods like the *Baum-Welch* or forward-backward algorithm [Rab90] can be used for this task. For a detailed description of probability estimation methods and optimal model construction in the context of pHMMs see sections 5.6 and 5.7 of [DEK98].

Sequence alignment and database searching with pHMMs

The most important application of a pHMM representing a family of sequences is finding new sequences in a database that show a high similarity to the members of this family. Given a database of sequences and a pHMM, the sequences can be aligned to the model. Here the pHMM can be seen as a generative model and a sequence is viewed as a sequence of observations (emitted symbols). As shown in Figure 2.14, in a pHMM one sequence of observations can be generated by different hidden state sequences. If we want to align a sequence $w \in \mathcal{A}^*$ to an already trained model λ, we are interested in the most probable sequence of hidden state transitions that generates this sequence of observations. Thus, when aligning w to λ, a sequence of *match*, *insert*, and *delete* states will be obtained. The determination of the best (most likely) sequence of state transitions is essentially equal to solving the HMM decoding problem which can be solved with the *Viterbi* algorithm in $\mathcal{O}\left(N^2 \cdot T\right)$ time and $\mathcal{O}\left(N \cdot T\right)$ space, applying dynamic programming. Here N denotes the length of w and T is the number of states in λ. Finally, an ordered score can be determined from the comparison of the probability of the most likely state sequence to the probability of random sequences.

[8] `http://hmmer.wustl.edu/`

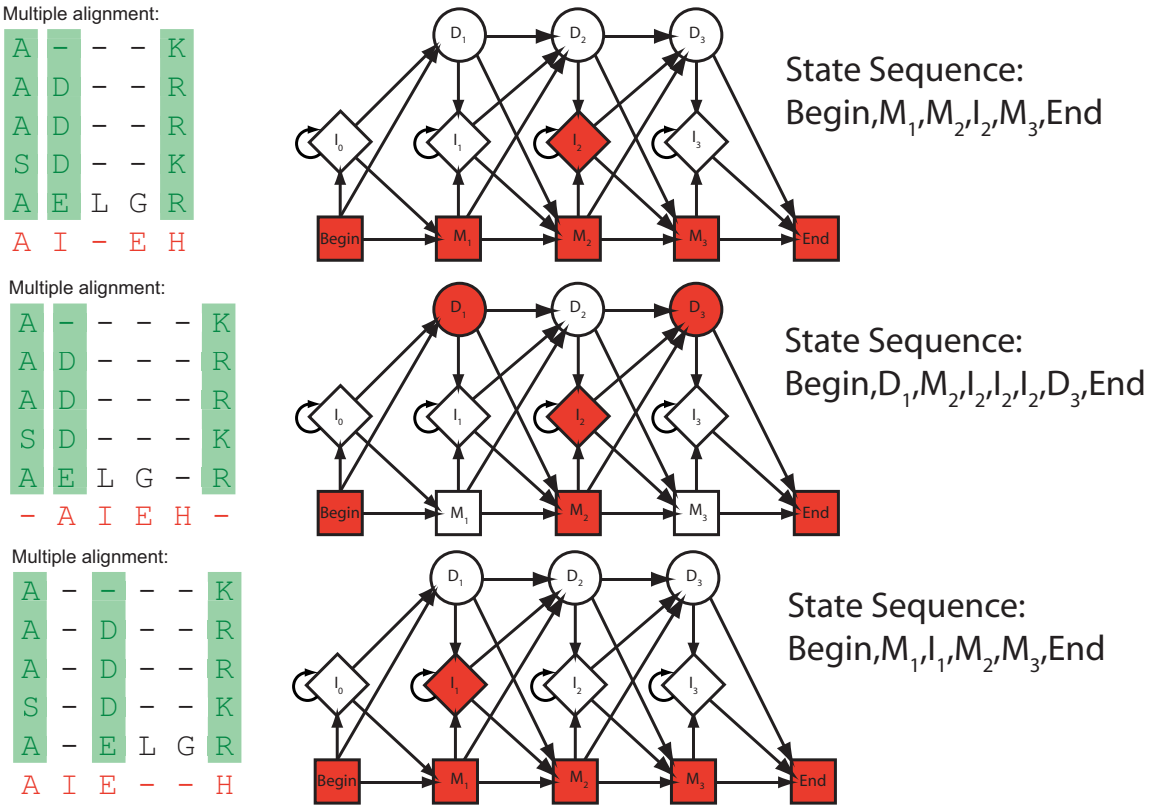


Figure 2.14: A pHMM can be seen as a process, that generates a sequence of characters with a certain probability, by emitting symbols in the symbol emitting *match* and *insert* states. Different paths through the model can generate the same sequence with different probabilities. In this example three possible paths and their state sequences to generate the sequence `AIEH` are shown. The three match states M_1,M_2, and M_3 correspond to the green marked columns in the multiple alignment. In each example, the visited states on the path are marked red.

Name (Release)	#Models	URL
Pfam (20.0)	8,296	`http://www.sanger.ac.uk/Software/Pfam/`
TIGRFAM (6.0)	2,946	`http://www.tigr.org/TIGRFAMs/`
SMART (5.0)	725	`http://smart.embl-heidelberg.de/`
SUPERFAMILY (1.69)	4,894	`http://supfam.org/SUPERFAMILY/`
CATH (3.0)	23,876	`http://www.cathdb.info/`
PANTHER (6.0)	36,298	`http://www.pantherdb.org/`

Table 2.5: Major existing pHMM collections

In a database search scenario with a pHMM λ, we can ask alternatively to the computation of the most probable state path, how likely is it that a certain sequence $w \in \mathcal{A}^*$ is generated by λ. That is, we have to compute the production probability $\mathbb{P}[w|\lambda]$. Since there can be more than one path through λ that generates w (see Figure 2.14), we have to take all paths into account that generate w (see Equation (2.39)). Observe that this is equivalent to solving the HMM evaluation problem and can be accomplished with the *Forward* algorithm in $\mathcal{O}\left(N^2 \cdot T\right)$ time and $\mathcal{O}\left(N \cdot T\right)$ space.

2.7.3 Profile HMM collections for sequence annotation and classification

Profile HMMs are especially successful for modeling of protein families and there is an increasing number of publicly available collections of such family models, see Table 2.5. A common aspect of all of these collections is, that they use the pHMM topology and model notations of the *HMMER* package [Edd98], such that they are compatible to and can be searched with the pHMM search software of the same name. Overall, it can be said that the HMMER software package has established a *de facto* standard in this field. In the following we give a brief overview of the most widely used collections of pHMMs, suitable for sequence annotation and protein family classification.

Pfam database

Pfam [BCD+04] is a large manually curated collection of multiple sequence alignments and derived pHMMs covering many common protein domains and families. Genome projects, including both the human and fly, have used Pfam extensively for large scale functional annotation of genomic data.

Each curated family in Pfam is represented by a seed and full alignment. The seed contains representative members of the family, while the full alignment contains all members of the family as detected with a pHMM constructed from the seed alignment. Such full alignments can be large, with the top 20 families containing over 2500 sequences each. The majority of known protein sequences come from just a few thousand protein families.

TIGRFAM database

The Institute for Genomic Research protein families database (TIGRFAM) [HSW03] is likewise to Pfam a collection of curated multiple sequence alignments (seed alignments) for protein families and

pHMMs built from the seeds. TIGRFAM contains predominantly equivalogs (functionally defined subfamilies). Protein family descriptions for use in protein annotation, including trusted score cutoff and noise cutoff values accompany each model. Proteins that score above the trusted cutoffs are believed to reside within the family and those falling below the noise cutoffs are believed to reside outside the family. The margin of error with respect to presence or absence of a protein within a TIGRFAM family is represented by the score range between noise and trusted cutoffs. Additionally the TIGRFAM database provides functional classification information in form of roles, in which models are classified, and cross referencing to the Gene Ontology classification system [Con06].

SMART database

The Simple Modular Architecture Research Tool (SMART) [LCP+06] can be used for the identification and annotation of genetically mobile domains and the analysis of domain architectures. It contains pHMMs for more than 500 domain families found in signaling, extracellular and chromatin-associated proteins. The domain models are extensively annotated with respect to phyletic distributions, functional class, tertiary structures and functionally important residues.

SUPERFAMILY database

The SUPERFAMILY database [GKHC01] provides structural (and hence implied functional) assignments to amino acid sequences at the protein superfamily level. It is a library of pHMMs representing 1539 protein superfamilies. Superfamilies are defined according to the structural classification of proteins database (SCOP) [AHB+04]. Each superfamily is represented by a group of pHMMs.

CATH Protein structure classification database

CATH [PTS+05] is a multi level hierarchical classification system, that classifies protein domain structures at four major levels, Class(C), Architecture(A), Topology(T), and Homologous superfamily (H). The level of homologous superfamilies groups together protein domains which are thought to share a common ancestor and can therefore be described as homologous. Similarities are identified either by high sequence identity or structure comparison. From multiple alignments of the homologous superfamilies pHMMs are constructed, such that each superfamily is represented by multiple pHMMs.

PANTHER classification system

The PRotein ANalysis THrough Evolutionary Relationships (PANTHER) classification system [MLUL+05] classifies proteins according to families and subfamilies. PANTHER defines families as groups of evolutionary related proteins and subfamilies as related proteins that also have the same function. Information about family and subfamily affiliations are derived from clustering of the UniProt protein database, using a BLAST-based similarity score. Each protein family is represented by a phylogenetic tree defining its subfamilies. Families and subfamilies are also represented by pHMMs and associated with functional ontology terms. For several families and subfamilies

additional information and associated data such as detailed biochemical interactions in canonical pathways are available.

2.8 Concluding remarks on sequence motif models

The different concepts for motif modeling presented in this chapter all have different advantages and disadvantages, making it difficult to choose one single method as best for all kinds of possible applications. See Table 2.6 for an overview of the advantages and disadvantages of the described motif models.

Advantages	Disadvantages
Regular expressions	
• easy to use due to the availability of efficient search engines • fast to match	• discrete motif descriptor • no scoring system, only binary response
PSSMs	
• provide a scoring system • needs less sequences for model construction than a pHMM • efficient to match using index structures	• matching is computationally more expensive than for regular expressions • limited expressiveness due to missing insertion and deletion model
Gribskov profiles	
• provide a scoring system • position specific insertion/deletion model	• not very common in practice • lack of publicly available models • tools for profile construction and searching were not further developed in the last decade and are poorly conceived
profile HMMs	
• full probabilistic model • provide a scoring system based on probabilistic theory • insertion and deletion model • very sensitive • widely used	• matching is very time consuming and hence it is difficult to use pHMMs on a large-scale • proper model training may become time consuming and challenging • a lot of sequences are needed to train a model adequately and to avoid overfitting

Table 2.6: Advantages and disadvantages of different motif modeling concepts.

Discrete sequence motif descriptors, such as consensus strings or regular expression based patterns are relatively easy to search, and standardized searching engines are available. Searching with these motif descriptors is relatively fast, since with a single mismatch parts of the text to be searched can

be skipped. Their severe drawback is, that they do not include a scoring system and give only a binary response. As a consequence they are often an inadequate concept to describe biological motifs. In contrast to regular expressions, PSSMs provide a kind of similarity score and increased sensitivity although searching with PSSMs is more complex and computationally expensive. Compared to pHMMs much less (aligned) sequences are needed to derive a meaningful PSSM which is definitely an advantage in practice, when modeling protein families with only a few number of known members. Another advantage of PSSMs is that they are a well studied, accepted and common motif model in sequence analysis and thus various publicly available resources of curated alignment blocks and already derived PSSMs exist. One severe drawback of PSSMs is, that they are fixed length motifs, which lack of an adequate insertion and deletion model. Hence their capabilities especially for modeling of longer regions is limited. This disadvantage is partially balanced in Gribskov's profile model, which extends the basic PSSM model by position specific gap costs. However, the pitfall with Gribskov profiles in practice is simply the nonexistence of publicly available models. Additionally the programs to build profiles from existing multiple alignments and to search with them are not very handy to use and were not further developed in the last decade.

The most successful motif model in computational biology are pHMMs. They are based on a fully probabilistic model and are capable to model insertions and deletions. Further on, they yield to be the most sensitive of the introduced motif models so far. The price to be paid when using pHMMs is an increased complexity and higher computational effort when building the model. To build a pHMM that achieves good classification accuracy, a lot of model parameters have to be trained properly. Consequently much more sequences are needed for an adequate training of a pHMM, compared to a PSSM, in order to avoid overfitting problems. Another problem, especially occurring when using pHMMs on a larger scale, are the running times of the *Viterbi* and *Forward* algorithms. These may make searching with pHMMs a time consuming process in practice. Especially in the absence of large cluster systems, searching with large collections of pHMMs on complete proteomes can become an infeasible task.

In the following chapters, we will focus on the efficient searching of PSSM based motif models. We will see, that some of the disadvantages of PSSMs compared to pHMMs, like the lack to model insertions and deletions, can be compensated, making PSSMs almost as sensitive as pHMMs and that the use of suffix based full text index structures lead to fast PSSM searching algorithms, that are well suited for large-scale PSSM matching tasks.

3 Fast algorithms for matching position specific scoring matrices

3.1 Introduction

As stated in the former chapter, PSSMs are a well known and successfully used concept for approximate motif modeling in sequence analysis. When searching with PSSMs in nucleotide or amino acid sequences, a high PSSM-score in some region of a sequence often indicates a possible biological relationship of this sequence to the family or motif characterized by the PSSM. There are several databases utilizing PSSMs for function assignment and annotation, e.g., PROSITE [HSL+04], PRINTS [ABF+03], BLOCKS [HGPH00], EMATRIX [WNB99], JASPAR [SAE+04], or TRANSFAC [MFG+03]. In addition, recently developed modeling concepts for more complex complete regulatory modules consisting of several transcription factor binding site, like the multiple-feature based approach of [PSTB05], also use PSSMs as atomic motif descriptors. While there are manifold applications that employs PSSMs and PSSM containing databases are constantly improved, there are only few improvements in the programs searching with PSSMs. E.g., the programs *FingerPrintScan* [SFA99], *BLIMPS* [HGPH00], *MatInspector* [QFWW95], and the method of [PSTB05] still use a straightforward $\mathcal{O}(mn)$-time algorithm to search a PSSM of length m in a sequence of length n. In [RJS02] the authors presented a method based on Fourier transformation. A different method introduced in [FB05] employs data compression. To the best of our knowledge there is no software available implementing these two methods. The most advanced program in the field of searching with PSSMs is *EMATRIX* [WNB00], which incorporates a technique called lookahead scoring. The lookahead scoring technique is also employed in the suffix tree based method of [DNM00]. This method performs a limited depth first traversal of the suffix tree of the set of target sequences. This search updates PSSM-scores along the edges of the suffix tree. Lookahead scoring allows to skip subtrees of the suffix tree that do not contain any matches to the PSSM. Unfortunately, the method of [DNM00] has not found its way into a widely available and robust software system. A method for the detection of transcription factor binding sites modelled with PSSMs utilizing suffix trees but no lookahead scoring was very recently described in [SSZ07]. In [Gon04], the development of new, more efficient algorithms for searching with PSSMs is considered an important problem, which still needs better solutions.

In this chapter, we briefly recall existing methods for searching with PSSMs and present a new, non-heuristic algorithm. With any non-heuristic PSSM searching algorithm, the performance in terms of sensitivity and specificity solely depends on the used PSSM and threshold, i.e. given a PSSM and threshold, all these algorithms give exactly the same results. For the generation of PSSMs from aligned sequences, numerous different methods were described in literature over the last decades

[GME87, TAK94, HH96, WNB99, KGR+03]. Some of them were already described in detail in section 2.5. The algorithms presented in this chapter can deal with all these types of PSSMs, since rather than improving PSSMs, we focus on improvements in terms of time and space efficiency when searching with PSSMs, independently of their underlying generation method. The overall structure of our proposed new search algorithm is similar to the method of [DNM00]. However, instead of suffix trees we use enhanced suffix arrays, a data structure which is as powerful as suffix trees (cf. [AKO04]) but provides several advantages over suffix trees, which make them more suitable for searching with PSSMs.

One of our algorithmic contributions is a new technique that allows to skip parts of the enhanced suffix array containing no matches to the PSSM. Due to the skipping, our algorithm achieves an expected running time that is sublinear in the size of the search space (i.e., the size of the nucleotide or protein database). As a consequence, our algorithm scales very well for large data sizes.

Since the running time of our algorithm increases with the size of the underlying alphabet, we developed a filtering technique, utilizing alphabet reduction, that achieves better performance especially on sequences/PSSMs over the amino acid alphabet.

When searching with a PSSM, it is important to determine a suitable threshold for a PSSM-match. Usually, the user prefers to specify a significance threshold (i.e., an E-value or a p-value) which has to be transformed into an absolute score threshold for the PSSM under consideration. This can be done by computing the score distribution of the PSSM, using well-known dynamic programming (DP, for short) methods, e.g., [Sta89, WNB00, Rah03, RMV03]. Unfortunately, these methods are not fast enough for large PSSMs. For this reason, we have developed a new, lazy evaluation algorithm that only computes a small fraction of the complete score distribution. Our algorithm speeds up the computation of the threshold by factor of at least 3, compared to standard DP methods. This makes our algorithm applicable for on-the-fly computations of the score thresholds.

3.2 Pattern matching with PSSMs

Recall, that a PSSM is an abstraction of a multiple alignment of related sequences and can be defined as a function $M : [0, m-1] \times \mathcal{A} \rightarrow \mathbb{R}$, where m is the length of M and $\mathcal{A}$ is a finite alphabet. We represent M by an $m \times |\mathcal{A}|$ matrix, in which each row reflects the frequency of occurrence of each amino acid or nucleotide at the corresponding position of the underlying alignment. See Figure 3.1 for an example.

From now on, let M be a PSSM of length m and let $w[i]$ denote the character of w at position i for $0 \leq i < m$. Further on, $w[i..j]$ denotes the substring of w starting at position i and ending at position j. We define $sc(w, M) := \sum_{i=0}^{m-1} M(i, w[i])$ for a sequence $w \in \mathcal{A}^m$ of length m. $sc(w, M)$ is the *match score of* w w.r.t. M. The *score range* of a PSSM is the interval $[sc_{\min}(M), sc_{\max}(M)]$ with $sc_{\min}(M) := \sum_{i=0}^{m-1} \min\{M(i, a) \mid a \in \mathcal{A}\}$ and $sc_{\max}(M) := \sum_{i=0}^{m-1} \max\{M(i, a) \mid a \in \mathcal{A}\}$. We define the *PSSM matching problem* as follows:

Definition 9 Given a sequence S of length n over alphabet $\mathcal{A}$, a PSSM M of length m and a score threshold th, the *PSSM matching problem* is to find all positions $j \in [0, n-m]$ in S and their assigned match scores, such that $sc(S[j..j+m-1], M) \geq th$ holds.

A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y	th_d	σ_d
-19	**92**	-45	-49	-30	-36	-38	-12	-41	-21	-22	-40	-46	-44	-44	-30	-25	16	-35	-34	**2**	**398**
5	-17	17	**22**	-28	-15	-7	-23	-8	-27	-21	26	18	-7	-13	-9	9	-19	-33	-25	**24**	**376**
7	-8	-29	-28	2	-25	-10	25	-23	-4	-5	-25	-32	-26	-25	-18	13	22	-11	**36**	**60**	**340**
-29	**99**	-55	-61	-42	-45	-47	-31	-52	-34	-36	-49	-56	-55	-55	-38	-35	-29	-44	-46	**159**	**241**
-14	-22	14	**22**	-28	9	-8	-26	15	-27	-20	-7	-26	-3	31	-13	5	-23	-30	-24	**181**	**219**
-25	-34	-25	-16	-37	-30	-15	-36	45	-34	-26	-18	-35	-9	**49**	-25	-26	-33	-39	-31	**230**	**170**
7	-8	-25	-24	-19	-23	-22	4	-15	-10	-8	-19	-29	-21	11	-13	**31**	31	-31	-22	**261**	**139**
-34	-27	-44	-43	60	-41	-8	-16	-38	-14	-17	-39	-51	-40	-36	-39	-35	-21	-1	**56**	**317**	**83**
7	**40**	-16	-14	-9	-14	-6	-17	14	-20	-15	-10	-24	-11	12	15	9	-13	-16	20	**357**	**43**
-7	**43**	16	-7	-27	-15	-9	-24	-5	-26	-18	-6	-25	25	13	25	-8	-21	-30	-24	**400**	**0**

Figure 3.1: Amino acid PSSM of length $m = 10$ of a zinc-finger motif. If the score threshold is $th = 400$, then only substrings beginning with C or V can match the PSSM, because all other amino acids score below the intermediate threshold $th_0 = 2$. That is, lookahead scoring will skip over all substrings starting with amino acids different from cysteine (C) and valine (V).

A simple algorithm for the PSSM matching problem slides along the sequence and computes $sc(w, M)$ for each $w = S[j..j + m - 1]$, $j \in [0, n - m]$. See Algorithm 1 and Figure 3.2 for an example. The running time of this algorithm is $\mathcal{O}(mn)$. It is used e.g., in the programs *FingerPrintScan* [SFA99], *BLIMPS* [HGPH00], *MatInspector* [QFWW95], and *MATCH* [KGR+03].

Algorithm 1: *SPsearch*

input : A sequence $S = s_0 \dots s_{n-1}$, a PSSM M of length m and a threshold th

output: All matching positions of M in S and their associated *matchscores*

```
1 for j ← 0 to n − m do
2     score ⟵ sc(S[j..j + m − 1], M) ;
3     if score ≥ th then
4         print "match at position j with score: score";
5     end
6 end
```

3.3 Improved running time through the usage of lookahead scoring

In [WNB00], lookahead scoring is introduced to improve the simple algorithm. Lookahead scoring allows to stop the calculation of $sc(w, M)$ when it is clear that the given overall score threshold th cannot be achieved. To be more precise, we define $pfxsc_d(w, M) := \sum_{h=0}^{d} M(h, w[h])$, $\max_d := \max\{M(d, a) \mid a \in \mathcal{A}\}$, and $\sigma_d := \sum_{h=d+1}^{m-1} \max_h$ for any $d \in [0, m - 1]$. $pfxsc_d(w, M)$ is the *prefix score of depth d*. σ_d is the maximal score that can be achieved in the last $m - d - 1$ positions of the PSSM. Let $th_d := th - \sigma_d$ be the *intermediate threshold* at position d. The correctness of lookahead scoring, not shown in [WNB00], is implied by the following Lemma:

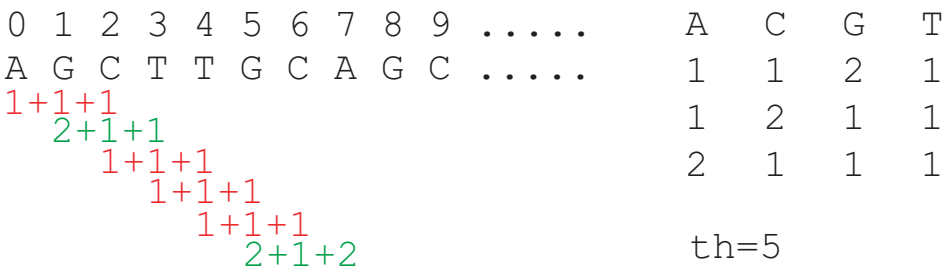


Figure 3.2: A straightforward solution for the PSSM searching problem is the *SPsearch* algorithm, which uses a sliding window technique. All subwords of the sequence of the PSSM length are scored completely according to the corresponding matrix values. If the score is equal to or exceeds the given threshold th, a match is reported at the starting position of the currently scored subword. In this example the given threshold is $th = 5$. Matching subwords are marked green, mismatching subwords are marked red.

Lemma 1 The following statements are equivalent:

(1) $pfxsc_d(w, M) \geq th_d$ for all $d \in [0, m-1]$,

(2) $sc\,(w, M) \geq th$.

Proof: (1)⇒(2): Suppose that (1) holds. Then $\sigma_{m-1} = \sum_{h=m}^{m-1} \max_h = 0$ and

$$sc\,(w, M) = \sum_{h=0}^{m-1} M(h, w[h]) = pfxsc_{m-1}(w, M) \geq th_{m-1} = th - \sigma_{m-1} = th.$$

(2)⇒(1): Suppose that (2) holds. Let $d \in [0, m-1]$. Then

$$\begin{aligned} sc\,(w, M) &= \sum_{h=0}^{m-1} M(h, w[h]) = \sum_{h=0}^{d} M(h, w[h]) + \sum_{h=d+1}^{m-1} M(h, w[h]) \\ &= pfxsc_d(w, M) + \sum_{h=d+1}^{m-1} M(h, w[h]) \end{aligned}$$

Hence $sc\,(w, M) \geq th$ implies $pfxsc_d(w, M) + \sum_{h=d+1}^{m-1} M(h, w[h]) \geq th$. Since $M(h, w[h]) \leq \max_h$ for $h \in [0, m-1]$, we conclude

$$\sum_{h=d+1}^{m-1} M(h, w[h]) \leq \sum_{h=d+1}^{m-1} \max_h = \sigma_d$$

and hence

$$pfxsc_d(w, M) \geq th - \sum_{h=d+1}^{m-1} M(h, w[h]) \geq th - \sigma_d = th_d.$$

□

The Lemma suggests a necessary condition for a PSSM-match which can easily be exploited: When computing $sc(w, M)$ by scanning w from left to right, one checks for $d = 0, 1, \ldots, m-1$, if the intermediate threshold th_d is achieved. If not, the computation can be stopped. See Figure 3.1 for an example of intermediate thresholds and their implications. A pseudocode formulation of the lookahead scoring algorithm (herein after called *LAsearch*) is given in Algorithm 2.

Algorithm 2: *LAsearch*

input : A sequence $S = s_0 \ldots s_{n-1}$, a PSSM M of length m, a threshold th
output: All matching positions of M in S and their associated *matchscores*

```
1  for d ← 0 to m − 1 do
2  |  th_d ⟵ th − Σ_{h=d+1}^{m−1} max{M(h, a) | a ∈ A} ;
   |  /* calculate the intermediate thresholds th_d = th − σ_d                    */
3  end
4  for j ← 0 to n − m do
5  |  score ⟵ 0 ;
6  |  for d ← 0 to m − 1 do
7  |  |  score ⟵ score + M(d, S[j + d]);
   |  |  /* score = pfxsc_{d−1}(S[j..j + d − 1], M) + M(d, S[j + d])            */
8  |  |  ;
9  |  |  if score < th_d then
10 |  |  |  break; //terminate when we miss an intermediate threshold
11 |  |  end
12 |  end
13 |  if score ≥ th then
14 |  |  print "match at position j with score: score";
15 |  end
16 end
```

If we assume that the row maxima of M can be determined in $O(1)$ time, such that the *a priori* calculation of the vector of intermediate thresholds can be accomplished in $\mathcal{O}(m)$ time instead of $\mathcal{O}(m|\mathcal{A}|)$, *LAsearch* runs in $\mathcal{O}(kn + m)$ time, where k is the average number of PSSM-positions per sequence position actually evaluated. In the worst case, $k \in \mathcal{O}(m)$, which leads to the worst case running time of $\mathcal{O}(mn)$, not better than the simple algorithm. However, k is expected to be much smaller than m, leading to considerable speedups in practice. In the best case, exact one character of each subword of length m of S has to be scored leading to $\mathcal{O}(m + n)$ running time.

3.3.1 Permuted lookahead scoring

The authors of [WNB00] also suggest a variant of (sequential) lookahead scoring, called permuted lookahead scoring, which indeed does not affect the worst case running time but can lead to an additional speedup in practice. The basic idea is to evaluate the PSSM in a permuted order with the aim to increase the likelihood of falling short of an intermediate threshold early. Lookahead scoring accesses the values of the PSSM sequentially from position 1 up to m. See interior loop

in Algorithm 2. However we can score the characters of a given subword w of length m in any order. Wu and coworkers suppose to reorder the rows of the PSSM according to the difference $D_i = |E_i - M_i|$ between the expected score $E_i = \sum_{a \in \mathcal{A}} M(i,a)f(a), \quad i \in [0, m-1]$ and the maximum score $M_i = \max\{M(i,a) \mid a \in \mathcal{A}\}$ of a row i, starting with the largest difference. Here $f(a)$ denotes the background probability of symbol a. Hence we can determine *a priori* a permutation $\pi = (\pi_0, \ldots, \pi_{m-1})$ of the rows of M such that $D_{\pi_i} \geq D_{\pi_j}$ holds, for any pair $i, j \in [0, m-1]$, $i < j$ and where π_i indicates the position to be evaluated in step i. That is, the intermediate thresholds are computed according to the order given by π as

$$th_d = th - \sum_{h=d+1}^{m-1} \max\{M(\pi_h, a) \mid a \in \mathcal{A}\} \text{ for any } d \in [0, m-1]. \tag{3.1}$$

Analogous to the calculation of *pfxsc* for sequential lookahead scoring, we compute a partial score, scoring d characters of w in the order given by π. Let

$$prtsc_d(s_j \ldots s_{j+m-1}, M) := \sum_{i=0}^{d} M(\pi_d, S[j + \pi_d]). \tag{3.2}$$

The substitution of the computation of function *pfxsc* by *prtsc* and the corresponding changes in the calculation of the intermediate thresholds lead to the permuted lookahead scoring variant for searching with PSSMs, shown in Algorithm 3. For the performance improvement of permuted lookahead scoring over sequential lookahead scoring achievable in practice, speedups between 5.8 and 20.6 %, depending on the stringency of th are reported in [WNB00]. Although this improvement is significant, we will see that the use of suffix based index structures in combination with sequential lookahead scoring lead to much higher performance improvements.

3.4 PSSM searching using suffix trees

Although the *LAsearch* and *permuted LAsearch* lead to a considerable speedup in practice, the benefit in times of exponentially increasing sequence databases is limited. The severe drawback of these techniques is, that the improvement does not affect the (exponentially increasing) search space and hence the running time is still linear in the size of the search space (i.e. length of the sequences to be searched). In analogy to traditional string matching, the improvements introduced by lookahead scoring can be compared to pattern preprocessing methods like the Knuth-Morris-Pratt [KMP77] or Boyer-Moore [BM77] algorithm that slide along the text to be searched and which running time is dominated by the text length. What we are really interested in, is an algorithm that runs independent of the sequence length n. This can be achieved with an indexing of the search space.

In the *SPsearch* as well as *LAsearch* algorithm, we observe that common prefixes of subwords are re-scored again and again when sliding along the sequence. We can avoid this by indexing all subwords based on their prefixes. A powerful index data structure known since the early seventies [Wei73] that became quite popular in the last years, with a wide range of possible applications [Apo85] in computational biology (cf. [Gus97]) is the suffix tree, which is well suited for our problem. A suffix tree is a Trie-like or PATRICIA-like [Mor68] data structure that exposes the internal structure of

Algorithm 3: *permuted LAsearch*

input : A sequence $S = s_0 \dots s_{n-1}$, a PSSM M of length m, a threshold th

output: All matching positions of M in S and their associated *matchscores*

```
1  compute permutation π = π_0, ... π_{m-1} of M such that D_{π_i} ≥ D_{π_j} for any i < j, i, j ∈ [0, m − 1];
2  for d ← 0 to m − 1 do
3      th_d ⟵ th − Σ_{h=d+1}^{m−1} max{M(π_h, a) | a ∈ A} ;
       /* calculate the intermediate thresholds in the permuted order */
4  end
5  for j ← 0 to n − m do
6      score ⟵ 0 ;
7      for d ← 0 to m − 1 do
8          score ⟵ score + M(π_d, S[j + π_d]);
           /* score = prtsc_{d−1}(S[j..j + m − 1], M) + M(π_d, S[j + π_d]) */
9          ;
10         if score < th_d then
11             break; //terminate when we miss an intermediate threshold
12         end
13     end
14     if score ≥ th then
15         print "match at position j with score: score";
16     end
17 end
```

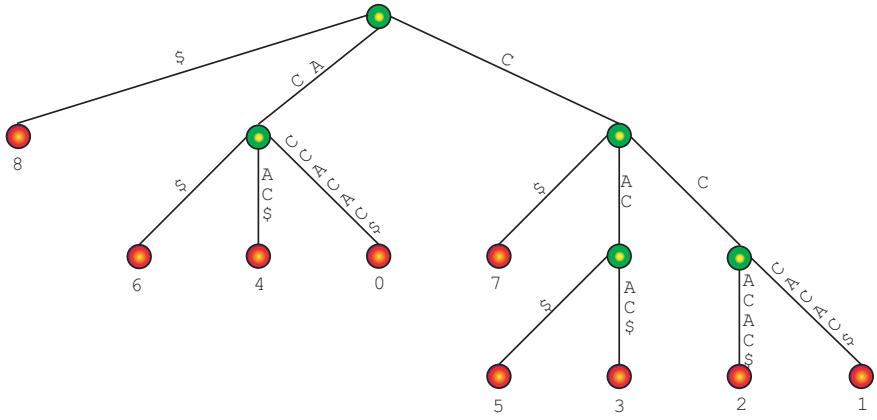


Figure 3.3: The suffix tree for the string $S = ACCCACAC\$$. Internal nodes are marked green, leaves are marked red. Observe that the concatenation of the edge labels on a path starting at the root node and ending at leave with leave number i results in the suffix of S starting at position i.

the underlying string in a deep way by containing all subwords of the string and allowing a very efficient access. More precisely:

Definition 10 Suffix tree
The suffix tree T for a string $S\$$ of length n is a rooted directed tree with exactly n leaves numbered 0 to $n-1$. Each internal node, excluding the root node, has at least two child nodes and each edge is labeled with a nonempty substring of S. No two edges out of a node can have edge-labels beginning with the same character. The key feature of the suffix tree is that for any leaf $i \in [0, n-1]$, the outcome of the concatenation of the edge-labels on the path from the root to leaf i exactly spells out the suffix of S that starts at position i.

The suffix tree can be constructed in linear time and space with several algorithms [Wei73, McC76, Ukk95]. Once constructed, it can be used to efficiently solve a wide range of string processing problems, e.g the exact matching of a pattern of length m in $O(m)$ time. Figure 3.3 gives an example of a suffix tree.

3.4.1 Dorohonceanu's algorithm

In [DNM00] the authors describe the usage of suffix trees to speed up the searching with PSSMs. Their non persistent implementation of suffix trees needs 17 byte space per input character on average. The basic idea of their method is to perform a limited depth first traversal of the suffix tree of the set of target sequences. In the traversal of the tree they update PSSM-scores along the edges and make use of the fact, that for a PSSM of length m all subwords of length m which have to be investigated are represented in the suffix tree up to depth m. If and only if the overall threshold th is reached or exceeded at depth m, the matching positions of the PSSM can be retrieved by

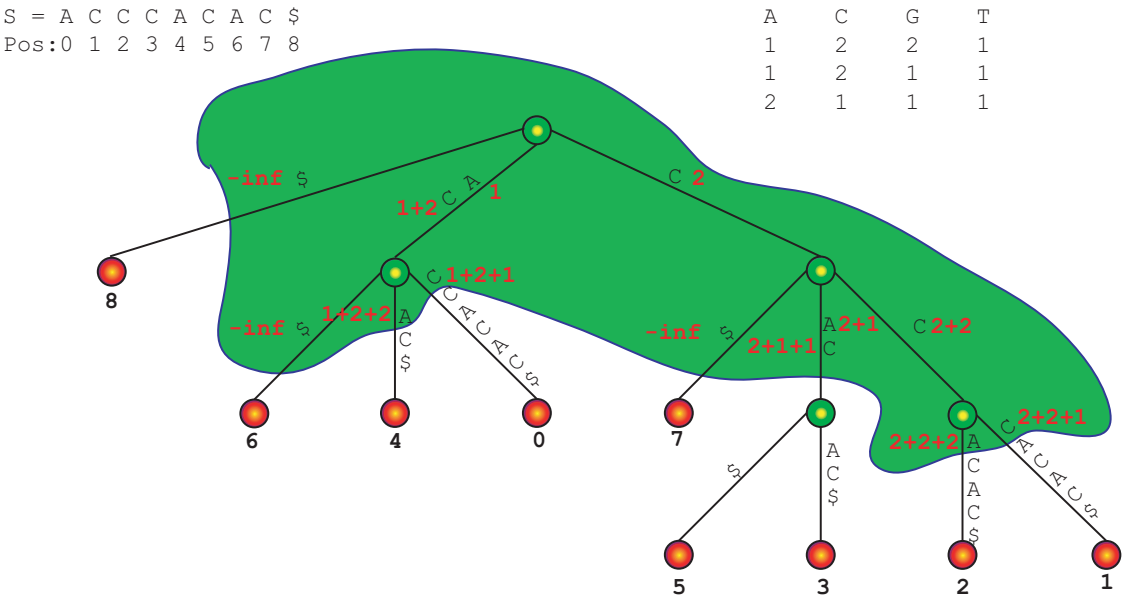


Figure 3.4: Using a suffix tree for searching with PSSMs. To score all subwords of the PSSM length m ($m = 3$ in this example), we have to perform a depth first traversal up to depth 3 (green marked part of the tree). This is a direct adaptation of the *SPsearch* algorithm described in Algorithm 1.

enumerating the leaf numbers in the subtree below. As we have already seen in the description of the lookahead scoring method, it is not necessary to score all subwords of length m completely, if an overall threshold th is given. Again we can use intermediate thresholds as early stop criterias for the subword scoring. For PSSM searching using suffix trees this means, that we essentially do not have to traverse the tree up to depth m completely (see Figure 3.4), when incorporating lookahead scoring. Lookahead scoring allows to skip subtrees of the suffix tree that do not contain any matches to the PSSM, by checking the intermediate thresholds while the traversal (see Figure 3.5 for an example). Suffix trees are also employed for searching with PSSMs in the very recently published *STORM* program [SSZ07]. *STORM* uses McCreight's algorihtm [McC76] for the construction of a non persistent suffix tree.

Analysis

The complexity analysis for Dorohonceanu's algorithm, not given in [DNM00], follows the same argumentations and leads to the same results as the analysis of the *ESAsearch* algorithm presented below. To avoid redundancies, we analyze the complexity of Dorohonceanu's algorithm together with the complexity of *ESAsearch* in section 3.5.1 on page 66.

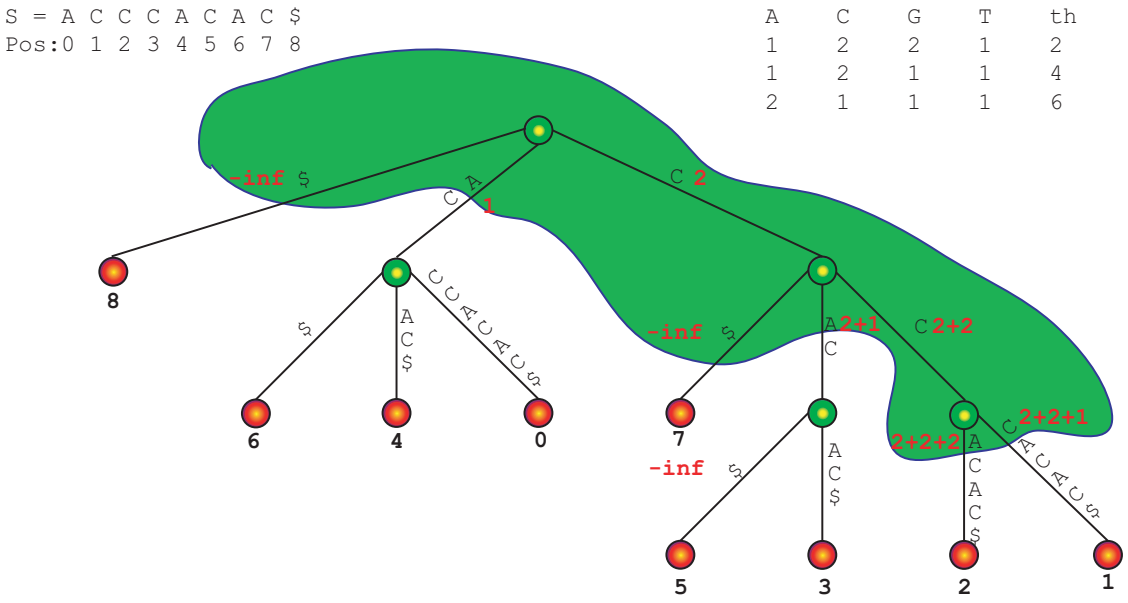


Figure 3.5: By incorporating lookahead scoring we can limit the depth first traversal of the suffix tree. Observe that, in contrast to the traversal shown in Figure 3.4, now subtrees can be skipped, if an intermediate threshold is missed. In this example we used an overall threshold of $th = 6$.

3.5 PSSM searching using enhanced suffix arrays: The ESAsearch algorithm

As demonstrated in [DNM00], a suffix tree is a powerful data structure, even for PSSM searching and its usage can lead to remarkable speedups, especially when the sequence space to be searched is large. Unfortunately, the method of [DNM00] has not found its way into a widely available and robust software system. Further on, an enhanced suffix array, a data structure as powerful as a suffix tree, provides several additional advantages over suffix trees, making it more suitable for searching with PSSMs:

- While suffix trees require about $12n$ bytes in the best available implementation (cf. [Kur99]), the enhanced suffix array used for searching with PSSMs only needs $9n$ bytes of space.

- While the suffix tree is usually only computed in main memory, the enhanced suffix array is computed once and stored on file. Whenever a PSSM is to be searched, the enhanced suffix array is mapped into main memory which requires no extra time.

- While the depth first traversal of the suffix tree suffers from the poor locality behavior of the data structure (cf. [GK95]), the enhanced suffix array provides optimal locality, because when searching with PSSMs it is sequentially scanned from left to right.

i	suf[i]	lcp[i]	skp[i]	$S_{suf[i]}$
0	1		12	aaaaccacac$
1	2	3	2	aaaccacac$
2	3	2	3	aaccacac$
3	7	1	6	acac$
4	4	2	6	accacac$
5	9	2	6	ac$
6	0	0	12	caaaaccacac$
7	6	2	9	cacac$
8	8	3	9	cac$
9	5	1	11	ccacac$
10	10	1	11	c$
11	11	0	12	$

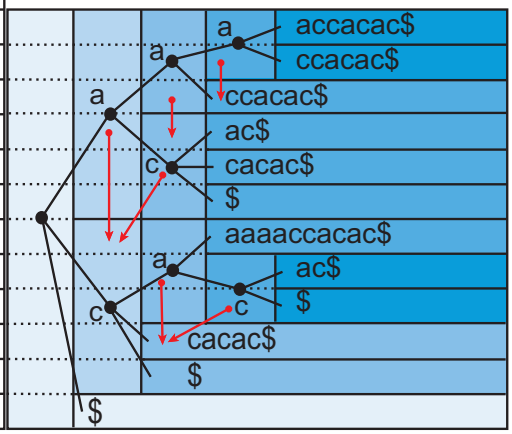


Figure 3.6: The enhanced suffix array consisting of tables suf, lcp, skp (left) and the suffix tree (right) for sequence $S =$ caaaaccacac. Some skp entries are shown in the tree as red arrows: If skp$[i] = j$, then an arrow points from row i to row j. For clarity, suffixes corresponding to suf$[i]$ are given in table $S_{\mathsf{suf}[i]}$.

The generic name *enhanced suffix array*, introduced in [AKO02] stands for a family of data structures, extending a suffix array with additional information. Suffix arrays are a well known data structure in literature. They were introduced in 1993 by Manber and Myers [MM93] and independently by Gonnet et al. under the name PAT array [GBYS92]. The enhanced suffix array for a given sequence S of length n consists of three tables suf, lcp, and skp. Let \$ be a symbol in $\mathcal{A}$, larger than all other symbols, which does not occur in S. suf is a table of integers in the range 0 to n, specifying the lexicographic ordering of the $n+1$ suffixes of the string $S\$$. That is, $S_{\mathsf{suf}[0]}, S_{\mathsf{suf}[1]}, \ldots, S_{\mathsf{suf}[n]}$ is the sequence of suffixes of $S\$$ in ascending lexicographic order, where $S_i = S[i..n-1]\$$ denotes the i-th nonempty suffix of the string $S\$$, for $i \in [0, n]$. See Figure 3.6 for an example. Given a suffix tree, suf can be constructed in $\mathcal{O}(n)$ time by a depth-first traversal of the tree. Recently published algorithms (cf. [KS03, KSPP03, KA03]) even allow a direct construction of suf in $\mathcal{O}(n)$ time, without first constructing a suffix tree. Table suf requires $4n$ bytes.

lcp is a table in the range 0 to n such that lcp$[0] := 0$ and lcp$[i]$ is the length of the longest common prefix of $S_{\mathsf{suf}[i-1]}$ and $S_{\mathsf{suf}[i]}$, for $i \in [1, n]$. See Figure 3.6 for an example. Table lcp can be computed in linear time given table suf [KLA+01]. In practice PSSMs are used to model relatively short, local motifs and hence do not exceed length 255. For searching with PSSMs we therefore do not access values in table lcp larger than 255, and hence we can store lcp in n bytes.

skp is a table in the range 0 to n such that $\mathsf{skp}[i] := \min(\{n+1\} \cup \{j \in [i+1, n] \mid \mathsf{lcp}[i] > \mathsf{lcp}[j]\})$. In terms of suffix trees, skp$[i]$ denotes the lexicographically next leaf that does not occur in the subtree below the branching node corresponding to the longest common prefix of $S_{\mathsf{suf}[i-1]}$ and $S_{\mathsf{suf}[i]}$. Figure 3.6 shows this relation. Table skp can be computed in $\mathcal{O}(n)$ time given suf and lcp. For the algorithm to be described we assume that the enhanced suffix array for S has been precomputed.

In a suffix tree, all substrings of S of a fixed length m can be scored with a PSSM by a depth first traversal of the tree. Using lookahead scoring, one can skip certain subtrees that do not contain matches to the PSSM. Since suffix trees have several disadvantages (see the introduction), we use enhanced suffix arrays to search PSSMs. Like in other algorithms on enhanced suffix arrays (cf. [AKO04]), one simulates a depth first traversal of the suffix tree by processing the arrays suf and lcp from left to right. To incorporate lookahead scoring while searching we must be able to skip certain ranges of suffixes in suf. To facilitate this, we use table skp. We will now make this more precise.

For $i \in [0, n]$, let $v_i = S_{\mathsf{suf}[i]}$, $l_i = \min\{m, |v_i|\} - 1$, and $d_i = \max(\{-1\} \cup \{d \in [0, l_i] \mid pfxsc_d(v_i, M) \geq th_d\})$. That is, d_i is the last position in the suffix v_i to be scored when scoring v_i from left to right, since at position $d_i + 1$ we fall short intermediate threshold th_{d_i+1}. Now observe that $d_i = m-1 \Leftrightarrow pfxsc_{m-1}(v_i, M) \geq th_{m-1} \Leftrightarrow sc\,(v_i, M) \geq th$. Hence, M matches at position $j = \mathsf{suf}[i]$ if and only if $d_i = m - 1$. Thus, to solve the PSSM searching problem, it suffices to compute all $i \in [0, n]$ satisfying $d_i = m - 1$. We compute d_i along with $C_i[d] = pfxsc_d(v_i, M)$ for any $d \in [0, d_i]$. d_0 and C_0 are easily determined in $\mathcal{O}(m)$ time. Now let $i \in [1, n]$ and suppose that d_{i-1} and $C_{i-1}[d]$ are determined for $d \in [0, d_{i-1}]$. Since v_{i-1} and v_i have a common prefix of length $\mathsf{lcp}[i]$, we have $C_i[d] = C_{i-1}[d]$ for all $d \in [0, \mathsf{lcp}[i] - 1]$. Consider the following cases:

- If $d_{i-1} + 1 \geq \mathsf{lcp}[i]$, then compute $C_i[d]$ for $d \geq \mathsf{lcp}[i]$ while $d \leq l_i$ and $C_i[d] \geq th_d$. We obtain $d_i = d$.

- If $d_{i-1} + 1 < \mathsf{lcp}[i]$, then let j be the minimum value in the range $[i + 1, n + 1]$ such that all suffixes $v_i, v_{i+1}, \ldots, v_{j-1}$ have a common prefix of length $d_{i-1}+1$ with v_{i-1}. Due to the common prefix we have $pfxsc_d(v_{i-1}, M) = pfxsc_d(v_r, M)$ for all $d \in [0, d_{i-1}+1]$ and $r \in [i, j-1]$. Hence $d_{i-1} = d_r$ for $r \in [i, j - 1]$. If $d_{i-1} = m - 1$, then there are PSSM matches at all positions $\mathsf{suf}[r]$ for $r \in [i, j - 1]$. If $d_{i-1} < m - 1$, then there are no PSSM matches at any of these positions. That is, we can directly proceed with index j. We obtain j by following a chain of entries in table skp: compute a sequence of values $j_0 = i$, $j_1 = \mathsf{skp}[j_0], \ldots, j_k = \mathsf{skp}[j_{k-1}]$ such that $d_{i-1} + 1 < \mathsf{lcp}[j_1], \ldots, d_{i-1} + 1 < \mathsf{lcp}[j_{k-1}]$, and $d_{i-1} + 1 \geq \mathsf{lcp}[j_k]$. Then $j = j_k$.

These case distinctions lead to the program *ESAsearch* (see Algorithm 4 and Function skipchain).

We illustrate the ideas of algorithm *ESAsearch*, formally described above, with the following example. Let M be a PSSM of length $m = 2$ over alphabet $\mathcal{A} = \{a, c\}$ with $M(0, a) = 1$, $M(0, c) = 3$, $M(1, a) = 3$, and $M(1, c) = 2$. For a given threshold of $th = 6$ we obtain intermediate thresholds $th_0 = 3$ and $th_1 = 6$. To search with M in the enhanced suffix array for sequence $S = \texttt{caaaaccacac}$ as given in Figure 3.6, we start processing the enhanced suffix array suf top down by scoring the first suffix $S_{\mathsf{suf}[0]} = \texttt{aaaaccacac\$}$ with M from left to right. For the first character of $S_{\mathsf{suf}[0]}$ we obtain a score of $pfxsc_0(S_{\mathsf{suf}[0]}, M) = M(0, a) = 1$ which is below the first intermediate threshold $th_0 = 3$. Hence we set $d_0 = -1$ and notice that we can skip all suffixes of S that start with character '$\texttt{a}$'. Further on, with a lookup in $\mathsf{lcp}[1] = 3$ we find that $S_{\mathsf{suf}[1]}$ and $S_{\mathsf{suf}[0]}$ share a common prefix of length 3 and $d_0 + 1 = -1 + 1 < \mathsf{lcp}[1] = 3$ (second case described above). The next suffix that may match M with $th = 6$ is $S_{\mathsf{suf}[6]} = \texttt{caaaaccacac\$}$. Suffixes $S_{\mathsf{suf}[1]}, S_{\mathsf{suf}[2]}, \ldots S_{\mathsf{suf}[5]}$ can be skipped since they all share a common prefix with $S_{\mathsf{suf}[0]}$ of at least length 1. That is, they begin all with character '$\texttt{a}$' and would also miss the first intermediate threshold $th_0 = 3$ when scored. We find $S_{\mathsf{suf}[6]}$ by

Algorithm 4: *ESAsearch*

input : An enhanced suffix array for the sequence $S\$$ consisting of the tables suf, lcp and skp, a PSSM M of length m, and a threshold th.

output: All matching positions of M in S and their associated *matchscores*

```
1  for d ← 0 to m − 1 do
2      th_d ⟵ th − Σ_{h=d+1}^{m−1} max{M(h, a) | a ∈ A} ;
       /* calculate the intermediate thresholds th_d = th − σ_d                    */
3  end
4  depth ← 0;
5  i ← 0;
6  while i < n do
7      if n − m < suf[i] then
8          while (n − m < suf[i]) ∧ (i < n) do
9              i ← i + 1;
10             depth ← min{depth, lcp[i]};
11         end
12         if i ≥ n then return ;
13     end
14     if depth = 0 then score ← 0 else score ← C[depth − 1];
15     d ← depth − 1;
16     do
17         d ← d + 1;
18         score ← score + M(d, S_{suf[i]+d});
19         C[d] ← score;
20     while (d < m − 1) ∧ (score ≥ th_d);
21     if (d = m − 1) ∧ (score ≥ th) then
22         print"match at position suf[i] with score: score";
23         while i < n do
24             i ← i + 1;
25             if lcp[i] ≥ m then print"match at position suf[i] with score: score" else break;
26         end
27     else
28         i ← skipchain(lcp, skp, n, i, d);
29     end
30     depth ← lcp[i];
31 end
```

Function `skipchain`(lcp, skp, n, s, d)

input : Tables lcp and skp of an enhanced suffix array, $|S|$ denoted with n,an index i of the i-th smallest suffix, and depth d from where to start skipping.

output: An index j of the j-th smallest suffix with $j > i$.

```
1  begin
2      if i < n then
3          j ← i + 1 ;
4          while (j ≤ n) ∧ (lcp[j] > d) do
5              j ← skp[j] + 1 ;
6          end
7      else
8          j ← n ;
9      end
10     return j ;
11 end
```

following a chain of entries in table skp: skp[1] = 2, skp[2] = 3, and skp[3] = 6. When scoring $S_{\mathsf{suf}[6]}$ we compute $pfxsc_0(S_{\mathsf{suf}[6]}, M) = M(0, c) = 3$ and $pfxsc_1(S_{\mathsf{suf}[6]}, M) = M(0, c) + M(1, a) = 6$ and store them for reuse in $C[0]$ and $C[1]$. Since $d_6 = 1 = m - 1 = 1$ holds, we report suf[6] = 0 with score $sc\left(S_{\mathsf{suf}[6]}, M\right) = pfxsc_1(S_{\mathsf{suf}[6]}, M) = 6$ as a matching position. With lookups in lcp[7] = 2 and lcp[8] = 3 we notice that $S_{\mathsf{suf}[7]}$ and $S_{\mathsf{suf}[8]}$ share a common prefix of at least two characters with $S_{\mathsf{suf}[6]}$. Hence we report suf[7] = 6 and suf[8] = 8 with score $C[1] = 6$ as further matching positions. We proceed with the scoring of $S_{\mathsf{suf}[9]}$. Since lcp[9] = 1 holds, we obtain the score for the first character 'c' from array C with $pfxsc_0(S_{\mathsf{suf}[9]}, M) = C[0]$. After scoring the second character of $S_{\mathsf{suf}[9]}$, $pfxsc_1(S_{\mathsf{suf}[9]}, M) = 5 < th_1 = 6$ holds and we miss the second intermediate threshold and continue with the next suffix. The last two suffixes $S_{\mathsf{suf}[10]}$ and $S_{\mathsf{suf}[11]}$ in suf do not have to be considered since their lengths are smaller than $m = 2$ (not counting the sentinel character \$) and therefore they cannot match M. We end up with matching positions 0, 6, and 8 of M in S with match score 6. To find these matches, we processed the enhanced suffix array suf top down and scored suffixes from left to right, facilitating the additional information given in tables lcp and skp to avoid re-scoring of characters of common prefixes of suffixes and to skip suffixes that cannot match M for the given threshold.

3.5.1 Analysis

The C_i arrays can be stored in a single $\mathcal{O}(m)$ space array C as any step i only needs the C_i specific to that step. C_i solely depends on C_{i-1}, and $C_i[0..d-1] = C_{i-1}[0..d-1]$ holds for a certain $d < m$, i.e., the first d entries in C_i are known from the previous step, and thus C can be organized as a stack. No other space (apart from the space for the enhanced suffix array) depending on input size is required for *ESAsearch*, leading to an $\mathcal{O}(m)$ space complexity.

The worst case for *ESAsearch* occurs, if $th \leq sc_{\min}(M)$ (M matches at each position in S), and no suffix of S shares a common prefix with any other suffix. In this case lookahead scoring does not

give any speedup and every suffix must be read up to depth m, leading to an $\mathcal{O}(nm)$ worst case time complexity. This is not worse but also not better than the complexity for *LAsearch*. Next we show that, *independent* of the chosen threshold *th*, the overall worst case running time boundary for *ESAsearch* drops to $\mathcal{O}(n+m)$ under the assumption that

$$n \geq |\mathcal{A}|^m + m - 1 \tag{3.3}$$

holds.

The shorter the common prefixes of the neighboring suffixes, the slower *ESAsearch* runs. Thus to analyze the worst case, we have to consider sequences containing as many different substrings of some length q as possible. Observe that a sequence can contain at most $|\mathcal{A}|^q$ different substrings of length $q > 0$, independent of its length. To analyze the behavior of *ESAsearch* on such a sequence, we introduce the concept of suffix-intervals on enhanced suffix arrays, similar to lcp-intervals as used in [AKO04].

Definition 11 An interval $[i, j]$, $0 \leq i \leq j \leq n$, is a *suffix-interval* with offset $\ell \in \{0, \ldots, n\}$, or *$\ell$-suffix-interval*, denoted $\ell\text{–}[i, j]$, if the following three conditions hold:

1. $\mathsf{lcp}[i] < \ell$
2. $\mathsf{lcp}[j+1] < \ell$
3. $\mathsf{lcp}[k] \geq \ell \quad$ for all $k \in \{x \mid i+1 \leq x \leq j\}$

An *lcp-interval*, or ℓ-interval, with lcp-value $\ell \in \{0, \ldots, n\}$ is a suffix-interval $\ell\text{–}[i, j]$ with $i < j$ and $\mathsf{lcp}[k] = \ell$ for at least one $k \in \{i+1, \ldots, j\}$.

Every lcp-interval $\ell\text{–}[i, j]$ of an enhanced suffix array for text S corresponds to an internal node v in a suffix tree for S, and the length of the string spelled out by the edge labels on the path from the root node to v is equal to ℓ. Leaves are represented as singleton intervals, $\ell\text{–}[i, j]$ with $i = j$. We say that suffix-interval $\ell\text{–}[i, j]$ embeds suffix-interval $\ell^+\text{–}[k, l]$, if and only if $\ell^+ > \ell$, $i \leq k < l \leq j$, and if there is no suffix-interval $\ell'\text{–}[r, s]$ with $\ell < \ell' < \ell^+$ and $i \leq r \leq k < l \leq s \leq j$. As an example for ℓ-suffix-intervals, consider the enhanced suffix array given in Figure 3.6. $[0, 5]$ is a 1-suffix-interval, because $\mathsf{lcp}[0] = 0 < 1$, $\mathsf{lcp}[5+1] = 0 < 1$, and $\mathsf{lcp}[k] \geq 1$, for all k, $1 \leq k \leq 5$. Suffix-interval $2\text{–}[3, 5]$ is embedded in $1\text{–}[0, 5]$, but $3\text{–}[0, 1]$ is not.

Consider an enhanced suffix array of a sequence which contains all possible substrings of length q. There are $|\mathcal{A}|$ 1-suffix-intervals, $|\mathcal{A}|^2$ 2-suffix-intervals, and so on. Consequently, up to depth q, there are a total of

$$E_q = \sum_{i=1}^{q} |\mathcal{A}|^i = \frac{|\mathcal{A}|^{q+1} - |\mathcal{A}|}{|\mathcal{A}| - 1} \tag{3.4}$$

ℓ-suffix-intervals ($1 \leq \ell \leq q$). This corresponds to the number of internal nodes and leaves in a suffix tree, which is atomic up to at least depth q under our assumptions. Note that due to this correspondence, statements on the complexity of *ESAsearch* also hold for the complexity of Dorohonceanu's suffix tree based PSSM searching algorithm described in section 3.4.1 on page 60.

Since we are considering sequences that contain all possible substrings of length q, there are $|\mathcal{A}|^d$ d-suffix-intervals at any depth d, $1 \leq d \leq q$. Let $d\text{–}[i, j]$ be a d-suffix-interval. We know that

$pfxsc_d(v_i, M)$ is a partial sum of $pfxsc_q(v_i, M)$, and because $v_i[0..d-1] = v_{i+1}[0..d-1] = \ldots = v_j[0..d-1]$, $pfxsc_d(v_i, M)$ is also a partial sum of $pfxsc_q(v_k, M)$ for $i \leq k \leq j$. That is, after *ESAsearch* has calculated $pfxsc_d(v_i, M)$ at depth d, at any suffix-interval $(d+1)\text{–}[r, s]$ embedded in $d\text{–}[i, j]$ it suffices to only calculate the "rest" of $pfxsc_q(v_k, M)$. At any depth d, the algorithm calculates $pfxsc_{d+1}(v_r, M) = pfxsc_d(v_i, M) + M(d, v_r[d])$, meaning that all prefix scores at depth $d+1$ in a d-suffix-interval can be computed from the prefix scores at depth d by $|\mathcal{A}|$ matrix look-ups and additions as there are $|\mathcal{A}|$ embedded $(d+1)$-suffix-intervals. There are $|\mathcal{A}|^d$ d-suffix-intervals at depth d. Hence, it takes *ESAsearch* a total of $|\mathcal{A}|^d \cdot |\mathcal{A}|$ matrix look-ups and additions to advance from depth d to $d+1$, and thus we conclude that the algorithm requires a total of $\mathcal{O}(E_q)$ operations to compute all scores for all substrings of length q.

Suppose that *ESAsearch* has read suffix v_i in some step up to depth $q-1$ such that character $v_i[q-1]$ is the last one read. If $\mathsf{lcp}[i+1] \geq q$ holds, then the algorithm has found a suffix-interval $q\text{–}[i, j]$ with a yet unknown right boundary j, otherwise $j = i$. *ESAsearch* reports all $\mathsf{suf}[k]$ with $k \in [i, j]$ as matching positions by scanning over table lcp starting at position i until $\mathsf{lcp}[k] < \mathsf{lcp}[i]$ (such that it finds $j = k-1$), and continues with suffix v_k at depth $\mathsf{lcp}[k]$. Hence processing such a suffix-interval requires one matrix look-up and addition to compute the score, and $j - i + 1$ steps to report all matches and find suffix v_k. Since suffix-intervals do not overlap, the total length of all suffix-intervals at depth q can be at most n, so the total time spent on reporting matches is bounded by n.

There are three cases to consider when determining the time required for calculating the match scores for a PSSM of length m. Let $p := m - q$.

1. If $p = 0$ ($\Rightarrow m = q$), then the time required to calculate all match scores is in $\mathcal{O}(E_q)$ as discussed above.

2. If $p < 0$ ($\Rightarrow m < q$), then none of the m-suffix-intervals are singletons since we assumed that the sequence under consideration contains all possible substrings of length q, i.e., there must be suffixes sharing a common prefix of length m, and the time required to calculate all match scores is in $\mathcal{O}(E_m)$.

3. If $p > 0$ ($\Rightarrow m > q$), then every m-suffix-interval can be a singleton, and all prefix scores for the PSSM prefix of length q are calculated in $\mathcal{O}(E_q)$ time. However, the remaining scores for the pending substrings of length p must be computed for *every* suffix longer than q, taking $\mathcal{O}(np)$ additional time, and leading to a total $\mathcal{O}(E_q + np)$ worst case time complexity for computing all match scores.

Note that a text containing $|\mathcal{A}|^q$ different substrings must have a certain length, which must be at least $|\mathcal{A}|^q$. In fact, a minimum length text that contains all strings of length q has length $n = |\mathcal{A}|^q + q - 1$. It represents a *de Bruijn sequence* [dB46] without wrap-around, i.e., a *de Bruijn* sequence with its first $q-1$ characters concatenated to its end. Since a *de Bruijn* sequence without wrap-around represents the minimum length worst case, we infer from Equation (3.4) that $E_q \in \mathcal{O}(n)$. Hence, if $m = q$, then it takes $\mathcal{O}(n)$ time to calculate all match scores. If $m < q$, then $E_m < E_q$ and thus it takes sublinear time. If $m > q$, it takes $\mathcal{O}(n + np)$ time.

i	$\mathsf{suf}[i]$	$\mathsf{lcp}[i]$	$S_{\mathsf{suf}[i]}$
0	5	0	aaccgtcttggc$
1	6	1	accgtcttggc$
2	1	1	agataaccgtcttggc$
3	3	1	ataaccgtcttggc$
4	0	0	cagataaccgtcttggc$
5	7	1	ccgtcttggc$
6	8	1	cgtcttggc$
7	11	1	cttggc$
8	16	1	c$
9	2	0	gataaccgtcttggc$
10	15	1	gc$
11	14	1	ggc$
12	9	1	gtcttggc$
13	4	0	taaccgtcttggc$
14	10	1	tcttggc$
15	13	1	tggc$
16	12	1	ttggc$
17	17	0	$

i	$\mathsf{suf}[i]$	$\mathsf{lcp}[i]$	$T_{\mathsf{suf}[i]}$
0	2	0	aaacaccc$
1	3	2	aacaccc$
2	4	1	acaccc$
3	6	2	accc$
4	1	0	caaacaccc$
5	5	2	cacccc$
6	0	1	ccaaacaccc$
7	7	2	ccc$
8	8	2	cc$
9	9	1	c$
10	10	0	$

Figure 3.7: Minimum sized enhanced suffix arrays for worst case analysis. Enhanced suffix arrays for text $S =$ `cagataaccgtcttggc`, consisting of all strings of length $m = 2$ over an alphabet of size 4 (left), and $T =$ `ccaaacaccc`, consisting of all strings of length $m = 3$ over an alphabet of size 2 (right). S and T are both *de Bruijn* sequences without wrap-around for the given alphabets.

We summarize the worst case running time of *ESAsearch* for preprocessing a PSSM M of length m, searching with M, and reporting all matches with their match scores, as

$$\mathcal{O}\left(n + n \cdot \max\left\{0, p\right\} + m\right).$$

Hence, the worst case running time is $\mathcal{O}\left(n + m\right)$ for $p \leq 0$, implying that this time complexity holds for any PSSM of length m and threshold on any text of length $n \geq |\mathcal{A}|^m + m - 1$, as already stated in Inequality (3.3).

In practice, large numbers of suffixes can be skipped if the threshold is stringent enough, leading to a total running time *sublinear* in the size of the text, regardless of the relation between n and m. *ESAsearch* reads a suffix up to depth m unless an intermediate score falls short of an intermediate threshold, and skips intervals with the same or greater lcp if this happens. Right boundaries of skipped suffix-intervals are found quickly by following the chain of skip-values (see function `skipchain`). It are these jumps that make *ESAsearch* superior in terms of running time to *LAsearch* in practice. The best case is indeed $\mathcal{O}\left(|A|\right)$ which occurs whenever there is no score in the first row of the PSSM that is greater than th_0.

See Figure 3.7 for examples of enhanced suffix arrays, constructed from texts S and T that consist of all strings of a certain length m over some alphabet. In these enhanced suffix arrays no suffix shares a prefix of length m with any other suffix, forcing *ESAsearch* to compute scores for each suffix. But with the intermediate scores available while processing the suffixes, it takes exactly E_m steps to compute the scores, as can be figured out by manually applying *ESAsearch* to the depicted

enhanced suffix arrays. For S, exactly $\frac{4^3-4}{4-1} = 20$, for T, exactly $\frac{2^4-2}{2-1} = 14$ operations are needed to compute *all* $|A|^m \leq n - m + 1$ possible scores (and to find all matches since S and T are both *de Bruijn* sequences without wrap-around). Only a single match is reported per matching substring, leading to $E_m \in \mathcal{O}(n)$ operations to be performed during the search phase.

3.6 Further performance improvements via alphabet transformations

Inequality (3.3) provides the necessary condition for $\mathcal{O}(n+m)$ worst case running time. We now assume that m in Inequality (3.3) identifies not the length of a PSSM, but the threshold dependent *expected reading depth* for some PSSM. We denote this expected depth by $m^*(th) \leq m$ and continue denoting the PSSM's length by m. As seen before, for PSSMs with length m, such that $p = m - m^*(th)$, the worst case running time is $\mathcal{O}(n + n \cdot \max\{0, p\} + m)$, but the expected running time is $\mathcal{O}(n+m)$, as on average we expect $p \leq 0$.

Inequality (3.3) with m substituted by $m^*(th)$ implies $\log_{|\mathcal{A}|}(n) \geq m^*(th)$. That is, to achieve linear worst case running time for the amino acid alphabet, $m^*(th)$ needs to be very small. For instance, if $n = 20^7$, then the search time is guaranteed to be linear in n only for PSSMs with a maximum length of 7, and expected to be linear for PSSMs with expected reading depth of 7. Observe that for $|\mathcal{A}| = 4$, $m^*(th)$ needs to be smaller or equal to 15 to achieve linear or sublinear running times. This provides the motivation to reduce the alphabet size by transforming $\mathcal{A}$ into a reduced size $\widehat{\mathcal{A}}$ such that $|\widehat{\mathcal{A}}| < |\mathcal{A}|$.

In practice, for reasonably chosen thresholds th, the performance of *ESAsearch* mainly depends on the fact that often large ranges of suffixes in the enhanced suffix array can be skipped. This is always the case if we drop below an intermediate threshold while calculating a prefix' score, and if that prefix is a common prefix of other suffixes. In terms of lcp-intervals, this means that we can skip all ℓ-intervals with $\ell \geq m^*(th)$ on average. In contrast to *suffix-intervals*, whose total count is in $\mathcal{O}(n^2)$, size and number of *lcp-intervals* depend on $|\mathcal{A}|$, as illustrated in Figure 3.8. We observe that smaller alphabet sizes imply (1) larger ℓ-intervals, and (2) an increasing number of ℓ-intervals for larger values of ℓ. Thus, by using reduced alphabets, we expect to skip larger and touch fewer lcp-intervals under the assumption that the average reading depth remains unchanged. Consequently, we expect to end up with an improved performance of *ESAsearch*. This raises the question for a proper reduction strategy for larger alphabets like the amino acid alphabet, and how this strategy can be incorporated into *ESAsearch*.

We now describe how to take advantage of reduced alphabets as fast filters in the *ESAsearch* algorithm. Let $\mathcal{A} = \{a_0, a_1, \ldots, a_k\}$ and $\widehat{\mathcal{A}} = \{b_0, b_1, \ldots, b_l\}$ be two alphabets, and $\Phi : \mathcal{A} \to \widehat{\mathcal{A}}$ a surjective function that maps a character $a \in \mathcal{A}$ to a character $b \in \widehat{\mathcal{A}}$. We call $\Phi^{-1}(b)$ the character class corresponding to b. For a sequence $S = s_1 s_2 \ldots s_n \in \mathcal{A}^n$ we denote the transformed sequence with $\widehat{S} = \Phi(s_1)\Phi(s_2)\ldots\Phi(s_n) \in \widehat{\mathcal{A}}^n$. Along with the transformation of the sequence, we transform a PSSM such that we have a one to one relationship between the columns in the PSSM and the characters in $\widehat{\mathcal{A}}$. We define the transformed PSSM $\widehat{M}$ of M as follows:

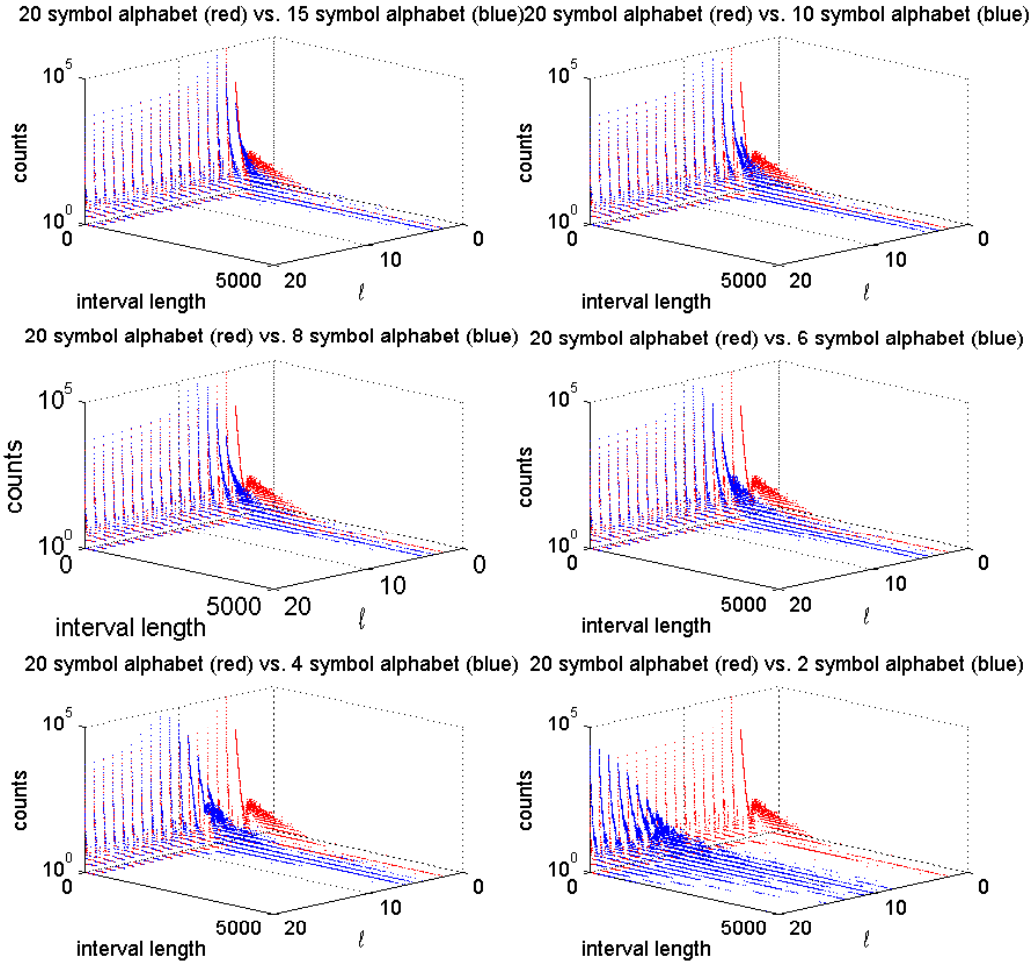


Figure 3.8: Numbers of ℓ-intervals for $\ell \in [1, 20]$ of different length for various reduced alphabets. We built the enhanced suffix array with sequences from the RCSB protein data bank (PDB) (total sequence length 4,264,239 bytes). The used reduced amino acid alphabets are given in Figure 3.10. Note that we limited the interval lengths in the figures to 5,000 to prevent distortion.

(A)denin	(C)ytosin	(G)uanin	(T)hymin	(P)urine	P(Y)rimidine
28.50	256.54	85.51	28.50	85.51	256.54
28.62	47.70	47.70	9.54	47.70	47.70
45.54	45.54	45.54	500.92	45.54	500.92
320.83	0.00	71.29	106.94	320.83	106.94
47.29	15.76	15.76	31.53	47.29	31.53
41.34	13.78	41.34	96.46	41.34	96.46
32.95	8.24	32.95	41.19	32.95	41.19
21.28	21.27	148.95	106.40	148.95	106.40
9.54	28.62	47.70	47.70	47.70	47.70

Figure 3.9: PSSM alphabet transformation. In the left PSSM M we used the normal four letter nucleotide alphabet $\mathcal{A} = \{A, C, G, T\}$ to describe a transcription factor binding site found in Hox A3 gene promotors. In the right PSSM $\widehat{M}$ we used a reduced two letter alphabet $\widehat{\mathcal{A}} = \{P, Y\}$ that differs only between purine (adenine or guanine) and pyrimidine (cytosine or thymine) nucleotides. Hence we have two character classes: $\Phi^{-1}(P) = \{A, G\}$ and $\Phi^{-1}(Y) = \{C, T\}$. Consequently $\widehat{M}(i, P) = \max\{M(i, a) \mid a \in \{A, G\}\}$ and $\widehat{M}(i, Y) = \max\{M(i, a) \mid a \in \{C, T\}\}$ for all $i \in [0, 8]$

Definition 12 Let M be a PSSM of length m over alphabet $\mathcal{A}$, and $\Phi : \mathcal{A} \to \widehat{\mathcal{A}}$ a surjective function. The transformed PSSM $\widehat{M}$ is defined as a function $\widehat{M} : [0, m-1] \times \widehat{\mathcal{A}} \to \mathbb{R}$ with

$$\widehat{M}(i, b) := \max\left\{M(i, a) \mid a \in \Phi^{-1}(b)\right\}. \tag{3.5}$$

Figure 3.9 gives an example of the relationship between M and $\widehat{M}$. $\widehat{S}$ can be easily determined from S in $\mathcal{O}(n)$ time, $\widehat{M}$ in $\mathcal{O}(|\mathcal{A}|m)$ time, given M. We define the set of matches to M on S and $\widehat{M}$ on $\widehat{S}$, respectively, as

$$\begin{aligned} MS &:= \{j \in [0, n-m] \mid sc(S[j..j+m-1], M) \geq th\} \\ \widehat{MS} &:= \left\{j \in [0, n-m] \mid sc\left(\widehat{S}[j..j+m-1], \widehat{M}\right) \geq th\right\}. \end{aligned}$$

Now observe that we can use matches of $\widehat{M}$ on $\widehat{S}$, for the computation of matches of M on S, since $MS \subseteq \widehat{MS}$. We prove that $MS \subseteq \widehat{MS}$ holds for all $th \in [sc_{\min}(M), sc_{\max}(M)]$ by proving the more general statement given in the following Lemma.

Lemma 2 $sc(w, M) \leq sc\left(\widehat{w}, \widehat{M}\right)$ holds for all $w \in \mathcal{A}^m$.

Proof:

$$\begin{aligned} sc(w, M) = \sum_{i=0}^{m-1} M(i, w[i]) &\leq \sum_{i=0}^{m-1} \max\left\{M(i, a) \mid a \in \Phi^{-1}(\Phi(w[i]))\right\} \\ &= \sum_{i=0}^{m-1} \widehat{M}(i, \Phi(w[i])) = sc\left(\widehat{w}, \widehat{M}\right). \end{aligned}$$

□

Thus the following implications follow directly

- $sc\,(w, M) \geq th \Rightarrow sc\left(\widehat{w}, \widehat{M}\right) \geq th$
- $i \in MS \Rightarrow i \in \widehat{MS}$

and we conclude: $MS \subseteq \widehat{MS}$ holds for $th \in [sc_{\min}(M), sc_{\max}(M)]$.

Hence we can search with $\widehat{M}$ in $\widehat{S}$ for prefiltering of matches to M in S, profiting of longer and larger ℓ-intervals in $\widehat{S}$ by extending algorithm *ESAsearch* as follows:

(1) Transform S into $\widehat{S}$ and build the enhanced suffix array for $\widehat{S}$;

(2) Construct $\widehat{M}$ from M;

(3) Compute $\widehat{MS}$ by searching with $\widehat{M}$ on the enhanced suffix array of $\widehat{S}$ using *ESAsearch*;

(4) For each $i \in \widehat{MS}$ re-score match with $\sigma = sc\,(S[i..i+m-1], M)$, and report i and σ if and only if $\sigma \geq th$.

As a further consequence of Definition 12 the maximum score values in each row of M and $\widehat{M}$ and thus the intermediate thresholds remain unchanged in the transformation process. Unfortunately the necessary PSSM transformation accompanying alphabet size reduction affects the expected reading depth $m^*(th)$ in such a way that it increases with more degraded alphabets, and therefore reduces the expected performance improvement. Due to maximization according to Equation (3.5) the matrix values in $\widehat{M}$ increase and we expect a decreased probability of falling short of an intermediate threshold early. Observe that there is a trade-off between increased expected reading depth and increased lcp-interval sizes at low reading depths. Therefore it is desirable to minimize the effect of maximization by grouping PSSM columns with similar score values, i.e., highly correlated columns. Since PSSMs reflect the properties of the underlying multiple alignment, we expect correlations of PSSM columns according to biologically motivated symbol similarities. Hence character correlation is the motivation for our alphabet reduction strategy.

3.6.1 Reduced amino acid alphabets

It is well known that various of the naturally occurring amino acids share certain similarities, like similar physiochemical properties. Accordingly, the complexity of protein sequences can be reduced by sorting these amino acids with similarities into groups and deriving a transformed, reduced alphabet [LFWW03]. These reduced alphabets contain symbols that represent a specific character class of the original alphabet.

Since PSSMs and the sequences to be searched have to be encoded over the same alphabet, we are more interested in a single reduced alphabet suitable for all PSSMs under consideration, than in PSSM-specific reduced alphabets. The latter implies an unacceptable overhead of index generation for sequences over PSSM-specific alphabets, even though it may result in a lower expected reading depth. The basis for our reduction of the 20-letter amino acid alphabet to smaller alphabets are

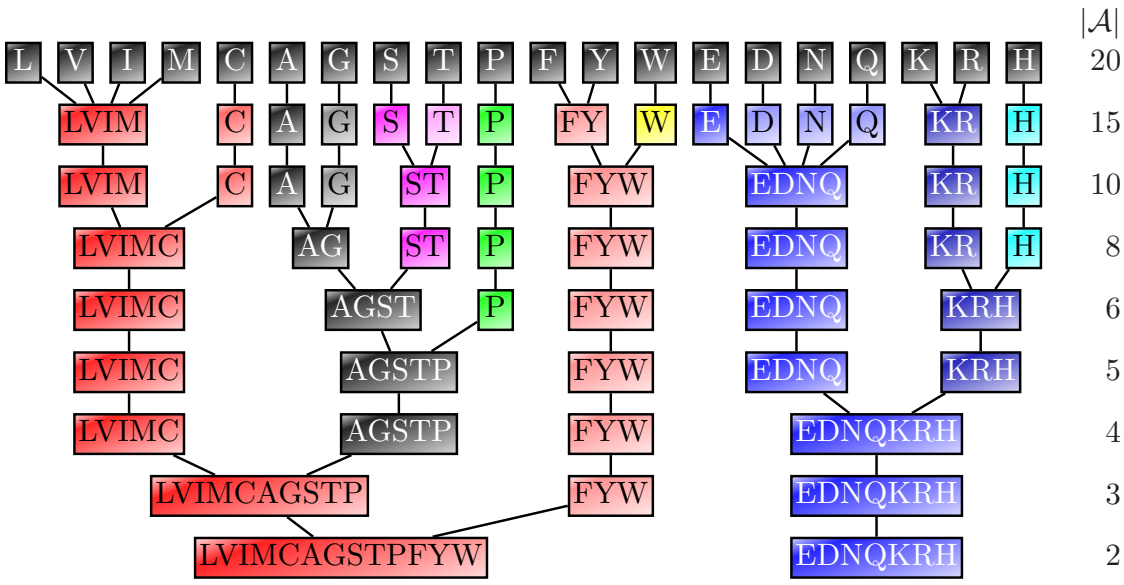


Figure 3.10: Reduction of the amino acid alphabet into smaller groups. Amino acid pairs are iteratively grouped together based on ther correlations $c_{a,b}$ (see Equation (3.6) for the definition of $c_{a,b}$), starting with the most correlated pairs, until al amino acids are divided into the desired number of groups. Here we used BLOSUM50 similarities for the determination of $c_{a,b}$. Observe that, hydrophobic amino acids, especially (LVIM) and (FYW) are conserved in many reduced alphabets. The same is true for the polar (ST), (EDNQ), and (KR) groups. The smallest alphabet contains two groups that can be categorized broadly as hydrophobic/small (LVIMCAGSTPFYW) and hydrophilic (EDNQKRH).

correlations indicated by the BLOSUM similarity matrix as described in [MWL00]. That is, amino acid pairs with high similarity scores are grouped together (see Figure 3.10 for an example). Let a and b be two amino acids and Y a 20×20 score matrix, then a measure of amino acid correlation $c_{a,b}$ between a and b can be defined as

$$c_{a,b} := \frac{\sum_{i=1}^{20} Y_{a,i} Y_{b,i}}{\left(\sum_{i=1}^{20} Y_{a,i}^2\right)\left(\sum_{i=1}^{20} Y_{b,i}^2\right)} \tag{3.6}$$

and amino acid pairs can be iteratively grouped together according to their correlations, starting with the most correlated pairs, until all the amino acids are divided into the desired number of groups.

3.7 A unifying view on SPsearch, LAsearch, and ESAsearch

In the following, we recapitulary take a unifying view on algorithms *SPsearch*, *LAsearch*, and *ESAsearch* focussing on similarities and differences. For the considered example given in Figure 3.11, let M be a PSSM of length $m = 3$ over the nucleotide alphabet $\mathcal{A} = \{A, C, G, T\}$ with $M(i, A) = 2$, $M(i, C) = 3$, $M(i, G) = 4$, and $M(i, T) = 5$ for any $i \in [0, 2]$ and $th = 12$ a given threshold. We obtain intermediate thresholds of $th_0 = 2$, $th_1 = 7$, and $th_2 = th = 12$. The sequence of length $n = 21$, to be scanned for matches of M is denoted with S and given as $S =$ `ACCCACCGTACGTAACACTGA`. With $W(S\$) = \{w | S\$ = vw \wedge |w| \geq m \wedge v, w \in \mathcal{A}^* \wedge \$ \notin \mathcal{A}\}$ we denote the set of suffixes of $S\$$ with a length of at least m. All three algorithms find all positions $j \in [0, n-m]$ in S and their assigned match scores, such that $sc\,(S[j..j+m-1], M) \geq th$ holds. To do so, *SPsearch* and *LAsearch* slide along the sequence, calculate $sc\,(w, M)$ for each $w \in \mathsf{words}_m(S)$, and report positions $j \in [0, |S| - 1]$ for which $sc\,(S[j..j+m-1], M) \geq th$ holds. This is equal to scoring the first m characters of each element of $W(S\$)$. Since the order in which suffixes of S are scored is neither relevant for *SPsearch* nor *LAsearch*, both algorithms can be viewed as operating on the suffix array suf of $S\$$. See (A) and (B) in Figure 3.11 for an example. Characters that have to be scored are marked red in this Figure. Characters, whose scoring is avoided when using lookahead scoring by falling short of an intermediate threshold are marked green. For $th = 12$ the only matching substrings of S are `CGT` and `CTG` occuring at text positions 6, 10, and 17. By incorporating information from table lcp we can reuse prefix scores of suffixes with common prefixes and avoid additional character scorings. $\mathsf{lcp}[i]$ gives us the length of the common prefix of suffixes $\mathsf{suf}[i-1]$ and $\mathsf{suf}[i]$. For an example, see the blue marked characters in section (C) in Figure 3.11. Facilitating information stored in table skp allows to directly skip ranges of suffixes for which no characters need to be scored and hence avoids to check the values in table lcp for suffixes in this ranges. This leads to algorithm *ESAsearch* and is shown in section (D) of Figure 3.11 by the yellow marked parts of the suffix array. Entries in table skp give the index position of the next suffix in suf to be considered.

By using lookahead scoring and information from tables lcp and skp, the total number of scored characters in this example can be reduced from 57 (*SPsearch* see (A)) to 19 (*ESAsearch* see (D)).

(A): *SPsearch*

i	suf[i]	$S_{suf[i]}$
0	13	AACACTGA
1	14	ACACTGA
2	0	ACCCACCGTACGTAACACTGA
3	4	ACCGTACGTAACACTGA
4	9	ACGTAACACTGA
5	16	ACTGA
6	20	A
7	3	CACCGTACGTAACACTGA
8	15	CACTGA
9	2	CCACCGTACGTAACACTGA
10	1	CCCACCGTACGTAACACTGA
11	5	CCGTACGTAACACTGA
12	10	CGTAACACTGA
13	6	CGTACGTAACACTGA
14	17	CTGA
15	19	GA
16	11	GTAACACTGA
17	7	GTACGTAACACTGA
18	12	TAACACTGA
19	8	TACGTAACACTGA
20	18	TGA

(B): *LAsearch*

i	suf[i]	$S_{suf[i]}$
0	13	AACACTGA
1	14	ACACTGA
2	0	ACCCACCGTACGTAACACTGA
3	4	ACCGTACGTAACACTGA
4	9	ACGTAACACTGA
5	16	ACTGA
6	20	A
7	3	CACCGTACGTAACACTGA
8	15	CACTGA
9	2	CCACCGTACGTAACACTGA
10	1	CCCACCGTACGTAACACTGA
11	5	CCGTACGTAACACTGA
12	10	CGTAACACTGA
13	6	CGTACGTAACACTGA
14	17	CTGA
15	19	GA
16	11	GTAACACTGA
17	7	GTACGTAACACTGA
18	12	TAACACTGA
19	8	TACGTAACACTGA
20	18	TGA

(C): *LAsearch* using lcp information

i	suf[i]	lcp[i]	$S_{suf[i]}$
0	13	0	AACACTGA
1	14	1	ACACTGA
2	0	2	ACCCACCGTACGTAACACTGA
3	4	3	ACCGTACGTAACACTGA
4	9	2	ACGTAACACTGA
5	16	2	ACTGA
6	20	1	A
7	3	0	CACCGTACGTAACACTGA
8	15	3	CACTGA
9	2	1	CCACCGTACGTAACACTGA
10	1	2	CCCACCGTACGTAACACTGA
11	5	2	CCGTACGTAACACTGA
12	10	1	CGTAACACTGA
13	6	4	CGTACGTAACACTGA
14	17	1	CTGA
15	19	0	GA
16	11	1	GTAACACTGA
17	7	3	GTACGTAACACTGA
18	12	0	TAACACTGA
19	8	2	TACGTAACACTGA
20	18	1	TGA

(D): *ESAsearch*

i	suf[i]	lcp[i]	skp[i]	$S_{suf[i]}$
0	13	0	21	AACACTGA
1	14	1	6	ACACTGA
2	0	2	5	ACCCACCGTACGTAACACTGA
3	4	3	3	ACCGTACGTAACACTGA
4	9	2	5	ACGTAACACTGA
5	16	2	5	ACTGA
6	20	1	6	A
7	3	0	21	CACCGTACGTAACACTGA
8	15	3	8	CACTGA
9	2	1	14	CCACCGTACGTAACACTGA
10	1	2	11	CCCACCGTACGTAACACTGA
11	5	2	11	CCGTACGTAACACTGA
12	10	1	14	CGTAACACTGA
13	6	4	13	CGTACGTAACACTGA
14	17	1	14	CTGA
15	19	0	21	GA
16	11	1	17	GTAACACTGA
17	7	3	17	GTACGTAACACTGA
18	12	0	21	TAACACTGA
19	8	2	19	TACGTAACACTGA
20	18	1	20	TGA

Figure 3.11: Algorithms *SPsearch* (A), *LAsearch* (B), *LAsearch* facilitating information from table lcp (C), and *ESAsearch* (D) in comparison. For a detailed explanation of color semantics, used PSSM, and threshold th, see text.

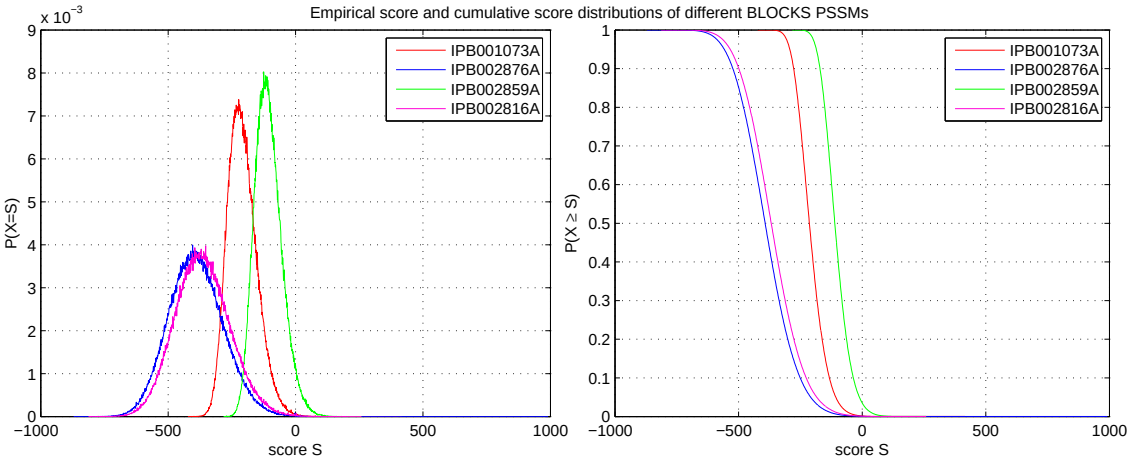


Figure 3.12: Empirical score ($\mathbb{P}[sc\,(w, M) = x]$, see left) and cumulative score ($\mathbb{P}[sc\,(w, M) \geq x]$, see right) distributions of different PSSMs from the PRINTS database. For the distributions, we searched with the PSSMs in the protein data bank (PDB) and sampled their match scores. Observe that for different PSSMs a fixed score cutoff (x-axis) corresponds to different probability values (y-axis).

3.8 Finding an appropriate threshold for PSSM searching

3.8.1 Probabilities and expectation values

The results of PSSM searches strongly depend on the choice of an appropriate threshold value th. A small threshold may produce a large number of false positive matches without any biological meaning, whereas meaningful matches may not be found if the threshold is too stringent. PSSM-scores are not equally distributed and thus scores of two different PSSMs are not comparable. This is even true for PSSMs taken from the same collection (see Figure 3.12).

It is therefore desirable to let the user define a significance threshold instead. The expected number of matches in a given random sequence database (E-value) is a widely accepted measure of the significance. We can compute the E-value for a known background distribution and length of the database by exhaustive enumeration of all substrings. However, the time complexity of such a computation is $\mathcal{O}\left(|\mathcal{A}|^m m\right)$ for a PSSM of length m. If the values in M are integers within a certain range $[r_{min}, r_{max}]$ of size $R = r_{max} - r_{min} + 1$, then dynamic programming (DP) methods (cf. [Sta89, WNB00, Rah03]) allow to compute the probability distribution (and hence the E-value) in $\mathcal{O}\left(m^2 R|\mathcal{A}|\right)$ time.

In practice the probability distribution is often not exactly, or completely calculated due to concerns of speed. E.g., in the *EMATRIX* system [WNB00] score thresholds are calculated and stored for probability values in the interval $\pi = 10^{-1}, 10^{-2}, \ldots, 10^{-40}$ only. Consequently, the user can only specify one of these p-value cutoffs. For the calculation of the p-value from a determined match score, *EMATRIX* uses log-linear interpolation on the stored thresholds. A different, commonly used strategy to derive a continuous distribution function uses the extreme value distribution with

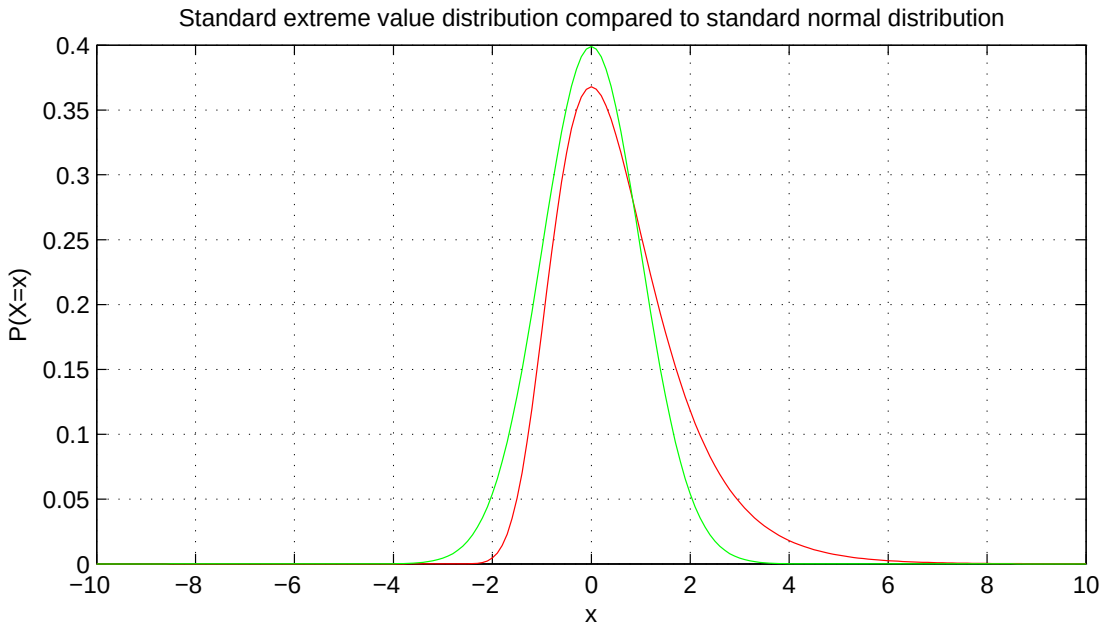


Figure 3.13: The standard extreme value distribution $p(x) = e^{-x-e^{-x}}$ (red) compared to the standard normal distribution $p(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$ (green).

estimated paramters λ and u (see Equation (3.7)) as an approximation [Cas88, EKM97, GW94] of high scoring matches. The extreme value distribution describes the limit distribution of suitably normalized maxima and is somewhat like a normal distribution, but with a positively skewed tail (see Figure 3.13). It is defined by the probability density function $p(x) = e^{-x-e^{-x}}$, and the probability that a random variable X exceeds x is $\mathbb{P}[X \geq x] = 1 - e^{-e^{-x}}$.

To use this distribution for sequence alignment scores, it has to be normalized such that the probability of a random score S exceeding x can be written as

$$\mathbb{P}[S \geq x] = 1 - e^{-e^{-\lambda(x-u)}}, \tag{3.7}$$

where parameter λ is also called the *decay*, or *scale* parameter, and u is called the *mode*.

Even though it is widely accepted that high-scoring local alignment score distributions of the popular position independent scoring systems PAM and BLOSUM can be well approximated by an extreme value distribution, this cannot be generalized for arbitrary PSSMs.

To check whether an extreme value distribution is a suitable approximation for the distribution of PSSM match scores, we sampled the match scores of PSSMs arbitrarily chosen from the TRANSFAC and BLOCKS database. We randomly shuffled 1000 human promotor sequences of length 1200, taken from the database of transcriptional start sites (DBTSS) and 1000 protein sequences of length 365 (= average sequence length in Uniprot-Swissprot), respectively, preserving their mono-symbol composition. From the derived random PSSM match scores we took the best score for each sequence and calculated the empirical cumulative distribution function. If the match scores S are extreme value distributed, an X-Y plot with $X = S$ and $Y = \ln(-\ln(S))$ should appear linear, since

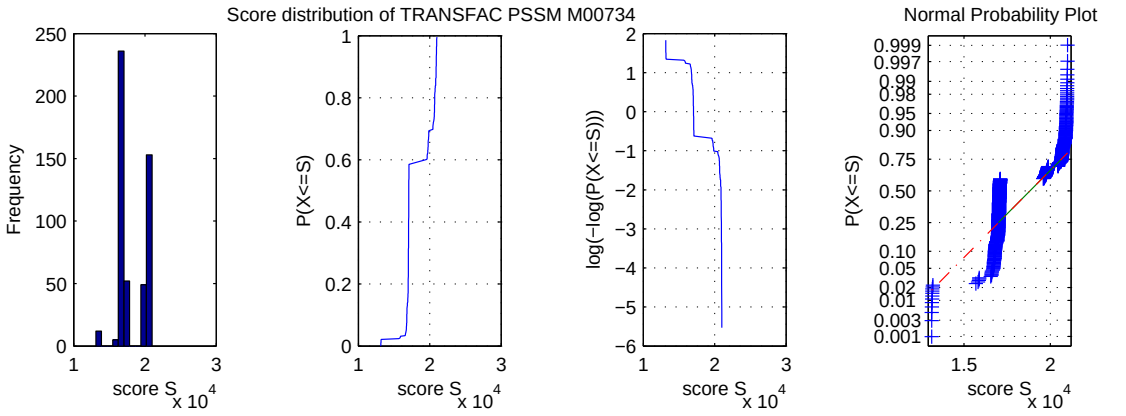


Figure 3.14: Histogram, cumulative score distribution function, X-Y plot, and normal probability plot of TRANSFAC PSSM M00734 (PSSM length $m = 9$).

$\ln\left(-\ln\left(e^{-e^{-\lambda(x-u)}}\right)\right) = -\lambda(x-u)$ holds. For the TRANSFAC PSSM shown in Figure 3.14, the X-Y plot clearly indicates that an extreme value distribution is not an appropriate approximation. For PSSM IPB003211A (see Figure 3.15) from the BLOCKS database, it seems as if the score distribution can be approximated quite well with an extreme value distribution. However, we then still have the problem of adequate parameter estimation for the distribution function.

Since we do not make any assumptions about the used PSSMs in our algorithm, neither about the type of scores, nor the score range, a proper approximation of the score distribution of arbitrary PSSMs is not possible, without time consuming simulations. That is why we are more interested in an exact solution and thus we focus on the efficient computation of an exact discrete score distribution.

3.8.2 Calculation of exact PSSM score distributions

While recent publications [Rah03, WNB00] focus on the computation of the complete probability distribution, what is required specifically for PSSM matching, is computing a partial cumulative distribution corresponding to an E-value resp. p-value specified by the user. Therefore, we have developed a new "lazy" method to efficiently compute only a small fraction of the complete distribution.

We formulate the problem we solve w.r.t. E-values and p-values: Given a user specified E-value η, find the minimum threshold $Tmin_E(\eta, M)$, such that the expected number of matches of M in a random sequence of given length is at most η. Given a user specified p-value π, find the minimum threshold $Tmin_P(\pi, M)$, such that the probability that M matches a random string of length m is at most π. The threshold $Tmin_E(\eta, M)$ can be computed from $Tmin_P(\pi, M)$ according to the equation

$$Tmin_E(\pi \cdot (n - m + 1), M) = Tmin_P(\pi, M). \tag{3.8}$$

Hence we restrict on computing $Tmin_P(\pi, M)$.

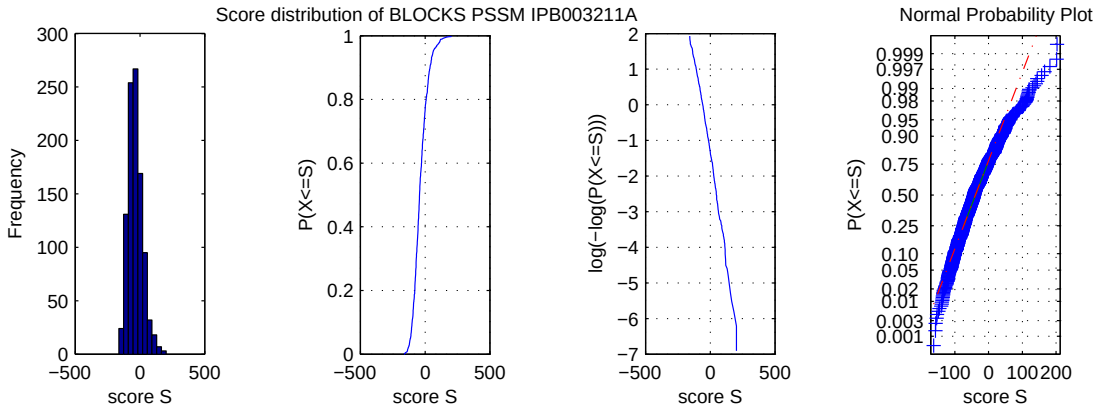


Figure 3.15: Histogram, cumulative score distribution, X-Y plot, and normal probability plot of a PSSM taken from the BLOCKS database (Accession: IPB003211A; PSSM length $m = 40$), describing the UreI protein of *Helicobacter pylori*, a proton gated urea channel [WESS00].

Since all strings of length m have a score between $sc_{\min}(M)$ and $sc_{\max}(M)$, we conclude

$$Tmin_{\mathcal{P}}(1, M) = sc_{\min}(M) \text{ and } Tmin_{\mathcal{P}}(0, M) > sc_{\max}(M).$$

To explain our lazy evaluation method, we first consider existing methods based on DP.

3.8.3 Evaluation with dynamic programming

We assume that at each position in sequence S, the symbols occur independently, with probability $f(a) = (1/n) \cdot |\{i \in [0, n-1] \mid S[i] = a\}|$. Thus a substring w of length m in S occurs with probability $\prod_{i=0}^{m-1} f(w[i])$ and the probability of observing the event $sc(w, M) = t$ is

$$\mathbb{P}[sc(w, M) = t] = \sum_{w \in \mathcal{A}^m : sc(w,M)=t} \prod_{i=0}^{m-1} f(w[i]). \tag{3.9}$$

We obtain $Tmin_{\mathcal{P}}(\pi, M)$ by a look-up in the distribution:

$$Tmin_{\mathcal{P}}(\pi, M) = \min\{t \mid sc_{\min}(M) \le t \le sc_{\max}(M), \mathbb{P}[sc(w, M) \ge t] \le \pi\}. \tag{3.10}$$

If the values in the PSSM M are integers in a range of width R, dynamic programming allows to efficiently compute the probability distribution. The dynamic programming aspect becomes more obvious by introducing for each $k \in [0, m-1]$ the *prefix* PSSM $M_k : [0, k] \times \mathcal{A} \to \mathbb{N}$ defined by $M_k(j, a) = M(j, a)$ for $j \in [0, k]$ and $a \in \mathcal{A}$.

Corresponding distributions $Q_k(t)$ for $k \in [0, m-1]$ and $t \in [sc_{\min}(M_k), sc_{\max}(M_k)]$, and $Q_{-1}(t)$, are defined by

$$Q_{-1}(t) := \begin{cases} 1 & \text{if } t = 0 \\ 0 & \text{otherwise} \end{cases}$$
$$Q_k(t) := \sum_{a \in \mathcal{A}} Q_{k-1}(t - M(k,a)) f(a) \tag{3.11}$$

We have $\mathbb{P}[sc\,(w, M) = t] = Q_{m-1}(t)$. The algorithm computing Q_k determines a set of probability distributions for $M_0, \ldots, M_k$. Q_k is evaluated in $\mathcal{O}\,(sc_{\max}(M)|\mathcal{A}|)$ time from Q_{k-1}, summing up to $\mathcal{O}\,(sc_{\max}(M)|\mathcal{A}|m)$ total time. See Figure 3.16 for an example.

If we allow for floating point scores that are rounded to ϵ decimal places, the time and space requirement increases by a factor of 10^{ϵ}. Conversely, if all integer scores share a greatest common divisor z, the matrix should be canceled down by z.

3.8.4 Restricted probability computation

In order to find $Tmin_{\mathcal{P}}(\pi, M)$ it is not necessary to compute the whole codomain of the distribution function $Q = Q_{m-1}$. We propose a new method only computing a partial distribution by summing over the probabilities for decreasing threshold values $sc_{\max}(M)$, $sc_{\max}(M) - 1, \ldots$, until the given p-value π is exceeded (see Figures 3.16, 3.17).

In step d we compute $Q(sc_{\max}(M) - d)$ where all intermediate scores contributing to $sc_{\max}(M) - d$ have to be considered. In analogy to lookahead scoring, in each row j of M we avoid all intermediate scores below the intermediate threshold th_j because they do not contribute to $Q(sc_{\max}(M) - d)$. The algorithm stops if the cumulated probability for threshold $sc_{\max}(M) - d$ exceeds the given p-value π and we obtain $Tmin_{\mathcal{P}}(\pi, M) = sc_{\max}(M) - d + 1$.

3.8.5 Lazy evaluation of the permuted matrix

The restricted computation strategy performs best if there are only few iterations (i.e., $Tmin_{\mathcal{P}}(\pi, M)$ is close to $sc_{\max}(M)$) and in each iteration step the computation of $Q_k(t)$ can be skipped in an early stage, i.e., for small values of k. The latter occurs to be more likely if the first rows of M contain strongly discriminative values leading to the exclusion of the small values by comparison with the intermediate thresholds. An example of this situation is given in Figure 3.1. Since $Q_k(t)$ is invariant to the permutation of the rows of M, we can sort the rows of M such that the most discriminative rows come first. We found that the difference between the largest two values of a row is a suitable measure for the level of discrimination since a larger difference increases the probability to remain below the intermediate threshold. Since the rows of M are scanned several times, we save time by initially sorting each row in order of descending score.

We divide the computation steps where the step d computes $Q(sc_{\max}(M) - d)$: In step $d = 0$ only the maximal scores $\max_i$, $i \in [0, m-1]$ in each row have to be evaluated.

In step $d > 0$ all scores $M(i,a) \geq \max_i - d$ may contribute to $Q(sc_{\max}(M) - d)$. Since in general a score value $M(i,a) \geq \max_i - d$ also gives contribution to $Q(sc_{\max}(M) - l)$ for $l > d$, we can

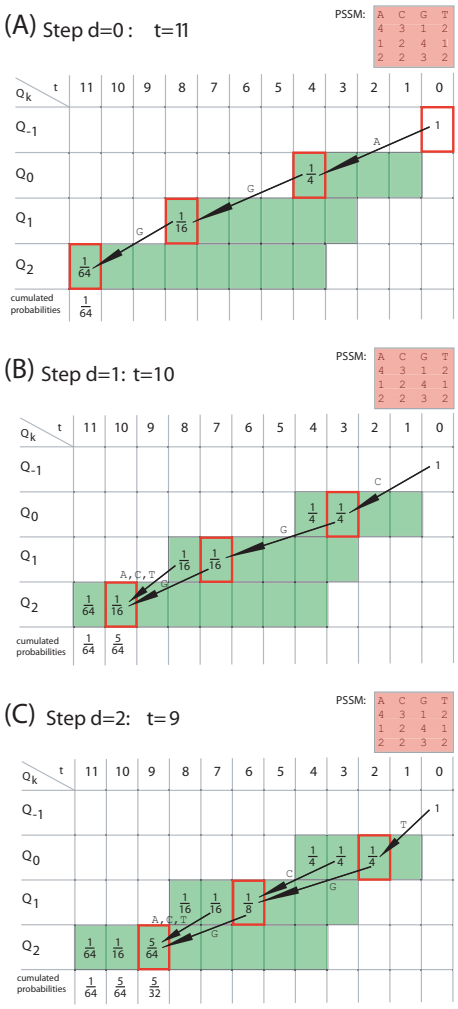


Figure 3.16: The simple DP scheme computes all probability vectors Q_0, Q_1, Q_2 completely within the green marked area, corresponding to score ranges of prefix PSSMs M_k. In contrast to the simple scheme, the restricted probability computation method computes only the upper end of the probability distribution until the given p-value threshold is exceeded, omitting parts of the green area. In this example we show how to compute the score threshold $Tmin_P(\pi, M)$ for PSSM M of length $m = 3$ and a score range of $[4, 11]$ corresponding to a given p-value threshold of $\pi = \frac{1}{8}$. For simplicity we assume a uniform character distribution of $f(A) = f(C) = f(G) = f(T) = \frac{1}{4}$. Cells of the matrix that are computed in the step actually under consideration are marked red. In step $d = 0$, see (A), the algorithm computes $Q_2(11)$ recursively for all paths through M that achieve a score of 11, i.e. $Q_2(11) = Q_1(8) \cdot f(G)$, $Q_1(8) = Q_0(4) \cdot f(G)$, $Q_0(4) = Q_{-1}(0) \cdot f(A) = 1 \cdot \frac{1}{4}$, since `AGG` is the only path achieving score 11. It follows $Q_2(11) = \frac{1}{64}$. In step $d = 1$ all paths achieving a score of $11 - d = 10$ to determine $Q_2(10)$ are computed, see (B). We conclude $Q_2(10) = \frac{1}{16}$. In this step, DP allows to reuse value $Q_1(8)$ without recomputation. In step $d = 2$, see (C) values $Q_1(7)$ and $Q_0(3)$ can be reused to compute $Q_2(9) = \frac{5}{64}$. In step $d = 2$ the cumulated probability $Q_2(11) + Q_2(10) + Q_2(9) = \frac{5}{32}$ exceeds the given p-value threshold of $\pi = \frac{1}{8}$, and the restricted probability computation method skips the rest of the computation. We obtain a score threshold of $th = 10$ correponding to the given p-value threshold $\pi = \frac{1}{8}$.

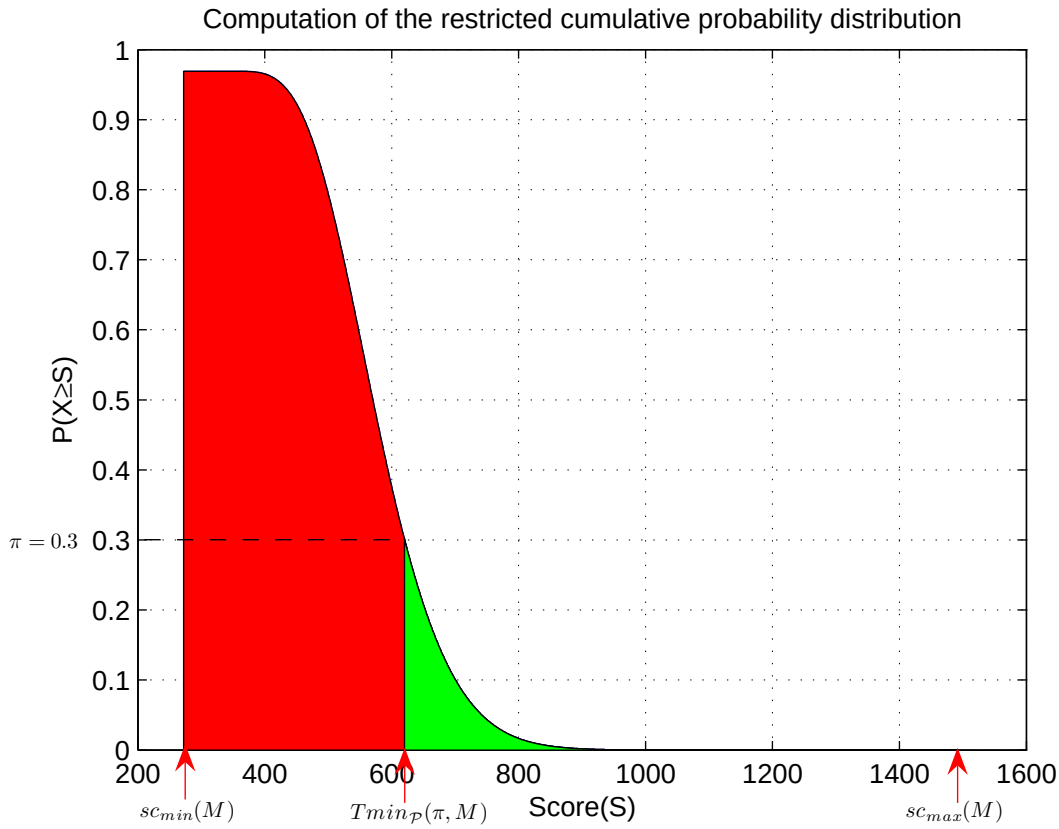


Figure 3.17: Computation of the partial cumulative distribution function. Observe that in order to determine $Tmin_{\mathcal{P}}(\pi, M)$ for $\pi = 0.3$ we do not have to calculate the complete distribution in the score range $[sc_{\min}(M), sc_{\max}(M)]$. It is sufficient to calculate only the upper end (green area) starting with $sc_{\max}(M)$ until $\mathbb{P}[X \geq S] \geq \pi$.

save time by storing $Q_i(\max_i - l)$ for $l > d$, in step d in a buffer and reusing the buffer in steps $d+1, d+2, \ldots$. This allows for the computation of $Q_k(sc_{\max}(M) - d)$ only based on the buffer and scores $M(i,a) = \max_i - d$ while scores $M(i,a) > \max_i - d$, $i \in [0, m-1]$, can be omitted. We therefore have developed an algorithm *LazyDistrib* employing lazy evaluation of the distribution. That is, given a threshold th, the algorithm only evaluates parts of the DP vectors necessary to determine $Q_k(th)$ and simultaneously saves sub-results concerned with score th in an additional buffer matrix *Pbuf* (instead of recomputing them later, see Figure 3.18). This is described by the following recurrence:

$$\begin{aligned} Q_k(th-d) &= Pbuf_k(th-d) + \\ &\qquad \sum_{a \in \mathcal{A}: M(k,a) \geq \max_k - d} Q_{k-1}(th - d - M(k,a)) f(a) \\ Pbuf_k(th-d) &:= \sum_{a \in \mathcal{A}: M(k,a) < \max_k - d} Q_{k-1}(th - d - M(k,a)) f(a) \end{aligned} \tag{3.12}$$

In the present implementation, the algorithm assumes independently distributed symbols. The algorithm can be extended to an order d-Markov model (w.r.t. the background alphabet distribution). This increases the computation time by a factor of $|\mathcal{A}|^d$.

The *modus operandi* of algorithm LazyDistrib

We illustrate the underlying ideas of algorithm *LazyDistrib* with Figure 3.18. In the example given in this Figure, we use the same PSSM M, character distribution, and p-value threshold $\pi = \frac{1}{8}$ as in Figure 3.16. However, in each row of the PSSM the scores are sorted in descending order, and the rows are sorted with the most discriminant row coming first (see coloured PSSMs for this relationship). Observe that the *LazyDistrib* algorithm evaluates the DP vectors non-recursively top-down. Cells computed in the actual step are marked red. In step $d = 0$ the algorithm computes $Q_2(11)$ by evaluating paths through the PSSM contributing to $Q_2(11)$, which is in this example only the high scoring path `GGA`. Intermediate results of $Q_0(4)$, $Q_1(7)$, and $Q_2(11)$ are collected in buffers $Pbuf_0(4)$, $Pbuf_1(7)$, and $Pbuf_2(11)$ first, and finally copied to the correponding cells in Q. See (A) for the situation after step $d = 0$ has been completed. In step $d = 1$, see (B), the algorithm computes $Q_2(10)$, starting in row $k = 1$ with the determination of $Pbuf_1(6)$ and $Q_1(6)$. That is, $Q_1(6) = Pbuf_1(6) = Q_0(4) \cdot f(A) + Q_0(4) \cdot f(C) + Q_0(4) \cdot f(T) = \frac{3}{16}$. Analogously $Q_2(10)$ and $Pbuf_2(10)$ are computed based on $Q_1(7)$ and $Q_1(6)$. Additionally $Pbuf_2(9)$ is filled for further reuse in subsequent steps $d+1$,$d+2, \ldots$. We compute $Pbuf_2(9) = Q_1(6) \cdot f(C) = \frac{3}{64}$. The algorithm can directly start in row $k = 1$ with the computation of $Q_1(6)$ instead of $Q_0(3)$ since a score of 3 cannot be achieved by the first prefix PSSM M_0. Only score 4 of M_0 contributes to $Q_2(10)$, scores 2 and 1 do not. In step $d = 2$, see (C), the algorithm computes $Q_2(9)$, starting in row $k = 0$. $Pbuf_2(9)$ is computed reusing the partial sum calculated in previous steps, such that $Pbuf_2(9) = \frac{3}{64} + Q_1(7) \cdot f(T) + Pbuf_1(5) \cdot f(A) = \frac{5}{64}$, and then copied to $Q_2(9)$. $Pbuf_1(4)$, $Pbuf_2(8)$, and $Pbuf_2(7)$ are filled based on $Pbuf_0(2)$, $Q_1(6)$, $Pbuf_1(5)$, and $Q_1(5)$ for further reuse. After step $d = 2$ the rest of the computation can be skipped since the cumulated probability $Q_2(11) + Q_2(10) + Q_2(9) = \frac{5}{32}$ exceeds the given p-value $\pi = \frac{1}{8}$ and we obtain a score threshold of $th = 10$ corresponding to π.

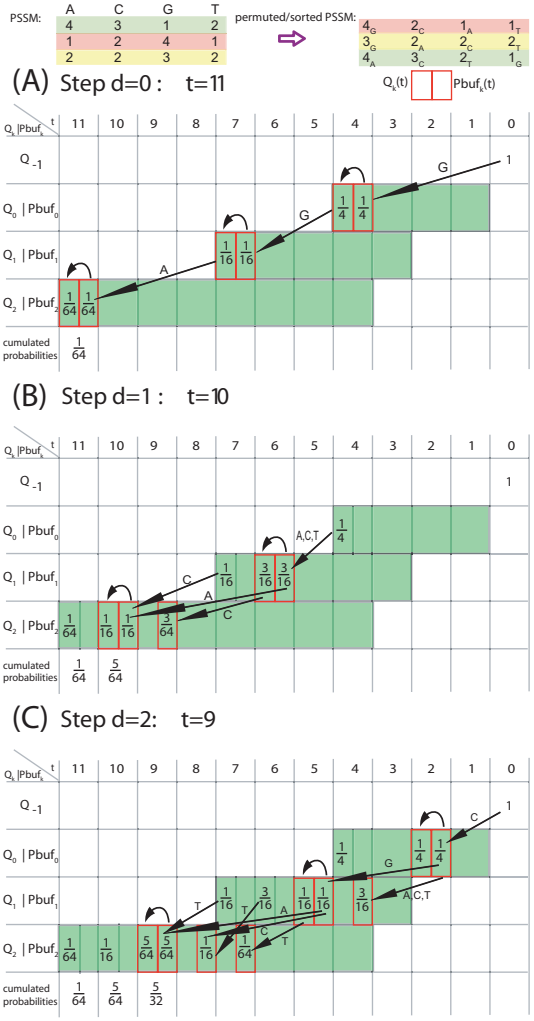


Figure 3.18: Probability computation using lazy evaluation of the DP matrix. For a detailed explanation, see example given in section 3.8.5 on the facing page.

3.9 Threshold independent PSSM matching: The k-best algorithm

In some application scenarios it is even difficult to specify a meaningful significance threshold for the search with PSSMs that differs appropriately between true positive and true negative matches. This is in particular true for relatively short PSSMs of low significance, like PSSMs representing transcription factor binding site motifs where the signal-to-noise ratio is very low. In such cases it is desirable to efficiently determine the, say, k best hits of a PSSM in a sequence without specifying a cutoff for their match score, E-value or p-value. Here k best means the k highest scoring PSSM matches. We now render this more precisely.

Definition 13 Let M be a PSSM of length m, T a text of length n, $k \geq 0$ the number of best matches to be computed. Further, $V = (S_0, p_0), (S_1, p_1), \ldots, (S_i, p_i), \ldots, (S_{n-m+1}, p_{n-m+1})$ denotes a sequence of score, position pairs for each of the potential matching positions of M in T, with $S_i = sc\,(T[p_i..p_i + m - 1], M)$ and $p_i \in [0, m - n + 1]$. We define a permutation $\pi : \{0, 1, \ldots, n - m + 1\} \rightarrow \{0, 1, \ldots, n - m + 1\}$ with $\pi(i) < \pi(j) \Leftrightarrow S_i \leq S_j$ and denote the inverse of π by π^{-1}. Then the *k-best matching problem* is to determine a sequence MS of length k with $MS = (S_{\pi^{-1}(0)}, p_{\pi^{-1}(0)}), (S_{\pi^{-1}(1)}, p_{\pi^{-1}(1)}), \ldots, (S_{\pi^{-1}(k-1)}, p_{\pi^{-1}(k-1)})$.

As a straightforward solution we could use the minimal possible score of the PSSM M under consideration as the threshold th. That is, $th = sc_{\min}(M)$. This guarantees to find all possible matches of M in S and no match which probably belongs to the k-best matches is missed. After the searching phase, the resulting matches then have to be sorted in descending order of their match score and finally the first k hits are reported.

Although with this approach, we find the k best PSSM matches, if S contains at least k subwords of length m, it is inapplicable in practice, especially for longer sequences. Using $th = sc_{\min}(M)$ has the corollary that $sc\,(w, M) \geq th$ holds for all $w \in \mathsf{words}_m(S)$. Thus, $n - m + 1$ matches have to be stored and sorted after the searching phase. A more severe drawback is, that for $th = sc_{\min}(M)$ we cannot make use of lookahead scoring, because according to Lemma 1 on page 55 the following implication holds:

$$\begin{aligned} & sc\,(w, M) \geq th \qquad \text{for all } w \in \mathsf{words}_m(S) \\ \Rightarrow \quad & pfxsc_d(w, M) \geq th_d \quad \text{for all } d \in [0, m-1] \wedge w \in \mathsf{words}_m(S). \end{aligned}$$

Hence we have to score each of the $\mathcal{O}\,(n)$ subwords $w \in \mathsf{words}_m(S)$ completely which takes $\mathcal{O}\,(m)$ time, leading to a time complexity of $O(mn)$ independent of k and we obtain no benefit from lookahead scoring.

To compute the k best matches of a PSSM more efficiently, we propose two new algorithms named *ESAsearchKb* and *LAsearchKb*, that dynamically adjust the used cutoff th while searching. Both are variants of the former described *ESAsearch* and *LAsearch* algorithms respectively. *ESAsearchKb* traverses an enhanced suffix array of the set of target sequences top down like *ESAsearch* whereas *LAsearchKb* operates on the concatenated target sequences and processes them from left to right. Both algorithms update th based on the match scores of PSSM matches found so far while processing.

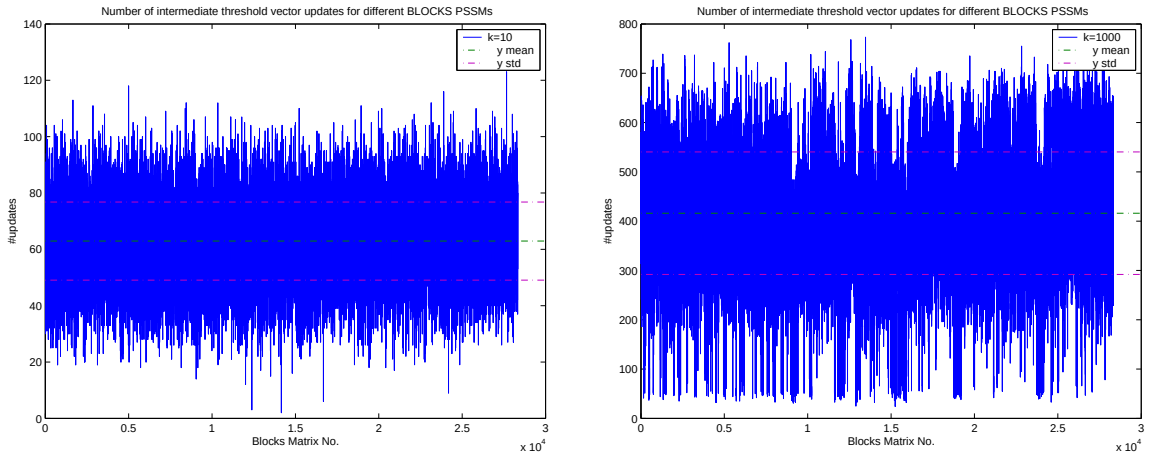


Figure 3.19: Number of intermediate threshold updates for different PSSMs from the BLOCKS database when using *ESAsearchKb* for values of $k = 10$ and $k = 1,000$. The mean (62.93 for $k = 10$ and 416.11 for $k = 1,000$) and the standard deviation (13.87 for $k = 10$ and 124.2 for $k = 1,000$) are shown in green and purple.

The algorithms start, with $th = sc_{\min}(M)$ until k matches to M are found. For simplicity we explain the algorithms in terms of sets, instead of using sequences as in Definition 13. We denote the set of match scores of matches found by MS, analogously. Along with MS we store the matching positions corresponding to the members of MS and update this list accordingly to updates of MS. Once k matches are found and $|MS| = k$ holds, *ESAsearchKb* and *LAsearchKb* determine the minimal matchscore $sc_{curmin}(MS) = \min\{MS\}$ in MS to update the threshold th. Both algorithms continue searching with $th = sc_{curmin}$. To use lookahead scoring, we additionally have to update the vector of intermediate thresholds, based on the new value of th. This can be done in $O(m)$ time if we determine and store the maximum values $\max_d = \max\{M(d,a) \mid a \in \mathcal{A}\}$ of each row in M *a priori*. For each subsequent matching substring w we check $sc(w, M) > th$[1] and update MS if necessary, by

1. removal of the lowest match score $sc_{curmin}(MS)$ from MS and
2. insertion of the new match score $sc(w, M)$.

From the updated set of match scores, we again determine $sc_{curmin}(MS)$ to update th. We apply this procedure whenever we find a new PSSM match with $sc(w, M) > th$, until we have processed our enhanced suffix array, or in case of *LAsearchKb*, the text, completely. Finally we sort the match scores included in MS and report them and their corresponding matching positions.

Whenever we update th we have to recompute $sc_{curmin}(MS)$. Consequently the determination of $sc_{curmin}(MS)$ is a critical point for the performance of both algorithms. To determine $sc_{curmin}(MS)$ efficiently, we could use a binary search tree as the data structure for the organisation of MS. This would allow us to retrieve $sc_{curmin}(MS)$ in $\mathcal{O}(log(k))$ time, where $k = |MS|$ and $log(k)$ is the

[1] Observe that we have to perform this check, because $sc(w, M)$ can also be equal to th.

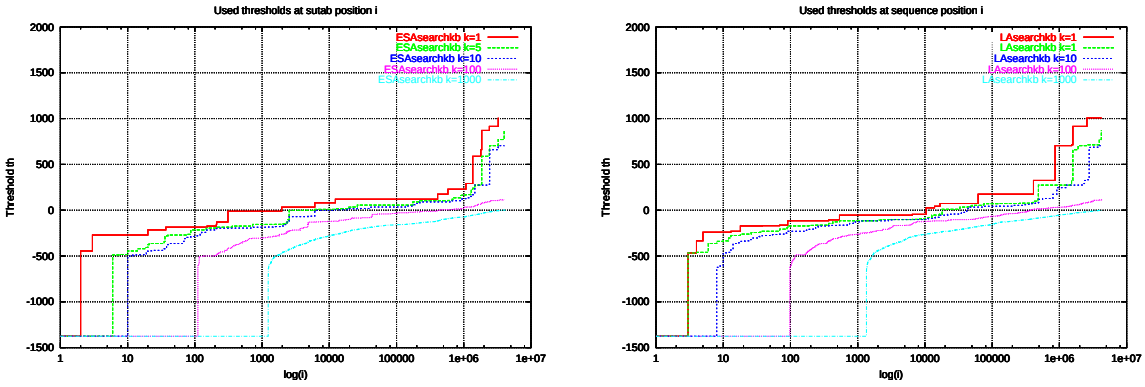


Figure 3.20: Increase of th , when processing the suffixes of S in the lexicographical order of the suffix array suf (left) and in the order of there occurence in S for different values of k and an arbitrarily chosen PSSM from the BLOCKS database (Accession: IPB001140A). The enhanced suffix array was built from protein sequences from the PDB database.

height of the tree. Thus this operation would be fast if the height of the tree is small. The drawback of using normal unbalanced binary search trees is, that they can degenerate. In such a case their performance may be not better than with a plain linked list.

A more appropriate data structure for our problem is a red-black tree. A red-black tree is a balanced binary search tree with one extra bit of storage per node, its color, which can be either *red* or *black*. By constraining the way that nodes may be colored on any path from the root to a leaf, red-black trees ensure that no path is more than twice as long as any other, so the tree is approximately balanced. It can be shown that a red-black tree with n internal nodes has height at most $2\log(n+1)$ (c.f. [TLRS01]), hence red-black trees make good search trees and are well suited for our problem to determine $sc_{curmin}(MS)$ efficiently. By using red-black trees we can guarantee to perform this operation in $\mathcal{O}(log(n))$ time. The same is true for node-insertion and -deletion operations. Tree-rebalancing operations, which are necessary after insert or delete operations to guarantee that the red-black tree properties are not violated, can also be accomplished in $\mathcal{O}(log(n))$ time. Hence red-black trees fit our requirements and we use them for the organization of the set of matches MS.

Both proposed algorithms - *ESAsearchKb* and *LAsearchKb* - perform best, if the determined threshold $th = sc_{curmin}(MS)$ quickly increases, while processing the enhanced suffix array top down and the concatenated sequences from left to right respectively. A higher threshold increases the likelihood of falling short of an intermediate threshold early, resulting in less scored characters and increased overall performance. For *ESAsearchKb*, in turn, this increases the likelihood to make use of common prefixes of suffixes and skip larger parts of the suffix array suf.

The increase of th while searching is influenced by k and the distribution of high PSSM scores in the text and in the enhanced suffix array respectively. The distribution in turn obviously depends strongly on the PSSM under consideration. As shown in Figure 3.19, the number of necessary threshold updates can strongly vary between different PSSMs. This is especially true for larger values of k.

For an example of the concrete changes of th for an arbitrarily chosen PSSM when searching with *ESAsearchKb* and *LAsearchKb* see Figure 3.20. Observe, that the different order in which suffixes are scored in *ESAsearchKb* and *LAsearchKb* has only marginal influence on the changes of th while processing, apparent by the similar shape of the graphs in Figure 3.20. We performed further analyses on the influences of different values of k on the number of threshold updates, number of touched suffixes when using *ESAsearchKb* and total running time (see Experiment 9 in the next section).

3.10 Implementation and computational results

We implemented *SPsearch*, *LAsearch*, *LAsearchKb*, *ESAsearch*, *ESAsearchKb*, and *LazyDistrib* in C. *SPsearch*, *LAsearch* and *ESAsearch* are capable to handle reduced alphabets. The program was compiled with the GNU C compiler (version 3.1, optimization option `-O3`). All measurements were performed on a 8 CPU Sun UltraSparc III computer running at 900MHz, with 64GB main memory (using only one CPU and a small fraction of the memory). Enhanced suffix arrays were constructed with the program `mkvtree`, see [Kur05b].

We performed nine experiments comparing different programs for searching PSSMs. Table 3.1 gives more details on the experimental input for Experiments 1-6. Results are given in Table 3.2 (Exp. 1-5) and Figures 3.21 and 3.22 (Exp. 6). For Experiment 7, see Figures 3.23 and 3.24. Figure 3.25 gives the results of Experiment 8. Results of Experiment 9 are given in Table 3.3.

In these experiments *ESAsearch* performed very well, especially on nucleotide PSSMs, see Experiments 2, 4, and 8. It is faster than *MatInspector* by a factor between 63 and 1,037, depending on the stringency of the given thresholds. The commercial advancement of *MatInspector*, called *MATCH*, was not available for our comparisons, but based on [MFG+03] we presume a running time comparable to *MatInspector*. Compared to *LAsearch*, *ESAsearch* is faster by a factor between 17 (MSS=0.80) and 196 (MSS=0.95) (see Experiment 2). On larger nucleotide sequences (see Experiment 4) the speedup factors increase, ranging from 58 (MSS=0.85) to 275 (MSS=0.95). See Table 3.1 for the definition of MSS. In the experiments using protein PSSMs, *ESAsearch* is faster than the method of [DNM00] by a factor between 1.5 and 1.8 (see Experiment 1). This is due to the better locality behavior of the enhanced suffix array compared to a suffix tree. For larger p-values *LAsearch* performs slightly better than *ESAsearch*. Increasing the stringency, the performance of *ESAsearch* increases, resulting in a speedup of factor 1.5 for a p-value of 10^{-40}. We explain this behavior by the larger alphabet size, resulting in shorter common prefixes and therefore smaller skipped areas of the enhanced suffix array. With increasing stringency of the threshold, the expected reading depth decreases, resulting in larger skipped areas of the enhanced suffix array. Compared to the *FingerPrintScan* program, *ESAsearch* achieves a speedup factor between 3.8 and 470, see Experiment 3. In comparison to *Blimps*, the PSSM-searching program of the BLOCKS database, *ESAsearch* is faster by a factor of 23 (see Experiment 5) for the chosen threshold. In Experiment 6 (see Figures 3.21 and 3.22), we measured the influence of alphabet reductions on the running time of *ESAsearch* and *LAsearch* when using protein PSSMs. Compared to the performance of *ESAsearch* operating on the normal 20 letter amino acid alphabet a speedup up to factor 2 can be achieved when using a 4 letter alphabet and a p-value cutoff of 10^{-20}. Observe that, when using *LAsearch*

	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5	Exp. 6
# searched sequences	59,021	30,964	19,111	1 (*H.s.* Chr. 6)	19,111	19,111
total length	20.2 MB	37.2 MB	4.3 MB	162.9 MB	4.3 MB	4.3 MB
sequence source	see [DNM00]	DBTSS 5.1	RCSB PDB	Sanger V1.4	RCSB PDB	RCSB PDB
sequence type/PSSM type	protein	DNA	protein	DNA	protein	protein
# PSSMs	4,034	220	11,411	577	28,337	11,411
PSSM source	see [DNM00]	MatInspector	PRINTS 38	TRANSFAC Prof. 6.2	BLOCKS 14.1	PRINTS 38
avg. length of PSSMs	29.74	14.21	17.32	13.33	26.3	17.37
index construction (sec)	41	146	10.2	586	10.2	10.2
mdc (sec)	1960	–	1486	–	11871	1486
MatInspector		×				
FingerPrintScan			×			
Blimps					×	
DN00	×					
LAsearch	×	×	×	×	×	×
ESAsearch	×	×	×	×	×	×
ESAsearch (reduced $\mathcal{A}$)						×
LAsearch (reduced $\mathcal{A}$)						×

Table 3.1: Performed experiments and experimental input. Overview of the sequences and PSSMs used in the performed experiments. For the experiments that use p-value or E-value cutoffs, we precomputed the cumulative score distributions and stored them on file. *mdc* is the time needed for this task. In Experiment 1 we measured the running time of the Java-program from [DNM00], referred to by *DN00*. We ran *DN00* with a maximum of 2 GB memory assigned to the Java virtual machine. *DN00* constructs the suffix tree in main memory and then performs the searches. For a fair comparison, we therefore measured the total running time, and the time for matching the PSSMs (without suffix tree construction). For Experiment 2, we implemented the matrix similarity scoring scheme (MSS) of *MatInspector* and matched the PSSMs against both strands of the DNA sequences with different MSS cutoff values. The MSS of PSSM M of length m and a sequence $w \in \mathcal{A}^m$ is defined as $\text{MSS} = \frac{sc(w,M)-sc_{\min}(M)}{sc_{\max}(M)-sc_{\min}(M)}$ and hence given an MSS cutoff value, the threshold th is determined as $th = \text{MSS} \cdot (sc_{\max}(M) - sc_{\min}(M)) + sc_{\min}(M)$. In literature the MSS is sometimes also called *functional depth* of a PSSM [BT04]. Instead of using the reverse strand we use the reverse complement $\overline{M}$ of the PSSM M, defined by $\overline{M}(i,a) = M(m-1-i,\overline{a})$ for all $i \in [0,m-1]$ and $a \in \mathcal{A}$, where $\overline{a}$ is the Watson Crick complement of nucleotide a. This allows to use the same enhanced suffix array for both strands. In Experiment 5 we used a PERL-based wrapper for the *Blimps* program shipped with the BLIMPS distribution to do bulk sequence searches. The overhead for the PERL interpreter call was found to be negligible. For Experiment 6 we used the reduced alphabets given in Figure 3.10. The last seven rows show which programs were used in which experiment.

Experiment 1: 4,034 PSSMs in 20.2 MB protein sequences				
p-value	*DN00* (total time)	*DN00* (search)	*LAsearch*	*ESAsearch* +41 sec.
10^{-10}	65,808	64,939	39,839	41,813
10^{-20}	38,773	37,706	23,786	24,378
10^{-30}	21,449	20,362	14,111	13,084
10^{-40}	9,606	8,533	8,067	5,374

Experiment 2: 220 PSSMs in 37.2 MB DNA			
MSS	*MatInspector*	*LAsearch*	*ESAsearch* +32 sec.
0.80	12,773	3,605	202
0.85	12,567	3,189	108
0.90	12,487	2,818	53
0.95	12,445	2,356	12
1.00	12,429	885	1

Experiment 3: 11,411 PSSMs in 4.3 MB protein sequences			
E-value	*FingerPrintScan*	*LAsearch*	*ESAsearch* +10.2 sec.
10^{-10}	4,733	3,423	1,244
10^{-20}	4,710	486	52
10^{-30}	4,706	27	10

Experiment 4: 577 PSSMs in 162.9 MB DNA		
MSS	*LAsearch*	*ESAsearch* +586 sec.
0.85	18,446	318
0.90	16,376	150
0.95	13,764	50
1.00	5,294	1

Experiment 5: 28,337 PSSMs in 4.3 MB protein sequences			
raw-*th*	*Blimps*	*LAsearch*	*ESAsearch* +10.2 sec.
945	271:30:16	16:03:12	11:35:58

Table 3.2: Results of Experiments 1-5. Experiment 1: Running times in seconds of the different PSSM searching methods at different levels of stringency, when searching for 4,034 amino acid PSSMs in 59,021 sequences (21.2 MB) from SwissProt. These are the same PSSMs and sequences used in the experiments of [DNM00]. Experiment 2: Running times in seconds of *MatInspector*, *LAsearch*, and *ESAsearch*, when searching 220 PSSMs on both strands of 37.2 MB DNA sequence data at different matrix similarity score (MSS) cutoffs. Experiment 3: Running times in seconds of *FingerPrintScan*, *LAsearch*, and *ESAsearch* when searching all 11,411 PSSMs from the PRINTS database in the RCSB protein data bank (PDB) for different E-values. Experiment 4: Running times in seconds of *LAsearch* and *ESAsearch* when searching 577 PSSMs in H. sapiens chr. 6 at different matrix similarity score (MSS) cutoffs. Experiment 5: Running times in `hh:mm:ss` of *Blimps*, *LAsearch*, and *ESAsearch* when searching all 28,337 PSSMs from the BLOCKS database in PDB. We used a raw score threshold of 945 as suggested in the *Blimps* documentation for searching large databases. For each experiment, the additional time needed for the construction of the enhanced suffix array is shown in the head of the *ESAsearch* column.

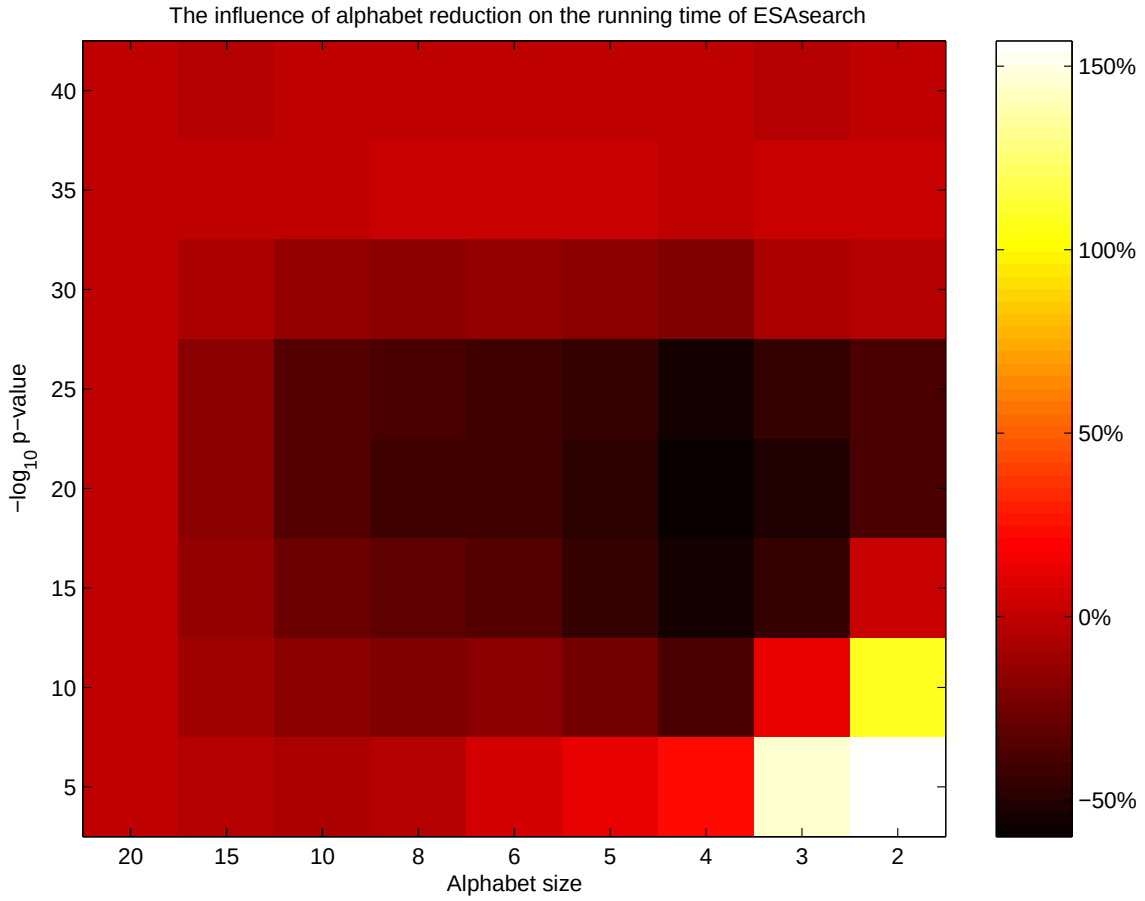


Figure 3.21: Experiment 6: Relative deviations of running time of *ESAsearch* when using reduced alphabets at different levels of stringency. We measured the relative percentage deviation with respect to the running time when using the standard 20 letter amino acid alphabet (= 0%). We searched with 11,411 PSSMs from the PRINTS database (Rel. 38) in the RCSB Protein Data Bank (PDB) with a total sequence length of 4.3 MB. In this example, the maximum performance improvement is achieved for an alphabet of size 4 and a p-value cutoff of $\pi = 10^{-20}$.

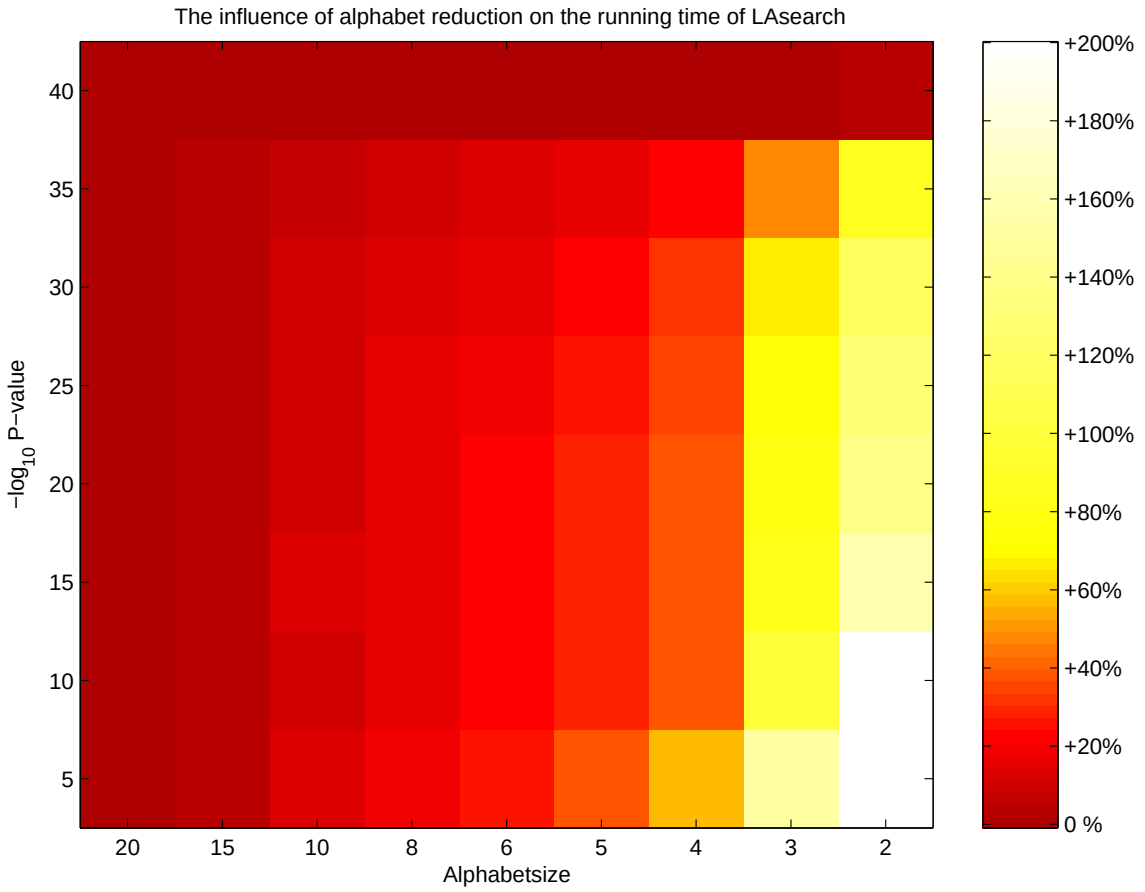


Figure 3.22: Experiment 6: Relative deviations of running time of *LAsearch* when using reduced alphabets at different levels of stringency. The experimental input and setup was the same as in Figure 3.21. For *LAsearch* alphabet reduction has a negative effect on the running time, indicated by brighter colors.

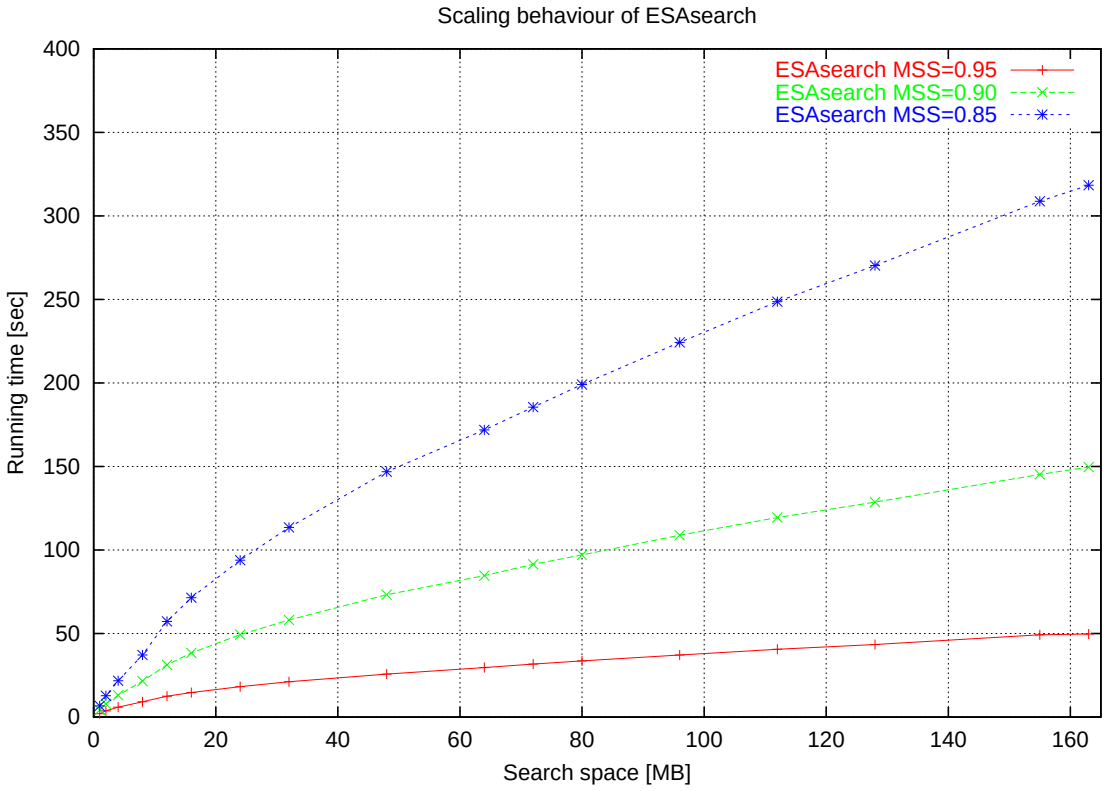


Figure 3.23: Experiment 7: Scaling behavior of *ESAsearch* when searching with 577 TRANSFAC PSSMs on subsets of human chromosome 6 of different sizes and with different matrix similarity cutoff values (MSS). The subsets are prefixes of human chromosome 6 of length 2^k for $k = 0, 1, 2, \ldots, 7$.

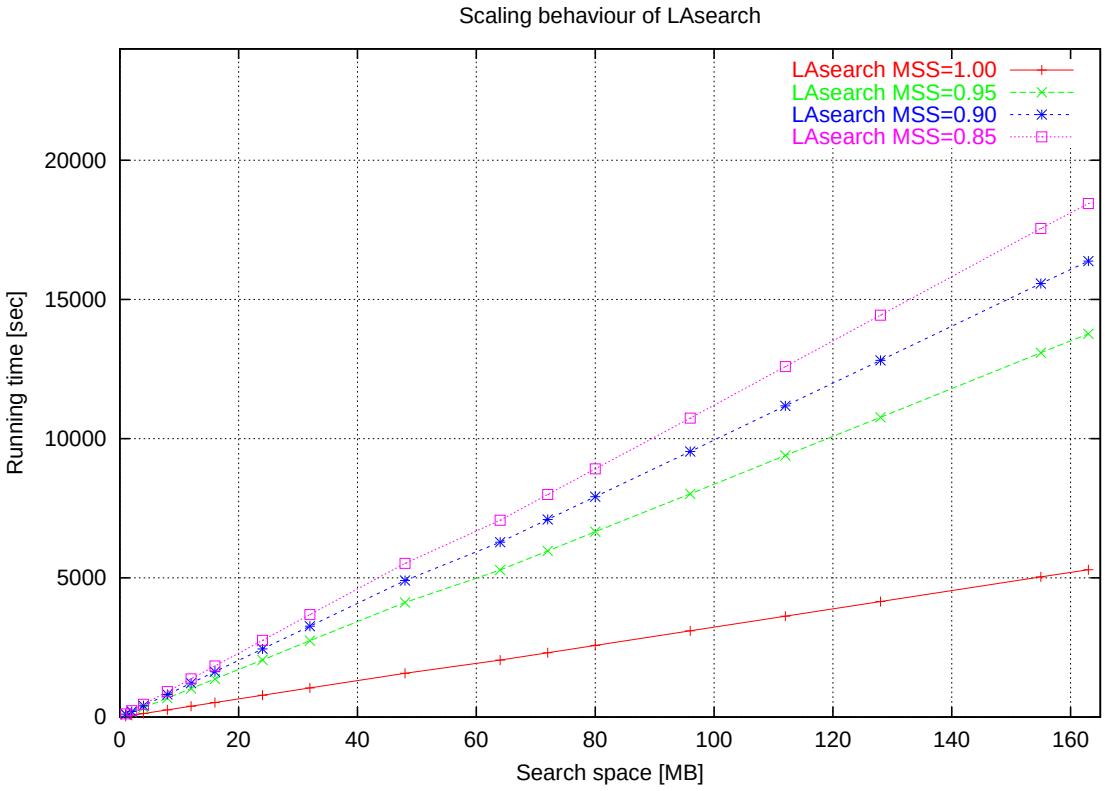


Figure 3.24: Experiment 7: Scaling behavior of *LAsearch* when searching with 577 TRANSFAC PSSMs on subsets of human chromosome 6 of different sizes and with different matrix similarity cutoff values (MSS). The subsets are prefixes of human chromosome 6 of length 2^k for $k = 0, 1, 2, \ldots, 7$.

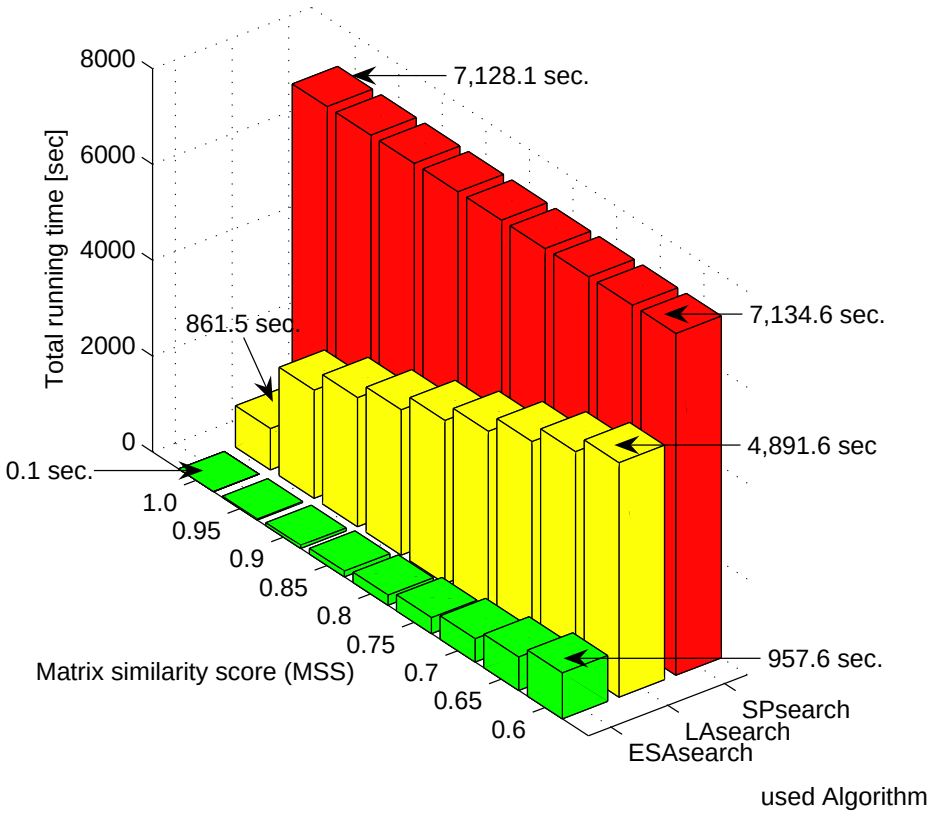


Figure 3.25: Experiment 8: Running times of *ESAsearch*, *LAsearch* and *SPsearch* when searching with 577 TRANSFAC PSSMs in the database of transcriptional start sites (DBTSS Rel. 4.0) containing 23,410 human and mouse promotor sequences with a total sequence length of 27MB. Measurements were performed for different matrix similarity score values (MSS), representing different levels of stringency.

(A) 577 TRANSFAC PSSMs in 27MB nucleotide sequences (DBTSS Rel 4.0):

	ESAsearchKb			
k	#th updates	#touched suffixes (%)	running time [sec]	speedup over *LAsearchKb*
1	24,115 (41.8;33.1)	43,883 (0.15)	20.7	65
10	54,391 (94.3;92.5)	111,252 (0.39)	46.5	36
100	244,261 (423.3;497.7)	227,208 (0.80)	93.7	23
1000	1,415,292 (2452.8;3,132.0)	474,184 (1.68)	210.8	13

	LAsearchKb	
k	#th updates	running time [sec]
1	10,249 (17.8;5.0)	1351.6
10	66,257 (114.8;36.2)	1710.5
100	506,606 (878.0;363.7)	2160.0
1000	3,451,915 (5,982.5;3,518.0)	2754.4

(B) 28,337 BLOCKS PSSMs in 4.3 MB protein sequences (PDB):

	ESAsearchKb			
k	#th updates	#touched suffixes (%)	running time [min]	speedup over *LAsearchKb*
1	522,918 (18.4;4.6)	1,609,021 (37.3)	452.6	1.36
10	1,783,023 (62.9;13.9)	1,956,327 (45.88)	588.9	1.26
100	6,628,970 (233.9;54.6)	2,079,215 (48.76)	649.0	1.33
1,000	11,789,765 (416.1;124.2)	2,149,793 (50.41)	723.9	1.31

	LAsearchKb	
k	#th updates	running time [min]
1	479,614 (16.9;3.8)	616.5
10	1,563,197 (55.1;9.6)	747.5
100	8,306,508 (293.1;66.8)	869
1,000	11,526,692 (406.8;118.6)	947

Table 3.3: Experiment 9: Measurements of the influence of different values of k on the number of threshold updates, number of touched suffixes, and the total running time for *ESAsearchKb* and *LAsearchKb*. We performed experiments on nucleotide PSSMs/sequences (see (A)) as well as on amino acid PSSMs/sequences (see (B)). We counted the total number of threshold updates for all PSSMs and calculated the mean and the standard deviation (values are given in brackets). Column *#touched suffixes* gives the number of suffixes on average (total number divided by number of used PSSMs) for which *ESAsearchKb* has to score at least one new character. The percentage value of all suffixes is given in brackets. For *ESAsearchKb*, the last column shows the speedup of *ESAsearchKb* over *LAsearchKb* for the same value of k. For algorithm *LAsearchKb* the number of touched suffixes is not shown, since *LAsearchKb* processes always the complete sequence, independent of the value of k.

(see Figure 3.22), alphabet reduction has a negative effect upto 4 times on the running time and hence using alphabet reduction in combination with algorithm *LAsearch* makes no sense in practice. This is due to the increased expected reading depth $m^*(th)$ for degraded alphabets, which is for *LAsearch* not counterbalanced by increased lcp-interval sizes, since no enhanced suffix array is used in this algorithm. Experiment 7 (see Figures 3.23 and 3.24) shows that the expected running time of *ESAsearch* is sublinear, whereas *LAsearch* runs in linear time. In Experiment 8, we compared the running times of *ESAsearch* and *LAsearch* with our own implementation of *SPsearch*. This experiment shows that the *SPsearch* algorithm, running in $\mathcal{O}(nm)$ time, although still widely used, is definitly inappropriate for larger PSSM matching tasks. In Experiment 9 we investigated the influences of different values of k on the number of threshold updates, number of touched suffixes when using *ESAsearchKb* and total running time (see Table 3.3). Here *ESAsearchKb* achieves a speedup of factor 13 (for $k = 1000$) upto 65 (for $k = 1$) over *LAsearchKb* on nucleotide PSSMs and a speedup of factor 1.3 on amino acid PSSMs. It has been shaped out in practice, that *ESAsearchKb* and *LAsearchKb* are especially useful when searching for transcription factor binding site motifs in large data sets, since it is difficult to specify a reasonable p-value or E-value cutoff for these short motifs of low significance. In a final experiment, we compared algorithm *LazyDistrib* with the DP-algorithm computing the complete distribution. *LazyDistrib* shows a speedup factor between 3 and 330 on our test set, depending on the stringency of the threshold (see Table 3.4).

We also note, that motivated by the first description of the *LazyDistrib* algorithm in [BSH$^+$04], in [MG06] the authors reimplemented *LazyDistrib* in the functional programming language Haskell, taking advantage of the built-in non strictness of the language. Whereas in our implementation in C some effort has to be spend on the simulation of laziness in an eager language, in Haskell this has not be adressed explicitly, since lazy evaluation is a built-in concept of the programming language. Accordingly, algorithm *LazyDistrib* can be formulated in Haskell on less than a page, while the C implementation consists of some hundred lines of code. Also remarkable, for an implementation in a functional programming language, is the reported speedup in [MG06]. The authors report for a test set containing a small fraction of PSSMs from the PRINTS database a speedup factor between 4.3 for a p-value of 10^{-10} and 172 for a p-value of 10^{-30} over the DP-algorithm computing the complete distribution. Though, this is in the same range as the speedup of our implementation of *LazyDistrib* with hand-coded laziness (see Table 3.4), the absolute running times differ significantly. E.g to compute a score threshold corresponding to a P-value cutoff of $\pi = 10^{-10}$ for $1,000$ PRINTS PSSMs, Malde and coworkers report a running time of 306 seconds using a 1.13 GHz system. The C implementation of *LazyDistrib* needs 485.3 seconds for all $11,411$ PRINTS PSSMs for the same p-value cutoff on a 900 MHz system.

3.11 PoSSuM software distribution

Our software tool *PoSSuMsearch* implements all algorithms and ideas presented in this work, namely *SPsearch*, *LAsearch*, *ESAsearch* and *LazyDistrib*. A user can search for PSSMs in enhanced suffix arrays built by `mkvtree` from the *Vmatch* package, as well as on plain sequence data in FASTA, GenBank, EMBL, or Swiss-Prot format. The search algorithm can be chosen from the command line.

p-value	simple DP	*LazyDistrib*	speedup factor
10^{-10}	1,486	485.8	3
10^{-20}	1,486	92.5	95
10^{-30}	1,486	8.9	166
10^{-40}	1,486	4.5	330

Table 3.4: Running times in seconds when computing score thresholds for all 11,411 PSSMs from the PRINTS database (Rel. 38), given different p-values. Running times given in this table are measurements performed with improved versions of the simple DP and *LazyDistrib* algorithms and thus are much lower than the times given in [BSH+04].

PSSMs are specified in a simple plain text format, where one file may contain multiple PSSMs. See Example in section A.4.1 on page 201. The alphabet a PSSM refers to, and alphabet character to PSSM column assignments can be specified on a per-PSSM basis for most flexible alphabet support. All implemented algorithms, except the k-best variants, support alphabet transformations. PSSMs can contain integer as well as floating point scores. To prevent rounding errors for integer based PSSMs, *PoSSuMsearch* uses integer arithmetics for these, resulting in an additional speedup on most CPU architectures. Searching on the reverse strand of nucleotide sequences is implemented by PSSM transformation according to Watson-Crick base pairing (see Definition 8 on page 23). Hence it is sufficient to build the enhanced suffix array for one strand only. This can then be used to search both strands.

The cutoff can be specified as p-value, E-value, MSS (matrix similarity score), or raw score threshold. If only the best matches with the highest scores need to be known, then *PoSSuMsearch* can be asked to report only the k highest scoring matches without even specifying an explicit cutoff. To do so, the search algorithms (*ESAsearchKb* and *LAsearchKb* variants) dynamically adapt the threshold during the search. When using p- or E-values, the score threshold is determined by either the new lazy dynamic programming algorithm (*LazyDistrib*), or read from file that stores the complete precalculated probability distribution. Background distributions can be specified arbitrarily by the user, or determined from a given sequence database. We provide a tool, *PoSSuMdist*, to generate a compressed file containing the complete precalculated probability distribution for a set of PSSMs.

PSSM matches can be sorted by specifying a list of sort keys, like p-value, match score, sequence number, and so on. The output formats of *PoSSuMsearch* print out all available information about a match, either in a human readable format, tab delimited, or in machine readable, XML-based CisML [HW04]. *PoSSuMsearch* as well as *PoSSuMdist* support multi-threading for a further reduction of running time on multi CPU machines. See Figures 3.26 and 3.27 for the additional speedup that can be achieved when facilitating multiple CPUs.

The *PoSSuM* software distribution includes the searching tool *PoSSuMsearch* itself, and additional tools to determine character frequencies from sequence data, for probability distribution calculation, and PSSM format converters for *TRANSFAC*, *BLOCKS*, *PRINTS*, and *EMATRIX* style PSSMs. See Table 3.5 for a list of included programs. Detailed descriptions of the programs included in *PoSSuM* software distribution, their command line interfaces, and used file- and output formats are given in the Appendix A.4 on page 199.

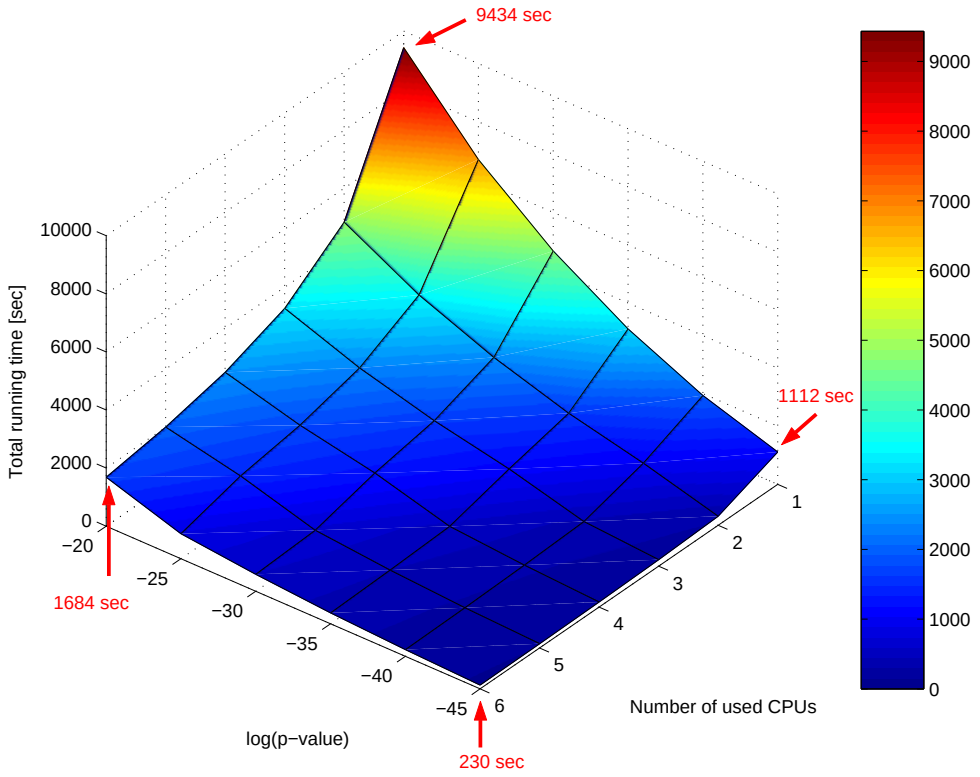


Figure 3.26: Scaling behaviour of the multithreaded version of *PoSSuMsearch* operating in *ESAsearch* mode. We measured the running time in seconds needed for searching with 24,291 protein PSSMs with an average length of 26.57 taken from the BLOCKS database (Rel. 13.0) in the RCSB protein data bank (PDB) containing 19,111 sequences with a total length of 4.3 MB. Measurements were performed for different p-value cut-offs and used numbers of CPUs.

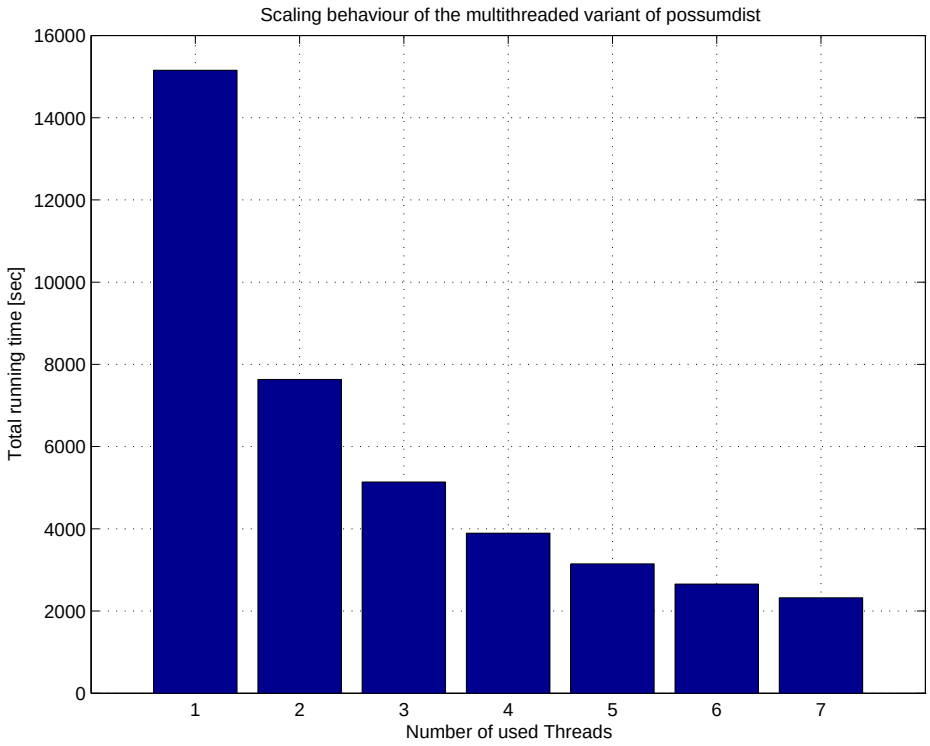


Figure 3.27: Scaling behaviour of the multithreaded variant of *PoSSuMdist*. We measured the time needed for the calculation of complete score distributions and their persistent, compressed storage. We used 28,333 protein PSSMs from the BLOCKS datbase (Rel. 14.1) with an average length of 26.3 using different numbers of CPUs.

Program	Description
PoSSuMsearch	Searching tool, that implements *ESAsearch*, *LAsearch*, *SPsearch*, *ESAsearchKb*, *LAsearchKb*, and *LazyDistrib* algorithms.
PoSSuMdist	Generates a compressed file containing the complete precalculated probability distribution for a set of PSSMs.
PoSSuMfreq	Utility program to calculate character frequencies from given sequence data, usable for *LazyDistrib* and *PoSSuMdist*.
transfac2gen	Converts PSSMs from TRANSFAC format into *PoSSuMsearch* compatible format.
prints2gen	Converts PSSMs from PRINTS format into *PoSSuMsearch* compatible format.
ematrix2gen	Converts PSSMs from EMATRIX format into *PoSSuMsearch* compatible format.
cdd2gen	Converts CDD and PsiBlast checkpoint file PSSMs into *PoSSuMsearch* format.
mkvtree	Program from the *Vmatch* package of S.Kurtz to build enhanced suffix arrays.

Table 3.5: Programs included in the *PoSSuM* software distribution.

3.12 Discussion and concluding remarks

We presented in this chapter a new non-heuristic algorithm, called *ESAsearch*, to efficiently find matches of PSSMs in large databases. Our approach preprocesses the search space, e.g., a complete genome or a set of protein sequences, and builds an enhanced suffix array that is stored on file. This allows the searching of a database with a PSSM in sublinear expected time. Our analysis of *ESAsearch* revealed sublinear runtime in the expected case, and linear runtime in the worst case for sequences not shorter than $|\mathcal{A}|^m + m - 1$, where m is the length of the PSSM and $\mathcal{A}$ a finite alphabet. The enhanced suffix array, on which the method is based, requires only $9n$ bytes. This is a space reduction of more than 45 percent compared to the $17n$ bytes implementation of [DNM00]. For a summarization of the time and space complexities of *ESAsearch*, *LAsearch*, and *SPsearch* see Table 3.6.

Since *ESAsearch* benefits from small alphabets, we presented a variant operating on sequences recoded according to a reduced alphabet. We also addressed the problem of non-comparable PSSM-scores by developing a method which allows the efficient computation of a matrix similarity threshold for a PSSM, given an E-value or a p-value. Our method is based on dynamic programming and, in contrast to other methods, it employs lazy evaluation of the dynamic programming matrix. For application scenarios, where it is difficult to specify a meaningful PSSM score cutoff, we developed variants of *ESAsearch* and *LAsearch*, that adjust dynamically the threshold th while searching and report the k- best matches of a PSSM without the need for the user to specify a cutoff value.

We evaluated algorithm *ESAsearch* with nucleotide PSSMs and with amino acid PSSMs. We performed various experiments in which, compared to the best previous methods, *ESAsearch* shows

speedups of a factor between 17 and 275 for nucleotide PSSMs, and speedups up to factor 1.8 for amino acid PSSMs. Comparisons with the most widely used programs even show speedups by a factor of at least 3.8. Alphabet reduction yields an additional speedup factor of 2 on amino acid sequences compared to results achieved with the 20 symbol standard alphabet. The lazy evaluation method is also much faster than previous methods, with speedups of a factor between 3 and 330. This new algorithm for accurate on-the-fly calculations of thresholds has the potential to replace formerly used approximation approaches.

Beyond the algorithmic contributions, we provide with the *PoSSuM* software distribution, a robust, well documented, and easy to use software package, implementing the ideas and algorithms presented in this chapter. The *PoSSuM* software distribution has already been successfully used in a large-scale study for the structural analysis of the core promoter in mammalian and plant genomes [FSD+05] and it constitutes the fundamental search engine for transcription factor binding site PSSMs in the CoryneRegNet software system for the reconstruction and comparison of transcriptional regulatory networks in prokaryotes [BBC+06, BRT06]. Further, *PoSSuMsearch* is integrated into the Genlight system [BMM+04] as a screening method for amino acid PSSMs from the PRINTS and BLOCKS databases.

Implementations	Time complexity	Space complexity	Comment
	Algorithm: *SPsearch*		
Matinspector, *MATCH*, *FingerPrintScan*, *Blocksearch*, *BLIMPS*, *PoSSuMsearch*	$\mathcal{O}(nm)$	$\mathcal{O}(n+m)$	$\mathcal{O}(nm)$ running time in the best-, average-, and worst case.
	Algorithm: *LAsearch*		
EMATRIX, *PoSSuMsearch*	$\mathcal{O}(kn)$	$\mathcal{O}(n+m)$	The worst case running time is $\mathcal{O}(nm)$, since in the worst case $k \in \mathcal{O}(m)$. In practice, k is expected to be much smaller than m, leading to considerable speedups. In the best case, exact one character of each subword of length m of S has to be scored leading to $\mathcal{O}(n+m)$ running time.
	Algorithm: *ESAsearch*		
PoSSuMsearch	$\mathcal{O}(n + n \cdot \max\{0, p\} + m)$	$\mathcal{O}(9n+m)$	The worst case running time is $\mathcal{O}(n+m)$, since $p \leq 0$ holds for any PSSM of length m and threshold on any text of length $n \geq \lvert\mathcal{A}\rvert^m + m - 1$. In practice, large numbers of suffixes can be skipped if the threshold is stringent enough, leading to a total running time *sublinear* in the size of the text, regardless of the relation between n and m.

Table 3.6: Summary of the time and space complexities of *SPsearch*, *LAsearch*, and *ESAsearch* when searching with a PSSM of length m in a text S of length n over alphabet $\mathcal{A}$.

4 PSSM family models for sequence family classification

4.1 Increasing the expressiveness of PSSM-based database searches

We have seen, that searching with PSSMs in combination with enhanced suffix arrays is very efficient and lead to algorithms with superior performance compared to existing methods. However, compared to other approximate motif description models, PSSMs have the severe drawback, that they are fixed length motifs, build from gapless (parts of) multiple alignments, that do not allow for possible gaps. Consequently, one single insertion or deletion in the sequence to be searched with a PSSM, can yield to a misleading overlooking (false negatives) of the motif represented by the PSSM. Hence, PSSMs are often only used to represent short highly conserved regions of nucleotide or amino acid sequences or regions that have a constant length, like transcription factor binding sites. Though, using only short PSSMs can increase the sensitivity in a database search, specificity decreases, since the significance of a PSSM match is correlated with its length (see Equation (3.9)). Accordingly, biological relevant matches of short PSSMs are difficult and sometimes impossible to distinguish from spurious ones and the explanatory power of a single short PSSM match is limited, in particular in larger data sets. This is a known problem e.g., in the field of transcription factor binding site prediction on whole genomes or chromosomes (see [RMV03]).

Alternatively, we could use a different motif model, e.g., one that allows for gaps, like Gribskov's PSSMs with position specific gap costs (see section 2.6 on page 36) or profile hidden Markov models (see section 2.7.2 on page 42). These types of models are less vulnerable to insertions and deletions caused by evolutionary mutation events and thus can be used to describe longer regions of sequences. On the downside, incorporation of gaps is computationally expensive and it is unclear how to efficiently use enhanced suffix arrays when allowing gaps. Further on, it is unknown, how to calculate exact statistics necessary to address the significance of matches of such models that incorporate gaps. They are based on approximations and empirically determined parameters [ABOH01, YH01].

Instead of using a completely different motif model, we propose to increase the power of PSSM based motif searches by using chains of multiple ordered PSSMs as a descriptor for a family of related sequences. The method, to be described, is a combination of the *ESAsearch* algorithm for fast PSSM matching with an efficient chaining algorithm incorporating ordering information of PSSMs and is in particular applicable for protein family classification and detection of distant homologies.

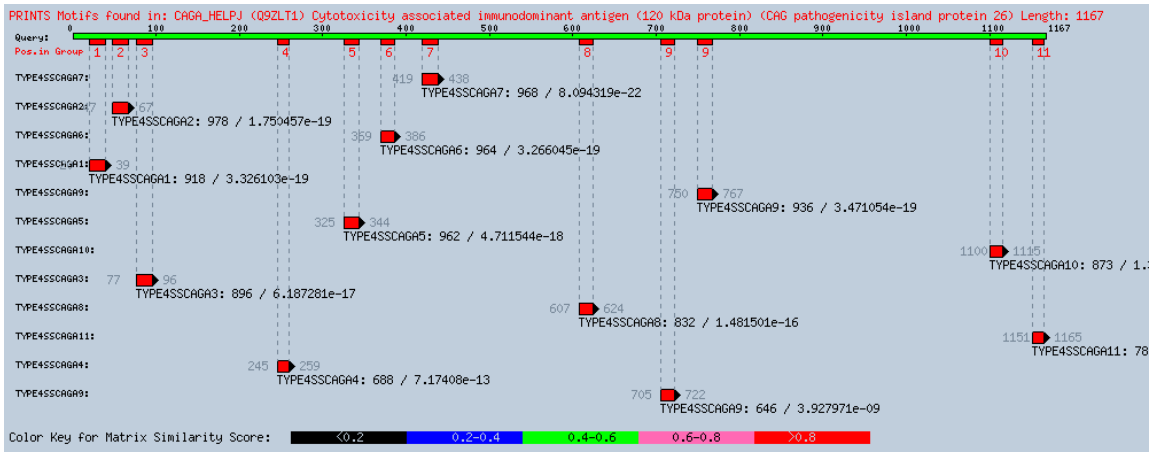


Figure 4.1: Conserved order of matches of a protein fingerprint describing the CagA exotoxin of the human pathogen *Helicobacter pylori* (PRINTS-ID: TYPE4SSCAGA), a well known virulence factor linked to the more severe forms of gastric ulcers and duodendal cancers caused by some strains of *H.pylori*. The PRINTS fingerprint representing the CagA family contains eleven PSSMs in a defined order. The group positions, shown in red in this figure, specify the order of PSSM motif occurrences in the alignment of the CagA family sequences from left to right. Figure taken from the Genlight system.

4.2 Using multiple ordered PSSMs for sequence classification

Sequence families can often be characterized by more than one motif derived from different conserved regions of the sequence. Reconsider that these motifs can be determined in a local multiple alignment of the family sequences or from gapless blocks, excised from a gap containing multiple alignment. We assume that the ordering of these motifs is evolutionary conserved and that family members will contain some or all the motifs, usually with a highly conserved ordering. For an example of order conservation see Figure 4.1. In this example the CagA exotoxin protein family is represented by eleven PSSMs determined from the multiple alignment of the family members. Observe that the order of PSSMs matches is conserved when they were screened versus a new member of the modeled family like the CAG_HELPJ protein sequence used in this example. This observation motivates an extension of our PSSM searching algorithms by incorporating ordering information of multiple PSSMs derived from a family of related sequences. Instead of single PSSM matches, our algorithm reports chains of PSSM matches that occur in a conserved order. Compared to approaches using a single motif only, with the incorporation of multiple PSSMs representing multiple conserved motifs of a sequence family in a specific order, the diagnostic power in a database search scenario like protein family classification, can be increased. See Figure 4.2 for the effect of using chains of multiple ordered PSSMs compared to single PSSMs. When reporting single high scoring PSSM matches only, we obtain multiple false positive hits, caused by the limited significance and explanatory power of short PSSMs and it remains difficult to confidently classify the sequence to a certain family based on these single PSSM matches (Figure 4.2 (A)). Hence, an effective search algorithm therefore has to take the cooccurrence and order of matching PSSMs of the same family into account and must

define a rankable score for the chain of matches. This increases the specificity of the database search and may reduce the number of false positive matches dramatically. In the example given in Figure 4.2 (B)) all false positive matches are eliminated. Here the top scoring chains are all true positives and exactly describe the five catalytic domains of the ARO multi domain protein of bakers yeast (*S.cerevisae*).

As apparent from Figure 4.2, by facilitating a set of multiple ordered PSSMs to describe a family of related sequences and by reporting of results with preserved ordering only, we can counterbalance the lack of specificity of short PSSMs. That is, we benefit of the increased sensitivity of short PSSMs, without losing specificity as it is the case when using single short PSSMs. Ordered sets of motifs adaptable for deviations of PSSMs, sometimes also called fingerprints, are available in form of alignment blocks in several database. The PRINTS [ACF$^+$00] and BLOCKS/BLOCKS+ databases [HGPH00, HP99] are two examples of large collections of such fingerprints representing specific protein families. BLOCKS for instance contains in its current release (Rel 14.3 April 2007) 29,068 PSSMs describing 5,900 protein families. In addition, ordered sets of alignment blocks which may serve as input for the construction of multiple ordered PSSMs can be easily excised from multiple alignments of related sequences. Hence, basically all protein family collections like Pfam [FMSB$^+$06], TIGRFAM [HSW03], Smart [LCP$^+$06], etc., which offer manually curated, high quality alignments of protein family members are applicable for deviation of multiple, ordered PSSMs constituting a descriptor for these families.

4.3 PSSM family models

From now on, we employ the term *PSSM family model* for an ordered set of PSSMs used as a descriptor for a family of related sequences. More precisely, let $A = A_1, A_2, \ldots, A_L$ be a sequence of L non-overlapping alignment blocks. These alignment blocks are excised from a multiple alignment and the sequence $A_1, A_2, \ldots, A_L$ reflects their order of occurrence in the alignment. See Figure 4.3 (A) for an example. We denote the start and end position of A_i, $i \in [1, L]$ in the multiple alignment by l_i and r_i, $l_i \leq r_i$, and define a binary ordering relation $\lhd$ on A, such that $A_i \lhd A_j$, $i, j \in [1, L]$ if and only if $r_i < l_j$. Then we define a PSSM family model as follows.

Definition 14 Let $A = A_1, A_2, \ldots, A_L$ be an ordered sequence of non-overlapping alignment blocks satisfying $A_i \lhd A_j$ for all $i, j \in [1, L] \wedge i < j$. A PSSM family model $\mathcal{M}$ of length L is a sequence of L PSSMs $\mathcal{M} = M_1, M_2, \ldots, M_L$ where M_i denotes the PSSM deviated from A_i, $i \in [1, L]$.

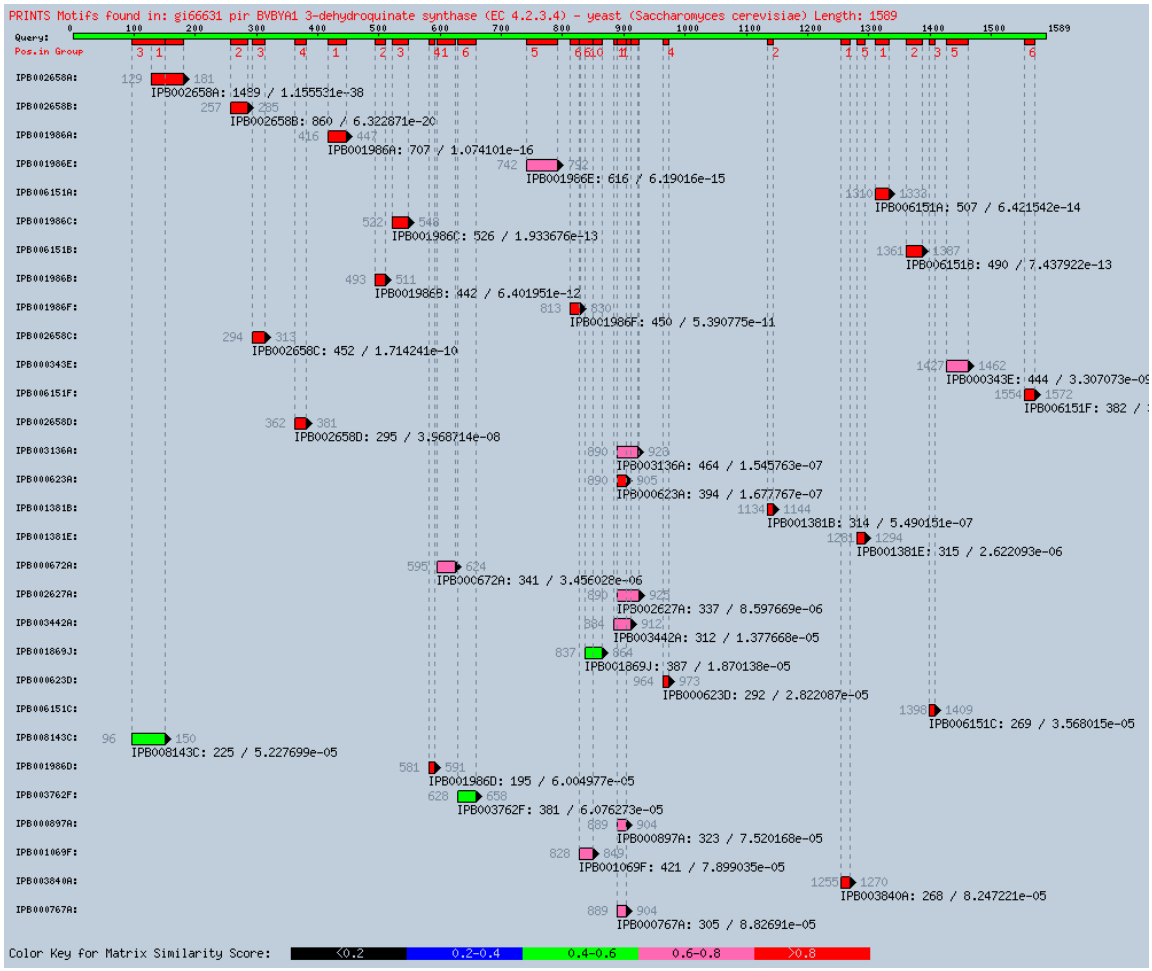
Let (l, i, j) denote a match to PSSM M_l, $l \in [1, L]$, $i \leq j$, of length $m_l = j - i + 1$ in a sequence S of length n. It holds $sc\,(S[i..j], M_l) \geq th_l$, where th_l is the threshold used for searching with M_l. Then the set containing all matches of all L PSSMs of $\mathcal{M}$ in S is defined as

$$MS_{\mathcal{M},S} := \{(l, i, j) \mid l \in [1, L] \wedge i, j \in [0, n-1] \wedge sc\,(S[i..j], M_l) \geq th_l\}. \tag{4.1}$$

For any tuple $h = (l, i, j) \in MS_{\mathcal{M},S}$ we employ the notation $h.l := l$, $h.i := i$, and $h.j := j$ to identify the components of h.

Definition 15 Let $\ll$ be a binary ordering relation on $MS_{\mathcal{M},S}$ such that $(l_r, i_r, j_r) \ll (l_s, i_s, j_s)$ if and only if $l_r < l_s$ and $j_r < i_s$. A match to a PSSM family model $\mathcal{M}$ of length L is a sequence of

(A) Reporting single PSSM matches



(B) Reporting of high-scoring chains only

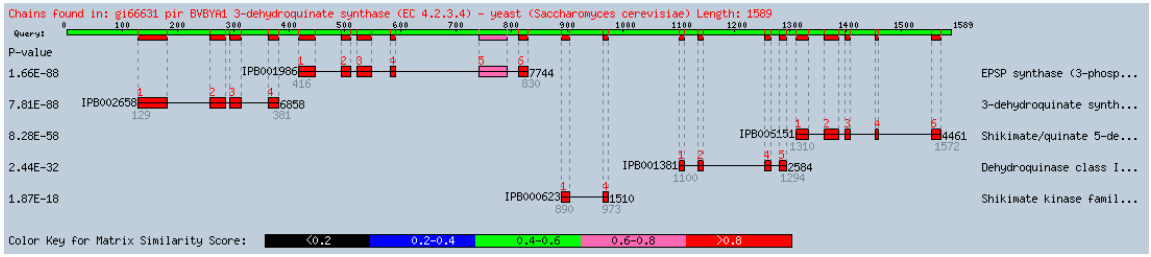


Figure 4.2: Screening of the ARO protein from yeast vs. the BLOCKS database. ARO is a pentafunctional protein that catalyzes five of seven steps in the chorismate biosynthesis pathway. Observe that when using single PSSMs, a lot of unrelated, spurious matches are found (A), whereas when using multiple ordered PSSMs and reporting high-scoring chains with preserved order only (B), the number of false positive matches is strongly reduced and each of the five domains of the ARO protein is correctly identified. For a detailed definition of high-scoring chains and chain scores, see corresponding text.

matches (or chain for short) $\mathcal{C}_{\mathcal{M},S} = h_1, h_2, \ldots, h_k$ such that $h_i \in MS_{\mathcal{M},S}$ for $1 \leq i \leq k$, and $k \leq L$, and $h_i \ll h_{i+1}$, and $h_i.l < h_{i+1}.l$ for $1 \leq i < k$. We call $\mathcal{C}_{\mathcal{M},S}$ a collinear, non-overlapping sequence, or chain, of length k and $h_i, i \in [1,k]$, a member or fragment of $\mathcal{C}_{\mathcal{M},S}$.

To incorporate a measure of match quality into PSSM family models, we compute the *chain score* of a chain. It is based on *fragment scores* assigned to each element of $MS_{\mathcal{M},S}$, expressing their quality. More precisely, let *fsc* be a function, that assigns a positive score to each fragment $h_i \in MS_{\mathcal{M},S}$, $1 \leq i \leq k$, in chain $\mathcal{C}_{\mathcal{M},S}$. This can be for example the p-value of h_i or its match score $sc\,(S[h_i.i..h_i.j], M_{h_i.l})$[1]. We define the *chain score* for chain $\mathcal{C}_{\mathcal{M},S} = h_1, h_2, \ldots, h_k$ as

$$csc\,(\mathcal{C}_{\mathcal{M},S}) := \sum_{i=1}^{k} fsc(h_i). \tag{4.2}$$

In the context of protein family classification, a sequence is searched with several PSSM family models, derived from multiple alignments representing different protein families. The classification into a certain family should be based on the quality of the *best* match of a sequence to the corresponding family model. Hence the first objective is to determine the *best* chain of PSSM matches in a sequence S for a given family model $\mathcal{M}$. This is stated in the following problem definition.

Definition 16 Given a PSSM family model $\mathcal{M} = M_1, M_2, \ldots M_L$ of length L, a sequence S of length n, L thresholds $th_1, th_2, \ldots, th_L$ used for searching with the PSSMs of $\mathcal{M}$ in S, and the set of PSSM matches $MS_{\mathcal{M},S}$ defined according to Equation (4.1) with their associated fragment scores, the *PSSM chaining problem* is to determine a chain $\mathcal{C}^*_{\mathcal{M},S}$ such that $csc\left(\mathcal{C}^*_{\mathcal{M},S}\right)$ is maximal among all possible chains. We call such a chain an *optimal* chain and denote its score with

$$csc^*{}_{\mathcal{M},S} := csc\left(\mathcal{C}^*_{\mathcal{M},S}\right) = \max\{csc\,(\mathcal{C}_{\mathcal{M},S}) \mid \mathcal{C}_{\mathcal{M},S} \text{ is a chain for } \mathcal{M} \text{ on } S\}. \tag{4.3}$$

With the definition of *optimal* chains and their chain scores we introduce a quantifiable, rankable criteria of match quality to our PSSM family model concept that allows to use PSSM family models for sequence classification. More precisely, let S be a sequence and $\mathcal{F} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_T\}$ be a collection of T PSSM family models, representing T distinct protein families. Further, let $csc^*{}_{\mathcal{F},S} := \max\{csc^*{}_{\mathcal{M}_i,S} \mid \mathcal{M}_i \in \mathcal{F}\}$ be the maximal score of all optimal chains in S over all family models in $\mathcal{F}$, then we classify S into the family represented by $\mathcal{M} \in \mathcal{F}$ if and only if $csc^*{}_{\mathcal{F},S} = csc^*{}_{\mathcal{M},S}$. That is, we classify the sequence under consideration into the family whose family model generates the highest scoring optimal chain. In practice it is often useful to employ a threshold constraint, like a minimal neccessary chain length, as a lower boundary for classification. That is, sequences not satisfying this constraint are rejected.

Since the PSSM chaining problem is a variation of the more general multi-dimensional fragment chaining problem (c.f. [AO03b, AO03a, AO05]) we solve the PSSM chaining problem by utilizing algorithms from this field. In the following we briefly reconsider the main ideas of existing solutions. We roughly follow with our description an introduction to two dimensional fragment chaining, given in [Kur05a] adapted to our concrete problem of finding an optimal match to a PSSM family model.

[1] Observe, that for a PSSM M that may generate non-positive match scores, i.e., $sc_{\min}(M) \leq 0$, these scores can be easily shifted to non-negative scores by adding $sc_{\min}(M)$.

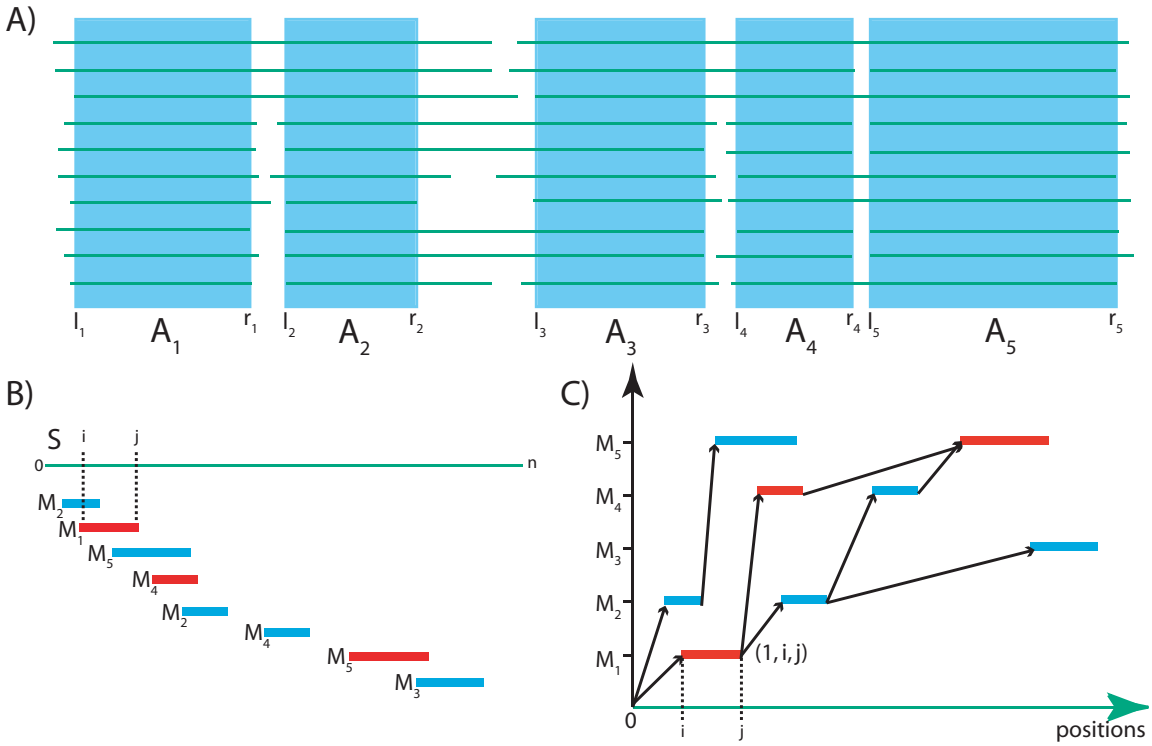


Figure 4.3: (A) Non-overlapping alignment blocks (cyan), excised from ungapped regions of a multiple alignment. Since $l_i \leq r_i < l_j \leq r_j$ for all $i, j \in [1, 5] \wedge i < j$ holds, $A = A_1, A_2, A_3, A_4, A_5$ is an ordered sequence of non overlapping alignment blocks suitable to construct a PSSM family model $\mathcal{M} = M_1, M_2, M_3, M_4, M_5$. (B) Matches of $M_i, i \in [1, 5]$, on sequence S, sorted in ascending order of their matching positions. (C) Graph based representation of the matches of $M_i, i \in [1, 5]$. An optimal chain of collinear non-overlapping matches is determined, by computing an optimal path in the directed, acyclic graph. Observe that not all edges in the graph are shown in this example and that the optimal chain (indicated here by their red marked members) is not necessarily the longest possible chain.

4.3.1 Computation of optimal PSSM chains

The problem of computing an optimal chain of PSSM matches can be solved with a well known graph-based algorithm described in [VA89], which is also used in the first version of the multiple genome alignment tool MGA [HKO02]. Adopted to the PSSM chaining problem, the ideas of the algorithm are as follows. Let $G = (V, E)$ be a directed, weighted, acyclic graph, where each vertex $v \in V$ represents an element of $MS_{\mathcal{M},S}$ and E is a set of weighted, directed edges. There is an edge pointing from vertex u to vertex v with score (weight) $fsc(v)$ if and only if $u \ll v$. See part (C) of Figure 4.3 for an example of such a graph. That is, there is an edge in G pointing from u to v if and only if u and v are collinear w.r.t. their matching positions in sequence S (see part (B) in Figure 4.3) and the positions of their corresponding PSSMs in the PSSM family model $\mathcal{M}$ (see part (A) in Figure 4.3). Hence all paths through G represent valid chains $\mathcal{C}_{\mathcal{M},S}$ according to Definition 15 and therefore matches to $\mathcal{M}$. An optimal chain corresponds to a path of maximal score. Since G is acyclic we can compute them as follows. Let $csc^*(v)$ be the maximum score of all chains ending with fragment v. As all fragment scores are positive, the following recurrence for the computation of $csc^*(v)$ holds:

$$csc^*(v) = fsc(v) + \max(\{0\} \cup \{csc^*(u) \,|\, u \ll v\}). \tag{4.4}$$

By utilizing dynamic programming, we can compute an optimal chain ${\mathcal{C}^*}_{\mathcal{M},S}$ and its chain score ${csc^*}_{\mathcal{M},S}$ in $\mathcal{O}(|V| + |E|)$ time. Let $K = |MS_{\mathcal{M},S}|$, then G contains K vertices connected by a maximum of $\sum_{i=1}^{K} i = \frac{K^2+K}{2}$ edges. Hence the time complexity can be rewritten as $\mathcal{O}\left(K^2\right)$ and the algorithm computing an optimal chain from K PSSM matches has a run time that is quadratic in the number of PSSM matches and takes $\mathcal{O}(K)$ space. In [AO05] the authors present an optimization that improves the $\mathcal{O}\left(K^2\right)$ time bound to $\mathcal{O}(K \cdot \log K)$ by considering the geometric nature of the problem. In the following, we briefly describe the main ideas of their algorithm. To compute $csc^*(v)$ according to Equation (4.4) we have to maximize the score over all $u \in MS_{\mathcal{M},S}$ satisfying $u \ll v$. Reconsider, that for any two PSSM matches $u, v \in MS_{\mathcal{M},S}$,

$$u = (l_r, i_r, j_r) \ll v = (l_s, i_s, j_s) \Leftrightarrow l_r < l_s \ \wedge \ j_r < i_s. \tag{4.5}$$

Furthermore in geometric terms, a PSSM match $v = (l_s, i_s, j_s) \in MS_{\mathcal{M},S}$ can be viewed as a line in a two dimensional space starting at position (i_s, l_s) and ending at position (j_s, l_s) (see Figure 4.3 part (C)). Hence to determine $csc^*(v)$ we have to maximize over all PSSM matches in the rectangle defined by the lower left corner point $(0, 0)$ and the upper right corner point (j_s, l_s), which is basically a two dimensional search problem. In [AO05] the authors show that this two dimensional search problem can be reduced to a one dimensional search problem by processing the elements of $MS_{\mathcal{M},S}$ in ascending order with respect to their matching positions in the sequence S or in the order of PSSMs as defined by $\mathcal{M}$. Hence, an efficient organization of the elements of $MS_{\mathcal{M},S}$ is advantageous and allows to reach the $\mathcal{O}(K \cdot \log K)$ time bound. This can be accomplished by using balanced binary search trees, for instance AVL- or Red-Black-Trees for the organization of $MS_{\mathcal{M},S}$. For a more detailed description of the algorithm see [AO03b, AO05].

Analysis of ESAsearch with fragment chaining

Given a PSSM family model $\mathcal{M} = M_1, M_2, \ldots, M_L$ consisting of L PSSMs of lengths $m_1, m_2, \ldots, m_L$ and a sequence of length n. Let $m^* = \max\{m_1, m_2, \ldots, m_L\}$ denote the length of the longest PSSM in $\mathcal{M}$ and K the total number of matches of all PSSMs M_i, $i \in [1, L]$. Then the time complexity for the combined algorithm composed of *ESAsearch* for PSSM match determination and the fragment chaining approach of [AO05] is $\mathcal{O}(L \cdot (n + m^* + K \cdot \log K))$. Although in practice K is much smaller than n, in the worst case $K \in \mathcal{O}(n)$ holds, leading to a total worst case time complexity of

$$\mathcal{O}(n + m^* + n \cdot \log n). \tag{4.6}$$

This time complexity holds for any PSSM of length m^* and threshold on any text of length $n \geq |\mathcal{A}|^{m^*} + m^* - 1$.

4.4 Integration of PSSM family models into PoSSuMsearch

To incorporate our concept of PSSM family models, we extended our search tool *PoSSuMsearch* by integrating the fragment chaining algorithm of [AO05] for chaining of matches to PSSMs.

In the first phase, *PoSSuMsearch* computes single PSSM matches for the PSSMs of a family model using algorithms *ESAsearch*, *LAsearch*, or even *SPsearch*, depending on the user's choice. In the second phase PSSM matches obtained in phase one and their ordering information are used to compute optimal chains of PSSM matches according to the order given in the family model. To formulate these orders, we augmented the PSSM file format to support grouping and ordering information for PSSMs enabling the description of PSSM family models. With these models *PoSSuMsearch* can compute and report for each sequence S and family model $\mathcal{M}$ the optimal chain $\mathcal{C}^*_{\mathcal{M},S}$ and its chain score $csc^*{}_{\mathcal{M},S}$. We added two modes of operation to *PoSSuMsearch*, namely *pssmsearch* and *seqclass* with the following semantics.

- Mode *pssmsearch* allows sequence classification based on a, typically small, library of PSSM family models. This mode requires a numeric argument k. Per family model the (up to) k best matching sequences are reported.
- Mode *seqclass* allows sequence classification based on a, typically large, library of PSSM family models. With user specified numeric argument k, per sequence the (up to) k best matching PSSM family models are reported.

Hence mode *pssmsearch* mimics the *modus operandi* of program *hmmsearch*, whereas mode *seqclass* is comparable to program *hmmpfam*. To further integrate PSSM family models into *PoSSuMsearch* seamlessly, we added additional filtering constraints like minimal (relative) chain length, output formats and sort keys operating on chains and there attributes instead of single PSSM matches. Graphical result visualization and a web user interface is available inside the Genlight system [BMM+04, BSS04] where *PoSSuMsearch* with PSSM match chaining is used as an integrated search method for the BLOCKS and PRINTS databases. For examples of visualized *PoSSuMsearch* results see Figure 4.2 on page 108.

4.5 Performance of PSSM family models for protein family classification

Detection of protein families in large databases is one of the principal research objectives in structural and functional genomics. Protein family classification can significantly contribute to the delineation of functional diversity of homologous proteins, the prediction of function based on domain architecture or the presence of sequence motifs as well as comparative genomics, providing valuable evolutionary insights.

To evaluate the suitability of *PoSSuMsearch* employing PSSM family models for fast and accurate protein family classification, we rigorously tested and validated our method in several database search scenarios. Therefore, we carried out extensive database searches with a large collection of protein families focusing on the ability to discriminate between homologs and non-homologs. For the experiments described in this chapter, we always used *PoSSuMsearch* operating in *pssmsearch* mode with algorithm *ESAsearch* for PSSM matching. For simplicity we refer to it as just *PoSSuMsearch* without mentioning each time that we use *ESAsearch* and chaining of PSSM matches.

To evaluate a database search method like *PoSSuMsearch*, we have to determine its sensitivity and specificity since the overall performance or quality of a method is always a combination of its sensitivity, also called coverage, and its specificity. One defines a method's sensitivity as its ability to detect as many true positive relationships (true members of the family described by the PSSM family model) and thus generate as few false negative results (erroneously missed members) as possible. Analogical, the specificity of a method is defined as its ability to select only sequences with a true relationship and thus to generate as few false positives (erroneously found members) as possible. We measured the performance of *PoSSuMsearch* in terms of sensitivity, specificity and running time and compared the results with a hidden Markov model based *state-of-the-art* approach. Database searches using profile hidden Markov model based approaches (c.f. [DEK98]) are yield to be very sensitive and specific [RV01]. Hence a performance evaluation assessing sensitivity and specificity of our PSSM family model based approach compared with a hidden Markov model based method is a meaningful and ambitious benchmark. We chose as a representative of methods employing HMMs the widely used *hmmsearch* program [Edd98] from Sean Eddy's *HMMER* package version 2.3.2. For the conducted experiments, we used the database search method evaluation framework PHASE4 [Reh02]. The method-independent *state of truth* essential for the expressiveness of the evaluation experiments is defined by sequences with known relationships taken from the SCOP (Structural Classification of Proteins) database [AHB+04] release 1.53. SCOP is a multi level hierarchy of protein sequences taken from the RCSB protein data bank (PDB). Thus, all sequences included in SCOP have a known tertiary structure and they can be classified into families based on their structural similarities, instead of sequence similarities. The classification hierarchy was constructed manually by expert knowledge and comparisons of structures and reflects both, structural and evolutionary relatedness of proteins. This unique feature makes the SCOP database a frequently used benchmark data set for database search methods [BCH98, KBH98]. Inside SCOP, families are organized into certain superfamilies, which again are members of certain folds. Families, superfamilies, and folds also constitute the three major levels of the hierarchy. For theses levels the following characteristics are assumed to be true:

Evaluation Scenario	Description
Experiment 1: **Very close relationship (family half one model)**	For each superfamily: For each family, half of its sequences are chosen as test sequences, and the remaining ones are chosen as training sequences. The sequences of the surrounding superfamily are ignored in the evaluation.
Experiment 2: **Close relationship (family halves one model)**	For each superfamily, half of the sequences of each of its families are chosen as training sequences and the remaining ones are chosen as test sequences.
Experiment 3: **Distant relationship (distant family one model)**	From a superfamily, each family in turn is chosen to provide the test sequences. The remaining families within that superfamily provide the training sequences.

Table 4.1: Evaluation scenarios used in the performed experiments to assess method sensitivity and specificity.

- **Family**
  Members of a family have a clear evolutionary relationship and hence a common or similar structure and function. Generally, this means that pairwise residue identities between proteins inside a family are 30% and greater. However, in some cases similar functions and structures provide definitive evidence of common descent in the absence of high sequence identity, e.g. many globins form a family although some members have sequence identities of less than 15%.

- **Superfamily**
  Members of a superfamily have probably a common evolutionary origin. Superfamilies contain proteins that have low pairwise sequence identities, but whose structural and functional features suggest that a common evolutionary origin and common function is probable. Examples for members that form a superfamily are actin, the ATPase domain of heat shock proteins, and hexakinase.

- **Fold**
  Proteins are defined as having a common fold if they have the same major secondary structures (alpha helices and beta sheets) in the same arrangement and with the same topological connections. Proteins placed together in the same fold category may not have a common evolutionary origin or function.

4.5.1 Employed data set and evaluation scenarios

To minimize the influence of redundancies, which are abundant in the SCOP database, on the results of our experiments we used the non-redundant *PDB90* subset[2] of SCOP (Rel. 1.53). This subset consists of a total of 4,861 amino acid sequences classified into 1,358 families and 853 superfamilies.

We performed three experiments with different evaluation scenarios to test our method's ability to detect (1) very close relationships, (2) close relationships, and (3) distant relationships. In these experiments we separated the SCOP sequences into different training- and test-sets. Table 4.1 and

[2]Subset of SCOP/PDB sequences with pairwise homology of less or equal 90 percent.

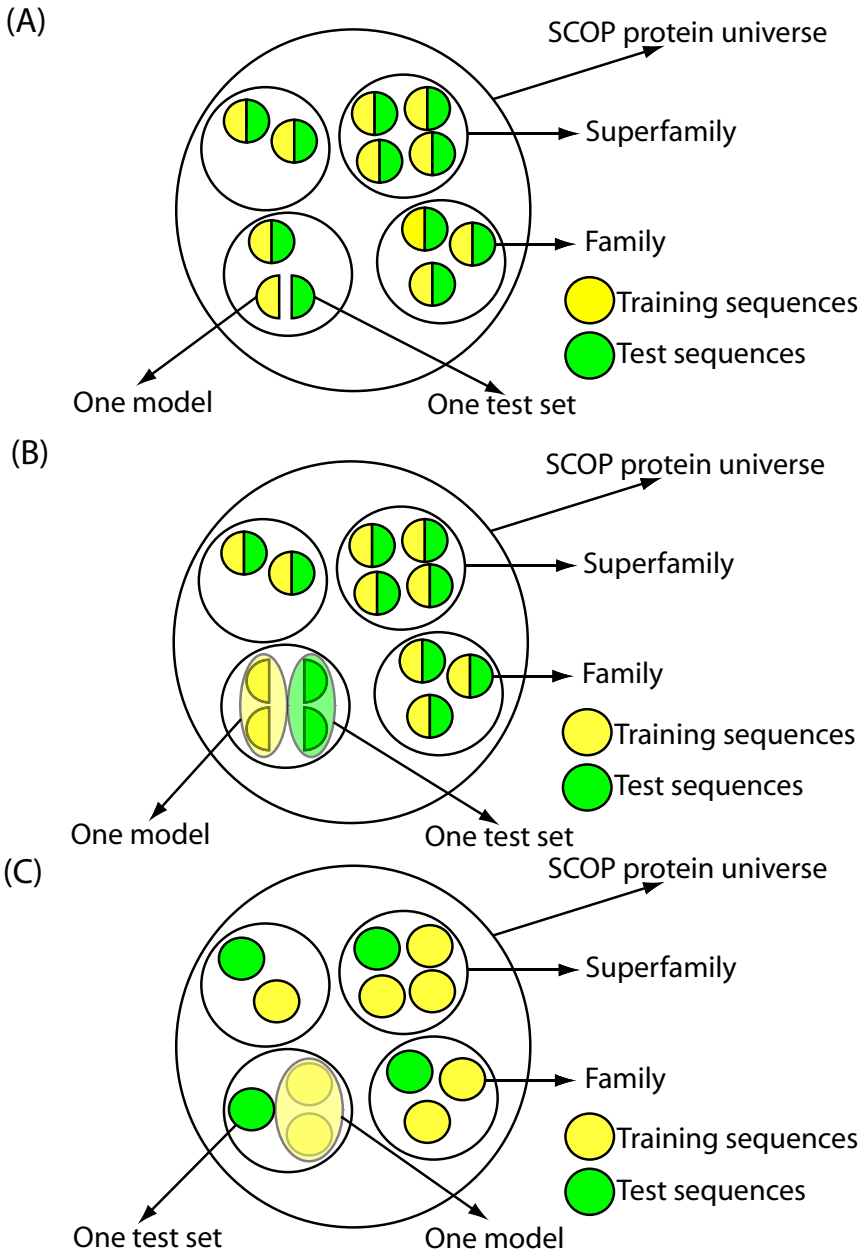


Figure 4.4: Construction of trainings- and test-sets for (A) very close relationships (family half one model scenario), (B) close relationships (family halves one model scenario), and (C) distant relationships (distant family one model scenario).

Figure 4.4 give more details on the construction process of training- and test-sets in the different evaluation scenarios. Since some superfamilies in SCOP contain only one family and some families consist only of very few member sequences, we employed the following constraints when selecting superfamilies and families for evaluation. Only superfamilies are selected that are comprised of at least two families. From these superfamilies, families were chosen to be test families, if both the family itself and the remainder of the superfamily contained at least 5 sequences each. Concrete numbers of resulting trainings/test sets in the different evaluation scenarios are given in the captions of the result figures shown in section 4.5.3 on the facing page. Note that training sequences are always ignored in the evaluation.

4.5.2 Model construction and scoring

From each training set we constructed a PSSM family model for use with *PoSSuMsearch* and a profile hidden Markov model for *hmmsearch* respectively. With these models, we subsequently search the sequences in the corresponding test set. Both model types are derived from a multiple alignment, which we compute from each training set using *CLUSTALW* [HTG+94] with default parameters. For deviation of calibrated profile hidden Markov models, we applied the programs *hmmbuild* and *hmmcalibrate* from the *HMMER* package. To construct PSSM family models, we first excised all ungapped blocks of length 6-12 from the multiple alignments retaining their order and deviated from the blocks PSSMs based on simple log-odds ratios according to Equation (2.7) in section 2.5.4 on page 28. For this, we estimated residue probabilities of observing a certain residue in a column of the alignment block from relative frequencies.

To score potential matches to a PSSM family model derived from a training set, we use *PoSSuMsearch* operating in *pssmsearch* mode. More precisely, let $\mathcal{M}$ be a PSSM family model of some training set and let *DB* denote the set of all sequences in *PDB90*, then we compute $csc^*_{\mathcal{M},S} = csc\,(\mathcal{C}^*_{\mathcal{M},S})$ for each $S \in DB$.

For the computation of high-scoring chains in our experiments, we define the fragment score for a match $h = (l, i, j)$ of PSSM $M_l, l \in [1, L]$, of length $m_l = j - i + 1$ in a sequence S of length n as

$$fsc(h) = \frac{-\ln(1 - (1 - \pi(h))^{n-m_l+1})}{\ln(n)}. \tag{4.7}$$

Here $\pi(h)$ denotes the probability for the event that PSSM M_l matches a random sequence of length m_l with at least score $sc\,(S[i..j], M_l)$ by chance. Observe that $\pi(h)$ can efficiently be determined in an exact manner with algorithm *LazyDistrib* described in section 3.8.5 on page 81. Thus, $(1-\pi(h))$ is the probability for the complementary event, that M_l does not match such a random sequence, and $(1-\pi(h))^{n-m_l+1}$ is the probability that there is no match in $n-m_l+1$ sequences of length m_l, corresponding to the number of possible different matching positions of M_l in a sequence of length n. Conversely, $1-(1-\pi(h))^{n-m_l+1}$ is the probability for the event, that there is at least one in $n-m_l+1$ random sequences of length m_l that matches M_l with a score of at least $sc\,(S[i..j], M_l)$. Since the fragment chaining algorithm computes chain scores by adding fragment scores (see Equation (4.2)), we take the logarithm to archive multiplication of probabilities $1-(1-\pi(h))^{n-m_l+1}$. These strongly depend on the sequence length n, therefore we divide them by $\ln(n)$ for compensation.

With fragment scores based on exact p-values according to Equation (4.7), we define the chain score of some chain $\mathcal{C}_{\mathcal{M},S} = h_1, h_2, \ldots, h_k$ of length k as

$$csc\,(\mathcal{C}) = \sum_{i=1}^{k} fsc(h_i). \tag{4.8}$$

Models constructed from training sets as described in section 4.5.1 on page 114 and Figure 4.4 on page 115, served as input for database searches with *hmmsearch* running in domain mode and *PoSSuMsearch*. For each model, we searched with both programs in the *PDB90* for sequences of the test-set corresponding to the used model. In these searches thresholds were set in a very relaxed way that resulted in reporting all sequences, irrespective of their score. More precisely, for *hmmsearch* we chose an E-value cutoff of 10 and for *PoSSuMsearch* we chose a single PSSM p-value cutoff of 0.1. Matches to a model were sorted in descending order of their achieved method specific score. As method specific scores we use for *PoSSuMsearch* given a family model $\mathcal{M}$ the best *chain score* ${csc^*}_{\mathcal{M},S}$ for a certain sequence S, computed according to Equation (4.8) and (4.7) respectively. Matches reported by *hmmsearch* were ranked by *sequence classification score*, the default result ranking score of *hmmsearch*. The *sequence classification score*, also called *overall model score*, is a log-odds score defined as

$$sc_{hmm} = \log_2 \left(\frac{\mathbb{P}[S|\lambda]}{\mathbb{P}[S|null]} \right). \tag{4.9}$$

Here $\mathbb{P}[S|\lambda]$ denotes the production probability that sequence S is generated by model λ and $\mathbb{P}[S|null]$ is the probability that S is generated by the null or background model expressing the probability of seeing S just by chance.

Finally, we obtain for each training set and model type (pHMM as well as PSSM family model) a list of matching sequences sorted in descending order of the method specific score and thus descending match quality. These lists of results are the foundation for the subsequent evaluation.

4.5.3 Performance evaluation and results

Assessment of sensitivity and specificity

To assess the sensitivity and specificity of our PSSM family model approach and to compare the classification accuracy with *hmmsearch* we process the lists of results computed by each method for each model top down, counting true- and false positive matches. This is feasible, since (true) family- and superfamily memberships are known from the SCOP classification. To provide an overall assessment of the methods' performances, we determined the percentage true positive value in all test sets (also called the *coverage*) for different counts of false positives and plotted the false positive counts versus the average percent coverage for the three different evaluation scenarios (see Figures 4.5, 4.6, and 4.7). This is a widely used method to measure the sensitivity and specificity of database search methods [ADRF04]. The resulting graphs (see Figures 4.5, 4.6, 4.7) describe how many percent true positives (y-axis) a method detects, if a certain number of false positives (x-axis) is accepted. In particular the percentage true positive value for 50 accepted false positives (FP50 value for short) is a commonly used value to characterize a database search method's performance in terms of sensitivity and specificity.

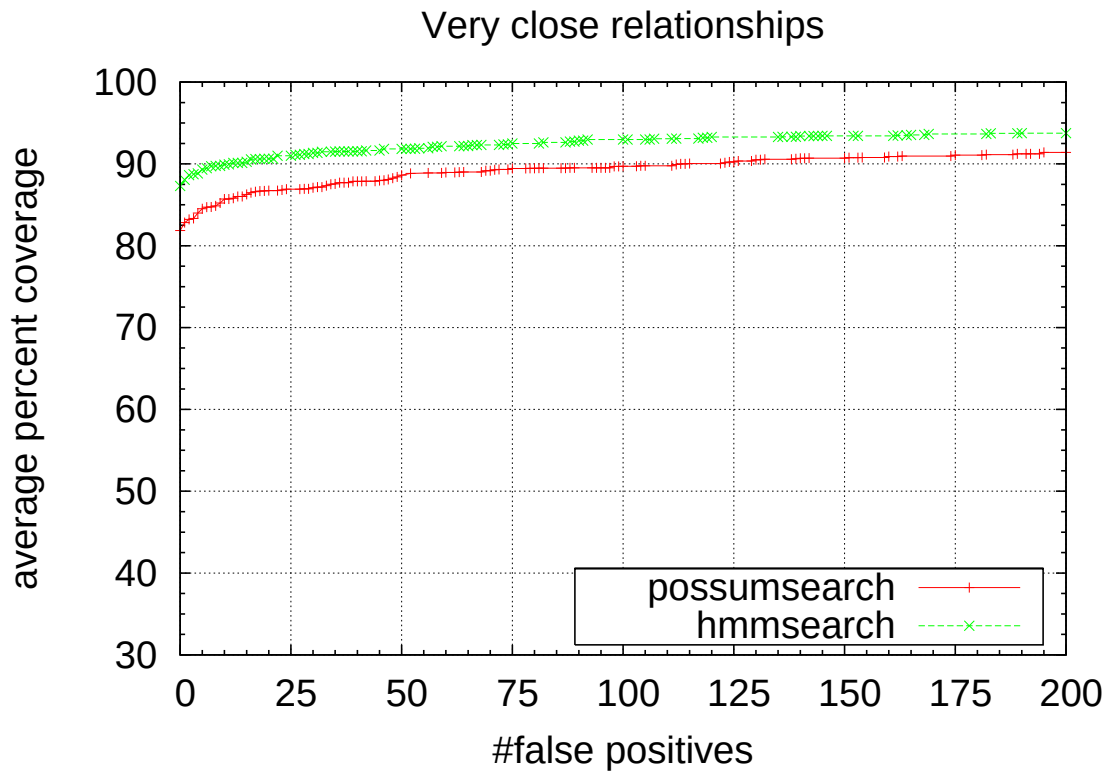


Figure 4.5: Experiment 1: Very close relationships (family half one model evaluation scenario). Classification performance of *PoSSuMsearch* and *hmmsearch* when detecting very closely related sequences. We used 258 models built from training sets representing 258 different protein families. The number of false positives is given on the x-axis, the y-axis gives the average percentage true positives. For details on training- and test-set generation see corresponding text.

In our experiments *PoSSuMsearch* reached an FP50 value of 88.6% when applied to very closely related proteins (Experiment 1, see Figure 4.5), 79.8% for closely related sequences (Experiment 2, see Figure 4.6), and 45.3% for distantly related sequences (Experiment 3, see Figure 4.7). For *hmmsearch* we achieved FP50 values reaching from 91.8% (very close relationships, see Figure 4.5 over 84.2% (close relationships, see Figure 4.6) down to 46.9% (distant relationships, see Figure 4.7). For a summary of detection rates for different numbers of allowed false positives see Table 4.2.

Running time and scalability

In a fourth experiment, we measured the running times and scaling behavior of *PoSSuMsearch* using PSSM family models and compared them to the hidden Markov model based *hmmsearch* program. To analyze the fraction of the total running time spent for chaining of PSSM matches, we also measured the running time of *PoSSuMsearch* without chaining for the same experimental setup.

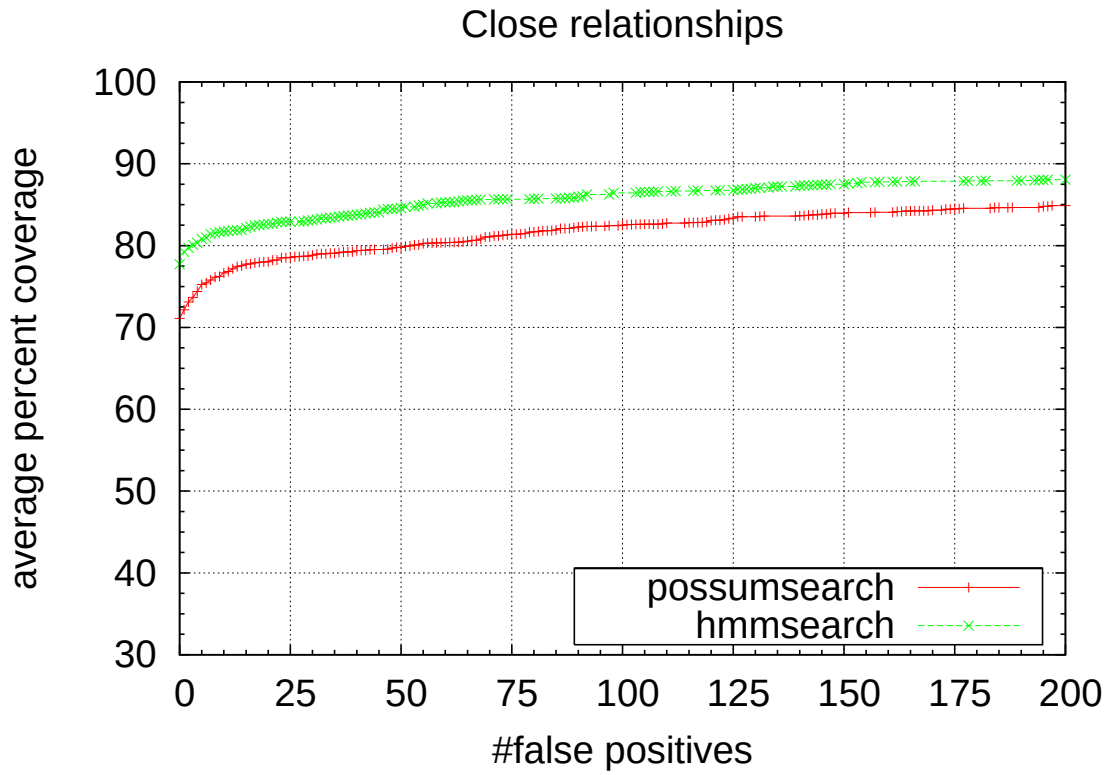


Figure 4.6: Experiment 2: Close relationships (family halves one model evaluation scenario). Classification performance of *PoSSuMsearch* and *hmmsearch* when detecting closely related sequences. We used 179 models made from the trainings sets in this experiment. For details on training- and test-set generation see corresponding text.

Evaluation scenario	*PoSSuMsearch* using PSSM family models			*hmmsearch*		
	FP0	FP25	FP50	FP0	FP25	FP50
Very close relationships	81.8%	86.9%	88.6%	87.3%	91.0%	91.8%
Close relationships	71.1%	78.6%	79.8%	77.7%	82.9%	84.2%
Distant relationships	35.0%	41.7%	45.3%	36.1%	43.9%	46.9%

Table 4.2: Average true positive detection rates of *PoSSuMsearch* using PSSM family models and *hmmsearch* for different numbers of allowed false positives (FP).

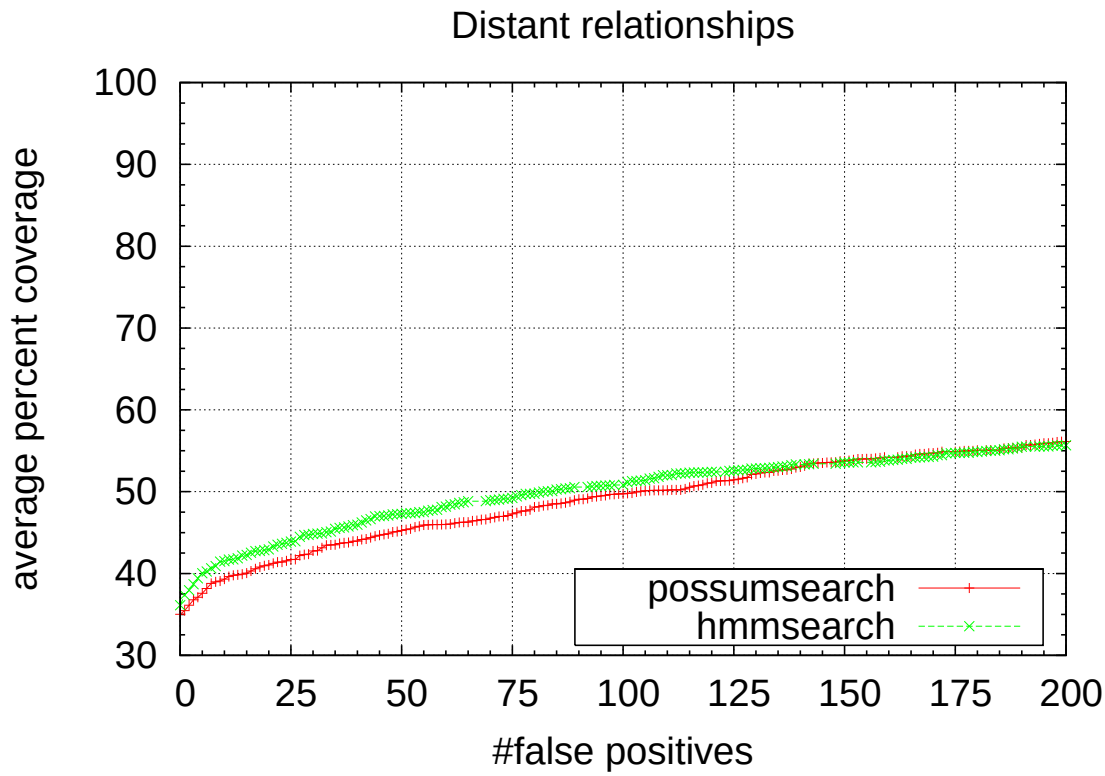


Figure 4.7: Experiment 3: Distant relationships (distant family one evaluation scenario). Classification performance of *PoSSuMsearch* and *hmmsearch* when detecting distantly related sequences. We used 320 models in this experiment. For details on training- and test-set generation see corresponding text.

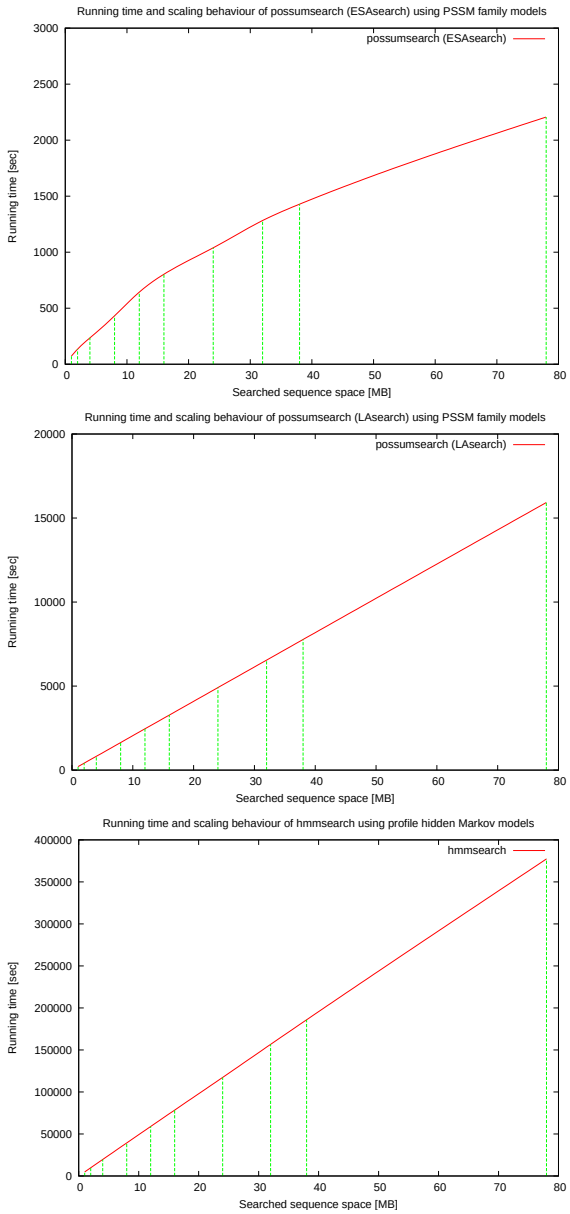


Figure 4.8: Experiment 4: Running times in seconds and scaling behavior of *PoSSuMsearch* operating in *ESAsearch* (top), *LAsearch* mode (center) and *hmmsearch* (bottom) when searching with 100 PSSM family models and profile hidden Markov models respectively, representing the first 100 protein families in *PFAM* on subsets of Swiss-Prot of different sizes.

For our experiment, we constructed PSSM family models for the first 100 protein families listed in Pfam (Rel. 21.0) [FMSB+06]. We excised alignment blocks of length 5-8 from the Pfam-seed alignments of these families and deviated PSSMs as described in section 4.5.2. This resulted in 100 PSSM family models consisting of 2,038 single PSSMs with an average length of 7.8. Along with the construction of PSSM family models we generated from the same seed alignments 100 pHMMs using *hmmbuild*. That is, we obtained 100 PSSM family models and pHMMs describing the first 100 Pfam protein families. We applied these models to *PoSSuMsearch* and *hmmsearch* respectively and measured the running time needed for searching on subsets of different sizes of the Swiss-Prot database (UniProtKB/Swiss-Prot Rel. 49.2) containing 212,425 amino acid sequences with a total sequence length of 78MB. For *hmmsearch* we used a moderately chosen E-value cutoff of 10^{-5}. Reconsider, that the running time of *PoSSuMsearch* depends on the stringency of the used cutoff, in particular when using algorithms *ESAsearch* or *LAsearch*. Hence for a fair comparison of running times of both methods, the cutoff for *PoSSuMsearch* has to be adjusted appropriately. Regrettably, E-values or p-values of different database search methods are in the majority of cases not comparable. In case of *PoSSuMsearch* with fragment chaining, this remains even more difficult due to the lack of accurate statistics for high-scoring PSSM chains[3]. However, manual inspection of results obtained for different levels of stringency revealed that for the majority of tested families a single PSSM p-value cutoff of $\pi = 10^{-4}$ underestimates the level of stringency of *hmmsearch* using an E-value cutoff of 10^{-5}. That is, *PoSSuMsearch* operates less stringent than *hmmsearch* and is not favored in terms of running time by operating on a higher level of stringency. Hence, we chose for our benchmark experiments a p-value cutoff of $\pi = 10^{-4}$ for searching with PSSM family models using *PoSSuMsearch*.

Measurements were performed on a 8 CPU Sun UltraSparc III computer with a CPU clock speed of 900MHz and 64GB main memory (using only one CPU and a small fraction of the memory). We measured the running times for *hmmsearch* and *PoSSuMsearch* operating in *ESAsearch* mode as well as in *LAsearch* mode (see Figure 4.8). Both *PoSSuMsearch* variants employ the fast chaining algorithm of [AO05] on the obtained PSSM matches to compute for each PSSM family model high-scoring chains. In this experiment *PoSSuMsearch* performed very well. Running times were in the range between 73 seconds for a 1MB subset of Swiss-Prot up to $2.2 \cdot 10^3$ seconds for the whole 78MB when employing *PoSSuMsearch* operating in *ESAsearch* mode and between 73.1 seconds (1MB) and $1.59 \cdot 10^4$ seconds (78MB) when using *LAsearch*. To accomplish the same task with *hmmsearch* utilizing 100 pHMMs built from the same seed alignments it took between $4.89 \cdot 10^3$ seconds (1MB) and $3.82 \cdot 10^5$ seconds (78MB) (see Figure 4.8). That is, *PoSSuMsearch* applying *ESAsearch* achieved speedup factors between 67 and 171 over *hmmsearch* and between 2.9 and 7.2 over *PoSSuMsearch* operating in *LAsearch* mode.

In our experiment *hmmsearch* shows a running time linear in the size of the searched sequence space (see Figure 4.8). This is an expected behavior induced by the applied *Forward* algorithm (see Equation (2.40)). By contrast, *PoSSuMsearch* operating in *ESAsearch* mode and employing chaining on the obtained PSSM matches shows clearly sublinear running time. We further found, that the overall running time is dominated by the time that algorithm *ESAsearch* needs to find matches for the PSSMs belonging to a family model. For example from the 2,178 seconds needed

[3] We will address the problem of score statistics of high-scoring PSSM chains in section 4.5.4 on the next page.

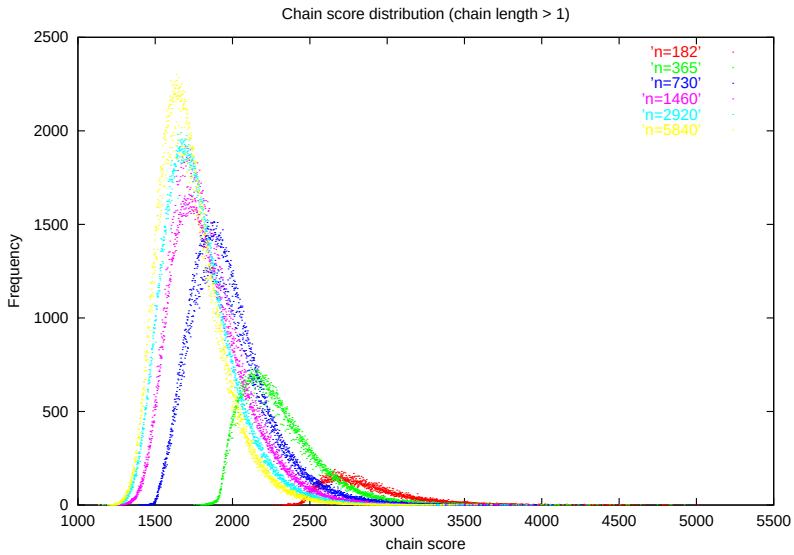


Figure 4.9: Histograms of $csc^*_{\mathcal{F},S}$ on 10^6 random sequences of constant lengths $n \in \{182, 365, 730, 1460, 2920, 5840\}$. The used collection of PSSM family models $\mathcal{F}$ contains 5,732 family models taken from BLOCKS Rel. 14.1.

by *PoSSuMsearch* to apply the 100 PSSM family models to the whole Swiss-Prot only 325 seconds (14.9%) were spent on chaining of PSSM matches.

4.5.4 The significance of PSSM chain scores

Although chain scores computed according to Equation (4.8) and (4.7) abstract from the underlying PSSM raw scores by using p-values and give a good rankable score, it is preferable to have a p-value or E-value as a measure of significance for a chain of PSSM matches. Such a p-value corresponding to a chain score $csc\,(\mathcal{C}_{\mathcal{M},S})$ should express the probability of obtaining a match to the PSSM family model $\mathcal{M}$ of at least score $csc\,(\mathcal{C}_{\mathcal{M},S})$ in a random sequence. This would allow more meaningful user specified p-value or E-value cutoffs instead of raw chain score cutoffs.

To define a meaningful p-value and hence to assess the significance of PSSM chain scores, we have to compute or at least approximate properly the chain score distribution. In literature very little is known about approximations of combined score distributions like our chain scores. In [BG98b] and [BG98a] the authors propose an intuitive method, implemented in the search tool *MAST* (Motif Alignment and Search Tool), for combining sources of evidence (matches of multiple motifs characterizing a sequence family) that yields a p-value for the complete evidence (membership of a sequence to this family). They use the product of p-values of single motif matches to derive a combined p-value. This is basically similar to our definition of chain scores according to Equation (4.8) and (4.7) except that we take the sequence length n into account and add logarithms of probabilities instead of multiplying probabilities. However, in contrast to our PSSM family models, the method of [BG98b] does not take the order of match occurrences into account and it reveals unclear how this

additional constraint influences the distribution of chain scores and the accuracy of the combined p-values[4].

To get an idea of the shape of this distribution, we sampled scores of high scoring chains on a large data set of random data. In this experiment, we used the chain score function as defined in Equation (4.8) and (4.7). To analyze potential dependencies of the chain score on the length of the matched sequence, we generated sets of 10^6 random sequences each, for different sequence lengths. As lengths we chose multiples of the average sequence length in Swiss-Prot determined as $n_{avg} = 365$, namely 182, 365, 730, 1460, 2920, and 5840, resembling the length spectrum of proteins. In these random data sets we retained the relative amino acid frequencies of Swiss-Prot. We searched with 5,732 PSSM family models ($|\mathcal{F}| = 5,732$) taken from the BLOCKS database (Rel. 14.1) consisting of 28,333 single PSSMs with an average length of 26.3 on our data sets of random sequences using *PoSSuMsearch* operating in *seqclass* mode for $k = 1$ and a relaxed p-value cutoff for a single PSSM match of 10^{-2}. That is, we computed the chain score of a match to a PSSM family model and tabulated for each of our random sequences S score $csc^*{}_{\mathcal{F},S}$, the score of the highest scoring chain of all family models. For these scores we calculated the distribution. See Figure 4.9 for the results of our sampling experiments. Although we already tried to compensate for dependencies of chain scores on the length n of the matched sequence by division by $\ln(n)$ (see Equation (4.7)), such an dependency still exists. This is in particular apparent in the histogram shown in Figure 4.10. For this Figure we sampled chain scores $csc^*{}_{\mathcal{M},S}$ for two PSSM family models describing two TIGRFAM protein families and containing a different number of PSSMs of length 6 to 10 on two sets of 10^6 random sequences of length 184 and 1472 respectively . We notice that the distribution of high chain scores depends on the sequence length and it is much more likely to achieve a high chain score in a longer sequence than in a shorter one. Also the number of PSSMs in a family model seems to influence the chain score, such that models consisting of a higher number of single PSSMs incline to achieve higher chain scores. For an example see the blue and red histogram in Figure 4.10. A more detailed analysis of the distribution of high PSSM chain scores (see Figure 4.11) revealed a further interesting aspect. As indicated by the X-Y plots in Figure 4.11, distributions of high PSSM chain scores derived from a single PSSM family model can be approximated quite well with an extreme value distribution. This may lead in the future, if an acceptable length normalization of chain scores can be found, to an approximation of the chain score distribution on a per model basis and in turn to reliable statistics with p-values and E-values corresponding to chain scores.

4.6 Accelerating HMM based database searches with PSSM family models

Profile hidden Markov models (pHMMs) are currently the most popular modeling concept for protein families. They provide very sensitive family descriptors, and sequence database searching with models from major pHMM collections has become a standard task in today's sequence analyses and genome annotation pipelines. On the downside, database searching for pHMMs with programs like *hmmsearch* or *hmmpfam* is computationally expensive. The application of the programs to com-

[4]Interestingly, incorporation of match order is mentioned as a promising constraint to increase specificity in the outlook of [BG98b].

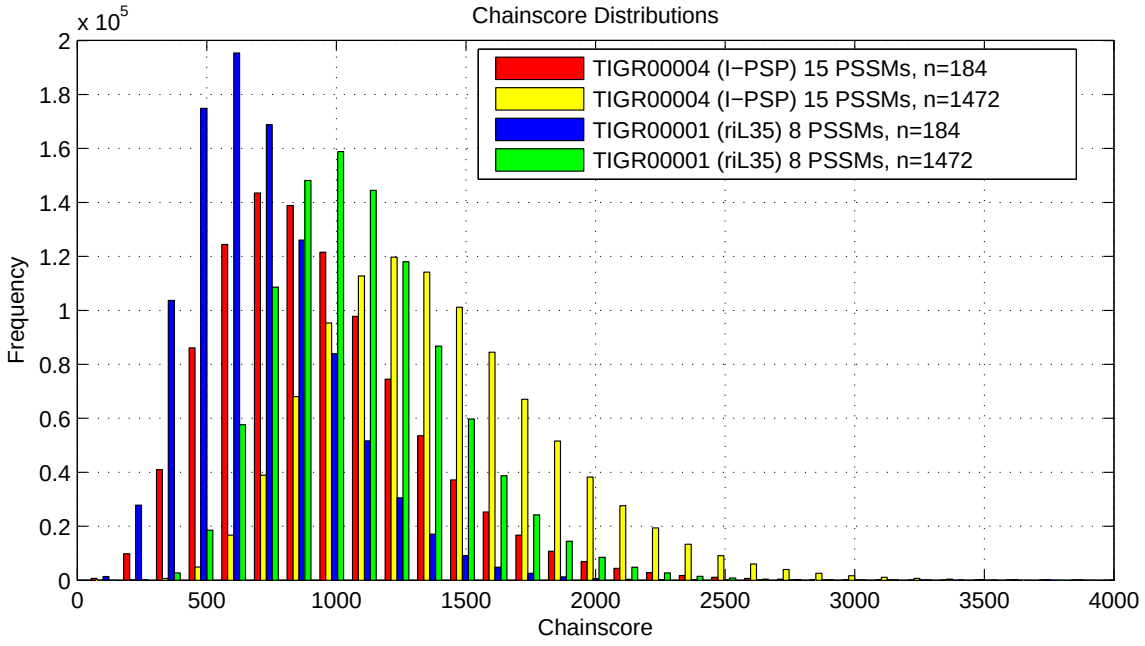


Figure 4.10: Histogram of $csc^*_{\mathcal{M},S}$ for two PSSM family models containing a different number of single PSSMs on 10^6 random sequences of lengths 184 and 1472. The two investigated PSSM family models describe TIGRFAM protein families TIGR00001 (modeled with 8 single PSSMs) and TIGR00004 (modeled with 15 single PSSMs).

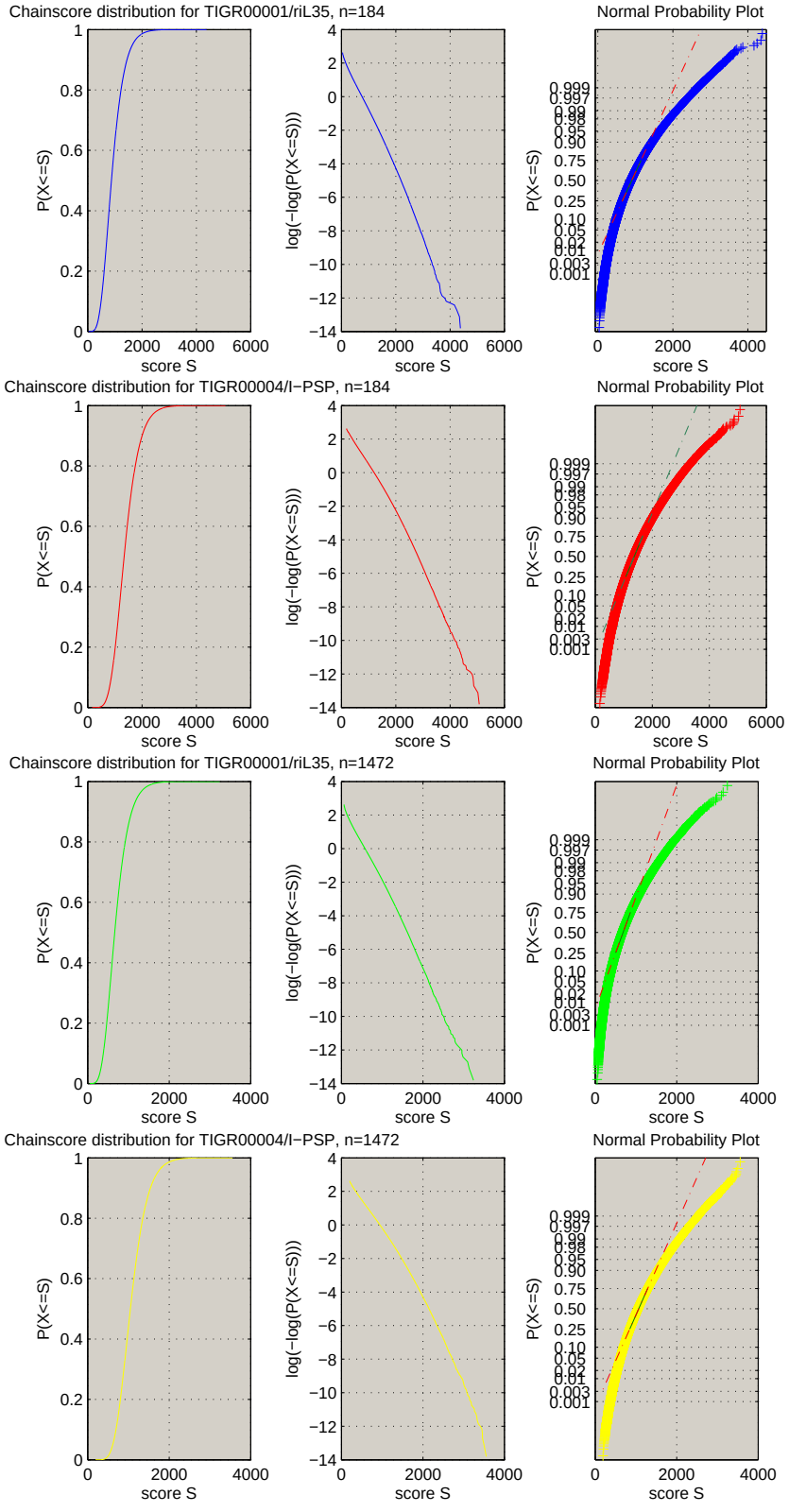


Figure 4.11: Cumulative chain score distributions, X-Y plots, and normal probability plots of PSSM family models for two TIGRFAM families (TIGR00001 and TIGR00004) for two different sequence lengths 184 and 1472. Color assignments are the same as in Figure 4.10.

plete proteomes, or even whole protein databases like Swiss-Prot or UniProtKB/TrEMBL, demands massive amounts of compute resources or highly specialized hardware [Tim06].

We propose a new method to speed up *hmmsearch*. Our approach employs the simpler PSSM family models and fast PSSM matching using algorithm *ESAsearch* to filter the search space for subsequent database searches with pHMMs corresponding to this families.

4.6.1 Model specific trusted- and noise cutoffs

Accompanied to the traditional user defined E-value cutoffs to control the significance level of a database search, major protein family databases use additional cutoffs to judge between true positive and false positive matches. For instance pHMMs of protein families from the Pfam or TIGRFAM databases contain additional *trusted-* and *noise-cutoffs*. These cutoffs, set with expert knowledge of the modeled family by the database curators, are used by programs from the *HMMER* package as cutoff values for the *sequence classification score* (see Equation (4.9)) for searches on a defined significance level. The *trusted cutoff* is the lowest score for sequences included in the protein family described by the model. Hence, it is assumed as a lower score boundary for true members of the family and is therefore often used in automatic annotation pipelines of genome annotation systems. Contrary, the *noise cutoff* is the highest score known so far of a sequence not belonging to the family. For this reason it can be seen as an upper score boundary for sequences assumed not to belonging to the family. This cutoff is often used for manual searches with increased sensitivity. The range between both cutoffs marks a gray zone in which the obtained matches require manual inspection. Using predefined *trusted-* or *noise cutoffs* recorded in the model entries, searches with different levels of stringency and in contrast to E-value cutoffs independent from the size of the searched sequence space can be performed.

In the following, we demonstrate that even in the absence of accurate statistics and significance values for PSSM chain scores, we can map *trusted* and *noise cutoffs* to single *PoSSuMsearch* PSSM p-value cutoffs, allowing *PoSSuMsearch* and *hmmsearch* to operate on a similar level of sensitivity. This permits to use *PoSSuMsearch* as a pre-filter for search space reduction for the compute intensive hidden Markov model based *hmmsearch*. The intention behind this filtering and search space reduction approach with PSSM family models is an expected reduction of overall running time of the combined approach consisting of *PoSSuMsearch* and subsequent *hmmsearch* over direct *hmmsearch*. To achieve this, we propose the subsequently described procedure.

4.6.2 PSfamSearch: Search space reduction with PSSM family models

We start by searching with a pHMM representing a protein family in a large protein database like Swiss-Prot using *hmmsearch* with the model's *trusted cutoffs* and tabulate all matching sequences. From the seed alignment of the employed pHMM we construct a PSSM family model as described in section 4.5.2 and use this family model to iteratively search Swiss-Prot using *PoSSuMsearch*. In each iteration we relax the p-value cutoff until we find all sequences also detected by *hmmsearch* using the models *trusted cutoff (TC)* and *noise cutoff (NC)* respectively. With this procedure we determine p-value cutoffs denoted by π_{TC} and π_{NC} corresponding to the pHMMs *trusted cutoff* and *noise cutoff* in terms of sensitivity. That is, we operate with *PoSSuMsearch* and our calibrated PSSM

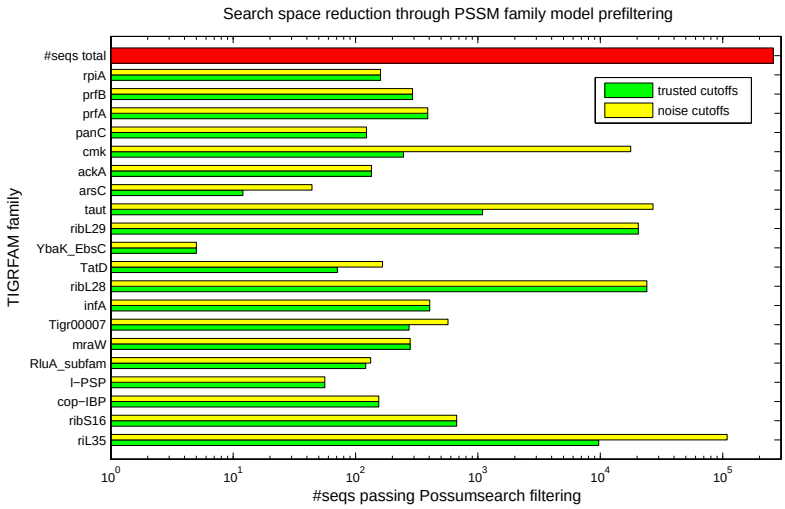


Figure 4.12: Search space reduction through PSSM family model pre-filtering. We measured the number of sequences passing pre-filtering of the search space with *PoSSuMsearch* using PSSM family models (x-axis, logscale). p-value cutoffs are adjusted to find at least the same matches as *hmmsearch* using *trusted-* and *noise cutoffs* for the first 20 protein families of the TIGRFAM database (Rel. 6.0). The red bar shows the total number of sequences in the used Swiss-Prot release 51.7 (259,034 protein sequences with a total length of $\sim$ 122MB) needed to be searched by direct *hmmsearch* without filtering.

family model on the same level of sensitivity as *hmmsearch* employing the pHMM, but with possibly reduced specificity. Observe that the set of matching sequences detected by *PoSSuMsearch* using cutoff π_{TC} or π_{NC} may be a super-set of the set of sequences detected by *hmmsearch* employing the pHMMs *trusted-* and *noise cutoff*. However, since we are interested in using PSSM family models searched with *PoSSuMsearch* as a pre-filter for search space reduction for *hmmsearch*, sensitivity is more important than specificity. Once π_{TC} and π_{NC} are computed on a large protein database like Swiss-Prot, they can be stored together with the PSSM family model on file for reuse. That is, for further searches with *hmmsearch* using the model's *trusted-* or *noise cutoff* we can use *PoSSuMsearch* using cutoff π_{TC} or π_{NC} as a filter and apply the compute intensive *hmmsearch* only on sequences that contain matching chains to the PSSM family model. Sequences that contain no matching chains are thus filtered out. Since sequences containing matching chains constitute only a small fraction of all sequences to be searched and since *PoSSuMsearch* is much faster than *hmmsearch*, we expect a reduced overall running time.

From now on we use the term *PSfamSearch* to denote the combined approach consisting of *PoSSuMsearch* using PSSM family models for pre-filtering and subsequent application of *hmmsearch* on the filtered sequence set.

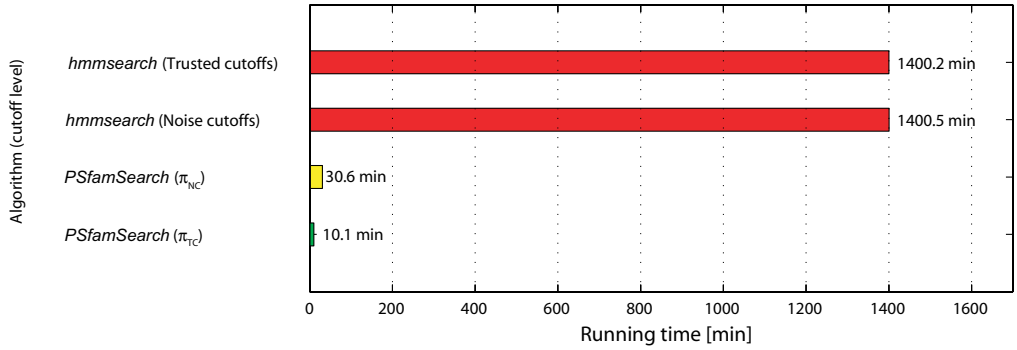


Figure 4.13: Running time reduction by PSSM family model based pre-filtering. We measured the total running time in minutes needed to search with models representing the first 20 protein families in TIGRFAM, in the complete Swiss-Prot database (Rel. 51.7) for direct *hmmsearch* (red bars) and the family model filtering approach with *PoSSuMsearch* (yellow and green bars) for the two different significance levels given by *trusted-* and *noise cutoffs*.

4.6.3 Evaluation and computational results

We tested *PSfamSearch* with the first 20 out of 2,946 pHMMs of the TIGRFAM database (Rel. 6.0) on the complete Swiss-Prot database (Rel. 51.7, $\sim$ 122MB protein sequence data). We determined *PoSSuMsearch* p-value cutoffs corresponding to *hmmsearch trusted cutoffs* as well as *noise cutoffs* with the iterative procedure described above. We measured the search space reduction (see Figure 4.12) and the total running times needed by *PSfamSearch* and compared them with *hmmsearch* operating on the unfiltered data set (see Figure 4.13). Running times for the filtered approach are total running times including times needed for search space reduction with *PoSSuMsearch* and subsequent application of *hmmsearch* on the filtered filtered sequence space. In these experiments PSSM family model based filtering reduces the search space and hence the overall running time dramatically. For example, for TIGRFAM family *YbaK_EbsC* (TIGRFAM Accession: TIGR00011) only 5 sequences remain after the filtering step and are handed over to *hmmsearch* to score them instead of all 259,034 Swiss-Prot sequences without filtering. Filtering with p-value cutoffs corresponding to the less stringent *noise cutoffs* revealed in the worst case (family *ril35*, TIGRFAM Accession: TIGR00001) even still a search space reduction of $\sim$ 50%.

The overall running time needed for searching is reduced from 1,400.2 minutes required by standard *hmmsearch* to only 10.1 minutes for *PSfamSearch* when using *trusted cutoffs*. This is a speedup of factor 138. Using *noise cutoffs* the achieved speedup factor is still $\sim$ 45.

We explicitly note, that we obtain with *PSfamSearch* and direct *hmmsearch* operating on the full sequence set, exactly the same results. Hence, *PoSSuMsearch* works in this scenario as a perfect, lossless filter. This is not too surprising, since thresholds were trained/adjusted on the same set of

sequences that was searched afterwards employing these thresholds. This raises the question, how well the calibrated p-value cutoffs generalize to sequences not included in the training set used for threshold determination.

4.6.4 Cutoff calibration strategies

The determination of a proper family specific p-value cutoff is crucial for the sensitivity as well as speedup of *PSfamSearch*. A too stringent cutoff results in too radical search space reduction which in turn has the effect, that *PSfamSearch* misses to many matches. Contrary, a too relaxed cutoff affects the obtained speedup factor negatively due to insufficient reduction of the search space. In the following we investigate and evaluate three different strategies for cutoff calibration. Namely,

- cutoff calibration based on family seeds,
- cutoff calibration based on *hmmsearch* matches obtained on a smaller sample set (Swiss-Prot),
- cutoff calibration based on UniProtKB/TrEMBL results with training- and test-set separation.

Cutoff calibration based on family seeds

To employ family seeds for the calibration of p-value cutoffs, we derived PSSM family models from the families' seed alignments and adjusted the p-value cutoffs for *PoSSuMsearch* so that all members of the seed alignment of a protein family were found by the model. Subsequently, we used these calibrated cutoffs on the UniProtKB/TrEMBL database (Rel. 35.0) containing 3,874,166 protein sequences comprising 1,260,291,226 amino acids with *PSfamSearch* and compared the achieved results with direct *hmmsearch* using the models *trusted cutoffs*. Detailed results for the first 20 TIGRFAM protein families are given in Table 4.3. In this experiment, direct *hmmsearch* returned for all 20 families a total of $7,588$ matches scoring above the *trusted cutoff*. Using *PSfamSearch* we obtained $7,005$ matches also detected by direct *hmmsearch* and missed 583. Although on average overall tested TIGRFAM families, *PSfamSearch* using p-value cutoffs trained on family seeds returned 88.86% of all direct *hmmsearch* results, for some diverse families the determined cutoffs were too stringent. An example for such a family is *cop-IBP*. For this family *PSfamSearch* missed with the determined cutoff of 0.00021 more than 96% (207 of 214) of the sequences detected by *hmmsearch*. The same problem arose for family *taut* for which more than 47% (80 of 168) of the *hmmsearch* matches were missed. We identified for this behavior the following two main reasons:

- Some protein families are too diverse to be represented properly by a single seed alignment. These families are defined by *trusted cutoffs* much lower than the scores obtained for the seed sequences. Hence, the sequences included in the seed alignment are not a representative sample for the family and thus inappropriate for cutoff calibration.
- Some seed alignments sometimes simply does not contain enough sequences for a proper representation of the family. Such an example is family *cop-IBP* for which the seed alignment contains 4 sequences only.

We conclude that family seeds are not well suited for p-value cutoff calibration.

TIGRFAM family	#matches (*hmmsearch* using TC)	#seqs in red. space	% of total seq. space	#matches in red. space	matches in red. space[%]	#missed	missed[%]	*PoSSuMsearch* Cutoff π_{TC}	#seed seqs.
riL35	242	51,842	1.33815	241	99.59	1	0.41	2.65E-004	24
ribS16	328	310	0.00800	310	94.51	18	5.49	2.44E-006	20
cop-IBP	214	8	0.00021	7	3.27	207	96.73	1.25E-006	4
I-PSP	517	520	0.01342	470	90.91	47	9.09	1.25E-006	19
RluA_subfam	1,038	1748	0.04512	1033	99.52	5	0.48	3.81E-006	16
mraW	349	343	0.00885	337	96.56	12	3.44	1.00E-007	8
Tigr0007	223	287	0.00741	223	100	0	0	1.25E-006	23
infA	327	380	0.00981	298	91.13	29	8.87	1.82E-005	13
ribL28	372	16,278	0.42017	355	95.43	17	4.57	1.69E-004	22
TatD	563	856	0.02210	558	99.11	5	0.89	1.95E-006	18
Ybak_EbsC	255	297	0.00767	249	97.65	6	2.35	3.05E-006	17
ribL29	487	28,883	0.74553	451	92.61	36	7.39	1.69E-004	27
taut	168	88	0.00227	88	52.38	80	47.62	3.81E-006	6
arsC	302	233	0.00601	231	76.49	71	23.51	1.00E-007	6
ackA	438	438	0.01131	415	94.75	23	5.25	1.00E-008	8
cmk	344	362	0.00934	342	99.42	2	0.58	1.25E-006	8
panC	358	374	0.00965	344	96.09	14	3.91	1.00E-008	5
prfA	244	609	0.01572	244	100	0	0	1.00E-008	11
prfB	472	697	0.01799	469	99.36	3	0.64	1.00E-008	6
rpiA	347	348	0.00898	340	97.98	7	2.02	1.25E-006	23
Avg:	**379.4**	**5,245.05**	**0.14**	**350.25**	**88.86**	**29.15**	**11.16**	**3.38E-005**	**14.2**

Table 4.3: Results of *PoSSuMsearch* cutoff calibration based on seed alignment members for first 20 TIGRFAM models. Detection rates measured on UniProtKB/TrEMBL.

Cutoff calibration based on Swiss-Prot matches

As a second strategy for cutoff calibration, we analyzed the usability of cutoffs determined from a relatively large training set. Therefore, we determined *PoSSuMsearch* p-value cutoffs corresponding to *hmmsearch trusted cutoffs* on the Swiss-Prot database for the first 20 protein families listed in TIGRFAM. With these cutoffs we searched in the complete UniProtKB/TrEMBL database (Rel. 35.0) and compared the results of *PSfamSearch* with direct *hmmsearch* using *trusted cutoffs*. Direct *hmmsearch* returned for all 20 families a total of 7,588 matches scoring above the *trusted cutoff*. Using *PSfamSearch* we obtained 7,487 matches also detected by direct *hmmsearch*. That is, our filtering approach returned 98.67% of the results detected by direct *hmmsearch*, but in a fraction of running time. See Table 4.4 for the detailed results.

The fact that *PSfamSearch* missed a few matches (2.13%) is caused by the incomplete representation of some of the families in Swiss-Prot, making it impossible to derive a meaningful cutoff for the whole family based on Swiss-Prot sequences only. This is in particular true for family *cop-IBP*. For this family *PSfamSearch* missed 51 out of 214 (23.83%) *hmmsearch* matches with the employed p-value cutoff. We further note, that the majority of matches missed by *PSfamSearch* achieved an *hmmsearch sequence classification score* near the *trusted cutoff* boundaries. That is, caused by the incomplete representation of some families in Swiss-Prot, p-value cutoffs for *PoSSuMsearch* were chosen too stringently. Another disadvantage of using *hmmsearch* matches for a pHMM obtained on Swiss-Prot for cutoff calibration is, that it reveals still unclear how well the determined cutoff can be generalized for new family members not contained in Swiss-Prot. It may be the case, that the complete family consists of Swiss-Prot sequences and the cutoff is then adjusted to find exactly these sequences.

Overall it seems more appropriate to adjust *PoSSuMsearch* cutoff values on a more complete set, probably UniProtKB/TrEMBL itself, or even more ideally on the set of all true family members known so far and to demonstrate the generalization abilities of determined cutoffs with clearly separated training- and test-sets.

Cutoff calibration based on UniProtKB/TrEMBL results with training- and test-set separation

As a third strategy for model parameter determination, we built PSSM family models from the families' seed alignments for the first 20 families listed in TIGRFAM and calibrated the p-value cutoffs and minimal chain lengths to match all sequences of a training set containing half of the sequences returned by direct *hmmsearch* on UniProtKB/TrEMBL using the profile hidden Markov models' trusted cutoffs. That is, we adjusted sensitivity according to the sensitivity level of *hmmsearch* operating with trusted cutoffs. Employing these models and cutoffs in a database search on complete UniProtKB/TrEMBL, *PSfamSearch* returned more than 99.7% of the original results determined by *hmmsearch*, including their E-values and scores. Only 14 of 7,574 matches (0.23%) were missed. With p-value cutoffs calibrated to match the sensitivity level of *hmmsearch* using *noise cutoffs*, *PSfamSearch* even detected 99.8% of the *hmmsearch* while missing only 18 out of 9,137 sequences. See Figure 4.14 and Tables 4.5 and 4.6 for detailed results of this experiment.

It took *PSfamSearch* only $\sim$ 146 minutes on one UltraSPARC III CPU running at 900Mhz, to search with the first 20 TIGRFAM families, instead of more than 7 days for direct *hmmsearch* using

No.	TIGRFAM family	#matches *PSfamSearch*	#matches *hmmsearch* using TC	#missed	missed[%]
1	*riL35*	241	242	1	0.41
2	*ribS16*	324	328	4	1.22
3	*cop-IBP*	163	214	51	23.83
4	*I-PSP*	510	517	7	1.35
5	*RluA_subfam*	1,035	1,038	3	0.29
6	*mraW*	349	349	0	0
7	*TIGR00007*	223	223	0	0
8	*infA*	321	327	6	1.83
9	*ribL28*	372	372	0	0
10	*TatD*	563	563	0	0
11	*Ybak_EbsC*	249	255	6	2.35
12	*ribL29*	485	487	2	0.41
13	*taut*	152	168	16	9.52
14	*arsC*	302	302	0	0
15	*ackA*	438	438	0	0
16	*cmk*	343	344	1	0.29
17	*panC*	358	358	0	0
18	*prfA*	244	244	0	0
19	*rpfB*	471	472	1	0.21
20	*rpiA*	344	347	3	0.86
total:		**7,487(98.67%)**	**7,588(100%)**	**101(1.33%)**	**avg: 2.13**

Table 4.4: Comparison of results obtained with *PSfamSearch* and direct *hmmsearch* when searching with first 20 TIGRFAM models on UniProtKB/TrEMBL. Cutoffs for *PSfamSearch* were calibrated based on *hmmsearch* matches on Swiss-Prot using the models *trusted cutoffs*. Columns five and six give the total number and percentage of matches missed by *PSfamSearch*. In this experiment *PSfamSearch* detected 98.67% of the matches detected by *hmmsearch*.

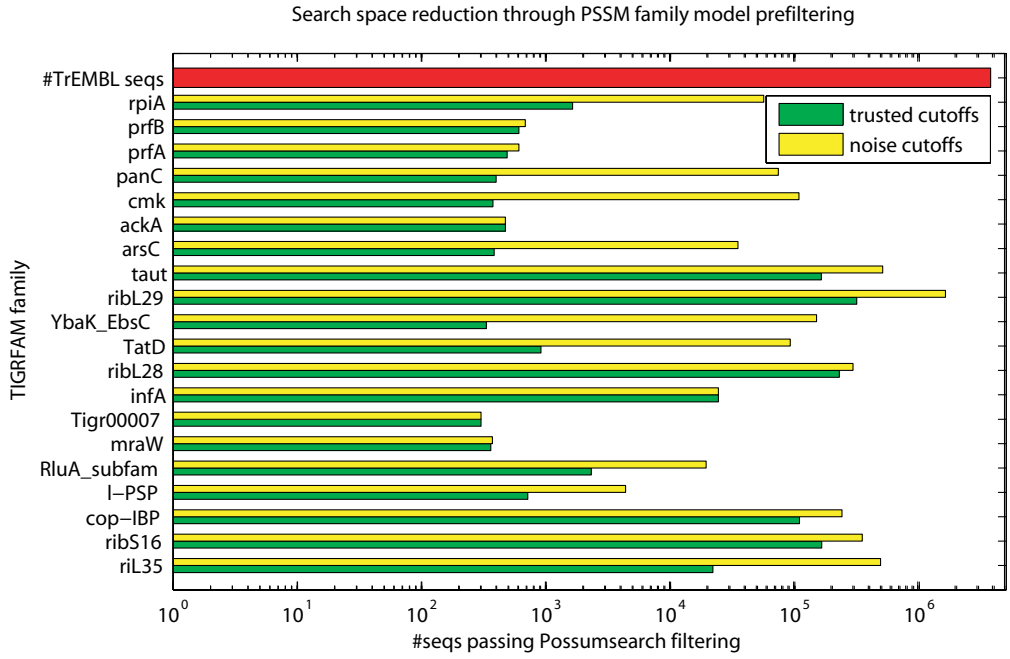


Figure 4.14: Reduction of UniProtKB/TrEMBL achieved by PSSM family model filtering for the first 20 TIGRFAMs families. Green (yellow) bars indicate the effective number of sequences to be searched with *hmmsearch* (x-axis, logscale) when using p-value cutoffs adjusted to match *trusted cutoffs* (*noise cutoffs*). The red bar shows the total number of sequences in the UniProtKB/TrEMBL (3,874,166 protein sequences with a total length of $\sim$ 1.26GB) needed to be searched by direct *hmmsearch* without filtering.

the models *trusted cutoffs*. That is, *PSfamSearch* achieves a speedup of factor $\sim$ 72 over direct *hmmsearch* while retaining more than 99.7% of the original results. Using the less stringent *noise cutoffs* *PSfamSearch* reduces the search space to only 5.24% of the original search space size with a sensitivity of 99.7% (see Table 4.6) and a speedup of factor of 15.2 over direct *hmmsearch*. Extrapolated to all 2,946 TIGRFAM families we estimate a running time of $\sim$ 14.9 days for *PSfamSearch*, and 3.02 years for direct *hmmsearch* using the models *trusted cutoffs*.

Overall, we observe, that cutoff calibration on a test set determined from search results of the pHMM on UniProtKB/TrEMBL outperforms the former mentioned calibration strategies and leads to cutoffs with very well generalization characteristics. Accordingly this strategy is well suited for determination of cutoffs with good sensitivity and search space reduction characteristics.

TIGR family	#seqs in red. Space	% of total seq. space	P-value cutoff	min. chain length	#found	#missed	found[%]	missed[%]
riL35	21,878	0.56	4.44E-05	2	239	3	98.76	1.24
ribS16	164,203	4.24	1.46E-05	2	328	0	100.00	0.00
cop-IBP	108,907	2.81	1.08E-04	2	213	1	99.53	0.47
I-PSP	710	0.02	1.82E-05	4	514	3	99.42	0.58
RluA_subfam	2,303	0.06	1.82E-05	4	1,038	0	100.00	0.00
mraW	358	0.01	2.44E-06	4	348	1	99.71	0.29
TIGR00007	299	0.01	5.96E-06	4	223	0	100.00	0.00
infA	24,280	0.63	1.69E-04	3	326	1	99.69	0.31
ribL28	227,473	5.87	1.01E-03	3	371	1	99.73	0.27
TatD	907	0.02	7.45E-06	4	561	2	99.64	0.36
Ybak_EbsC	330	0.01	1.16E-05	3	255	0	100.00	0.00
ribL29	313,713	8.1	2.27E-05	1	487	0	100.00	0.00
taut	163,417	4.22	1.46E-05	1	167	1	99.40	0.60
arsC	382	0.01	1.00E-06	2	302	0	100.00	0.00
ackA	470	0.01	1.00E-06	4	438	0	100.00	0.00
cmk	373	0.01	7.45E-06	4	343	1	99.70	0.30
panC	396	0.01	2.44E-06	6	358	0	100.00	0.00
prfA	485	0.01	1.00E-07	6	244	0	100.00	0.00
rpfB	603	0.02	1.00E-06	7	472	0	100.00	0.00
rpiA	1,629	0.04	3.55E-05	4	347	0	100.00	0.00
Average:	51,655.8	1.33	7.48E-005	3.55			99.78	0.22

Table 4.5: Results of p-value cutoff calibration based on *hmmsearch* matches obtained on UniProtKB/TrEMBL using *trusted cutoffs*. Cutoffs were calibrated such that half of the sequences (training set) pass *PoSSuMsearch* filtering. Column 2 and 3 give the absolute number and percentage of sequences passing the filter. Numbers of found and missed family sequences on complete UniProtKB/TrEMBL are given in column 6 and 7.

TIGR family	#seqs in red. Space	% of total seq. space	P-value cutoff	min. chain length	#found	#missed	found[%]	missed[%]
riL35	488102	12.6	3.31E-004	2	265	2	99.25	0.75
ribS16	348187	8.99	6.46E-004	3	329	0	100	0
cop-IBP	239167	6.17	1.26E-003	4	223	0	100	0
I-PSP	4351	0.11	3.55E-005	3	675	5	99.26	0.74
RluA_subfam	19315	0.5	2.84E-005	3	1496	0	100	0
mraW	369	0.01	4.77E-006	4	355	1	99.72	0.28
TIGR00007	299	0.01	5.96E-006	4	270	0	100	0
infA	24280	0.63	5.17E-004	3	329	2	99.4	0.6
ribL28	293362	7.57	1.58E-003	4	375	1	99.73	0.27
TatD	91600	2.36	1.36E-004	4	950	0	100	0
Ybak_EbsC	149252	3.85	3.31E-004	4	329	2	99.4	0.6
ribL29	1621658	41.86	1.01E-003	2	507	0	100 0	
taut	507491	13.1	5.55E-005	1	334	1	99.7	0.3
arsC	34857	0.9	2.84E-005	2	327	1	99.7	0.3
ackA	470	0.01	1.00E-006	4	465	0	100	0
cmk	107685	2.78	8.67E-005	3	366	1	99.73	0.27
panC	73506	1.9	1.82E-005	2	408	0	100	0
prfA	604	0.02	1.00E-007	5	275	0	100	0
rpfB	678	0.02	1.00E-007	4	482	0	100	0
rpiA	56059	1.45	6.94E-005	3	359	2	99.45	0.55
Average:	203,064.6	5.24	3.07E-004	3.2	Total: 9,119	Total: 18	99.77	0.23

Table 4.6: Results of p-value cutoff calibration based on *hmmsearch* matches obtained on UniProtKB/TrEMBL using *noise cutoffs*. Cutoffs were calibrated such that half of the sequences (training set) pass *PoSSuMsearch* filtering. Column 2 and 3 give the absolute number and percentage of sequences passing the filter. Numbers of found and missed family sequences on complete UniProtKB/TrEMBL are given in column 6 and 7.

4.7 Discussion and concluding remarks on performed experiments

In this chapter we presented the combination of the formerly introduced *ESAsearch* algorithm with a fast fragment chaining approach to efficiently search with PSSM family models in large data sets. We extended our search tool *PoSSuMsearch* with the algorithm of [AO05] and evaluated the performance of the combined method in terms of sensitivity and specificity as well as total running time. In addition, we compared the obtained results to a *state of the art* pHMM based approach represented in our experiments by the well known *hmmsearch* program from the *HMMER* package.

The experiments assessing the sensitivity and specificity in different evaluation scenarios show that for protein classification on the family and superfamily level, PSSM family models achieved a classification performance only marginally inferior to the performance of pHMMs, which yield to be the most sensitive modeling approach for detecting distant homologies. Although PSSM family models are much simpler than the full probabilistic pHMMs, the measured FP50 value of PSSM family models is only 3.2 percentage points below the FP50 value achieved by *hmmsearch* in the experiment evaluating the method's ability to detect very close relationships (see Figure 4.5). In the experiments assessing the detection performance of close and distant relationships the advance of *hmmsearch* over *PoSSuMsearch* was even only 4.4 and 1.6 percentage points respectively, when accepting 50 false positive matches. Hence, PSSM family models perform nearly as accurate as pHMMs. Additionally, there are indications that the classification performance of PSSM family models for protein family assignment can be further improved. Observe, that the PSSM family model construction process is really straightforward yet and in the performed experiments, simple log-odds ratios are used for PSSM deviation (see section 2.5.4 on page 28) from excised alignment blocks instead of the more sophisticated methods incorporating pseudo-counts described in sections 2.5.7 on page 33 and 2.5.8 on page 34. Preliminary results, not shown in this thesis, indicate, that PSSM construction methods using pseudo-counts increase the classification performance significantly. Another starting point for further improvements is how ungapped alignment blocks are excised from the underlying multiple alignment. One can think of using ungapped but overlapping tiles for PSSM deviation instead of non overlapping blocks. This should give a better coverage of the multiple alignment and may lead to a PSSM family model representing the sequence family more accurately. Additionally, the distance between blocks or tiles in the alignment could be incorporated into the chain score function. This should increase the specificity of PSSM family models.

Still an open problem is the efficient determination of accurate statistics for PSSM chain scores without the need for time consuming sampling. Although chain scores as defined by Equation (4.8) and (4.7) performed well for sequence classification (see Experiments 1 to 3) the score sampling on random sequences clearly showed a strong dependency on the length of the matched sequence. At the time of this writing it is not clear, if this problem can be solved by additional normalizations that finally may lead to a continuous distribution function for high chain scores.

The surprisingly well performance of PSSM family models for protein family classification in terms of sensitivity and specificity appears in an even brighter light, when the total running time needed by *PoSSuMsearch* and *hmmsearch* to accomplish the same task is taken into account. For the setup of experiment 4 (see section 4.5.3 on page 118), it took *PoSSuMsearch* less than 40 minutes to search with 100 PSSM family models built from the first 100 Pfam protein families on the complete Swiss-Prot database using a p-value cutoff of $\pi = 10^{-4}$, whereas *hmmsearch* employing an E-value

cutoff of 10^{-5} needed more than 4 days ($\sim$ 105 hours) for this task. That is *PoSSuMsearch* achieved a speedup of factor 171 over *hmmsearch*. Observe, that we measured the running time only for the first 100 out of 8957 families listed in the current Pfam release 21.0. By linear extrapolation of the measured running time to all 8957 families listed in Pfam Release 21.0, we assume a running time for searching with all family models on Swiss-Prot for *hmmsearch* of $\sim$ 391 days compared to only $\sim$ 54.8 hours for *PoSSuMsearch*.

In the experiments using PSSM family models for search space reduction for *hmmsearch*, the combined approach (*PSfamSearch*) using *PoSSuMsearch* for pre-filtering and subsequently *hmmsearch* also performed very well.

Since the achieved speedups as well as the sensitivity of *PSfamSearch* strongly depend on the chosen p-value cutoff, we tested different strategies for threshold determination. In our experiments, the achieved speedups of *PSfamSearch* were in the range between 72 and 138 when using p-value cutoffs corresponding to *trusted cutoffs* and between 15.2 and 45 for p-values adjusted to match the significance level of the models' *noise cutoffs*. The highest speedup factor for *PSfamSearch* over *hmmsearch* of 138 was obtained when searching with models for the first 20 TIGRFAM families on Swiss-Prot with p-value cutoffs calibrated according to *hmmsearch* matches on Swiss-Prot using *trusted cutoffs* (see section 4.6.3 on page 129). With a clear separation of training- and test-sets for cutoff determination, necessary to derive cutoffs with good generalization characteristics, *PSfamSearch* also achieved speedups between 72 and 15.2 with more than 99.7 sensitivity when searching with models for the first 20 TIGRFAM families on complete UniProtKB/TrEMBL (see section 4.6.4 on page 132). Extrapolated to all 2,946 TIGRFAM models we expect a reduction of running time from more than $\sim$ 2.84 years for direct *hmmsearch* using *trusted cutoffs* to only $\sim$ 15 days for *PSfamSearch*.

In particular, the extremely long running times and the linear time scaling behavior[5] of pHMM based methods employing the *Forward*, *Backward*, or *Viterbi* algorithm (see Figure 4.8) make them more and more challenging and sometimes even infeasible to dispose in today's sequence database search scenarios. In the future this problem will get even more tightening as sequence databases still grow at an exponential rate. Additionally new, revolutionary high-throughput sequencing techniques like 454 sequencing [MEA+05] will certainly amplify this growth in the near future. Nevertheless, pHMM based database searches are an indispensable, standard task in today's genome annotation pipelines. For instance the majority of member databases of the InterPro classification system [MAA+07], a widely used system for protein annotation purposes, employ family information in form of pHMMs. The applied classification procedure *InterProScan* [QSP+05] includes searches with all pHMMs from the Pfam [FMSB+06], TIGRFAM [HSW03], Superfamily [GKHC01], PIRSF [WNH+04], Gene3D [YMM+06], Smart [LCP+06], and Panther [MLUL+05] databases. Especially these pHMM based database searches render *InterProScan* into a very compute intensive application whose employment on a large scale is even challenging on huge cluster systems.

To solve this dilemma much effort has been spent on improving the running time of pHMM based database search tools. Some approaches for improvement use parallelism techniques and/or fast, extended, CPU specific instructions sets, like SSE/SSE2 (Streaming Single Instruction/Multiple Data Extensions) [WQC06, Dep03]. Also discussed is the application of pruning techniques (c.f.

[5] Linear in the length of the searched sequence.

[Plö05]), like the employment of the *Beam-Search* algorithm [Low76] instead of the full *Viterbi* algorithm. The probably most successful accelerations available, are the commercial DeCypher© and BioBoost©HMMer solutions sold by TimeLogic® and progeniq® respectively. They implement among other things the program *hmmsearch* in hardware on special hardware acceleration boards using *Field Programmable Gate Arrays* (FPGAs). In benchmark experiments published by TimeLogic® a speedup up to factor 180 for a single DeCypher© accelerator board over standard *hmmsearch* is reported [Tim06]. progeniq® reports for the BioBoost©HMMer board a speedup of factor 40 over standard *hmmsearch* running on an AMD Athlon 64 3500+ [Pro07]. Considering, that our experiments revealed for *PoSSuMsearch* speedups up to factor 171 over standard *hmmsearch*, and for *PSfamSearch* up to factor 138, we observe that our purely software based acceleration of *hmmsearch* compares well with what is achieved by costly, specialized hardware solutions like DeCypher© or BioBoost©HMMer. We note, that this speedup comes from an algorithmic as well as a conceptual advancement:

- the speed of index based PSSM searching, and
- the astonishing fact that pHMMs can be approximated well with the simpler PSSM family models and achieve a similar performance for protein family classification as the widely used more complex pHMMs.

For these reasons, we make up our discussion and concluding remarks with a comparison of PSSM family models and pHMMs focusing on similarities and differences.

4.7.1 Comparison of pHMMs and PSSM family models

Consider that a PSSM is essentially equivalent to a pHMM consisting of a linear sequence of *match* states only, with state transition probabilities of 1 between them, as described in section 2.7.2 and shown in Figure 2.12. That is, each *match* state corresponds to a column in the multiple alignment and hence a row in the PSSM. It emits a symbol from the output alphabet with a certain probability depending on the probability/score distribution in the corresponding PSSM row. This perception also holds for PSSM family models like the model $\mathcal{M} = M_1, M_2, M_3$ given in Figure 4.15 consisting of three PSSMs of lengths $|M_1| = 4$, $|M_2| = 2$, and $|M_3| = 3$. In contrast to a pHMM where each *match* state is connected with an *insert* state and each *match* state can be skipped by a *delete* state (see Figure 2.11 on page 43), in the PSSM family model arbitrary insertions are only allowed between single PSSMs, and *delete* states allow only to skip complete PSSMs. That is, PSSM family models combined with the employed chaining approach allow

- arbitrary insertions between single PSSMs of a family model and
- arbitrary deletions of complete PSSMs[6].

Consequently a PSSM family model is, compared to a pHMM, a similar but more restrictive modeling approach for a family of related sequences. In addition, PSSM family models are not necessarily

[6] This holds at least for our chaining approach and definition of chain scores (see section 4.8 on page 117). However, one can also think of a *local* chaining incorporating some kind of gap penalties instead of the currently used more *global* one.

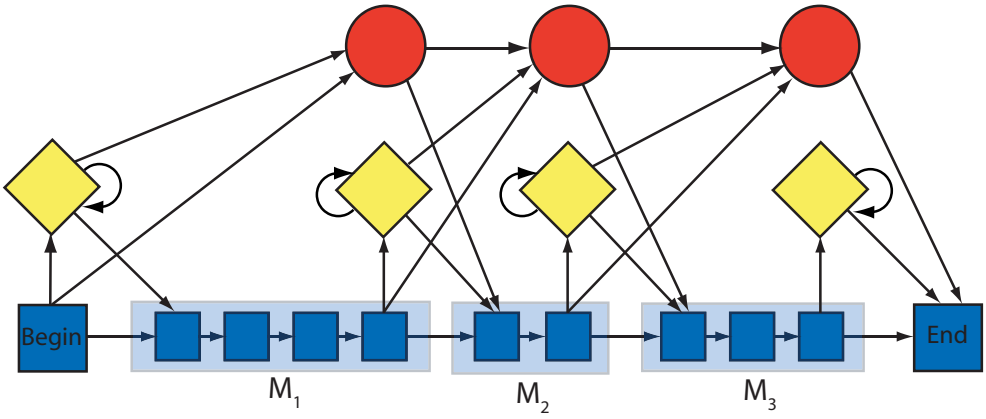


Figure 4.15: A pHMM like view on PSSM family models. Shown is a PSSM family model $\mathcal{M} = M_1, M_2, M_3$ consisting of 3 PSSMs. Likewise to the common pHMM graph view, blue squares denote *match* states, *insert* states are drawn as yellow diamonds and *delete* states are given by red circles. Valid state transitions are drawn as unlabeled black arcs. Observe that each path through $\mathcal{M}$ starting in the *Begin* state and ending in state *End* corresponds to a valid chain of PSSM matches and hence a match to $\mathcal{M}$ according to Definition 15 on page 107.

fully probabilistic, since they can consist of PSSMs containing arbitrary score values. Also in a PSSM family model, transitions from/to an *insert* or *delete* state are unweighted. Hence much less parameters have to be trained from the underlying data. This, in turn allows the construction of meaningful PSSM family models from multiple alignments containing much fewer aligned sequences than are necessary for proper pHMM derivation.

5 Genlight - a system for interactive, high-throughput, differential genome analysis

5.1 Motivation

Even today and more severe in the future, advancements in high-throughput sequencing techniques that reduce the time needed for sequencing an organism's genome from several years to a few days, will lead to a growing gap between data collection and data interpretation. With the increasing amount of data that needs to be analyzed there is not only a strong demand for efficient computational methods generating accurate and reliable results, but also for integrative approaches and systems that allow to rapidly apply and combine several analysis methods in a user-friendly fashion, even in data rich application scenarios. The support of different analysis methods for the same task does not only introduce more flexibility, but also allows to identify method specific weaknesses in certain application scenarios more quickly. Once such deficiencies are identified, the ability of combining different methods may allow to balance them and hence increase the overall quality of the results. In addition, with increasing numbers of complete genome sequences, tasks are shifting from single gene to complete genome or proteome analyses, and many new questions regarding similarities and differences between the sequenced organisms arise in multiple genome comparison approaches. An even strong commercial interest exists in genome comparisons of pathogenic organisms, since they can lead to new insights in the principles of pathogenity and infection [GFB+01, HDB98]. Pathogen genome sequencing projects have provided a wealth of data in this field that need to be set into context of pathogenicity and the outcome of infections to understand and interfere with deseases caused by microbial pathogens.

One of the new challenging questions is the differentiation between species specific and common genes [HdlTV03, Koo03]. This is also a fundamental questions in the target-based approach in the development of either narrow-spectrum or broad-spectrum antibiotics. For instance, among the key criterias that must be met by an anti-microbial drug target are

- target pathogen spectrum,
- target selectivity,
- target essentiality, and

- target function, i.e., the biochemical function of the target needs to be characterized. This includes among other things the gathering of information about structure, potential active and binding sites, etc.

Genes satisfying these criteria and hence making a promising anti-microbial drug target can be identified and evaluated by comparing all relevant pathogen genomes with the host genomes. Genes that show to be conserved in these large-scale comparisons across different pathogens often turn out to be essential and hence may represent target candidates for new broad-spectrum antibiotics. Differential or subtractive analyses can reveal those genes that are conserved in all or most of the pathogenic bacteria but not in eukaryotes. These are the most obvious candidates for drug targets. Species-specific genes, also identifiable by differential genome comparisons, may offer the possibility to design drugs against a particular, narrow group of pathogens.

Different studies [DDSS01, HDB98] already proved the potential of differential genome analyses, often also called differential comparative genomics, especially in combination with the analysis of functional relevant sequence motifs or domains describable with one of the motif models introduced in chapter 2, to detect new drug target candidates. Such procedures often include the application of a variety of bioinformatics methods and searches in different databases to retrieve a maximum of information about the sequence or gene under consideration. To be feasible in practice, especially on a larger, genomic scale, integrated and scalable solutions are necessary that support the user in this data rich problem environment. Unfortunately, the number and flexibility of existing systems is not sufficient or to the least very limited. Hence there is a strong need for new integrated solutions.

In the following section, we will give a brief overview of existing and conceptional related systems and explain why they are not well suited to solve our sketched problem scenario.

5.1.1 Genome annotation systems: Related concepts with different focus

Although the integration of various bioinformatics methods and automated sequence homology searches are widely used techniques in genome annotation systems, such as *Magpie* [GS96b, GS96a], *PEDANT* [FAH+01], *genomeSCOUT* [SCK00] and *GenDB* [MGM+03], the objective of these systems mostly focuses on textual annotation of genes only. An important point, often neglected in existing systems, is the querying and mining of stored data, especially query capabilities that allow to combine different, derived attributes or characteristics. More precisely, high-level queries like the following, combining several attributes of a gene or protein, are hardly possible.

Which outer membrane proteins involved in metabolism M, and linked to apoptosis from the pathogen organism A are highly conserved in pathogen organism B but lack a counterpart in apathogen organism C and host organism D?

The deficiencies of existing systems to answer such queries is often founded in the way they internally organize and store data. For instance the *GenDB* system, a widely used genome annotation system for prokaryotes, uses a proprietary object relational mapping layer that allows a persistent storage of the used object oriented data model in an underlying relational database management system. Although this mapping layer admits an easy and almost seamless storage of objects in an relational database while abstracting from the underlying relational data model, the relational data model generated from the applications object model is not well suited to be queried with rela-

tional database query languages like SQL, any longer. As a consequence the retrieval of non-trivial information becomes problematic. Especially more complex queries joining multiple attributes of different database tables, are nearly impossible to formulate or suffer at least from bad response times. To solve this issue, an annoying and often redundant programming overhead on the application level is needed for implementing higher-level functionalities, which could be easily accomplished with standard SQL-queries and a well designed, query optimized, relational data model[1].

The commercial genome annotation system *genomeSCOUT*, which is no longer available due to limited commercial success, stored its data in simple ASCII formatted flat files and used the data integration system *SRS* [EUA96] for information retrieval and basic data mining tasks. Although *SRS* basically offers some flexibility aimed to easily integrate proprietary data from for example in-house sequencing projects, this benefit is only of practical use with extensive programming skills and knowledge of the system.

The aforementioned deficiencies of genome annotation systems like insufficient data querying and mining capabilities should not brush off these systems. We just reveal that traditional genome annotation systems are simply not designed for extensive querying and mining of data.

When we focus on automated differential genome comparisons, very little is found in literature on that topic. To the best of our knowledge, only three noncommercial, initial attempts have been made to develop computational systems, that support these kinds of analyses, namely Seebugs [BDD98], *FindTarget* [CGK01], and *Difftool* [CGK02]. All of these provide limited functionality and flexibility, i.e., they are very limited in their supported sequence comparison methods, and neither integrate additional databases for sequence motif analyses, nor do they support subsequent analyses including generated results inside the systems. That is, they do not allow reusability of derived results for the step by step modeling of more compley analysis workflows. Further, all systems operate on precalculated data and do not allow for interactive on-the-fly analyses.

The deficiencies of existing systems were the motivation for the development of the Genlight system [BSS04, BMM+04], a versatile and powerful system for interactive high-throughput sequence analysis and differential comparative genomics with extensive data querying capabilities fulfilling the subsequently described requirement definitions.

5.2 Requirement definitions and design goals

Genlight follows the overall paradigma of a highly integrated system, suited to perform a wide range of large-scale sequence analysis tasks in an interactive way with features to combine, reuse, and query derived results. In particular, Genlight was developed to fulfill the following requirement definitions:

- support for the discovery or prioritization of potential new drug targets *in silico* by highly automated differential comparative analyses and user specified selection criteria;
- automatic genomic scale analyses in reasonable time, without the need for specialized hardware or large and expensive cluster systems;

[1] We note at this point that this is a prevalent problem of object relational mapping solutions, that try to store hierarchically organized objects in a flatter relational database schema.

- interactive as well as asynchronously executed large-scale analyses;
- integration of a wide range of bioinformatics analysis methods, each applicable on genomic scale data;
- scalability;
- integration of various publicly available sequence and motif databases;
- structured storage of computed results, that allow for extensive querying and mining;
- support for user defined queries and filters operating on generated data whose results are persistently stored inside the system;
- reusability of generated results to allow protocol based step by step modeling of more complex analysis workflows;
- concurrent multi user capabilities with project and access control management;
- dynamic presentation and visualization of computed results through an easy to use, but still flexible, platform independent interface;
- data import and export capabilities which support commonly used exchange formats.

Although each of the requirements listed above can be individually achieved with existing software solutions, to the best of our knowledge, no publicly available system exists combining all requirements in a single integrated approach.

5.3 System architecture and implementation

The Genlight system consists of four major parts as shown in Figure 5.1:

- a web-based user interface for the communication with the system,
- the Genlight server, providing queuing, scheduling, and dispatching capabilities,
- client components to carry out various bioinformatics analysis tasks in an asynchronous way,
- and a database component for storing, modifying, and accessing data.

To allow asynchronously executed, large scale sequence analysis tasks in an interactive system, Genlight uses a distributed client server approach. The core of the system, i.e., the client and server components, implementing the distributed execution engine with its queuing, scheduling and dispatching components are written in the C programming language and access the database via PostgreSQL's native C language interface *libpq*. To ensure a maximum of robustness and fault tolerance, which is in particular important in a distributed system, the server as well as the client component are implemented using multi threading for connection supervision and make use of backlog techniques. This allows to detect the failure of a compute node, deactivate this node in the

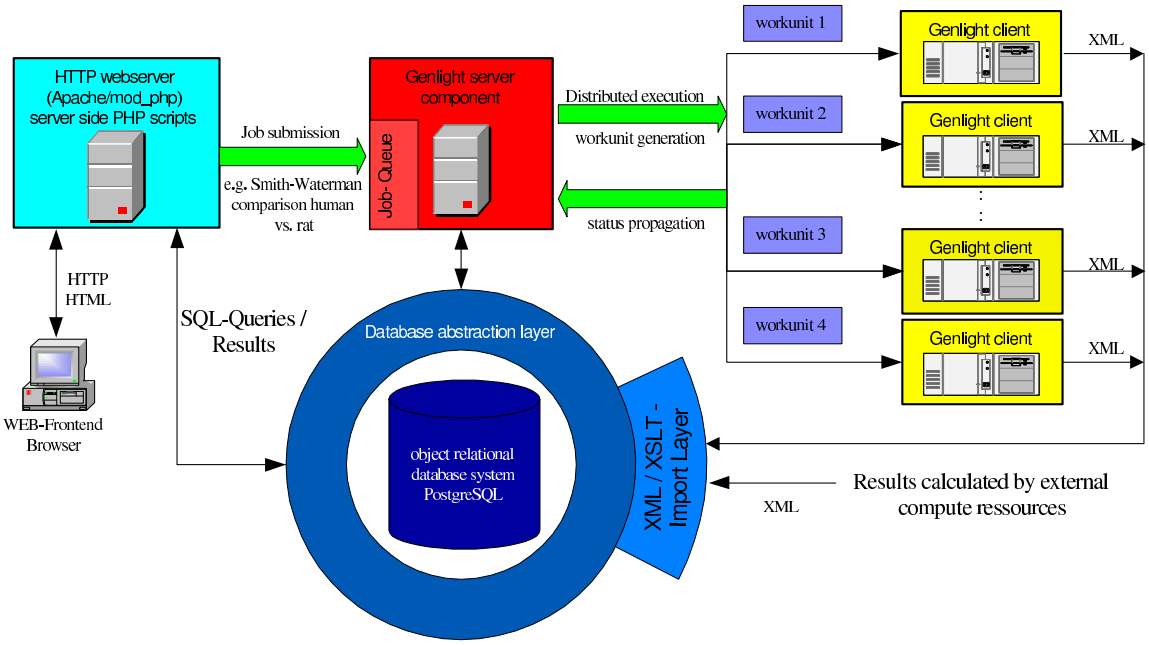


Figure 5.1: A schematic overview of the Genlight system architecture

virtual compute cluster, and allow to resubmit the assigned task to a different node. Further, run time errors of an integrated analysis method do not affect the client application.

The system is capable to serve multiple users. This is achieved with, among other things, the transaction mechanisms of the underlying database system. For persistent data storage and access using SQL queries, Genlight employs the ORDBMS (Object Relational Database Management System) PostgreSQL, though any other full SQL99 compliant DBMS (Database Management System) should also work. The system makes use of PostgreSQL's object oriented features like inheritance (see section 5.5 on page 158) and transaction capabilities to ensure data integrity and consistency.

The web interface is written in the server side scripting language PHP. PHP scripts retrieve data from the underlying PostgreSQL database and generate dynamic HTML pages which are subsequently delivered by the Apache web server to the user's web browser. Dynamic visualizations of results are performed using the GD graphics library.

In the following we describe the underlying concepts of the main parts of Genlight and the functionalities they provide.

5.4 Concepts and functionality

5.4.1 The set oriented concept

The structured storage, ensuring reusability of generated results, is a critical point for the protocol based step by step modeling of complex experiments and workflows often neglected in bioinformatics applications. In Genlight the reuse of derived results is a central concept, anchored in the basic

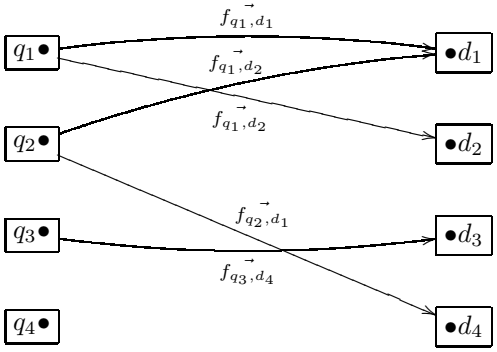


Figure 5.2: A bipartite graph as a model for a *Hit-set*.

system design. It is achieved by a set oriented data model with only two basic data object types: *Seq-sets* and *Hit-sets*. A *Seq-set* $\mathcal{Q} = \{S_1, S_2, \ldots, S_n\}$ is basically an ordered set[2] of $n = |\mathcal{Q}|$ sequences over a predefined alphabet $\mathcal{A}$, which is usually the nucleotide or amino acid alphabet. All sequences $S_i \in \mathcal{A}^+$, $i \in [1, n]$ are of one kind, either nucleic acid or protein. That can be for example all proteins of a certain organism in the order of their occurrence in the organisms genome. A specialized form of a *Seq-set* is the *Cluster-set* which contains for each sequence entry additional information that allows a partitioning of the *Seq-set* into sub sets. This allows to model the clustering of *Seq-sets*. Although *Seq-sets* contain additional sequence specific informations for each sequence, like ID, length, molecular weight and in case of amino acid sequences molar absorption coefficient and isoelectric point, etc., we neglect these additional informations in the following remarks for reasons of simplicity.

A *Hit-set* is a set of sequence pairs, defined by a comparison operation between two *Seq-sets* and its user defined parametrization, e.g., the set of all sequence pairs detected by a homology search between two *Seq-sets*. Observe that a sequence comparison operation between a single query sequence and a set of sequences to be searched (compared) establishes a one to many relationship. Consequently, in case of comparing two *Seq-sets* $\mathcal{Q}$ and $\mathcal{D}$, the resulting *Hit-set* defines a many to many relationship $\mathcal{H}_{\mathcal{Q},\mathcal{D}} \subseteq \mathcal{Q} \times \mathcal{D} = \{(q, d) \mid q \in \mathcal{Q} \wedge d \in \mathcal{D}\}$ between sequences from $\mathcal{Q}$ and $\mathcal{D}$ and hence can be seen as a directed, weighted, bipartite graph as shown in Figure 5.2 with vertices corresponding to sequences of the two *Seq-sets* and edges corresponding to the pair relationship weighted with a feature vector $\vec{f_{q,d}}$. The feature vectors $\vec{f_{q,d}}$ contain additional information, further characterizing the specific sequence pair (q, d) (e.g. statistical significance of the relationship, alignment scores, percentage of identity inside aligned region, etc.).

5.4.2 Operations on Seq-sets and Hit-sets

Genlight supports various operations that can be applied to *Seq-sets* and *Hit-sets*. The result of each operation is again a new *Seq-set* or *Hit-set*. A *Hit-set* filter, for instance, which can be pre-defined in the system or user defined, generates a new *Hit-set* with sequence pairs satisfying the respective filter condition. Filter criteria for *Hit-sets* can be any of the attributes associated with a sequence pair, like E-value, score(-ranges), rank, alignment coverage rate, percentage of identity/positives,

[2]For sake of simplicity we speak of ordered sets instead of tuples.

etc. Two *Hit-sets* may be combined with a filter to determine, for example, bi-directional best hits, where "best" can be defined on method specific ranking or other attributes. Additionally, Genlight comes with a collection of predefined filters for more complex filtering tasks. A detailed list of predefined filters and their semantics are given in the Appendix in Table A.5.

Sequence filters generate new *Seq-sets* and extraction operations convert a *Hit-set* to a new *Seq-set* depending on specified criteria (see Table 5.1). This procedure follows the software engineering concept of *compositionality* and allows an interactive step by step modeling of complex workflows as schematically drafted in Figure 5.3.

Using a combination of comparison, filter, and extraction operations, several proteomes, say A, B, and C, can easily be screened for proteins common to proteome sets A and B but nonexistent in proteome set C. Moreover, all possible intersections of A, B, and C can be calculated. Evidence of proteins with similar functions can be defined by combinations of several homology search results (e.g., unidirectional or bidirectional best hits), even generated by different homology search methods. Further on, the results of different sequence comparison methods can be combined with Boolean operators. With this concept the results of different alignment methods can be taken into account as evidence factors for the detection of homologous genes and weaknesses in the heuristics of a single method, which result in a false negative detection of homologous sequences, can be balanced.

The implemented project management, provides fundamental access control features and allows to store *Seq-sets* and *Hit-sets* on a per-user basis. Frequently used *Seq-sets* and *Hit-sets*, like major sequence databases as GenBank or UniProtKB/TrEMBL, model organism comparisons, etc., can be made available system-wide. The administrative features are complemented by a quota system, which allows to assign resources on a per-user and per-method basis. It is therefore possible to restrict the number of *Seq-sets* and *Hit-sets* in a project or to limit the size of a *Seq-set* in a comparison operation.

5.4.3 Integrated sequence analysis methods

Several different algorithms have been developed over the last decades to compare biological sequences and determine a concrete measure of their distance or similarity in order to deduce a common or similar biological function (c.f. [SW81, AGM$^+$90, AMS$^+$97, Pea99, ZSWM00]).

The dynamic programming methods for global (i.e., the Needleman-Wunsch algorithm) or local alignments (i.e., the Smith-Waterman algorithm) allow to obtain the optimal alignment under a given scoring schema, in time proportional to the product of the lengths of the two sequences being compared. With exponentially increasing sequence database sizes, complete exhaustive similarity searches based on full dynamic programming are no longer feasable in reasonable time. This problem was the motivation that has led to the development of the *FASTA* [Pea99] and *BLAST* (Basic Local Alignment Search Tool) [AGM$^+$90, AMS$^+$97] alignment programs, which became the most widely used algorithms in database searches and comparative sequence analysis. One important aspect, which is often overlooked, is that they are based on *heuristics*. They achieve improved performance compared to a full dynamic programing approach like the Smith-Waterman algorithm [SW81] by sacrificing some sensitivity. *BLAST* and *FASTA* reduce the problem by selecting the sequences in a database search that are thought to share significant similarity with the query

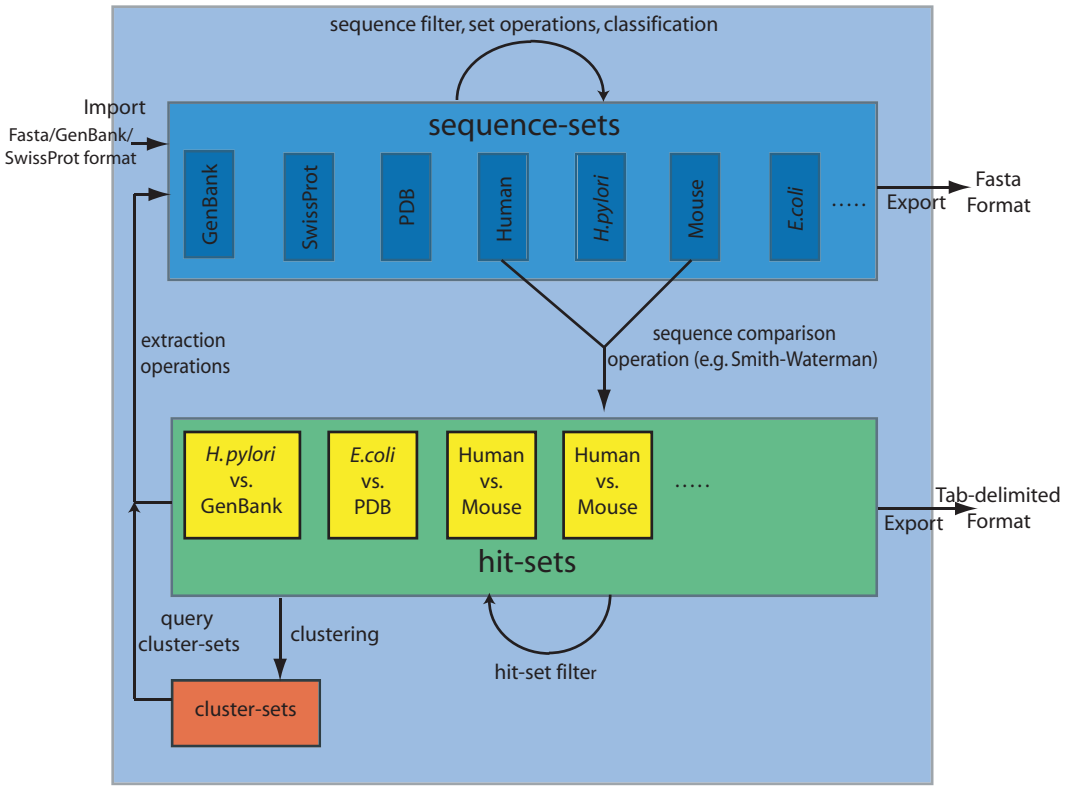


Figure 5.3: Genlight's operational model. The reuse of results is anchored in Genlight's operational design allowing a step by step modeling of complex analysis tasks. E.g., *Seq-set* filtering and classification operations result in new *Seq-sets* and a filtering operation applied to a *Hit-set* generates a new *Hit-set* for further reuse, containing only sequence pairs satisfying the filtering constraints.

Operation	Result
	Seq-set operations
filter by domain/motif composition	all sequences with a specified sequence motif or a combination of sequence motifs
SCOP filter	all sequences with user defined sequence similarity to a SCOP class, fold, superfamily, family or protein
taxonomy filter	all sequences that belong to a given taxon (if taxonomy information is available)
filter by length	sequences satisfying length constraints
intersect	all sequences that are present in at least two *Seq-sets*
union/merge	merges two or more *Seq-sets*
	Hit-set filter
filter by attribute values	all pairs of a *Hit-set* satisfying the filter condition. Filter condition is a boolean expression over attribute values
best hit filter	selecting the best hits depending on method specific rankings
two-way-best hit filter	selecting bidirectional best hit pairs depending on method specific rankings
text pattern filter	all pairs that contain a given pattern in the query or hit, or in both descriptions of the two sequences of a *Hit-set* entry
full query seq. length matches	all pairs with an aligned region length equal to the length of the query sequence
full hit seq. length matches	all pairs with an aligned region length equal to the length of the hit sequence
	extraction operations
query sequences with homologs	generates a new *Seq-set* of sequences that have a homolog in a *Hit-set*
homologs	generates a new *Seq-set* of sequences that are determined as homologs in a *Hit-set*
query sequences with no homologs	generates a new *Seq-set* of sequences that have no homolog in a *Hit-set*
homologs	generates a new *Seq-set* of sequences from DB-set that are not present in a *Hit-set*
	cluster set operations
differential cluster analysis	selects all clusters that contain sequences satisfying boolean expression over *Seq-sets* membership

Table 5.1: An excerpt of available operations on hit-sets and seq-sets.

sequence, and by locating the similar regions in the sequences. These selective steps allow to confine the computationally expensive sequence alignment methods based on dynamic programming only to a subset of the database sequences and to restrict the search for the best local alignment to only subregions of the sequences. Because of concerns of speed they estimate the similarity between the sequences in an approximate manner, and thus introduce a risk of missing similarities that are not detectable with the underlying heuristics.

In Genlight we integrated a wide range of sequence comparison algorithms, including methods based on full dynamic programming as well as algorithms employing heuristics. Almost all algorithms of the *BLAST* [AMS+97], and *FASTA* [Pea90, Pea94] family as well as the traditional Smith-Waterman algorithm [SW81] are integrated into Genlight. This enables the user to freely choose a sequence comparison method depending on available compute resources, problem and data sizes, and experimental requirements. In particular, we will see in section 5.4.8 that Genlight allows for the application of computationally expensive operations on a larger scale than other systems, due to bundling of available resources.

The results of sequence comparison operations are stored in *Hit-sets* and these homology information can directly serve as input for the probabilistic clustering algorithm *Tribe-MCL* [EvDO02]. *Tribe-MCL* relies on the Markov cluster algorithm of [vD00] for large-scale assignment of proteins into families based on precomputed sequence similarity information. We modified it, so that *Tribe-MCL* directly utilizes sequence similarity information stored in *Hit-sets*. Results of the clustering are stored in *Cluster-sets*. Thus, it integrates seamlessly into Genlight and allows to cluster even whole proteomes in seconds or minutes.

Adjacent to the integrated sequence comparison methods, Genlight can even compute features of single sequences, like the ability of an amino acid sequence to form a coiled-coil conformation. Coiled-coil structures are 2 to 5 stranded bundles of α-helices which are stabilized by hydrophobic and other interactions [NS76]. They are common in extracellular matrix molecules to connect different subunits in oligomeric proteins or in regulatory proteins like transcription factors. Coiled-coil domains are characterized by a heptad repeat pattern in which residues in the first and fourth position are hydrophobic, and residues in the fifth and seventh position are predominantly charged or polar. This pattern can be used to predict coiled-coil domains in amino acid sequences with computational methods. Genlight makes use of the *COILS* program [LVDS91, Lup96] to detect potential coiled-coil regions of protein sequences.

In particular useful for wet-lab work is Genlight's capability to determine basic sequence features, like a sequence's G/C content, molecular weight, molar absorption coefficient, isoelectric point, or charge.

5.4.4 Integrated protein domain and family databases

Protein evolution has employed a repertoire of a few thousand elementary modules or domains, which form the building blocks of today's proteins. Since structure and molecular function is largely conserved within domain families, computational methods for domain identification have become powerful tools in sequence function annotation, structure-function analysis. Searching for conserved domains can be helpful in particular to

Method	Explanation
BLASTN	Nucleotide Blast: Nucleotide query vs. nucleotide DB
BLASTP	Protein Blast: Protein query vs. protein DB
BLASTX	Translated nucleotide query vs. protein DB
TBLASTN	Protein query vs. translated nucleotide DB
TBLASTX	Translated query vs. translated nucleotide DB
psiBLAST	Position specific iterated Blast: Protein query vs. protein DB
FASTA	Nucleotide query vs. nucleotide DB or protein query vs. protein DB
FASTX/Y	Nucleotide query vs. protein DB
TFASTA	Translated nucleotide query vs. translated nucleotide DB
SSEARCH	Smith-Waterman algorithm: Nucleotide query vs. nucleotide DB or protein query vs. protein DB
rpsBLAST	Reverse position specific Blast: Protein query vs. CDD models
hmmpfam	Protein query vs. HMM database, like Pfam, TIGRFAM or Smart
PoSSuMsearch	Protein query vs. PRINTS and BLOCKS databases employing PSSM family models and fast fragment chaining as described in chapter 4 on page 105
COILS	Detection of coiled-coiled regions in proteins
Tribe-MCL	Markov based clustering of protein sequences

Table 5.2: An excerpt of supported sequence analysis methods.

- locate functional domains within a protein,
- predict the function of a protein whose function is unknown,
- establish evolutionary relationships across protein families,
- predict the structure of a protein of unknown structure.

High quality functional and structural annotation information about protein domains and protein families is available in several manually curated databases. Genlight integrates these heterogeneous data sources and their specific screening and search methods in one common environment and allows to rapidly combine derived results. More precisely, for the discovery of conserved domains, we integrated (i) the hidden Markov model based databases Pfam [FMSB+06], TIGRFAM [HSW03], Smart [LCP+06], CATH [PTS+05], and Superfamily [GKHC01], (ii) National Center for Biotechnology Information's (NCBI for short) PSSM based conserved domain database (CDD for short) [MBADS+05], and (iii) the PSSM family model based databases PRINTS [AMG+06] and BLOCKS [HGPH00]. The CDD is a collection of sequence alignments and PSSMs representing protein domains conserved in molecular evolution and hence defines the features that are conserved within each domain family. Therefore, the CDD can serve as a classification resource that groups proteins based on the presence of these predefined domains. To identify conserved domains in a protein sequence by screening versus the CDD, Genlight employs the reverse position specific *BLAST* variant *rpsBLAST*. With *rpsBLAST* the query sequence is compared to a *psiBLAST* generated PSSM prepared from the underlying conserved domain alignment. A screening versus CDD can also reveal insights in the structure of a protein, since CDD entries are linked to three dimensional structure data of the molecular modeling database MMDB [WAC+07]. This allows the user to identify the

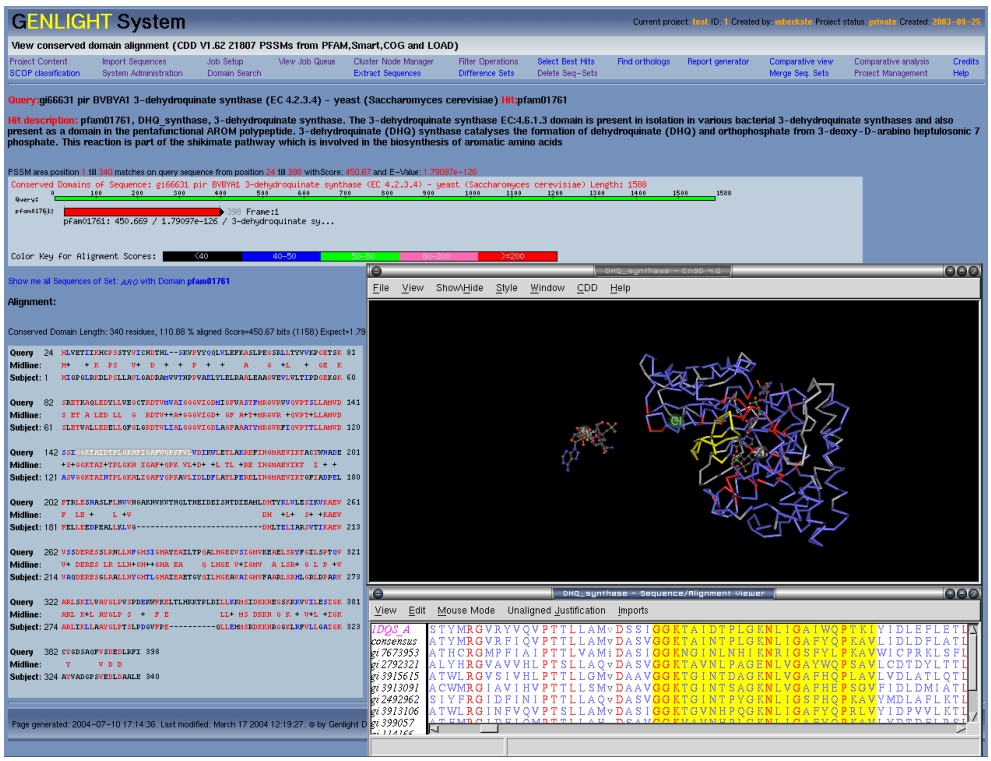


Figure 5.4: Visualization of the three dimensional location of a protein sequence (marked gray in the pairwise alignment and yellow in the multiple CDD alignment and the MMDB structure model) using the external viewer application *Cn3D*.

3D location of conserved regions of the protein query with external viewer applications like *Cn3D* [MBPS+02] (see Figure 5.4 for an example) and to directly retrieve three dimensional model data for further structure based studies.

To search in HMM based databases like Pfam, TIGRFAM, etc., Genlight makes use of the *hmmpfam* [Edd98] program from the HMMer package. The database PRINTS and BLOCKS are searched with *PoSSuMsearch* employing PSSM family models and fast chaining of PSSM matches.

A further advantage of the integration of a variety of different databases and search methods is the ability to balance method specific deficiencies in the detection of certain homologies and the incompleteness of protein family databases. As shown in Figure 5.5, different methods and screenings versus different databases reveal different results. Hence it is often not sufficient to screen the sequences under consideration only against one database. Genlight can easily perform searches in several different databases and allows to access the persistently stored results in an integrated manner.

5.4.5 Supported protein classification schemes

When dealing with complete proteomes of multiple organisms, the focus may shift from detailed single protein to complete proteome analyses, depending on the level of detail necessary to an-

Figure 5.5: Searches in different databases reveal different results. In this example a multi domain protein from *S.cerevisae* consisting of five functional domains, was screened versus Pfam, TIGRFAM, and Smart using method *hmmpfam*, and versus PRINTS and BLOCKS using *PoSSuMsearch*. Observe that the screening against TIGRFAM detects only three and searching in PRINTS using *PoSSuMsearch* even only one domain. Responsible for these varying results are missing signatures/models in some databases.

swer a certain type of question. For this purpose, functional classification systems allow a broader view on and comparison of an organism's genome or proteome by classifying genes in a relatively small number of functional categories. Hereby, the number of available categories and hence the employed abstraction level depends on the classification schema utilized. For maximal flexibility the integration of different classification systems with different abstraction levels is essential. For the functional classification of sequences, Genlight integrates the COG (Cluster of Orthologous Groups) [TKL97, TNG+01] database containing annotated clusters of prokaryotic proteins and its eukaryotic complement KOG (euKaryotic cluster of Orthologous Groups) [TFJ+03] including their crude, but widely used functional classification schema. The COG/KOG databases are an attempt to classify the complete complement of proteins (both predicted and characterized) encoded by complete genomes. Each COG and KOG respectively is a group of three or more proteins that are inferred to be orthologs, i.e., they are direct evolutionary counterparts and assumed to share a common function. The COG release integrated into Genlight consists of 4,873 COGs, which include 136,711 proteins (71% of all encoded proteins) from 50 bacterial genomes, 13 archaeal genomes, and 3 genomes of unicellular eukaryotes. The eukaryotic counterpart KOG includes proteins from 7 eukaryotic genomes: three vertebrates (the nematode *C.elegans*, the fruit fly *D.melanogaster* and *H.sapiens*), one plant (*A.thaliana*), two fungi (*S.cerevisae* and *S.pombe*), and the intracellular microsporidian parasite *Encephalitozoon cuniculi*. The KOG version integrated into Genlight consists of 4,852 clusters of orthologs, which include 59,838 proteins, or approximately 54% of the 110,655 analyzed eukaryotic gene products. Classification of nucleotide as well as protein sequences into one of the 25 functional COG/KOG categories can be performed with Genlight due to homology to COG/KOG sequences. Inside COG/KOG these categories are further classified into 4 top-level categories:

1. Information Storage and Processing;
2. Cellular processes and signaling;
3. Metabolism;
4. Poorly characterized.

Although this functional classification is very crude, it is still widely used, in particular when dealing with prokaryotic sequences. See Figure 5.6 for an example of the classification of the *C.glutamicum* proteome into COG categories with Genlight. A more detailed functional classification of sequences can be achieved with the Pfam clan [FMSB+06] or TIGR role functional classification systems also integrated into Genlight.

Pfam clans allow a grouping of single Pfam families into a hierarchical classification called clans and hence provide a hierarchical view of a diverse range of proteins families. A clan contains two or more Pfam families that have arisen from a single evolutionary origin. Evidence of their evolutionary relationship is usually determined by similar tertiary structures, or when structures are not available, by common sequence motifs. Pfam clans provide a level of detail which is a little bit broader than the protein family level. In its latest release, Pfam contains 262 different clans consisting of 1676 single Pfam families. Classification into clans in Genlight is performed due to homology to Pfam models. In contrast to Pfam clans, TIGR roles are a more detailed two level classification concept

Functional class	Functional Category	#COG Clusters in this category	#distinct matched clusters	#classified seqs	percentage	pie chart
INFORMATION STORAGE AND PROCESSING:						
[J]	Translation, ribosomal structure and biogenesis	234	123	154	5.04 %	
[A]	RNA processing and modification	21	1	1	0.03 %	
[K]	Transcription	173	53	126	4.12 %	
[L]	Replication, recombination and repair	215	72	126	4.12 %	
[B]	Chromatin structure and dynamics	8	0	0	0.00 %	
Sum (this category only):				**407**		
CELLULAR PROCESSES AND SIGNALING:						
[D]	Cell cycle control, cell division, chromosome partitioning	45	13	19	0.62 %	
[Y]	Nuclear structure	1	0	0	0.00 %	
[V]	Defense mechanisms	46	11	35	1.14 %	
[T]	Signal transduction mechanisms	104	20	44	1.44 %	
[M]	Cell wall/membrane/envelope biogenesis	171	54	98	3.21 %	
[N]	Cell motility	40	3	3	0.10 %	
[Z]	Cytoskeleton	7	0	0	0.00 %	
[W]	Extracellular structures	0	0	0	0.00 %	
[U]	Intracellular trafficking, secretion, and vesicular transport	93	15	27	0.88 %	
[O]	Posttranslational modification, protein turnover, chaperones	171	53	85	2.78 %	
Sum (this category only):				**311**		
METABOLISM:						
[C]	Energy production and conversion	238	77	122	3.99 %	
[G]	Carbohydrate transport and metabolism	208	77	133	4.35 %	
[E]	Amino acid transport and metabolism	234	118	183	5.99 %	
[F]	Nucleotide transport and metabolism	86	50	67	2.19 %	
[H]	Coenzyme transport and metabolism	156	77	116	3.79 %	
[I]	Lipid transport and metabolism	82	36	63	2.06 %	
[P]	Inorganic ion transport and metabolism	193	85	163	5.33 %	
[Q]	Secondary metabolites biosynthesis, transport and catabolism	71	16	26	0.85 %	
Sum (this category only):				**873**		
POORLY CHARACTERISED:						
[R]	General function prediction only	673	146	260	8.51 %	
[S]	Function unknown	1347	156	201	6.58 %	
Sum (this category only):				**461**		
Total number of classified sequences:				**2052**	**67.12 %**	
Number of unclassified sequences (no COG homolog found):				**1005**	**32.88 %**	

Figure 5.6: Functional classification of the *C.glutamicum* proteome based on homology to the COG database with Genlight. In this example, the classification criteria, which can be user defined, was a *BLASTP* hit with an E-value of at most 10^{-5} and the additional requirement that the matching region covers at least 50 percent of the matched COG sequence. Assignment of functional categories was performed based on the highest scoring hit.

Classification schema	Abstraction level	Classification criteria
COG/KOG functional categories	25 categories organized in 4 top-level categories.	Best *BLAST*, *FASTA*, *SSEARCH* hit vs. COG/KOG sequence database satisfying additional constraints (e.g. E-value or coverage).
TIGR roles	Two level classification with 105 sub roles organized in 21 main roles.	Best hit to TIGRFAM model library satisfying E-value constraint.
Pfam clans	262 clans representing 1676 Pfam families.	(Best) hit to Pfam model library satisfying E-value constraint.
CATH protein structure classification	Hierarchical classification of protein domain structures at the four major levels: (C)lass, (A)rchitecture, (T)opology, and (H)omologous superfamily.	(Best) hit to CATH pHMM model library satisfying E-value constraint.
SCOP structural classification	Hierarchical classification of proteins at the class, fold, superfamily, and family level.	(Best) *psiBLAST* hit vs. SCOP sequence database satisfying additional E-value constraint.

Table 5.3: Supported classifications schemas, abstraction level, and employed classification criteria.

consisting of main roles and sub roles and allow to classify proteins on the basis of matches to TIGRFAM pHMM family models. Currently this classification schema distinguishes 105 sub roles organized into 21 main role categories.

Beyond the above mentioned functional classification systems, Genlight supports two classification schemas focusing on structural similarities and differences, namely the multi level hierarchical classification systems SCOP and CATH.

For a recapitulating overview of supported classification schemas and the classification criteria employed inside Genlight, see Table 5.3.

5.4.6 Gene ontologies: a unifying vocabulary for cross database queries

For historical reasons, different database use different terminologies and naming conventions, introducing an artificial heterogeneity which makes it complicated to query these resources in a combined fashion. For example, if we were searching for new targets for antibiotics, we might want to find all the gene products that are involved in bacterial protein synthesis, and that have significantly different sequences or structures from those in humans. Currently, one database describes these molecules as being involved in "translation", whereas another uses the phrase "protein synthesis" and hence without knowledge about these different naming conventions, it is difficult to find functionally equivalent terms and thus related or equivalent sequences. An attempt to overcome this problem is the Gene Ontology (GO for short) project [Con00]. GOs provide a controlled vocabulary to describe genes and gene products, addressing the problems resulting from different terminologies currently used in different databases. Therefore, GO contains three structured controlled vocabularies (ontologies) that describe gene products in terms of their associated biological processes, cellular

components, and molecular functions in a species-independent manner. The usage of GO terms by collaborating databases enables uniform queries across them.

In Genlight, such terms can be assigned by the system, inferred from the respective assignment of the integrated databases. Genlight contains mapping to GO terms for entries from the Pfam, TIGRFAM, Smart, and PRINTS databases, and hence allows to query results from these resources using GO terms. Since GO is a structured ontology, queries at different levels of abstraction are possible. For instance, one can use GO terms to find all gene products in an organisms genome that are involved in signal transduction, or one can zoom in on all the receptor tyrosine kinases.

5.4.7 User defined sequence databases

In addition to the databases integrated into Genlight described above, the system allows to import any sequence collection available in GenBank, Swiss-Prot, or FASTA format. Such an user defined sequence collection can contain just a few sequences that should be analyzed with Genlight's integrated analysis methods, a complete proteome, or even a whole sequence database like Swiss-Prot or UniProtKB/TrEMBL. Imported sequence collections are treated as normal private *Seq-sets* in the user's project workspace or can be made available as a system-wide resource by the Genlight administrator. This means that major public sequence databases, genomes and proteomes of model organisms of interest, or proprietary in-house sequence data can be imported and made accessible system-wide if required. This concept saves resources and avoids data redundancy.

5.4.8 Asynchronous distributed execution of sequence analysis tasks

The comparison of whole genomes or proteomes, or their use as query sets for searches in large databases like GenBank or Swiss-Prot is a challenging and time consuming task. To compare, for instance, the mouse proteome to the human proteome by pairwise sequence comparison, 53,847 (International Protein Index (IPI) release 3.28, April 2007 [KDW+04]) single homology searches with programs like *BLAST*, *FASTA*, or the time-consuming Smith-Waterman full alignment method versus the human proteome set comprised of 68,020 protein sequence (IPI 3.28, April 2007) have to be performed. To handle such comparison tasks in a multi-user capable, interactive system with the need of guaranteed, adequate response times, the individual comparison calculations have to be done asynchronously. To accomplish this, Genlight uses queuing mechanisms and a distributed client-server approach with multiple compute clients carrying out parts of comparison- or other sequence analysis tasks (see Figure 5.1 on page 145). After submission of such a task, it is asynchronously executed by the distributed execution system. This process neither influences other users of Genlight, nor does it block the interactive work with the system while processing. Once computation has finished, the results are directly accessible inside the system. Sequence analysis tasks can be added to, suspended from execution, or deleted from the systems job queue at any time. Even changes to job priorities affecting execution order are possible. The complete queue management can be performed with a comfortable web interface.

To process queued entries, the system has its own scheduling and dispatching component, which allows a parallel, distributed execution of comparison jobs and can form a virtual cluster system of regular workstations for high throughput analysis tasks. This allows to use existing compute re-

Query Set	DB Set	Method	Running time [hh:mm:ss]
H.pylori	*H.influenzae*	*BLASTP*	00:00:32
H.pylori	*V.cholerae*	*psiBLAST* (10 iterations)	00:03:22
L.innocua	*L.monocytogenes*	*BLASTN*	00:00:27
H.pylori	CDD	*rpsBLAST*	00:03:41
S.typhimurium	*A.thaliana*	*BLASTP*	00:03:48
H.pylori	Pfam	*hmmpfam*	04:41:33
H.sapiens	*M.musculus*	*BLASTP*	02:17:30

Table 5.4: Running times for different comparison methods using the Genlight virtual cluster system with 25 SUN UltraSparc II CPUs on different workstations.

sources and often eliminates the necessity for a dedicated compute cluster. The integrated dispatcher splits sequence comparison tasks between two *Seq-sets* into smaller work units of user defined size which are subsequently distributed to the available compute nodes, thus balancing the overall load over the available compute resources. The two major strengths of this approach are the complete integration into one system without the need for difficult to install third party batch-queuing systems and a high robustness of the system. The latter is achieved by methods to insure data integrity, like a backlog technique, transactions, and connection supervision, during distributed execution. Compute nodes can be added to and deleted from the virtual cluster system by starting or stopping the Genlight client component on a workstation, via the cluster node management interface (see Figure 5.7). The cluster node management interface also provides information about each compute node, the node's status, and the overall progress of the task currently processed.

The possibility to temporarily include or exclude certain computers at any time, makes the virtual cluster very flexible. For instance, departmental workstations can be excluded during working hours and included during the night to use idle compute-power.

Since the Genlight client application is available for different hardware architectures and operating systems, and platform independent communication between Genlight-server and compute clients is implemented, the compute resources of computers running different operating systems can be bundled in one heterogeneous virtual cluster system. Up to now, Genlight supports (and is tested on) Sparc/Solaris, x86/Solaris, x86/Linux, and Mips/IRIX platforms. The distributed computing approach allows comparisons of complete genomes or proteomes in short time periods. The overall running time is nearly inversely proportional to the number of CPUs used (see Figure 5.8). For concrete examples of running times of different sequence comparison and analysis methods using Genlight's virtual cluster system see Table 5.4.

5.5 Database schema

Genlight uses the ORDBMS (Object Relational Database Management System) PostgreSQL for data storage and access, though Genlight has been designed with other SQL99 compliant DBMS in mind. The system makes use of PostgreSQL's object oriented features and transaction capabilities to ensure data integrity and consistency.

Figure 5.7: The virtual cluster management interface gives a detailed overview of the progress of a sequence analysis job, estimated duration, and real-time status of compute nodes. Users with system administrator privileges can start, stop, add, or remove compute nodes at any time.

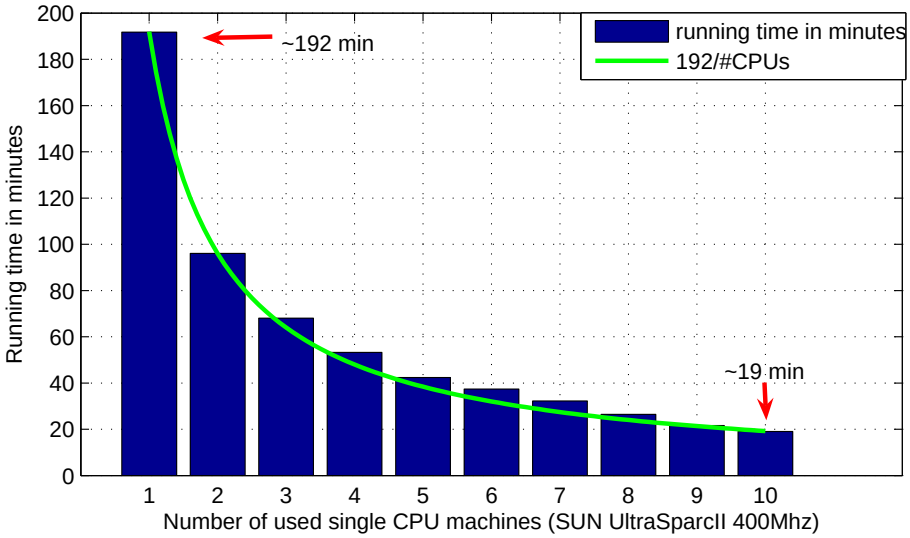


Figure 5.8: Scaling behavior of the distributed computing approach. Running times for a *BLASTP* comparison of *Helicobacter pylori* proteome consisting of 1487 proteins and the complete Swiss-Prot database for different numbers of CPUs used.

The interactive character of Genlight, its concurrent multi user capabilities, and its need to store calculated data on demand requires more complexity in the implementation of the data model than it would be the case in single user systems with pre-calculated, static data. In particular the "dynamics" introduced by the requirement to store data from user specified sequence analysis operations on demand, have to be supported in the data model.

In the following we describe by example the set oriented concept of Genlight's data model and show how this functional concept is represented in the physical data model. Recall, that the set oriented concept consisting of *Seq-sets* and *Hit-sets*, described in section 5.4.1 on page 145, is one of Genlights fundamental concepts. The information contained in *Seq-sets* and *Hit-sets* is persistently stored in database tables. In case of *Hit-sets*, the database tables reflect the method-specific attributes of the sequence comparison methods that generated the data contained in a *Hit-set*. For this purpose, Genlight employs method specific template tables for the various supported sequence comparison methods. Each time a *Seq-set* or *Hit-set* needs to be generated, by import, comparison of *Seq-sets*, or through the application of one of the *Seq-set* and *Hit-set* operations described in section 5.4.2 on page 146 by the user, a new database table for this *Seq-set* or *Hit-set* is automatically created as a child table by table inheritance from the method specific template table. Hence, this process of instantiation results in a newly generated table, which can be seen as an instance of the template. This template instance is then unambiguously referenced by a catalog table entry that stores additional parameters like generating method, e.g., for a *Hit-set* table the sequence comparison method used, parametrization of the method, etc. Accordingly, catalog tables contain information globally characterizing a complete template instance instead of each of the instance's entries. They are further necessary for the organization and administration of template instances. We now explain the concept of template instantiation with a small excerpt of Genlight's data model. Figure 5.9 gives a

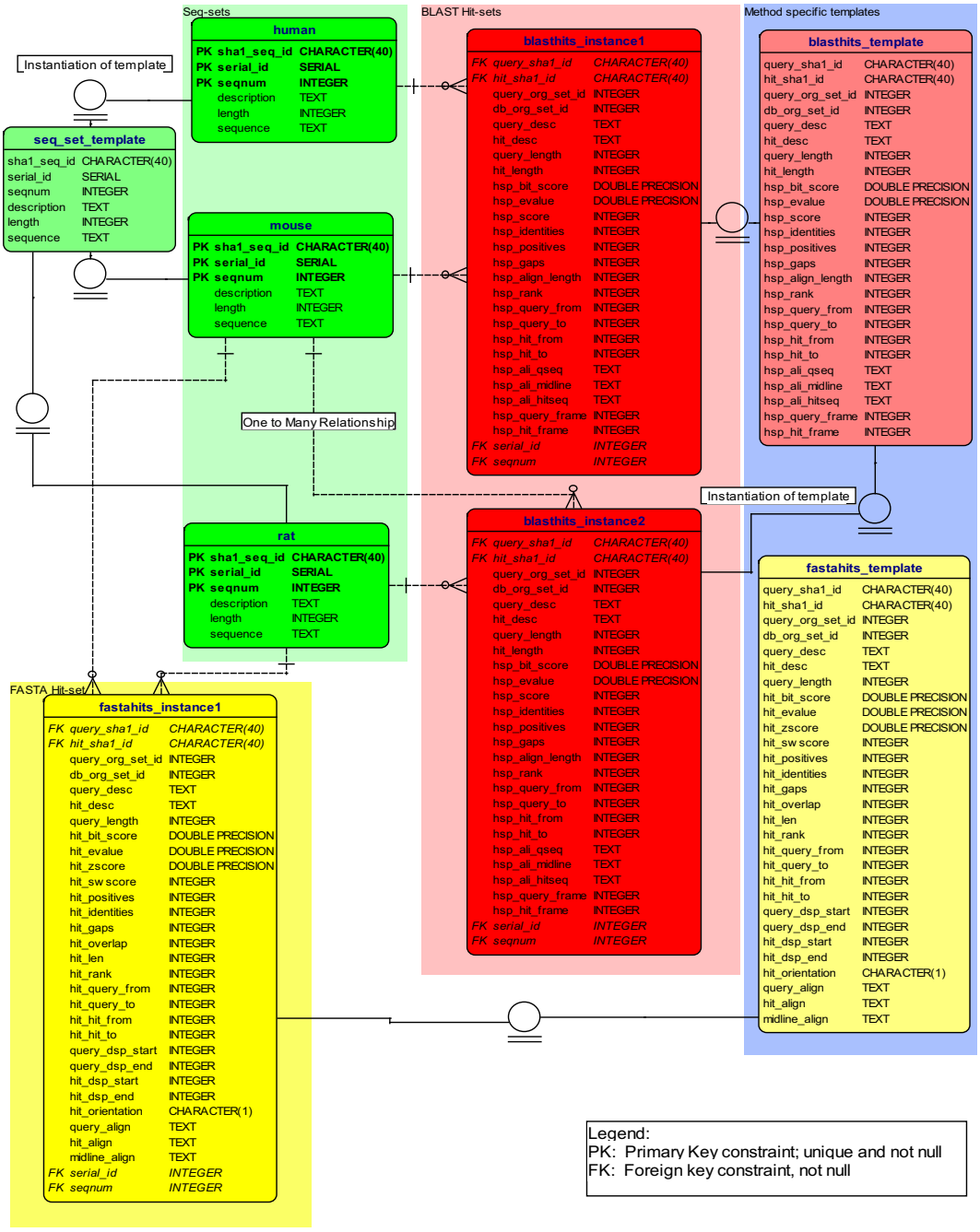


Figure 5.9: Instantiation of template tables and relationships between database tables representing *Seq-sets* and *Hit-sets*. For details see corresponding text.

snapshot of the database schema containing *Seq-sets* and *Hit-sets*. The three sequence sets `human`, `mouse` and `rat` are instantiated from the *Seq-set* template table `seq_set_template`. Template instantiation is shown by arcs annotated with a circle/equal sign connecting template and instance. A *Seq-set* contains beside the attributes DESCRIPTION, LENGTH, and SEQUENCE three unique primary keys, namely the sequence identifier SHA1_SEQ_ID, the serial SERIAL_ID for auto numbering of entries necessary for sequential processing of a *Seq-set*, and SEQNUM denoting the position of an entry in the set of sequences at import time of this *Seq-set* [3]. At import time SHA1_SEQ_ID acts even as a primary key constraint for the template table `seq_template`, but this constraint can be violated through set operations over time. However, SHA1_SEQ_ID uniquely identifies a sequence entry in a *Seq-set*. In section 5.5.1 we will give more details about the SHA1_SEQ_ID identifier concept.

Further shown in Figure 5.9 are three method specific *Hit-sets*, namely `blasthits_instance1`, `blasthits_instance2`, and `fastahits_instance1` instantiated from the two template tables `blasthits_template` and `fastahits_template`, respectively. Observe, that a *Hit-set* $\mathcal{H}_{\mathcal{Q},\mathcal{D}}$ is the result of a sequence comparison operation between two *Seq-sets*, say $\mathcal{Q}$ and $\mathcal{D}$ and defines a relation between the sequences of $\mathcal{Q}$ and the sequences of $\mathcal{D}$. This is reflected in the data model by attributes QUERY_SHA1_ID and HIT_SHA1_ID of a *Hit-set* table which are foreign key constraints for the SHA1_SEQ_ID sequence identifier of two *Seq-sets*. For example, table `blasthits_instance1` defines a relation between sequences from *Seq-sets* `human` and `mouse`, and table `fastahits_instance1` defines a relation (homology as detected by *FASTA*) between *Seq-sets* `mouse` and `rat`. Between a *Seq-set* $\mathcal{Q}$ and a *Hit-set* $\mathcal{H}_{\mathcal{Q},\mathcal{D}}$ defined over this $\mathcal{Q}$, there is a *one to many relationship*, since one sequence from $\mathcal{Q}$ may match multiple sequences in $\mathcal{D}$. Since a *Hit-set* $\mathcal{H}_{\mathcal{Q},\mathcal{D}}$ defines a *one to many relationship* for both involved *Seq-sets* $\mathcal{Q}$ and $\mathcal{D}$, it establishes a *many to many relationship* between sequences from $\mathcal{Q}$ and sequences from $\mathcal{D}$.

5.5.1 The internal sequence identifier concept

A central point for a database driven sequence analysis system is the ability to uniquely identify a single sequence. For this purpose, many different identifier concepts have been developed in recent years. Such developments were in particular furthered by the maintainers of large public sequence collections like GenBank or UniProtKB/TrEMBL. Since Genlight can use sequence data from any resource, even proprietary in-house sequences, it cannot rely on the existence of a specific public identifier like a *GenBank accession number* or *Swiss-Prot protein id*. Therefore, Genlight needs its own internal identifier concept that allows to uniquely identify a sequence, taking the following attributes into account:

- the sequence itself,
- its annotation/description and
- its *Seq-set* membership at import time.

This allows to differentiate between distinct entries for the same sequence even if they share the same description in two different *Seq-sets*, which is essential for some of Genlight's *Seq-set* set

[3] Observe that, though being identical at import time, SERIAL_ID and SEQNUM can differ over time for instance due to application of set operations.

operations. To combine these three attributes into one unique identifier, Genlight computes SHA-1 hash keys from these attributes. These 160 bit long keys act as primary keys in the database tables representing a *Seq-set* (see attribute SHA1_SEQ_ID in table `human` in Figure 5.9) and hence as foreign keys in database tables representing a *Hit-set* (see attributes QUERY_SHA1_ID and HIT_SHA1_ID in table `blasthits_instance1` in Figure 5.9).

5.5.2 The handiness of the set oriented concept

We will now give some examples for the handiness of the set oriented concept, and describe exemplarily how its implementation in the physical data model allows to answer biological relevant questions using standard SQL-queries easily. Such queries are automatically generated by the scripts implementing Genlight's web interface depending on user specified criteria.

User defined Hit-sets filtering

Filtering of *Hit-sets* based on user defined criteria is a straightforward task in Genlight. Assume that we are only interested in homologous sequence pairs satisfying a certain E-value constraint, say having an E-value lower than 10^{-10}. Let $\mathcal{H}_{\mathcal{A},\mathcal{B}}$ be a *Hit-set* containing pair relationships resulting from a *BLAST* based comparison between the two *Seq-sets* $\mathcal{A}$ and $\mathcal{B}$, then the following SQL-query selects all entries of $\mathcal{H}_{\mathcal{A},\mathcal{B}}$ satisfying this constraint.

SELECT * **FROM** $\mathcal{H}_{\mathcal{A},\mathcal{B}}$ **WHERE** `hsp_evalue` $\leq$ 10^{-10};

Identification of conserved gene orders and genome rearrangements

When analyzing complete genomes or proteomes of prokaryotes, a common task is not only the identification of homologous sequences, but also the identification of conserved gene orders since they reveal information about global genome rearrangements, such like, translocations, inversions, duplications or deletions. These genome-wide mutations are believed to be more neutral than local mutations such as substitutions, insertions, and deletions. Therefore, phylogenetic investigations of genome rearrangement events are less biased by the hypothesis of neutral evolution [PH88]. Genlight can support the detection of genome rearrangements as follows. Let $\mathcal{A}$ and $\mathcal{B}$ be two *Seq-sets* containing genes from two prokaryotic genomes in the order of their occurrence in the genome and a *Hit-set* $\mathcal{H}_{\mathcal{A},\mathcal{B}}$ containing homology results from a *BLAST* based comparison[4] of sequences from $\mathcal{A}$ versus sequences from $\mathcal{B}$. Then, the SQL-Query

SELECT `t1.seqnum,`
`t2.seqnum,`
`t3.hsp_bit_score`
FROM $\mathcal{A}$ `t1,`
$\mathcal{B}$ `t2,`
$\mathcal{H}_{\mathcal{A},\mathcal{B}}$ `t3`
WHERE `t1.sha1_seq_id = t3.query_sha1_id` **AND**
`t2.sha1_seq_id = t3.hit_sha1_id` **AND**

[4] *FASTA* or *Smith-Waterman* based homology information can also be used.

```
          t3.hsp_rank = 1
ORDER BY t1.seqnum  ASC;
```

determines triples consisting of two sequence numbers identifying a gene's or protein's position in each of the genomes and an alignment score as a measure of the sequences homology. Due to the constraint HSP_RANK=1, only the top ranked hit, if it exists, for each sequence from $\mathcal{A}$ is considered. Genlight allows to visualize these informations in form of a XY plot directly, with optional color coding of hit quality. In such a XY plot, conservation of gene order and genomic rearrangements become directly visible (see Figure 5.10).

Determination of bidirectional-best hits

Sequence comparison methods like *BLAST* or *FASTA* introduce a kind of asymmetry into sequence comparisons due to their implied scoring functions. That is, a query/hit pair of sequences (S_1, S_2) resulting from a unidirectional comparison of S_1 versus S_2 may achieve a different score than the reverse pair (S_2, S_1). This causes problems in homology based function assignments based on *best* hits, since although S_2 may be the highest scored (best) homolog found for S_1 when using S_1 as the query, this neither necessarily implies that S_1 is found with the same score nor that this is the highest score, when using S_2 as query. This issue is often addressed by using bidirectional-best hits instead of unidirectional-best hits. A bidirectional-best hit is defined as follows.

Definition 17 Let $\mathcal{A} = \{g \mid g \text{ is gene of organism 1}\}$, $\mathcal{B} = \{h \mid h \text{ is gene of organism 2}\}$ be two *Seq-sets* and $S_{\mathcal{A},\mathcal{B}} : \mathcal{A} \times \mathcal{B} \longrightarrow \mathbb{R}$, $S_{\mathcal{B},\mathcal{A}} : \mathcal{B} \times \mathcal{A} \longrightarrow \mathbb{R}$ be two scoring functions that assign to each pair $g \in \mathcal{A}$, $h \in \mathcal{B}$ a score expressing the homology between g and h, and h and g respectively. Then, pair g, h is a bidirectional best pair or hit iff. $S_{\mathcal{A},\mathcal{B}}(g,h) \geq S_{\mathcal{A},\mathcal{B}}(g,h')$ for all $h' \in \mathcal{B}$ and $S_{\mathcal{B},\mathcal{A}}(h,g) \geq S_{\mathcal{B},\mathcal{A}}(h,g')$ for all $g' \in \mathcal{A}$.

With Genlight's set oriented data model, method specific bidirectional-best hits can be easily determined from two reverse, unidirectional comparisons. More precisely, let $\mathcal{A}$ and $\mathcal{B}$ be two *Seq-sets* and $\mathcal{H}_{\mathcal{A},\mathcal{B}}$, $\mathcal{H}_{\mathcal{B},\mathcal{A}}$ be two *Hit-sets* containing homology results from two unidirectional comparisons between sequences from $\mathcal{A}$ and sequences from $\mathcal{B}$. Then, the subsequently given SQL-query selects pair relationships satisfying the bidirectional-best hit criteria.

```
SELECT t1.query_desc,
       t2.query_desc,
FROM H_{A,B} t1,
     H_{B,A} t2,
WHERE t1.query_sha1_id = t2.hit_sha1_id  AND
       t1.hit_sha1_id = t2.query_sha1_id  AND
       t1.hsp_rank = t2.hsp_rank  AND
       t2.hsp_rank = 1;
```

Observe, that unidirectional-best hits are specified with the attribute constraint HSP_RANK=1. Attribute HSP_RANK ranks the pair relationships found by the comparison method according to their method specific score. This is due to concerns of speed and allows to retrieve the best pair of a *Hit-set* efficiently and avoids to maximize the score over several entries.

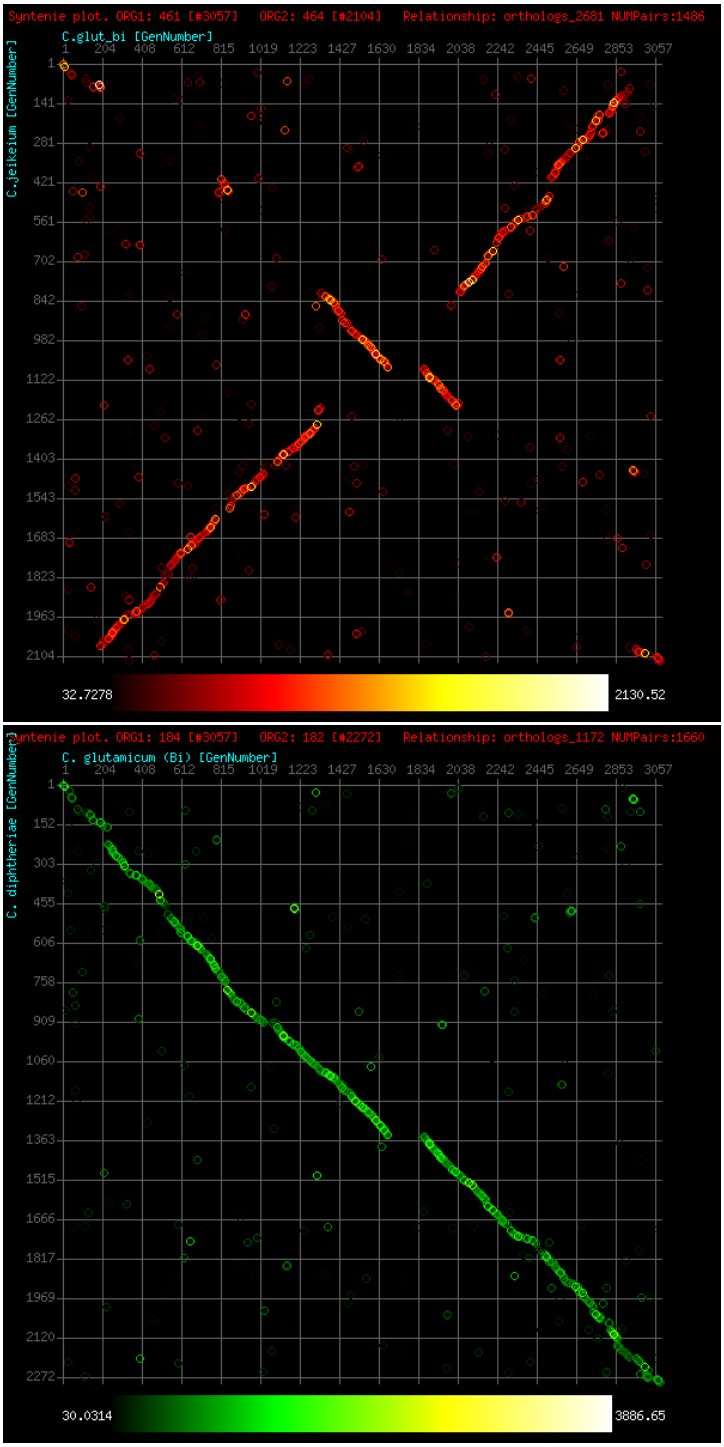


Figure 5.10: Visualization of genomic rearrangements between the actino bacterias *C.glutamicum* and *C.jeikeium* (top), and *C.glutamicum* and *C.diptheriae* (bottom) using Genlight. The upper plot clearly shows genomic inversions. Responsible for the gap visible in the lower plot is a bacteriophage inserted into the genome of *C.glutamicum* [KBB+03]. Relationships between genes were determined with a *Hit-set* containing bidirectional-best *BLAST* hit information. Data points are colored based on alignment bit scores.

Differential comparative analyses

Differential comparative genome analyses have been successfully used to identify species specific genes or genes responsible for a certain phenotype [RZG+02]. They can help, for instance, to identify genes responsible for the pathogenicity of an organism by comparison of the organism's genome with closely related apathogenic organisms. Such analyses involve the inclusion of genes with a homolog in organisms sharing a certain phenotype (the pathogenicity) and an exclusion of genes with a homolog in an organism not showing this phenotype (being apathogenic). Assume that we have the proteomes of two organisms A and B showing a certain phenotype and the proteome of a third organism C not showing this phenotype. Then, it would be interesting to identify genes of A also existing in B, but not in C under the assumption that these genes, or at least some of them, are responsible for the observed common phenotype of A and B.

Such questions can be answered with Genlight's data model using standard SQL queries. More precisely, let $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ be three *Seq-sets* and $\mathcal{H}_{\mathcal{A},\mathcal{B}}$ and $\mathcal{H}_{\mathcal{A},\mathcal{C}}$ be two *Hit-sets* defining a (homology based) relationship between members from $\mathcal{A}$ and $\mathcal{B}$ and $\mathcal{A}$ and $\mathcal{C}$ respectively. Then SQL-query

```
SELECT sha1_seq_id   FROM A
      INTERSECT ( SELECT query_sha1_id  FROM H_{A,B})
      EXCEPT    ( SELECT query_sha1_id  FROM H_{A,C});
```

determines all sequence identifiers of sequences in $\mathcal{A}$ that have a counterpart in $\mathcal{B}$ defined by $\mathcal{H}_{\mathcal{A},\mathcal{B}}$ and no counterpart in $\mathcal{C}$ according to $\mathcal{H}_{\mathcal{A},\mathcal{C}}$. To determine the desired result we employ the SQL concept of select chaining. That is, we combine several **SELECT** statements with **INTERSECT** and **EXCEPT** clauses. The semantics of these clauses are analogical to the homonymous operations in set theory.

Select chaining also enables us to ask queries of type "Which genes of $\mathcal{A}$ have a homolog in $\mathcal{B}$ **OR** $\mathcal{C}$?". The logical **OR** can be modeled using the **UNION** clause as follows:

```
SELECT sha1_seq_id  FROM A
      INTERSECT (( SELECT query_sha1_id  FROM H_{A,B})
                   UNION
                 ( SELECT query_sha1_id  FROM H_{A,C})
                );
```

5.5.3 More complex queries using computed sequence attributes

Beside the concept of *Hit-sets* for the pairwise comparison of sequences, Genlight integrates various protein family and motif databases with their specific search methods and sequence classification schemas (see section 5.4.4 on page 150 and section 5.4.5 on page 152). Once these database searches are performed for the sequences of some *Seq-set*, their results are persistently stored in database tables also instantiated from method specific templates. The computed information can then be combined for more complex analysis tasks. Figure 5.11 shows an excerpt of Genlight's data model after the sequences of *Seq-set* `human` were screened versus databases TIGRFAM, Pfam, Smart, CDD, COG, and PRINTS.

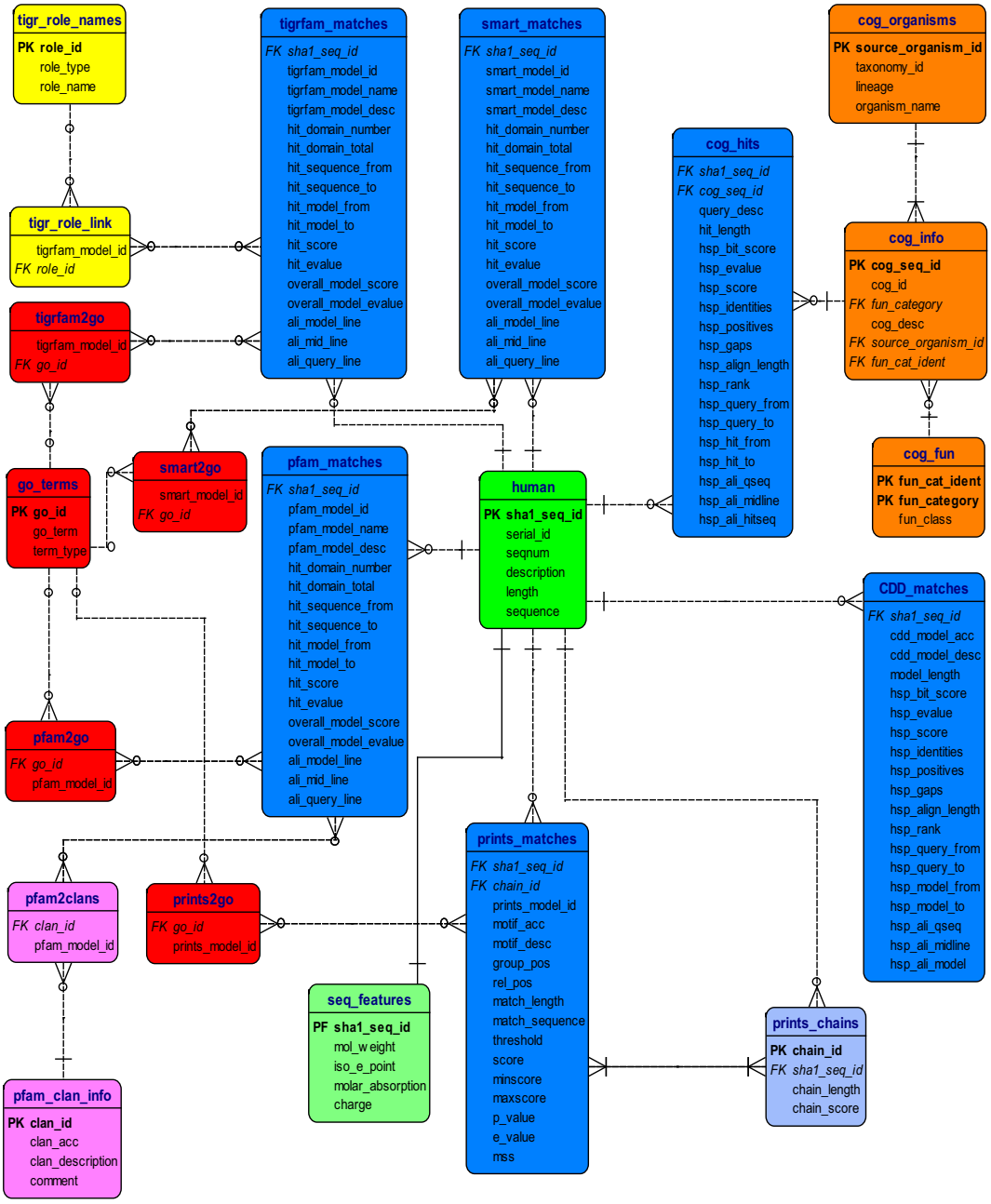


Figure 5.11: An excerpt of Genlights data model showing relationships between a *Seq-set* (table `human`), results of different sequence analysis methods (tables `tigrfam_matches`, `pfam_matches`, `prints_matches` etc.), gene ontology mappings (tables `pfam2go`, `prints2go` etc.), and functional classification schemas (tables `tigr_role_names`, `tigr_role_link` etc.). For details, see corresponding text.

By incorporation of the results of these database searches and classification information, it is now possible to perform more complex queries on the data. Assume that we are interested in all human proteins containing a death domain, since we know in advance that death domains are related to apoptosis, the programmed cell death in multi cellular organisms. The death domain [CI95] is a heterodimerization domain present in several proteins involved in apoptotic signal transduction. Over-expression of these proteins usually leads to cell death. Since death domains constitute a heterogeneous domain family, they are described by several Pfam models belonging to Pfam clan *Death Domain Superfamily*. Thus to identify proteins with a death domain, we should query the human protein *Seq-set* for sequences classified with a certain confidence to this Pfam clan instead of looking for matches to a single *Death Domain* Pfam model. This is accomplished with the following SQL query[5]:

```
SELECT description
FROM human
WHERE sha1_seq_id  IN
  ( SELECT query_sha1_id
    FROM pfam_matches
    WHERE hit_evalue ≤ 10^-5  AND
              pfam_model_id  IN
                  ( SELECT pfam_model_id
                    FROM pfam2clans
                    WHERE clan_id  IN
                       ( SELECT clan_id
                         FROM pfam_clan_info
                         WHERE clan_description='Death Domain Superfamily'
                       )
                  )
  );
```

To take only sufficiently confident classifications into account, we restrict the Pfam matches to those having an E-value lower or equal to 10^{-5}. In the above query we make use of sub queries instead of joining the involved tables. Here all sub queries may return multiple rows which are processed by the comparison operator **IN**. Operator **IN** is similar to the $\in$ operator in mathematics.

We already addressed the problem of different terminologies used in different databases and described the *Gene Ontology* controlled vocabulary as an approach to solve this issue. In the following, we will show how we can make use of GO terms to collect results from different data sources employing different naming conventions. Observe, that in the above query example we looked for death domains as single indicator for a sequence to be involved in apoptosis and the only evidence factor for the existence of such a domain, taken into account, was a hit to a Pfam model. Basically, we face here two problems.

1. It is unlikely that the death domain is the only domain linked to apoptosis, which is a process that involves an orchestrated series of biochemical events. Hence, there may be proteins not containing a death domain, but which nevertheless play a central role in the process of cell death. With the above query we would fail to detect these proteins.

[5]For the following examples we use the data model as given in Figure 5.11.

2. Domain/motif databases, even large ones like Pfam, are often incomplete in certain areas or contain suboptimal models which do not detect all related sequences (of a family). Hence, a different model taken from a different database may further detect proteins involved in apoptosis.

To overcome these pitfalls we have to consider results from searches in multiple different databases. To address the problem of different naming conventions we use GO terms for querying instead of attributes of certain database entries. This procedure implies that entries of databases like Pfam, TIGRFAM, etc. can be mapped to GO categories. Such mappings, which are built with expert knowledge and are freely available for certain domain and motif databases, are also integrated into Genlight. Therefore, it is possible to (i) look up from the gene ontology (all) entries linked to apoptosis, (ii) to look up with the mapping information the model identifiers of databases like Pfam, TIGRFAM, or Smart, and (iii) to finally return the sequences matching these models. This can be performed with the following SQL-query.

```
SELECT description FROM human WHERE sha1_seq_id IN (
   (SELECT query_sha1_id FROM pfam_matches WHERE hit_evalue ≤ 10^-5 AND pfam_model_id IN (
       SELECT pfam_model_id FROM pfam2go WHERE go_id IN (
          SELECT go_id FROM go_terms WHERE name='apoptosis'
          )
       )
   )
   UNION
   (SELECT query_sha1_id FROM tigrfam_matches WHERE hit_evalue ≤ 10^-5 AND tigrfam_model_id IN (
       SELECT tigrfam_model_id FROM tigrfam2go WHERE go_id IN (
          SELECT go_id FROM go_terms WHERE name='apoptosis'
          )
       )
   )
   UNION
   (SELECT query_sha1_id FROM smart_matches WHERE hit_evalue ≤ 10^-5 AND smart_model_id IN (
       SELECT smart_model_id FROM smart2go WHERE go_id IN (
          SELECT go_id FROM go_terms WHERE name='apoptosis'
          )
       )
   )
   UNION
   (SELECT DISTINCT t1.sha1_seq_id FROM prints_matches t1, prints_chains t2
    WHERE t1.chain_id=t2.chain_id AND
          t2.p_value ≤ 10^-3 AND
          t2.chain_length > 2 AND
          t1.prints_model_id IN (
             SELECT prints_model_id FROM prints2go WHERE go_id IN (
                SELECT go_id FROM go_terms WHERE name='apoptosis'
             )
          )
   )
);
```

This query selects all sequences (more precisely their descriptions) from our *Seq-set* `human` that contain a match of reasonable quality to one of the databases Pfam, TIGRFAM, Smart, and PRINTS

(stored in tables `pfam_matches`, `tigrfam_matches`, `smart_matches`, etc.) by chaining **SELECT** statements, specific to retrieve certain database search results, with the **UNION** clause. Additionally, to be picked up by the query, sequences have to match database entries that are linked with the mapping tables `pfam2go`, `tigrfam2go`, `smart2go`, and `prints2go` via attribute GO_ID to the GO term *apoptosis*.

5.5.4 Genlight as a data warehouse

Figures 5.9 on page 161 and 5.11 on page 167 clearly show that Genlights data model is not fully normalized, according to normalization forms known in relational database theory [Ken83]. This process of *denormalization* is common for systems intended for *OLAP* (On Line Analytical Processing). In contrast to *OLTP* (On Line Transaction Processing) systems, which are designed for handling a high volume of transactions, like inserting/deleting or modifying relatively small amounts of data, and hence often employing highly normalized data models, *OLAP* systems are primarily designed for read-only reporting and data analysis. E.g., for the purpose of data analyses a high level of normalization often has an disadvantageous effect on query response times, since normalization splits related data into several database tables, which, have to be combined again for querying, using costly table joining operations.

For Genlight short query response times are essential due to the interactive character of the system. Consequently the employed data model, which is optimized for short query response times by intended denormalization through the introduction of data redundancy[6] has more characteristics of a typical *OLAP* than an *OLTP* system. This is not in contradiction to the fact, that Genlight also carries out several data generating analysis tasks, which lead to extensive data insertions and modifications, since these tasks are executed asynchronously by the distributed execution approach, and hence are independent from guaranteed response and completion times.

Typical representatives of *OLAP* like systems, are the so called data warehouses. They follow an information integration concept and integrate data from heterogeneous and distributed sources into one system to allow a global view on the data by enabling source data comprehensive queries and data analyses. Following this view, Genlight also shares several characteristics with data warehouses. In particular, by the persistent storage of computed results in a query-optimized data model that allows to combine heterogeneous information in complex queries, Genlight builds, while analyzing a certain organism's genome or proteome, an integrated information resource from the computed sequence analysis results. This resource can be accessed by using standard SQL queries even from external applications as well as by a user-friendly web interface automatically generating the needed SQL queries and thus hiding the internals of the underlying data model from the user.

5.6 The Genlight user interface

For interaction with the system, we developed a flexible and powerful web-interface that, while containing a high information density, is still user-friendly and easy to use. It allows the user to set up sequence comparison jobs and to perform all operations on *Seq-sets* and *Hit-sets* described in this

[6] Genlight also employs B-Tree indexing of certain attribute fields to ensure short query response times.

chapter, or to import/export *Seq-set* and *Hit-set* information in a completely interactive way. For operations that cannot be handled interactively like the comparison of large *Seq-set* sets which are processed asynchronously by the distributed execution engine, the processing state and progress of computation is directly visible in the interface. Once computation has finished, these informations are dynamically updated and the newly computed results are available for further analyses. This is a major advantage in comparison to other systems, which use pre-computed data and present it in a static way. The generated results can be viewed in tabular as well as in graphical form, using intuitive visualizations. Throughout this thesis, we already presented visualizations generated by Genlight in various figures. For example the sequence to structure mapping figures in chapter 2 (see Figures 2.1 on page 9 and 2.2 on page 10), the visualizations of single PSSM matches and matches to PSSM family models in Figure 4.2 on page 108, the demonstration of functional classification of sequences into COG/KOG categories (see Figure 5.6 on page 155), or the visualization of genome rearrangements shown in Figure 5.10 on page 165.

The central place of Genlight's user interface, which is also the entry point after successful authentication and project selection, is the overview page of a project's workspace (see Figure 5.12). It shows all *Seq-sets*, *Hit-sets* and user defined filters available in the selected project and gives the user a comprehensive overview of already performed or ongoing analyses. Indicated are the processing states of certain analyses, i.e., which analyses for which *Seq-sets* are already computed, which are in processing state or waiting in the systems job-queue to be executed by the distributed execution engine.

From the project overview page, the information contained in a certain *Seq-set* or *Hit-set* is quickly accessible, following a hyper text reference. Information contained in a *Hit-set* is conveniently displayed in tabular (see Figure 5.13) as well as in graphical forms (see Figure 5.14). Descriptions of sequences in a *Hit-set* can be searched using exact patterns or regular expressions, allowing a quick navigation even in large *Hit-sets*. If the user is interested in further details of a *Hit-set* entry, like concrete alignment information, Genlight generates this information on the fly by comparing the two involved sequences of a *Hit-set* entry with the sequence comparison method that has been used to generate this specific *Hit-set*. Alignment informations are due to concerns of space not stored in the underlying database. Alignments are presented in colored textual as well as in graphical form (see Figure 5.15). In case of a comparison with a nucleotide sequence being involved, additional information about start- and stop-codons and potential open reading frames are dynamically determined and presented in graphical form (see Figure 5.16).

Results that arose from screenings of a *Seq-set* versus one of the integrated domain and family databases are also presented in a comprehensive manner including additional derived information like *Gene Ontology* classifications. This allows a unifying view on the determined database search results. For an example see Figure 5.17 which shows the results of a search with a protein sequence from *H.pylori* versus the PRINTS database using *PoSSuMsearch*. Observe that both database matches, although being different, are classified to the same GO categories.

For searches in databases with structure information available, Genlight can retrieve structure information from the source database and allows to visualize these information using structure viewer applications like *Jmol* or *Rasmol*. This allows for direct mapping of sequence features, like homology

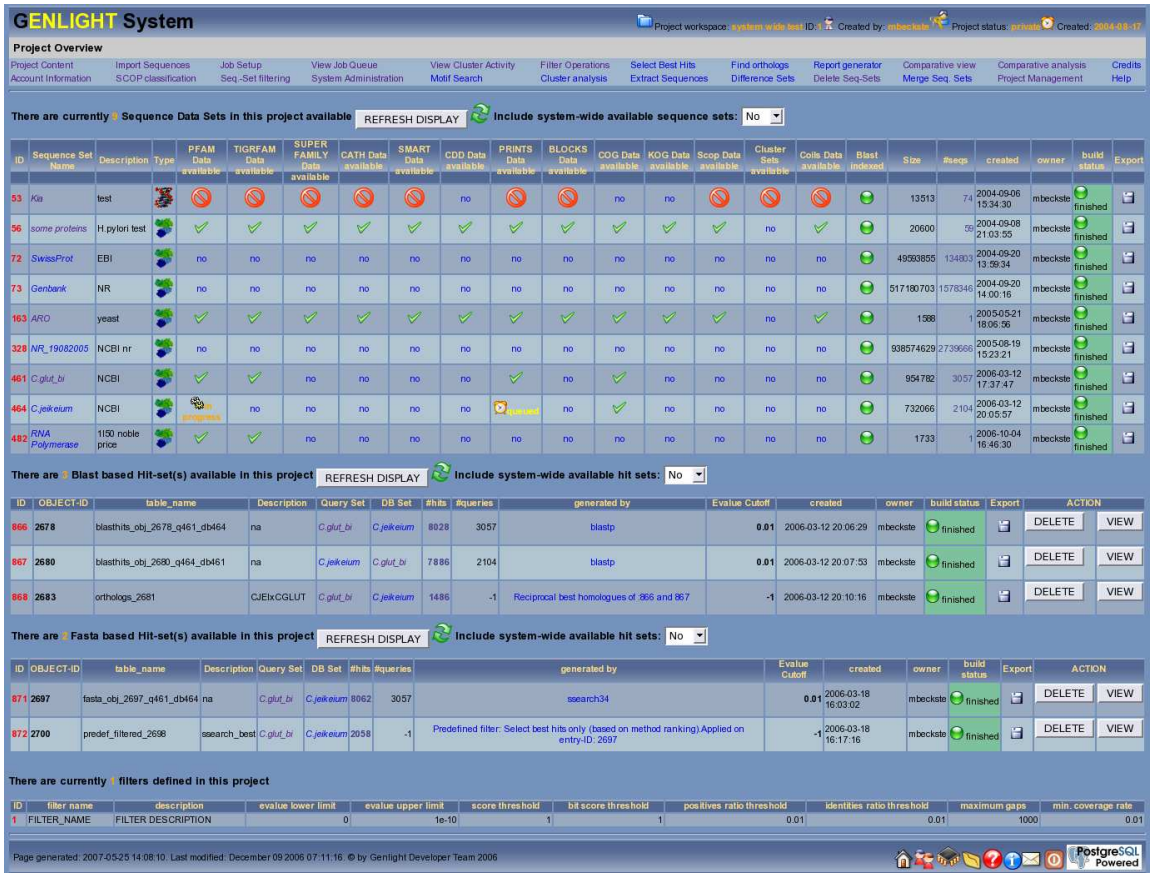


Figure 5.12: Genlight's project workspace overview, showing information about *Seq-sets* and *Hit-sets* of the current project. In the upper right corner, information about the current project is displayed, like its name, the project id, project owner, privacy status, and the creation time. The top of the page shows the Genlight navigation menu. This menu is present on almost all pages of the system and allows an easy and quick navigation to different sections of the system. The upper table gives information about available *Seq-sets* and the availability of screening results versus databases like Pfam, TIGRFAM, Smart, etc., whereas the two centered tables show computed *Hit-sets*. The table at the bottom of the page shows user defined *Hit-set* filters available in this project.

Figure 5.13: Tabular view of the content of a *Hit-set*. For each entry (matching sequence pair), several attributes characterizing the entry are displayed. Entries can be displayed in sorted order using different sort keys, like E-value, score, rank, number of identities, number of positives or query/hit description.

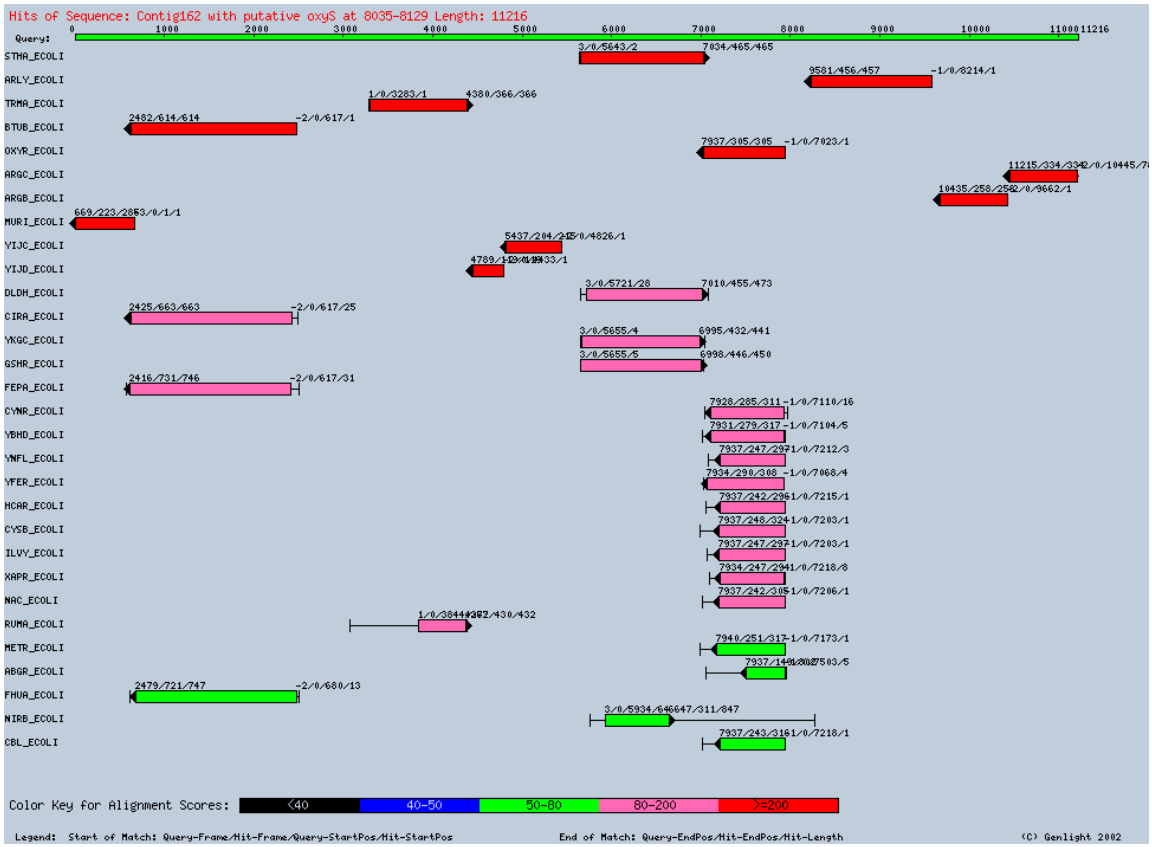


Figure 5.14: Graphical view of a *Hit-set*. Shown are matches resulting from a *BLASTX* search of a contig sequence of the alpha proteobacteria *Serratia proteamaculans* versus the *Escherichia coli* proteome. Matches are colored according to the obtained alignment score.

Figure 5.15: Visualization of *on the fly* generated *BLASTP* alignment information for a *Hit-set* entry. In the coloured textual alignment, identical amino acids are marked red, similar amino acid (positives) are marked blue. Gaps are shown as black dashes.

Figure 5.16: Visualization of a *FASTY* alignment with additional start/stop codon and open reading frame information. Potential open reading frames are marked as blue arrows, start (stop) codons are shown as small green (red) boxes in the upper part of this figure. In this example, the aligned regions correspond to a potential open reading frame in the frame one translation of the shown nucleotide sequence.

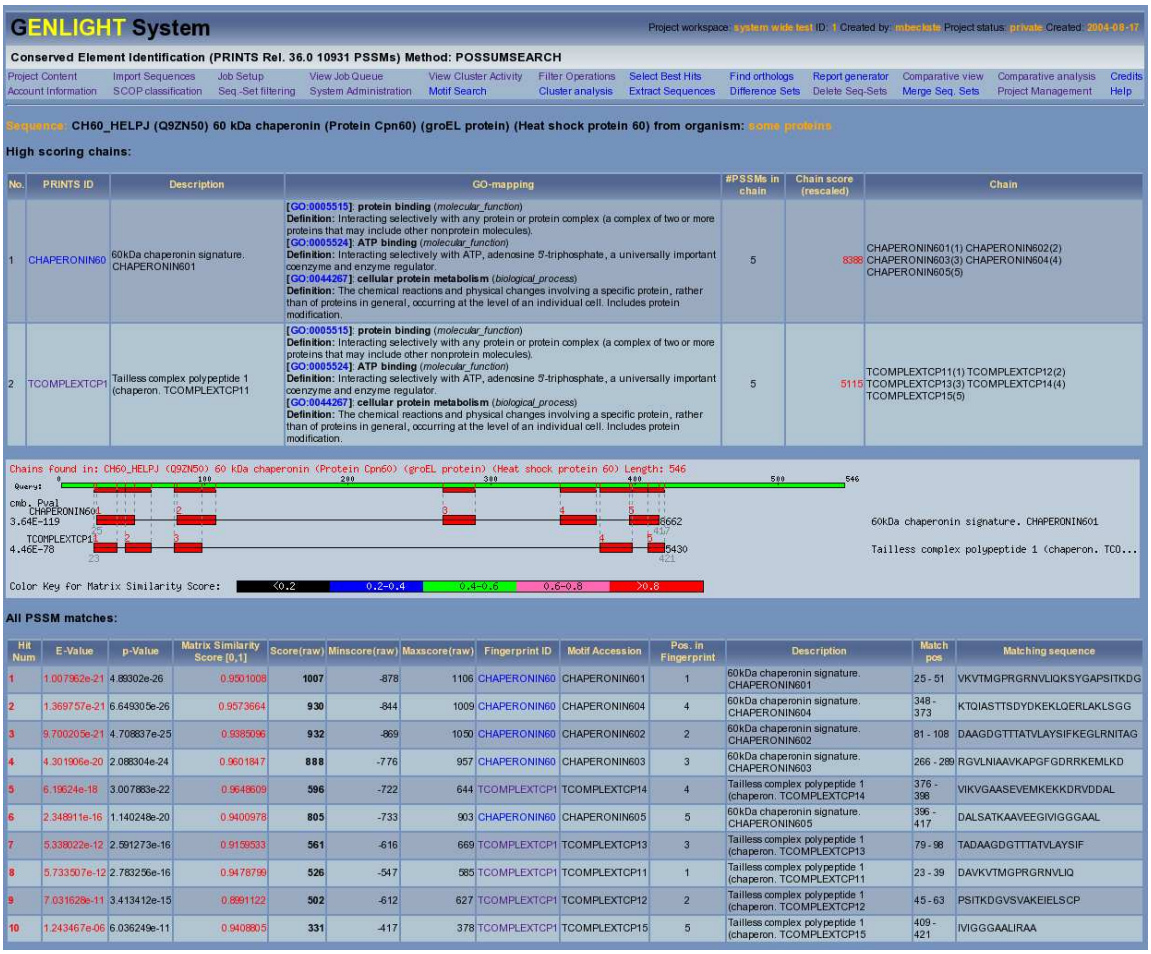


Figure 5.17: Visualization of database search results from a screening versus the PRINTS database using *PoSSuMsearch*. Genlight allows to map obtained results directly to GO categories.

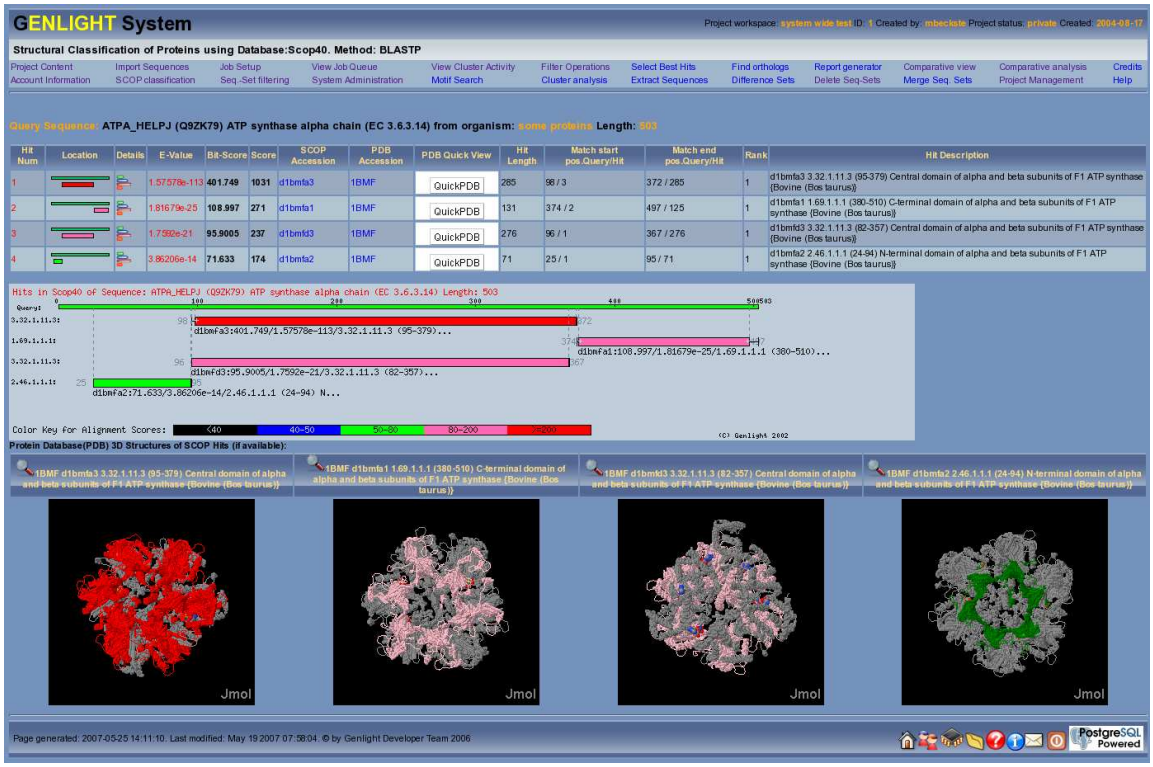


Figure 5.18: Mapping of homology information determined by a *PsiBLAST* search versus the SCOP sequence database on PDB structures corresponding to SCOP entries.

information to the structure models, by automatically generating viewer specific script files. For an example see Figure 5.18.

Wherever possible Genlight also references to external data sources allowing the user to retrieve additional information not contained inside the system. E.g., references to web pages describing a certain family complement the information of the publicly available HMM collections integrated into Genlight with a variety of additional information. These information are quickly accessible from Genlight's interface following a single hyper text reference.

5.7 Genlight case studies

5.7.1 Detection and analysis of the Smh gene family in maize

In [MBG+03] we detected a new gene family in maize (*Zea mays*), called *Single myb histone (Smh)* family, with the help of the Genlight prototype system. We screened GenBank, ZmDB [DRF+03] and Pioneer Hi-Bred (PHI)[7] expressed sequence tag (EST) databases with Genlight's built-in sequence comparison and domain search methods for the occurrence of the myb-like domain of human telomeric protein TRF1. TRF1 binds to repeats at chromosome ends and has homology to the DNA-

[7] http://www.pioneer.com/

binding domain of the Myb family of transcription factors, but unlike most Myb related proteins, TRF1 carries only one rather than multiple Myb-like binding motifs. We identified several maize ESTs that encoded proteins with a single N-terminal myb-like domain. Together, the EST and additional cDNA library screens uncovered cDNAs from five related genes. The deduced protein sequences from five different full-length cDNAs revealed a family of small basic proteins. The cDNAs belonged to an uncharacterized gene family, the *Smh* gene family. Detailed sequence analysis with Genlight revealed a number of surprising features of *Smh* genes. The most remarkable aspect was their triple-motif structure, which has not been previously described in any system, plant, animal, fungal, or bacterial. Namely, *Smh* genes have

- (a) an N-terminal myb like or SANT domain of the homeodomain-like superfamily of 3-helical-bundle-fold proteins,
- (b) a central region with homology to the globular domain of linker histones H1/H15, and
- (c) a strong prediction signature for a coiled-coil domain near the C-terminus.

See Figure 5.19 for the gene model of *Smh1* and an excerpt of the underlying Genlight analysis results. Table 5.5 gives more details about all members of the *Smh* family.

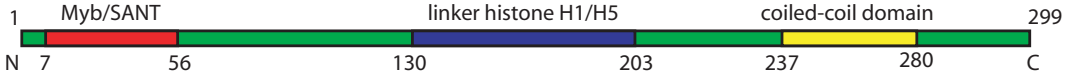
Additional large scale database searches with Genlight versus GenBank and Swiss-Prot revealed that *Smh*-type genes are plant specific and include a gene family in *Arabidopsis thaliana* and one gene (PcMYB1) of parsley (*Petroselinum crispum*). Various wet-lab experiments with a chosen member of the *Smh* family (*Smh1*) showed the ability to bind telomeric DNA repeats in vitro.

5.7.2 Analysis of Xenopus laevis expressed sequence tag clusters

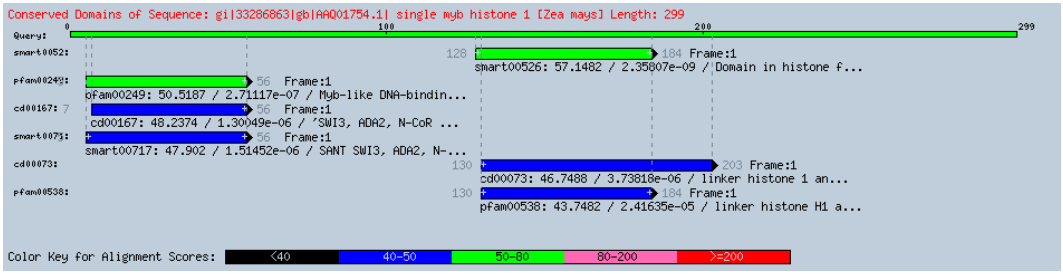
Agglomerative clustering of Expressed Sequence Tags (EST) sequences is a widely used method for analyzing the transcriptome of a genome. Especially in organisms where the genome sequence is not (yet) sequenced, the EST data is a valuable source of information. In [SBB$^+$] Genlight was used for extensive analysis of 31,353 tentative contig (TC) and 40,877 singleton sequences resulting from clustering and assembly of all 350,468 ESTs of the african claw frog *Xenopus laevis* as were available in November 2003. *Xenopus laevis* is a major model organism for early embryonic vertebrate development. Since its genome is not fully sequenced yet, in particular due to problems introduced by its pseudo-tetraploidity, extensive analysis of available EST information seems to be promising and may lead to new insights in embryonic vertebrate biology.

Our analysis of *X.laevis* ESTs described in [SBB$^+$] focused on the identification of full length contigs, representing potential new, yet unknown genes from *X.laevis*. Therefore, sequences were subject to *BLASTX* and *FASTY* homology searches with Genlight in NCBI's non-redundant protein database (NR), the proteomes of five major model organisms (*H.sapiens*, *M.musculus*, *R.norvegicus*, *C.elegans*, *D.melanogaster*), *X.laevis* and the closely related *X.tropicalis*. We chose *FASTY*, which is a version of the *FASTA* program that compares a DNA sequence to a protein sequence database by translating the DNA sequence in all six reading frames, since it allows in contrast to *BLASTX* for frame shifts. As EST sequences contain many sequencing errors, and even the assembly of clusters cannot correct all of these, frame shift tolerant database searching with *FASTY* should, though

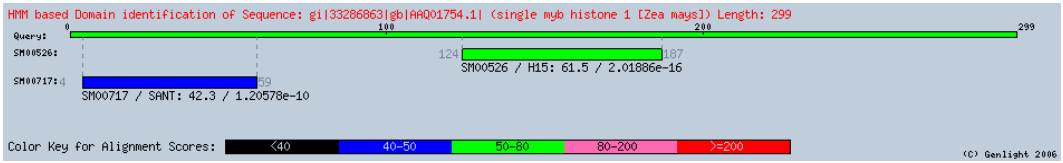
(A): *ZmSMH1* gene model/domain structure



(B): Results of screening vs. CDD



(C): Results of screening vs. Smart



(D): Results of coiled-coil prediction with program *COILS*

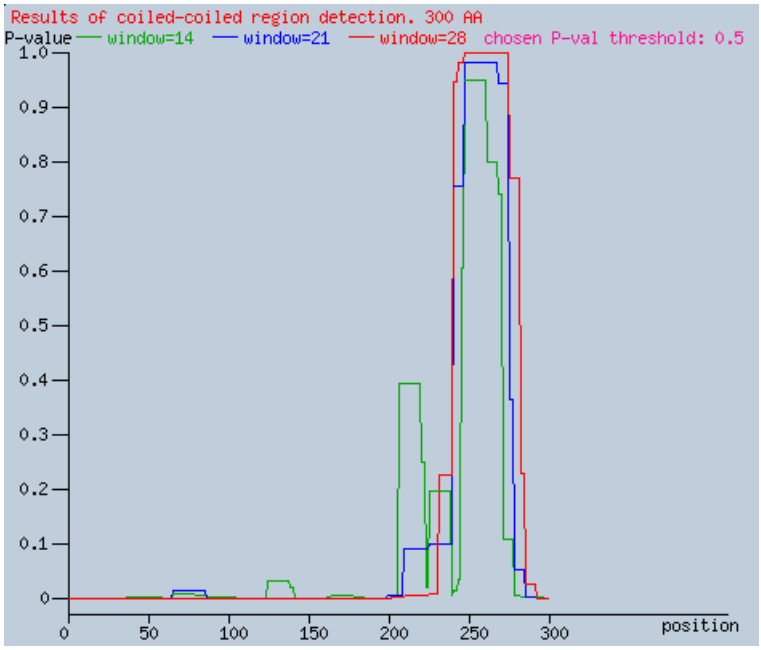
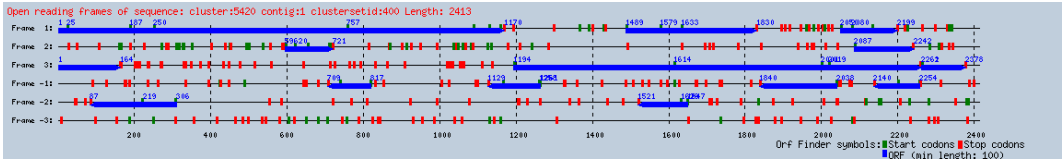


Figure 5.19: An excerpt of Genlight analysis results for ZmSMH1. For the derived domain structure of ZmSMH1 see (A). Database searches vs. CDD and Smart (see (B) and (C)) show near the N-terminus the myb-like or SANT domain and more centered the linker histone 1 and 5 domain. The analysis of potential coiled-coil formations reveals a strong prediction signature near the C-terminus (see (D)). Plot (D) shows the probability for coiled-coil formation using windows of width 14, 21, and 28.

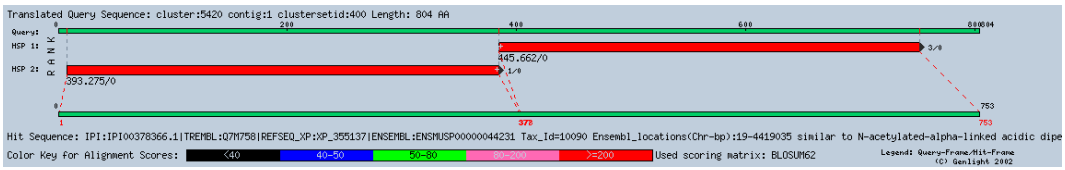
Gene	GenBank Accession No.	Protein (length;mass;pI)	Domain	Location
ZmSMH1	AY271659	299;32.6;9.07	SANT	R7-L56
			H15	D130-K203
			CC	K237-D280
ZmSMH3	AY280629	285;31.3;9.45	SANT	K7-L56
			H15	G111-V173
			CC	E219-E264
ZmSMH4	AY280631	288;31.3;9.33	SANT	K7-L56
			H15	P115-I173
			CC	V229-S260
ZmSMH5	AY280630	286;31.4;8.71	SANT	R7-M56
			H15	K120-V182
			CC	M226-V286
ZmSMH6	AY280632	298;33;8.78	SANT	R7-M56
			H15	N127-K200
			CC	M236-A297

Table 5.5: *Smh*-type genes and predicted protein features. Domain names and database identifiers for SANT are cd00167, SW13, ADA2, N-CoR, and TFIIIB DNA-binding domains from NCBI's conserved domain database (CDD); H15 is cd00073, linker histone1 and 5 domains, from CDD or smart00526, domain in histones families 1 and 5 from Smart; CC is the coiled-coil domain, which is indicated for any region where a peak probability exceeds 0.8. Column 3 gives the length of the amino acid sequence, the molecular weight in kilo Dalton [kD] and the isoelectric point.

(A): Open Reading Frames of *X.laevis* contig sequence



(B): *BLASTX* alignment of *X.laevis* contig sequence



(C): *FASTY* alignment of *X.laevis* contig sequence

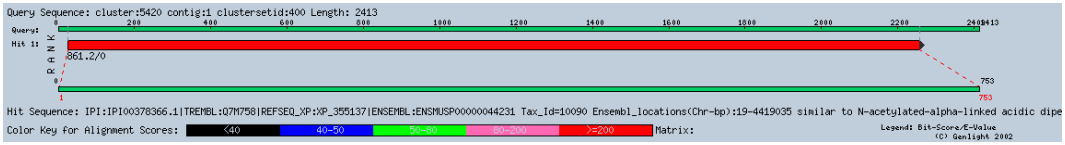


Figure 5.20: Comparison of a *BLASTX* alignment with corresponding full length *FASTY* alignment, generated with Genlight. Open reading frames are indicated by blue boxes in (a), start and stop codons by green and red boxes, respectivly. The assembled contig sequence has a frame shift at position 1150 from frame 1 to frame 3, generating two distinct HSPs in the *BLASTX* alignment (b). *FASTY* clearly corrects this frame shift and generates a full alignment (c).

more CPU-intensive, maximize the length of the resulting alignments and hence allow to identify full length contigs even if they contain frame shifts. For an example of the differences between *BLASTX* and *FASTY* screening results for a contig sequence containing a frame shift, see Figure 5.20.

In particular, for these large-scale *BLASTX* and *FASTY* homology searches, the distributed execution approach used in Genlight proved to be very powerful, and reduced, in combination with the application of external compute resources, the overall time needed for these tasks dramatically. For these analyses, we disposed external compute resources at the Center of Biotechnology of Bielefeld university as well as a huge cluster system at the supercomputing facility of Florida state university. Results determined with external resources were easily integrated using Genlight's XML/XSLT import layer (see Figure 5.1).

For full length contig identification different *Hit-set* filters were defined and applied to the *Hit-sets* containing *FASTY* screening results versus the model organisms. *FASTY* hits were categorized into four classes, representing the quality of the full length matches (see Figure 5.21).

- (1) Class 1 hits are defined as matches covering 100% of the sequence of a known protein. Additionally, the matched protein sequence has to start with a conserved methionine has to end at a conserved STOP codon.

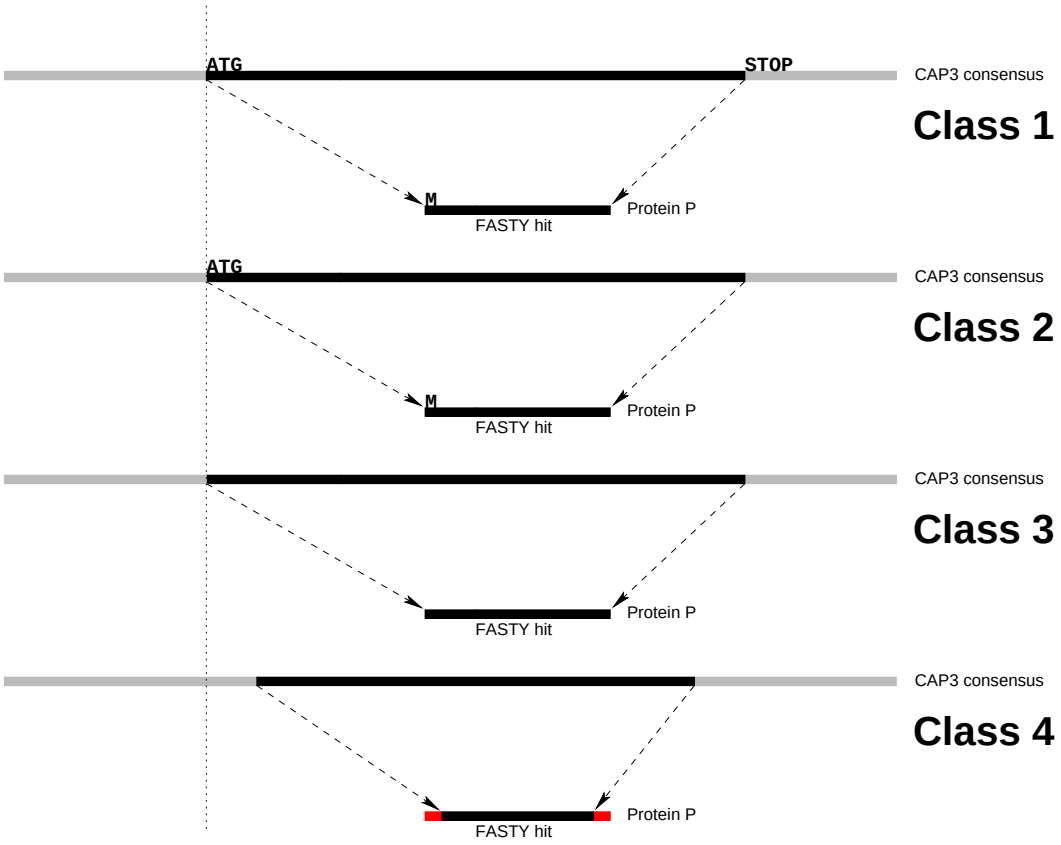


Figure 5.21: ESTs derived from different clones were compared to protein databases using *BLASTX* and *FASTY* and hits were categorized in 4 categories. Class 1 hits had to match the whole protein sequence and start with an ATG in the TC and an methionine (M) in the protein and the hit had to end at a STOP codon. Class 2 hits had to match the whole protein sequence, start with an ATG in the TC and M in the protein. Class 3 had to match the full protein sequence (without further restrictions), class 4 had to cover the protein over almost its full length, allowing the match to start or end maximal 10 amino acids after/before the start or end of the protein.

Class	NR	Human	Mouse	Rat	Fruit fly	C.elegans	X.laevis	X.tropicalis
Comparison method: BLASTX								
1	3,942	1,760	1,765	1,455	219	140	2,918	495
2	5,050	2,067	2,076	1,736	311	233	3,104	541
3	7,792	2,647	2,919	2,592	392	283	3,898	590
4	12,389	5,587	5,841	3,078	2,071	1,856	5,024	1,033
Comparison method: FASTY								
1	5,139	2,347	2,337	1,930	268	190	3,862	660
2	6,243	2,692	2,671	2,248	383	296	4,119	721
3	9,576	3528	3,774	3,374	473	357	4,967	796
4	14,094	6,467	6,701	6,341	2,249	1,918	5,701	1,241

Table 5.6: Number of *X.laevis* contigs with full length *BLASTX* and *FASTY* hits in the non-redundant protein database (NR), five model organisms, and available *X.laevis* and *X.tropicalis* proteins. Lower quality categories include sequences from higher, more stringent categories.

- (2) Class 2 hits are defined as matches covering 100% of the sequence of a known protein. Additionally, the matched sequence has to include the initial methionine.
- (3) Class 3 hits are matches capable of covering 100% of the matched protein sequence with no additional constraints.
- (4) Class 4 hits are matches that cover the protein over almost its full length, allowing the match to start or end up to 10 amino acids after or before the start or end of the protein respectively.

Table 5.6 shows the numbers of full length sequences matching proteins for each model organism.

For a functional classification of the clustered *X.laevis* data set, a non-redundant sequence set was built by selecting in each cluster a single contig. This resulted in 26,187 sequences. This non-redundant data set was then classified based on homology to known proteins from the KOG database using *SSEARCH* (Smith-Waterman) with an E-value cutoff of 10^{-5}. 17,624 sequences (67.3%) had a hit against the KOG database under these constraints and could be assigned a functional category, see Figure 5.22.

5.7.3 Identification of potential drug targets in Helicobacter pylori

Subsequently, we describe the application of the Genlight system to detect potential drug targets in the human pathogen *Helicobacter pylori* using Genlight's integrated sequence analysis methods and capabilities for differential genome analyses. Parts of this study were carried out by the department of BioChem Informatics of Intervet Innovation GmbH, our project partner in the development of the Genlight system, and were published in [BMM+04].

H.pylori is a spiral shaped bacterium living in the stomach and duodenum of humans and in other mammalians [THHM92]. Uncontrolled *H.pylori* infections are a major factor for duodenal ulcers,

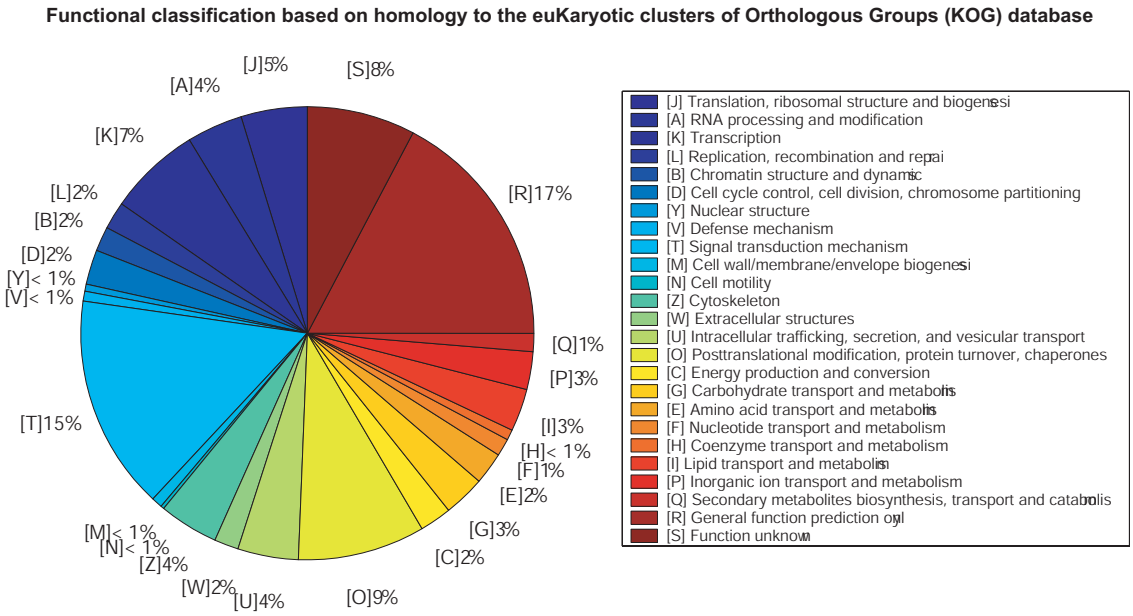


Figure 5.22: Functional classification of 26,187 non redundant *X.laevis* sequences based on similarity determined with *SSEARCH* to the euKaryotic clusters of Orthologous Groups (KOG) database. Classification criteria was best hit with an E-value of at least 10^{-5}.

gastric ulcers, stomach cancer, and non-ulcer dyspepsia [Mar02]. The sequencing of the *H.pylori* genome (strains *H.pylori* 26695 and *H.pylori* J99) offers the chance to develop highly specific treatments against *H.pylori* infections [TWK$^+$97, ALM$^+$99]. With the idea of minimizing toxicological effects, a perfect drug target protein should have low similarity to eukaryotic proteins [GK99]. Such genes are the most obvious candidates for drug targets. The strategy of this study was therefore to find all *H.pylori* proteins with low similarity to known eukaryotes.

The *H.pylori* J99 proteome consisting of 1,487 protein sequences was compared to various eukaryotic proteome sets (see Table 5.7) using the *BLASTP* sequence comparison method integrated into Genlight. From the resulting *Hit-sets* all *H.pylori* proteins that have no homolog in one of the eukaryotes with a *BLASTP* bit-score of at least 30 were extracted. After this initial filtering step only 226 *H.pylori* sequences remained.

In a subsequent analysis step the remaining 226 protein sequences were screened for putative drug/vaccine targets using Genlight's integrated protein family and motif databases Pfam, TIGRFAM, Smart, PRINTS, BLOCKS, and CDD. UreI, a well known drug target [STLDR98, BMSLDR01], which served as an internal control, was detected within this sequence set. UreI encodes an activated urea channel enabling urea access to intrabacterial urease at acidic pH. UreI is necessary for the survival of *H.pylori* at $pH < 4.0$ [MRHSM02].

Organism	**Number of protein sequences**
H.sapiens (IPI)	43,426
M.musculus (IPI)	40,742
R.norvegicus (IPI)	33,028
A.thaliana	26,192
C.elegans	22,439
D.melanogaster	16,106
S.cerevisae	6,195
P.falciparum	5,257
S.pombe	5,037
E.cuniculli	1,908
G.theta	451
Total	200,781

Table 5.7: Eukaryotic proteome sets used in the comparative analyses.

Pfam/BLOCKS Acc.	Description
PF06160	Septation ring formation regulator, EzrA
PF07432	Histone H1-like protein Hc1
PF01098	Cell cycle protein
PF00493	MCM2/3/5 family
PF01189	NOL1/NOLP2/sun family
PF03568	Peptidase family C50
IPB001182	Cell cycle protein

Table 5.8: Pfam and BLOCKS protein families known to be involved in the cell cycle process in bacteria.

Figure 5.23: Results of database searches with *H.pylori* protein FTSW_HELPJ in BLOCKS using *PoSSuMsearch* (top) and Pfam using *hmmpfam* (bottom). The results of both methods reveal strong evidence for FtsW family membership.

Identification of cell cycle proteins

Vital processes, like the process of cell division, are of special interest for drug development. Proteins involved in these processes are quite often fundamental and therefore are putative drug targets. In order to find such putative targets the remaining 226 protein sequences were screened for several families known to be involved in the cell cycle process in bacteria. More precisely, we queried the screening results for hits to the protein families given in Table 5.8. This query resulted in a potential target, the cell division protein FtsW. See Figure 5.23 for matches of this protein versus BLOCKS and Pfam models. FtsW is a polytopic membrane protein that is required for cell division in *E.coli* [KBD94, IJI$^+$89] and that is present in virtually all bacteria having a peptidoglycan cell wall [LA02, ISW$^+$89, HGPM98]. It is also discussed in the context of chemotherapeutic intervention of *M.tuberculosis* [DDBB02].

Identification of surface proteins

Surface proteins playing a role in pathogen-host interactions represent potential targets for vaccination [SLZA$^+$02]. To find such putative targets within the specific *H.pylori* proteins, the 226 protein sequences were analyzed for the appearance of surface exposed proteins using an *hmmpfam* screening versus Genlight's integrated Pfam database. Thirteen potential outer membrane proteins were found in this screening (data not shown here, see [BMM$^+$04]). These proteins could serve as potential candidates for vaccination.

The detection of UreI, FtsW, and outer membrane proteins clearly demonstrates the ability of Genlight to detect potential drug targets with its built in sequence analysis methods. In particular the differential or subtractive genome comparison approaches possible with Genlight allow to cut down a genome scale data set to a manageable size focusing on genes or proteins with interesting, user defined characteristics. UreA, UreB, VacA, and other well known pharmaceutical targets were not included in our analyses, since they do not pass the initial filtering step due to their significant similarity to eukaryotic proteins [SJ99]. Observe that this finding is in accordance to our strategy to detect only proteins with very low similarity to eukaryotic proteins.

5.8 Concluding remarks on Genlight

In this chapter, we described Genlight an interactive system for high-throughput sequence analysis with specific features for differential comparative genome analysis. Due to its distributed client server concept that allows an asynchronous execution of sequence analysis and comparison tasks, it is suitable for large scale analysis of nucleotide as well as protein sequence data in reasonable time. Genlight can facilitate available compute resources and build a virtual cluster system from normal department workstation computers. The overhead necessary for managing compute nodes inside the virtual cluster is negligible leading to an excellent scaling behavior with an overall running time nearly inversely proportional to the number of used CPUs. This approach allows sequence analyses on a scale otherwise only achievable with large and expensive cluster systems. Genlight contains integrated scheduling and queuing components. Consequently, there is no need for specialized third party software, which is often difficult to install and maintain.

A unique property of Genlight is its powerful, set oriented data model, that anchors the reusability of results in the system design and allows a protocol based step by step modeling of complex workflows. Genlight integrates and automates a variety of sequence analysis methods, including our PSSM matching software *PoSSuMsearch*, in a common environment, thus saving hours of tedious work that would otherwise be needed for performing all analysis tasks sequentially and manually in a non-integrated fashion. Due to the persistency of computed results and extensive query capabilities, Genlight can also be used as a data warehouse for sequence data and allows to build organism specific information resources. For easy access to and interactive work with the system we implemented a powerful, platform-independent web interface that allows to present derived results in a clearly arranged way and employs various result visualizations. We also remark, that Genlight is not just a prototype developed in an academic environment. It is a system ready for production and has already been installed and is successfully used for different research projects in the pharmaceutical industry. Furthermore its merits were already evaluated and have been proven valuable in several scientific studies [MBG+03, SBGA04, MMBB05, SBB+].

5.8.1 Potential future developments and system extensions

Although Genlight already supports a variety of sequence analysis methods and integrates various databases into a common environment, there are still several methods and databases which have not found their way into Genlight yet, but which would definitely fit in the context of the system and could benefit from the developed generic concepts and the high-throughput analysis infrastructure.

Integration of PoSSuMsearch and PSfamSearch as additional screening methods for pHMM based databases

In Genlight *PoSSuMsearch* is currently only used for searching with PSSM family models from the PRINTS and BLOCKS databases. We have shown that using PSSM family models in combination with fast index based PSSM searching as implemented in *PoSSuMsearch*, is an alternative to time consuming pHMM based methods in the context of protein family assignment and classification. In section 4.6.2 on page 127 we further demonstrated with the combined approach *PSfamSearch*, that

PSSM family models derived from pHMM seed alignments can be used for search space reduction and hence to speed up the search with pHMMs dramatically. Therefore, it is reasonable to integrate *PoSSuMsearch* as well as *PSfamSearch* as additional search methods for pHMM based databases like Pfam, TIGRFAM, CATH, Superfamily, or Smart into Genlight. Due to the generic and modular structure of Genlight, the integration of such new analysis methods is straightforward and can be easily accomplished. One can also think about the integration of further major pHMM based protein family collections, like for example the Panther database [MLUL+05].

Up to now, analysis methods and databases integrated into Genlight show a bias towards the analysis of amino acid sequences. However, noncoding DNA and RNA sequences attracted an increased attention in the last years since it became clear that they play a central role in many regulatory processes. To illustrate the potential importance of this non-protein coding genes consider the human genome. About 5% of the genome is evolutionarily conserved with respect to rodent genomic sequences, and therefore is inferred to be functionally important [Rat04, Mou02], but only about one-third of the sequence under such selection is predicted to encode proteins [Con01]. In [ENC04], the authors state that the collective knowledge about putative functional, noncoding elements, which represent the majority of the remaining functional sequences in the human genome is remarkable underdeveloped.

To the types of analyses necessary for identification and function predicition of these noncoding elements belong, among other things, the search for

- *cis*-regulatory elements, like transcription factor binding sites,
- microRNAs and their target sites, and
- structurally conserved noncoding RNAs.

Detection of transcription factor binding sites

The search for transcription factor binding sites can be efficiently accomplished with *PoSSuMsearch* using PSSMs from collections like TRANSFAC. Unfortunately, the licensing conditions of the commercial TRANSFAC database, the major collection of binding site signatures used in this field, does not allow an integration into a publicly available system like Genlight. Possibly, the actually relatively small open access database JASPAR [SAE+04] will offer a free alternative signature resource in the future, when the number of contained binding site signatures increases.

Methods for miRNA prediction

MicroRNAs (miRNAs for short) [Ruv01] are short single-stranded RNA molecules of about 21-23 nucleotides that post-transcriptionally regulate the expression of target genes by binding to the target mRNAs, and inhibit translation or facilitates cleavage of the mRNA. They are known to be involved in several regulatory processes and hence are of increasing importance also for the pharmaceutical industry. For their prediction, programs like *mirscan* [LGY+03] or *PoMiR II* [NKKZ06] were developed, which probably can be integrated into Genlight as additional analysis methods for DNA/RNA sequences.

Methods for miRNA target prediction

Widely used tools for the prediction of potential microRNA target sites are programs like *RNAhybrid* program [Reh06, RSHG04], *PicTar* [KGP+05], *miranda* [EJG+03], or *targetscan* [LSJR+03]. They compute the ability of a given miRNA to bind to a given target mRNA. Such programs could be integrated into Genlight as sequence analysis methods, establishing a pair relationship between two sets of nucleotide sequences, comparable to the already supported pairwise alignment methods *BLAST* and *FASTA*. Their results could then be represented in form of a method specific *Hit-set* table.

Integration of methods for the detection of structurally conserved noncoding RNAs

With the detection of microRNAs and other structurally conserved noncoding RNAs and their involvement in several central regulatory processes, a lot of effort was spent on the development of computational methods and tools for their reliable detection. In the following we briefly introduce existing methods and resources whose integration into Genlight could be reasonable. One important resource from this field is the Rfam collection of known structurally conserved noncoding RNAs [GJMM+05]. Rfam describes families of noncoding RNAs with covariance models [ED94] and can be searched with the *INFERNAL* program [Edd02]. In Genlight, *INFERNAL* could be integrated as an additional database screening method, comparable to the already integrated *PoSSuMsearch* and *hmmpfam* methods.

A different approach, combining description of and searching for structurally conserved noncoding RNAs is used in the thermodynamic matcher approach described in [HHG06]. This may also lead in the future to collections of programs suitable for searching for specific structurally conserved RNAs and may offer an alternative to the covariance models used in Rfam.

A program for the detection of new structurally conserved RNAs, which works in the absence of an available structure model is the *RNAz* program [WHS05]. *RNAz* predicts structurally conserved and thermodynamically stable RNA secondary structures in multiple sequence alignments. It can be used in genome wide screenings to detect new functional RNA structures, as found in noncoding RNAs and cis-acting regulatory elements of mRNAs.

Integration of metabolic pathway information

In recent years the Kyoto Encyclopedia of Genes and Genomes (KEGG for short) [KGH+06] has developed to a major resource for metabolic pathways, bio molecular interactions and reaction networks. A proper integration of the information contained in the KEGG database into Genlight would allow for a mapping of proteins to pathway data and hence a quick assessment of existing metabolic pathways in an organism's genome. Moreover, sophisticated differential screenings on the pathway level between certain organisms would become possible.

Provision of Genlight as a web service

Under the assumption of sufficiently available compute resources one can think about the development of a web service API for Genlight's analysis capabilities. This would allow an external user, or even an external system to participate from the implemented infrastructure and the distributed execution approach by sending sequence data via the web service to Genlight and receive, for instance, all results obtained from a database search in one or even all of the integrated motif and protein family databases.

6 Conclusions and prospects

6.1 Concluding remarks

The contributions made with this thesis can be subdivided into two parts, namely an algorithmic and a software engineering part. We presented not only several algorithmic contributions to the field of sequence analysis using PSSMs as approximate motif descriptors, but also implemented these newly developed algorithms in our search tool *PoSSuMsearch*. With the Genlight system, we further developed a production ready system, integrating *PoSSuMsearch* and a variety of existing sequence analysis methods in a common and user-friendly environment suitable for interactive high-throughput sequence analysis tasks.

Our first main contribution is the development of a new non-heuristic algorithm, called *ESAsearch*, to efficiently find matches of PSSMs in large databases. *ESAsearch* facilitates persistently stored enhanced suffix arrays for search space indexing and allows to search a database with a PSSM in sublinear expected time. We presented a detailed complexity analysis which revealed sublinear running time in the expected case, and linear running time in the worst case for sequences not shorter than $|\mathcal{A}^m + m - 1|$, where m is the length of the PSSM and $\mathcal{A}$ a finite alphabet. We tested algorithm *ESAsearch* in various experiments on nucleotide as well as amino acid data. In these experiments *ESAsearch* shows speedups of factor between 17 and 275 compared to the best previous methods for nucleotide PSSMs, and speedups up to factor 1.8 for amino acid PSSMs. Comparisons with the most widely used programs that all use variants of the *SPsearch* algorithm for searching with PSSMs even show speedups by a factor of at least 3.8.

Since *ESAsearch* benefits from small alphabets, we developed a variant employing alphabet reduction. In our performance experiments alphabet reduction yields an additional speedup factor of 2 on amino acid sequences compared to results achieved with the 20 symbol standard alphabet.

Our second main contribution addresses the problem of non-comparable PSSM-scores of different PSSMs. Therefore, we developed with algorithm *LazyDistrib* a new method for efficient computation of a matrix similarity threshold for a PSSM given an E-value or a p-value. *LazyDistrib* is based on dynamic programming and in contrast to other methods, it employs lazy evaluation of the dynamic programming matrix. It is much faster than existing methods and reaches speedups of a factor between 3 and 330 depending on the stringency of the threshold. In contrast to other methods, which often use approximations to determine a PSSM threshold from a user specified E-value or p-value, *LazyDistrib* is exact and allows accurate on-the-fly calculations of thresholds. For application scenarios, where it is difficult to specify meaningful PSSM score thresholds, we developed two variants, *ESAsearchKb* and *LAsearchKb*, that adjust dynamically the threshold while searching and report the k highest scoring matches for a PSSM.

In our third main contribution, we addressed the problem of limited expressiveness of single PSSM matches and introduced the concept of PSSM family models. To efficiently search with such models, we combined algorithm *ESAsearch* with a fast fragment chaining approach. We performed several experiments assessing the sensitivity and specificity of our PSSM family model approach for protein family classification and assignment. In these experiments PSSM family models achieved a classification performance only marginally inferior to the performance of pHMMs on the family and superfamily level, which yield to be the most sensitive modeling approach for detecting distant homologies and define the *state of the art* in this field. For distant relationships the percentage of true positives when allowing 50 false positives was only 1.6 percentage points below the value achieved by *hmmsearch*. These results are astonishing, since PSSM family models are much simpler models than the fully probabilistic pHMMs. From our experimental results, we conclude that PSSM family models perform nearly as accurate as pHMMs for protein family classification. In addition there are indications that the classification performance can be further improved by using more sophisticated methods for PSSM construction. The major advantage of using PSSM family models instead of pHMMs is the dramatically reduced running time needed for searching with these models compared to searching with pHMMs. This is due to the use of algorithm *ESAsearch* for fast PSSM matching. In a comparable experimental set up, our search tool *PoSSuMsearch*, also implementing the combination of *ESAsearch* and the fragment chaining algorithm of [AO05], achieved a speedup of factor 171 over *hmmsearch*.

With the *PoSSuM* software distribution we provide a well documented software package implementing the ideas and algorithms for efficient searching with PSSMs. Some of the included programs also support multi-threading and hence benefits from multiple CPUs for further speedups. The *PoSSuM* software distribution has already been successfully used in [FSD+05], and is an integrated analysis method in the Genlight and CoryneRegNet [BBC+06, BRT06] software systems.

Motivated by the surprisingly well classification performance of PSSM family models compared to pHMMs, and the fact that database searching with pHMMs in particular on a larger scale is a challenging and time consuming task, we developed the idea of using *PoSSuMsearch* with PSSM family models to speedup time consuming pHMM based database searches. Therefore, we designed and implemented *PSfamSearch* which uses *PoSSuMsearch* with PSSM family models to pre-filter the search space for subsequent application of *hmmsearch* on the filtered sequence set. Our benchmark experiments revealed speedups up to factor 138 for *PSfamSearch* over standard *hmmsearch* and hence may offer a purely software based alternative to highly specialized, costly, hardware based acceleration solutions like DeCypher© or BioBoost©HMMer. In several experiments we tested the impact of the chosen p-value cutoff on the achieved speedups as well as sensitivity of *PSfamSearch*. We tested different strategies for threshold determination. The most promising strategy uses a clear separation of matches obtained on UniProtKB/TrEMBL into training- and test-sets. It computes cutoffs with good generalization characteristics, and using these cutoffs with *PSfamSearch* revealed speedup factors of 72 for p-value cutoffs corresponding to the HMMs *trusted cutoffs* and 15.2 for p-value cutoffs corresponding to *noise cutoffs*, compared to direct unfiltered *hmmsearch*, while retaining more than 99.7% of the original results. For the first 20 protein families listed in the TIGRFAM database, PSSM family model based pre-filtering using this strategy for cutoff determination allowed to reduce the search space to only 1.72% of the original search space on average. Extrapolated to all 2,946 models listed in the current TIGRFAM release we expect a reduction of

running time when searching with these models in UniProtKB/TrEMBL using the models *trusted cutoffs* from more than ≈ 2.84 years for direct *hmmsearch* to only ≈ 15 days for *PSfamSearch*. The measured speedup factors achieved by our purely software solutions *PoSSuMsearch* and *PSfamSearch* compare well with what is achieved by the costly, specialized DeCypher© hardware solution sold by TimeLogic®. Responsible for this speedup are an algorithmic as well as a conceptual advancement. The speed of index based PSSM searching with *ESAsearch* and the fact that pHMMs can be approximated well with the related, but much simpler, PSSM family models.

As a final contribution of this thesis, we describe Genlight, a production ready, interactive system suitable for various high-throughput sequence analysis tasks with a special focus on differential comparative genome analyses. Beyond a variety of other methods, Genlight integrates our database search tool *PoSSuMsearch* for efficient searching with PSSM family models from the PRINTS and BLOCKS databases in large sequence sets using algorithm *ESAsearch*. Unique features of Genlight are its set oriented generic data model anchoring the reusability of results in the basic system design and the integrated distributed execution engine allowing asynchronously executed large-scale analyses even in the absence of costly cluster systems. These features combined with a user-friendly interface make Genlight an extremely flexible system with proven value and usability in several scientific studies.

The majority of contributions made in this thesis were already evaluated and have been proven valuable for publication in the peer reviewing procedures of different journals or conference proceedings. I.e., the algorithms dealing with efficient searching with PSSMs and the *PoSSuM* software distribution have been published in [BSH+04, BHGK06]. We remark that [BHGK06] was designated as *highly accessed* by the journal publisher. Further, at time of this writing (June 2007) [BHGK06] is ranked at second position in the publishers ranking of the most accessed contributions of the last 12 months of all contributions from members from Bielefeld University in all *Biomedcentral* journals. The Genlight system is described in [BMM+04, BSS04] and is available for non-commercial use on `http://piranha.techfak.uni-bielefeld.de/`. The presented Genlight case studies in which the author of this thesis was involved appeared in [MBG+03, BMM+04, SBB+]. A manuscript describing the application of *PSfamSearch* as a fast alternative to *hmmsearch* is in preparation.

6.2 Prospects

Beside several ideas for future Genlight developments already mentioned in section 5.8.1 on page 188, a significant and still open problem is accurate statistics for PSSM chain scores without the need for time consuming sampling. Our experiments (see Figure 4.9 on page 123 and Figure 4.10 on page 125) clearly show a dependency of chainscores $\mathcal{C}^*_{\mathcal{M},S}$ and $\mathcal{C}^*_{\mathcal{F},S}$ on the length of the matched sequence. If this dependency can be eliminated by additional normalizations, this may finally lead to a continous distribution function for chain scores and hopefully to efficiently computable E-values and p-values expressing the statistical significance of a certain chainscore.

We have shown in this thesis that our concept of PSSM family models is well suited to describe protein families and to detect distant relationships. However, PSSM family models are not widely used in practice yet. This is predominantly founded in the unavailability of searchable collections of these models. Although we already converted the BLOCKS and PRINTS databases into a format

readable by *PoSSuMsearch* and use them inside the Genlight system, a future provision of widely used pHMM based databases in form of PSSM family models is reasonable and necessary to increase the level of popularity and the dispersal of *PoSSuMsearch*. Once these conversions have been done, these resources will become directly applicable for efficient searching using *PoSSuMsearch*. The conversion procedure may also include the proper computation of PSSM chainscore cutoffs, corresponding to pHMM *trusted* and *noise* cutoffs so that *PoSSuMsearch* can be used as a pre-filter for speeding up database searches with pHMMs.

A Appendix

A.1 The 20 letter amino acid alphabet

A	ALA	Alanine
V	VAL	Valine
L	LEU	Leucine
I	ILE	Isoleucine
F	PHE	Phenylalanine
P	PRO	Proline
M	MET	Methionine
D	ASP	Aspartic Acid
E	GLU	Glutamic Acid
K	LYS	Lysine
R	ARG	Arginine
S	SER	Serine
T	THR	Threonine
C	CYS	Cysteine
N	ASN	Asparagine
Q	GLU	Glutamine
H	HIS	Histidine
Y	TYR	Tyrosine
W	TRP	Tryptophan
G	GLY	Glycine

Table A.1: The twenty amino acids commonly found in proteins and their one-letter and three-letter coding.

A.2 PROSITE pattern entry

```
ID   CUTINASE_1; PATTERN.
AC   PS00155;
DT   APR-1990 (CREATED); NOV-1997 (DATA UPDATE); MAR-2005 (INFO UPDATE).
DE   Cutinase, serine active site.
PA   P-x-[STA]-x-[LIV]-[IVT]-x-[GS]-G-Y-S-[QL]-G.
NR   /RELEASE=46.4,178022;
NR   /TOTAL=20(20); /POSITIVE=20(20); /UNKNOWN=0(0); /FALSE_POS=0(0);
NR   /FALSE_NEG=0; /PARTIAL=0;
CC   /TAXO-RANGE=??EP?; /MAX-REPEAT=1;
CC   /SITE=11,active_site;
DR   P63880, CUT1_MYCBO , T; P63879, CUT1_MYCTU , T; P63882, CUT2_MYCBO , T;
DR   P63881, CUT2_MYCTU , T; P0A537, CUT3_MYCBO , T; P0A536, CUT3_MYCTU , T;
DR   P00590, CUTI1_FUSSO, T; Q96UT0, CUTI2_FUSSO, T; Q96US9, CUTI3_FUSSO, T;
DR   P41744, CUTI_ALTBR , T; P29292, CUTI_ASCRA , T; P52956, CUTI_ASPOR , T;
DR   Q00298, CUTI_BOTCI , T; P10951, CUTI_COLCA , T; P11373, CUTI_COLGL , T;
DR   Q8X1P1, CUTI_ERYGR , T; Q99174, CUTI_FUSSC , T; P30272, CUTI_MAGGR , T;
DR   Q8TGB8, CUTI_MONFR , T; Q9Y7G8, CUTI_PYRBR , T;
3D   1AGY; 1CEX; 1CUA; 1CUB; 1CUC; 1CUD; 1CUE; 1CUF; 1CUG; 1CUH; 1CUS; 1CUU;
3D   1CUV; 1CUW; 1CUY; 1CUZ; 1FFA; 1FFB; 1FFC; 1FFD; 1FFE; 1OXM; 1XZA; 1XZB;
3D   1XZC; 1XZD; 1XZE; 1XZF; 1XZG; 1XZH; 1XZJ; 1XZK; 1XZL; 1XZM; 2CUT;
DO   PDOC00140;
//
```

Figure A.1: PROSITE entry of a pattern describing a serine active site (PROSITE Accession: PS00155). The pattern description in form of a limited regular expression following the conventions as described in Section 2.4.2 on page 17 is given in the PA line.

A.3 PoSSuMsearch command line interface: Quick reference

Subsequently we give short explanations of the *PoSSuMsearch* command line options. Help on these options is also provided at the command line by calling *PoSSuMsearch* with option `-help`. For a detailed description of the available command line paramters see the complete manual of the *PoSSuM* software distribution provided in section A.4.

```
 * PoSSuMsearch 1.3.3-chaining 64bit, compiled on Feb 14 2007 at 14:41:21
-help        Show help screen.
-version     Show program version.
-db          Name of a database to search in, which can be either an enhanced
             suffix array, a Fasta, GENBANK, or EMBL file.
-pr          Name of a profile library file.
-protein     Use protein alphabet for input sequence.
-dna         Use DNA alphabet for input sequence.
-smap        Name of a symbol map file for input sequence alphabet.
-freq        Name of a frequency file.
-uniform     Assume uniform character distribution in input sequence.
-pdis        Name of a precalculated probability distribution file.
-lazy        Lazy probability evaluation.
-esa         Enhanced suffix array search algorithm (only applicable if the
             input is really an enhanced suffix array).
-lahead      Lookahead search algorithm.
-simple      Simple search algorithm.
```

```
-eval        An E-value to determine the threshold from.
-pval        A p-value to determine the threshold from.
-mssth       A matrix similarity score (MSS) to determine the threshold from.
-rawth       Raw threshold value.
-best        Search for k best matches for each PSSM.
-all         Search for all PSSMs, even if they fall below the given cutoff.
-dbsize      Size of database for E-value tuning (optional if -eval is used).
-realpha     Use reduced alphabet for searching protein PSSMs.
-sort        Sort output by key. Valid keys are i=identifier, a=accession,
             p=p-value, e=E-value, m=mss, s=score, n=sequence number,
             o=position, r=group ID, t=group position, g=group ID/position
             (=rt), l=chain length.
-pssmsearch  Sequence classification based on a, typically small, library of
             known family models. This option requires a numeric argument, k.
             Per family model, the (up to) k best matching sequences are
             reported.
-seqclass    Sequence classification based on a, typically large, library of
             known family models. This option requires a numeric argument, k.
             Per sequence, the (up to) k best matching family models are
             reported.
-mclen       Minimum chain length (default is 1).
-mrclen      Minimum relative chain length, reject chains shorter than a given
             fraction of its group size.
-format      Output format, one of "human" (default), "cisml", "tabs", "stats",
             or "null".
-fn          Search in forward direction (default).
-rc          Search for reverse complementary matches (only applicable on DNA).
-rn          Search for reverse non-complementary matches.
-fc          Search for complementary matches (only applicable on DNA).
-2           Search forward and reverse complementary (short for ‘‘-fn -rc’’).
-4           Search in all possible ways (short for ‘‘-fn -fc -rn -rc’’).
-ncompl      When reporting complementary matches, print out the matching
             sequences as appearing in the database instead of complementary.
-seqrange    Sequence range in which to search, given as min:max pair.
-mult        Multiplier for PSSM values (default is 1.0).
-csfun       Which chain score function to use. Valid functions are "pvalues"
             (default) and "ones" (longest chains win).
-nomatch     Don’t actually search, just set thresholds for benchmarking and
             print the time needed for that.
-qm          Suppress status messages.
-qw          Suppress warnings.
-q           Quiet execution, suppress warnings and status messages.
```

A.4 The PoSSuM software distribution

A.4.1 File formats

PoSSuMsearch and *PoSSuMdist* require PSSMs stored in an easy to read ASCII based file format, combining features supported by other PSSM formats. Converters are included in the *PoSSuM* software distribution to transform TRANSFAC and PRINTS PSSM libraries into *PoSSuM*-PSSM format, see Section A.4.5 on page 217. PSSMs of the BLOCKS database [HP99, HGPH00] or any

PSSMs in BLOCKS format can be used detouring a conversion into PRINTS format. Converters for this purpose are already available, e.g., in the *FingerPrintScan* [SFA99] package.

The PoSSuM-PSSM format

In *PoSSuM*-PSSM format, each line begins with a *tag*, followed by *one* white space character, followed by some data for that tag. All strings are case-sensitive. There must be no white space before the beginning of any tag. Lines may be empty to separate things. Comments are allowed and introduced by a `#` character at the beginning of a line, the whole line is considered as a comment then.

These are the general rules. Now, a PSSM is defined in multiple lines, from which the first one reads

`BEGIN` *type*

followed by some other lines making up the PSSM, and the last line

`END`

indicating the end of a PSSM. The *type* can be one of `INT` or `FLOAT`, depending on the values used in the scoring matrix. If no floating point values occur in the matrix, then *type* should bet set to `INT` to speed up the search as integers can be processed much faster on most machines than floats can be.

Valid tags between a PSSM's `BEGIN` and `END` lines are (in any order):

`ID` The identifier of the PSSM. This tag is required.

`AC` The accession of the PSSM.

`DE` A description; any number of `DE` lines are allowed per PSSM. Multiple description lines are concatenated in order of occurence and separated by full stops when displayed by *PoSSuM-search*.

`AL` An alphabet string. Each character in the string stands for one column of the PSSM, in given order and case-sensitive. The length of the alphabet string determines the width of the scoring matrix, that is, how many columns are expected to be defined.

`AP` A name of a predefined alphabet, either `PROTEIN` or `DNA`. Specifying `DNA` is equivalent to using an `AL` line with alphabet string `ACGT`; `PROTEIN` is equivalent to `AL ACDEFGHIKLMNPQRSTVWY` (note that the exact order of characters is important, thus the explicit specification of the alphabet strings).

Only one of `AL` or `AP` can be used for a PSSM, of course, but one of them is required.

`LE` The "length" of the PSSM, that is the number of rows. This tag is required.

The values of a scoring matrix are defined using `MA` tags, one line per matrix row. There must be as many rows as specified in the `LE` line, each containing as many values as there are characters in the alphabet, in the order imposed by the alphabet. All values are given either as integers or reals as specified by the `BEGIN` line, separated by white space. After the first `MA` line, only `MA`, empty, or comment lines, or `END` are permitted.

The format requires matrices being *grouped* for later, optional chaining. A group of PSSMs must be surrounded by `BEGIN GROUP` and `END` lines. If your application context does not require grouping, just start the library file with a `BEGIN GROUP` line and end it with a final `END` after the `END` of the last PSSM. This declares all PSSMs in the file to belong to the same group.

Internally, each PSSM is identified by a tuple of *group identifier* and *group position*. The group identifier is the position of a group within the profile library file and the group position is the position of a PSSM within its group. Both quantities are counted up from 0 while the profile library file is read, where the group position counter is reset to 0 for every new group. Group identifiers and positions can be used for sorting the output (see description for `-sort` in Section A.4.2) or for PSSM identification when post-processing the output by external programs.

A valid, artificial example for a PSSM library file is

```
BEGIN GROUP
BEGIN INT
ID Some matrix identifier
AC Some accession
DE A description describing the PSSM
DE Multiple description lines are possible
AP DNA
LE 3
#   A  C  G  T  was specified by "AP DNA"
MA  5 -1 -6  2
MA -4  4 -1 -5
MA  0 -3  3 -4
END

BEGIN FLOAT
ID Some other matrix identifier
DE Another description
AL AUCG
LE 2
#     A    U    C    G
MA  0.0 -3.5  3.2 -4.8
MA -4.2 -1.0  4.0 -5.8
END
END
```

Frequency file format

A frequency file consists of simple character/value pairs, one pair per line. It serves for proper probability distribution calculation for E- and p-values for the PSSMs based on a specific input sequence.

A line starts with a single character, followed by white space, followed by the relative frequency of that character in the input sequence. The relative frequency is a real number in the interval $[0, 1]$, the sum of all frequencies specified in one file should be 1.0, such that they constitute a sequence dependent character distribution.

Comments are allowed and introduced by a `#` character at the beginning of a line, the whole line is considered as a comment then. Empty lines are permitted.

A valid example for a uniform distribution on DNA data is

```
# Uniform nucleotide distribution.
A 0.25
C 0.25
G 0.25
T 0.25
```

Note that instead of "`T 0.25`" it would be equivalent to specify "`U 0.25`" or two separate lines reading

```
T 0.15
U 0.1
```

if the input sequence alphabet were to define "T" and "U" being the same. Frequencies of equivalent characters are summed up. See below for more information about symbol mappings.

See Section A.4.4 on page 216 for the description of a tool for determining relative frequencies of characters from an input sequence.

Custom symbol mappings

To work on some sequence, *PoSSuMsearch*, *PoSSuMdist*, and *PoSSuMfreqs* all need to know the sequence's underlying alphabet. If the input sequence is an enhanced suffix array built by `mkvtree`[1] from the *Vmatch* package (see `http://www.vmatch.de/`), then this information is stored in the suffix array project. If the input sequence is a plain text format like FASTA, though, the user must either provide a symbol mapping file, or use the command line options `-dna` or `-protein` to specify a predefined (built-in) alphabet.

The format of symbol mapping files is the same as the format used in *Vmatch*, which is because the *PoSSuM* software distribution is based on the same libraries as *Vmatch*. Each line consists of a string of characters that should be regarded as equal. E.g., the file

```
aA
cC
gG
tT
*
```

defines a case-insensitive DNA alphabet. The last line specifies a group of special wildcard characters (only "`*`" in the example). Actually the wildcard is just an ordinary character treated in a special way internally. Note that the symbol mapping parser is quite picky and *requires* the last line to be terminated by a newline character.

[1] Executable binary also included in the *PoSSuM* software distribution.

Internally, all characters read from the input sequence are mapped to integers, and all characters that appear on the same line of the symbol mapping file are mapped to the same integer such that there is really no difference between them internally.

Use of symbol mappings is important for several reasons:

- Validation of the input sequence (invalid characters can be detected and therefore never occur during searching or in the output).
- Special treatment for case-(in)sensitivity, or more generally, character classes (like "t" = "T" = "u" = "U"), is unnecessary because these cases are handled at the alphabet transformation level.
- The alphabet imposes an order on its characters by mapping them to integers (e.g., "a" and "A" may be mapped to 0, "c" and "C" to 1, etc.). Columns of PSSMs are ordered according to some alphabet, too (first column may stand for "A", second for "C", etc.). If the order of the input sequence alphabet is different from the PSSM's alphabet, then the columns of the PSSM can be reordered according to the order of the input sequence alphabet (otherwise, the user would be urged to provide his PSSMs with their columns in input sequence alphabet order), and this can be done with character classes being handled correctly (if the input sequence is encoded using the alphabet from the example above and the PSSM has some column for "U", then this column is read whenever a "t", "T", "u", or "U" appears in the input, additional columns for any of "t", "T", or "u" will be flagged as an error because of ambiguities).

A.4.2 PoSSuMsearch

Description

This is the main searching program. It implements *ESAsearch* for searching PSSMs in an enhanced suffix array, the lazy dynamic programming evaluation algorithm for threshold derivation from E- and p-values and the fast fragment chaining algorithm of [AO05] to compute high scoring chains of PSSM matches. Additionally, other search algorithms *LAsearch* and simple search for plain text formats such like FASTA are implemented. As an alternative to the lazy dynamic programming evaluation algorithm, a precalculated probability distribution generated by *PoSSuMdist* (see Section A.4.3 on page 214) can be used to derive PSSM thresholds.

Command line options

The searching program *PoSSuMsearch* is called as follows:

`possumsearch` [*options*]

Valid choices for *options* are

`-help`
: Show options and terminate with error code 0.

`-db` *dbfile*

Name of a database to search in, which can be either an enhanced suffix array, or a FASTA, GenBank, or EMBL file. The sequence must consist of characters over the alphabet as specified by the options `-dna`, `-protein`, or `-smap`, see below. This option is mandatory.

`-pr` *matrixfile*

Name of a profile library file. A "library" here is a collection of one or more PSSMs stored in the format as described in Section A.4.1 on page 200. This option is mandatory.

`-protein`

This option is equivalent to the option `-smap` *mapfile* where *mapfile* stores exactly the following 21 lines:

```
L
V
I
F
K
R
E
D
A
G
S
T
N
Q
Y
W
P
H
M
C
XBZ*
```

This specifies an alphabet of size 20 with additional wildcard symbols on the last line. See Section A.4.1 on page 202 or the *Vmatch* manual for a more detailed explanation of the format of symbol mapping files.

`-dna`

This option is equivalent to the option `-smap` *mapfile* where *mapfile* stores exactly the following 5 lines:

```
aA
cC
gG
tTuU
nsywrkvbdhmNSYWRKVBDHM
```

This specifies an alphabet of size 4 with additional wildcard symbols appearing in the fifth line. See Section A.4.1 on page 202 or the *Vmatch* manual for a more detailed explanation of the format of symbol mapping files.

`-smap` *mapfile*

Specify the file storing the symbol mapping. If the given *mapfile* cannot be found in the directory where *PoSSuMsearch* is run, then all directories specified by the environment variable `MKVTREESMAPDIR` are searched. If defined correctly, this contains a list of directory paths separated by colons (“`:`”).

If the file can't be found, an error message is reported and the program exits with error code 1. See Section A.4.1 on page 202 or the *Vmatch* manual for a more detailed explanation of the format of symbol mapping files.

`-freq` *freqfile*

Specify the file storing the relative frequencies of characters in the input sequence. See Section A.4.1 on page 201 for file format reference and Section A.4.4 on page 216 for a description of *PoSSuMfreqs*, a simple program for generating frequency files from a database.

`-uniform`

If no frequency file is available, this option can be specified to assume characters being distributed uniformly. Note that this option is not meant for regular use—for accurate results, determining the real character distribution and specifying it via `-freq` is mandatory.

`-pdis` *distfile*

Specify the file storing a precalculated probability distribution as generated by *PoSSuMdist* for fast computation of E- and p-values for the PSSMs. The file must match the profile library specified by `-pr` and the alphabet of the input sequence. Because frequency information was already used when the distribution was precalculated by *PoSSuMdist*, options `-freq` and `-uniform` are prohibited when using this option. See Section A.4.3 on page 216 and *PoSSuMdist* description for further information. Alternatively, `-lazy` can be used.

`-lazy`

Use lazy dynamic programming for fast computation of E- and p-values for the PSSMs as described in [BHGK06].

`-esa`

Search the PSSMs via *ESAsearch* as described in [BHGK06]. This option is only valid if the *dbfile* given to `-db` is an enhanced suffix array which must have been built by `mkvtree` beforehand.

`-lahead`

Search the PSSMs via *LAsearch* as described in [WNB00, BHGK06]. This option can be used with all kinds of input sequences.

`-simple`

Search the PSSMs via algorithm *SPsearch* as implemented in *FingerPrintScan*, *Blocksearch* [HH91], *Blimps*, *MatInspector*, and probably others. This option can be used with all kinds of input sequences, but should be used for debugging and benchmarking only due to its inferior efficiency.

`-eval` *E-value*

Specify E-value cutoff. This option must be combined with either `-lazy` or `-pdis`. E-value

calculation is based on p-values, it is simply the p-value times database size. If combined with `-seqrange`, the database size is still assumed to be the size of the whole input sequence, not just the range's size. Use `-dbsize` to modify the database size for E-value calculation.

`-pval` *p-value*

Specify p-value cutoff. This option must be combined with either `-lazy` or `-pdis`.

`-mssth` *similarity*

Specify a matrix similarity score (MSS) cutoff. MS-scores are PSSM scores rescaled to the interval $[0, 1]$ with the minimum reachable PSSM score corresponding to 0 and the maximum reachable PSSM score corresponding to 1. This scoring scheme is used in *MatInspector* and *Match* [KGR+03]. The MSS of PSSM M of length m and a sequence $w \in \mathcal{A}^m$ is defined as $\text{MSS} = \frac{sc(w,M)-sc_{\min}(M)}{sc_{\max}(M)-sc_{\min}(M)}$ and hence given an MSS cutoff value, the threshold th is determined as $th = \text{MSS} \cdot (sc_{\max}(M) - sc_{\min}(M)) + sc_{\min}(M)$. Note that because PSSM thresholds can be derived from *similarity* without use of probability distributions, they will not be calculated by default and E- and p-values will not be available in the results. If this information is required, also specify `-freq`, `-uniform`, or `-pdis` to tell *PoSSuMsearch* how the probability distributions for displaying E- and p-values should be obtained. If `-freq` or `-uniform` is used, a full probability distribution must be calculated for each matching PSSM which can be slow, using `-pdis` is the better choice then.

`-rawth` *threshold*

Specify a raw, global threshold for all PSSMs. As PSSM thresholds are set directly and hence no probability distributions are required to do so, the same discussion about E- and p-values as for `-mssth` applies.

`-best` k

Find the $k > 0$ best (meaning highest scoring) matches per PSSM. If there are less then k matches, only those are printed. This option can only be used in conjunction with `-esa` and `-lahead`. Searching with reduced alphabets on enhanced suffix arrays (options `-realpha` and `-esa`), however, does not work together with this option.

Note that since in general matches are found in different order for `-esa` and `-lahead`, their results may also slightly differ (e.g., this is the case when asking for, say, the best three matches, but there are actually a total of five best equally scoring matches in the database, then two of them never get reported—this is not a bug).

Also note that the threshold used for searching reported for each match (the value printed after "threshold" in human readable output, field 9 in tab delimited output, see Section A.4.2 on page 213) is quite useless if `-best` is specified.

`-all`

If a PSSM fails to code for the specified cutoff, e.g., if a p-value of 10^{-30} was specified, but the PSSM is only capable to code for a p-value of 10^{-20}, then that PSSM is not searched for by default and a warning is issued instead. If this option is specified, then in these cases the threshold is set to the maximum possible score the PSSM can yield, so it could match nonetheless (with a p-value higher than requested, though).

`-dbsize` *size*

Assume the database size is *size* as basis for E-value calculation. This option affects E-value calculation exclusively. By default, the original database size is used. Option `-seqrange` does not affect the default value.

`-realpha`

This option is specific to protein data/PSSMs. A problem with protein data is the large alphabet (when compared to DNA) involved which slows down the *ESAsearch* algorithm. A solution is to build an enhanced suffix array using a smaller custom alphabet which defines groups of amino acids as single representative characters, and to search the PSSMs on that reduced alphabet size index.

Usually PSSMs are converted to match the input sequence alphabet, such that an error would be issued when a protein PSSM was searched in an enhanced suffix array built with such a reduced alphabet. The problem then is that groups of distinct profile columns are mapped to the same sequence character representing a group, and when scoring that character there is no way to decide which of the PSSMs' columns should be used for scoring. So to handle this application case properly, the `-realpha` option must be specified. PSSMs are then read as if the input sequence was encoded by the standard protein alphabet, i.e., for enhanced suffix arrays as if they had been built using the `-protein` option, and for flat files (like FASTA) as if `-protein` had been passed to *PoSSuMsearch* (see description for `-protein` above). PSSMs are converted internally according to the reduced sequence alphabet and searched in the reduced sequence, the intermediate matches found are applied to the original PSSMs and original input sequence to calculate the correct match scores. Since reduced alphabets are specific to protein data, options `-rc` and `-fc` cannot be used together with `-realpha`, of course.

Note that for applying this option to an enhanced suffix array, it must have been built with the `-ois` option passed to `mkvtree`. To specify a reduced alphabet, write a symbol map file as described in Section A.4.1 on page 202 or in the *Vmatch* manual and pass that symbol map to `mkvtree` via the `-smap` option. Using this option for flat files doesn't make much sense but is still supported, use *PoSSuMsearch*'s `-smap` option then. Also don't expect any speed-up when using reduced alphabets with the *LAsearch* algorithm.

All *PoSSuMsearch* options retain their original semantics even if `-realpha` is specified, e.g., `-pval` specifies a p-value cutoff for the PSSMs as if they were searched directly in the protein data, hence optionally passed frequencies or precalculated distributions must refer to the standard protein alphabet. See Section A.4.6 for a complete example on how to use this option.

There is one drawback, though: if both `-esa` and `-best` are specified, then this option cannot be used. If `-best` is needed, use `-lahead` or a full alphabet size version of the index (i.e., no `-realpha`) instead.

`-sort` *keys*

Specify order in which matches should be reported. If this option is omitted, the output is not sorted in any special way. The *keys* argument is a string of keys the output is to be sorted by, priority in order of keys. Valid keys are

`i` PSSM identifier, sorted in lexical order. This is the string that is specified in the `ID` tag.

`a` PSSM accession, sorted in lexical order. This is the string that is specified in the `AC` tag.

`p` p-value of match, smallest first.

`e` E-value of match, smallest first.

`m` MSS of match, largest first.

`s` Score of match, largest first.

`n` Sequence number, smallest first.

`o` Position of match in sequence, smallest first.

`r` Group identifier.

`t` Group position.

`g` Group identifier and position, short for "`rt`".

E.g., to get results sorted by E-value in first place and sequence number in second place, specify "`-sort en`". Matches with both the same E-values and sequence numbers again are not sorted in any special way.

Note that a pair of group identifier and group position (sort key "`g`") always identifies exactly one PSSM, but a PSSM identifier together with its accession (sort keys "`ia`" or "`ai`") may not because multiple PSSMs with equal identifiers and accessions can be specified. If unsure, use "`iag`" instead of "`ia`" or "`aig`" instead of "`ai`" to make sure to have PSSMs with both the same identifier and accession separated in the output. Also note that specification of "`gia`" or "`gai`" is equivalent to only specifying "`g`" because if PSSMs are already sorted by group identifier and position in first place, then further sorting by PSSM identifier or accession is not possible (read: unnecessary). In other words, specification of "`g`" just separates matches by PSSM in order of occurence in the profile library, "`iag`" or "`aig`" arrange them in alphabetical order and then make sure to have distinct PSSMs with equal identifier and accession strings being separated.

`-pssmsearch` *k*

Sequence classification based on a, typically small, library of known family models. This option requires a numeric argument, k. Per family model, the (up to) k best matching sequences are reported. For this option matches to PSSMs are chained according to the order given in their corresponding "`GROUP`" definition. A group of PSSMs is also called a PSSM family model. Employing this options, *PoSSuMsearch* computes and rports high-scoring chains of PSSM matches instead of single PSSM matches. For fast computation of high-scoring chains the fragment chaining algorithm of [AO05] is applied with chain scores defined according to Equation (4.8) and Equation (4.7). Using this option, the mode of operation is comparable to *hmmsearch* from the *HMMER* package.

`-seqclass` *k*

Sequence classification based on a, typically large, library of known family models. This option requires a numeric argument, k. Per sequence, the (up to) k best matching family models are reported. Likewise to "`-pssmsearch`" this option also employs fragment chaining. Its semantics are comparable to *hmmpfam* from the *HMMER* package.

`-mclen` *k*

Filtering of obtained high scoring chains based on chainlength. Only chains consisting of at least *k* PSSM matches are reported.

`-mrclen` *k*

Filtering of obtained high scoring chains based on relative chainlength. Here *k* specifies the chain length necessary for a reported result relative to the number of PSSMs in the corresponding family model. $k = 0.66$ means, that the minimum length of a chain to be reported has to be at least two third of the number of PSSMs specified in the family model.

`-format` *fmt*

Specify output format, where *fmt* is one of

`human`	a human readable multiline format,
`cisml`	CisML [HW04], an XML-based format,
`tabs`	tab delimited output (see Section A.4.2 on page 213), or
`null`	no output.

`-fn`

Search on forward strand (default). This option works with any alphabet and replaces the `-fwd` option of previous versions of *PoSSuMsearch*. See also `-rc`, `-rn`, and `-fc`. See Section A.4.2 on page 211 and Figure A.2 for a more detailed explanation on the options concerning search directions.

`-rc`

Search for reverse complementary matches. This option disables the default of searching on the forward strand—specify an extra `-fn` to search on both strands, or use option `-2` or even `-4`. This is a DNA specific option, i.e., the input alphabet must be a DNA alphabet (or in better words, a DNA *compatible* alphabet, not necessarily generated via the `-dna` option of *PoSSuMsearch* or `mkvtree`), and, of course, the PSSMs should encode DNA motifs.

The reason for the DNA specificity is that internally the PSSMs' columns are exchanged ("A"-column with "T"- or "U"-column and "C"-column with "G"-column), their rows are reversed in order, and then a usual search in forward direction is done. Because *PoSSuMsearch* supports arbitrary alphabets to be used for both, input sequences and PSSMs, the column exchange must be done carefully, the compatibility of both alphabets must be checked, and character classes must be recognized ("T" and "U" could be distinct characters in the input sequence). Note that *PoSSuMsearch* does not attempt to recognize whether the alphabets are strictly DNA or not, it just tries to find those columns unambiguously labelled with characters from the DNA alphabet and exchanges them. If columns cannot be exchanged for some reason, the program will tell so and exit with error code 1. See Section A.4.2 on page 211 and Figure A.2 for a more detailed explanation on the options concerning search directions. See also option `-ncompl`.

A more general, non-DNA specific approach could be implemented by requiring the user to explicitly specify the columns to be exchanged instead of autodetecting them, but it would be harder to use without any gain in practical use. Expect no problems when using option `-dna` throughout.

`-rn`

Search for reverse matches. The PSSMs' rows are reversed in order and a usual search in forward direction is done. This option works with any alphabet. See Section A.4.2 on the next page and Figure A.2 for a more detailed explanation on the options concerning search directions.

`-fc`

Search for complementary matches. Alike `-rc`, this is a DNA specific option, i.e., the input alphabet must be a DNA alphabet and the PSSMs should encode DNA motifs. Internally, the PSSMs' columns are exchanged as with `-rc` and a usual search in forward direction is done, but the PSSMs' rows are *not* reversed in order. This option replaces the (misnamed) option `-rev` of previous versions of *PoSSuMsearch*. See Section A.4.2 on the facing page and Figure A.2 for a more detailed explanation on the options concerning search directions. See also option `-ncompl`.

`-2`

Short for `-fn` and `-rc`. If searching for forward and reverse complementary matches, this option is usually what you want. See Section A.4.2 on the next page and Figure A.2 for a more detailed explanation on the options concerning search directions.

`-4`

Short for all of `-fn`, `-rc`, `-rn`, and `-fc`. See Section A.4.2 on the facing page and Figure A.2 for a more detailed explanation on the options concerning search directions.

`-ncompl`

By default, the matching sequences for matches on the complementary strand (options `-rc` and `-fc`) are printed out complemented, i.e. *not* as they appear in the input sequence. E.g., `ACG` is a matching sequence to the first PSSM in Section A.4.1 on page 201 with a threshold of 12 on the forward strand. A reverse complementary matching sequence to the same PSSM with threshold 12 is `CGT`, possibly occuring somewhere else on the reverse complementary strand. In this case, the string that really occurs in the input sequence is `GCA` since the input sequence is always considered to be the forward strand. Without this option, the matching sequence will be printed as `CGT`, and as `GCA` otherwise. See Section A.4.2 on the facing page and Figure A.2 for a more detailed explanation on the options concerning search directions.

`-seqrange` *range*

When using `-lahead` or `-simple`, search only in a range of sequences, not all. The *range* is specified as `min:max` pair, including the borders. A range of `30:39` will search only in sequences 30 to 39, including sequence 30 and 39. Sequence numbers always start at 0.

`-mult` *factor*

For probability distribution calculation, the values of the scoring matrices are scaled by the value of *factor*. The default value of *factor* is 1.0. To speed up calculation of E- and p-values at the price of loss of precision and to reduce disk space when writing the distribution to file using *PoSSuMdist*, choose a value from interval $(0, 1)$. This effects in a compression of PSSM score ranges and thereby a reduction of computation time for the probability distribution calculation. E.g., a value of 0.1 speeds up the probability distribution calculation for a PSSM

by approximately a factor of 10 (because the PSSM's score range is only a tenth of the original range then), but this also means that every 10 consecutive score values achievable by a PSSM are condensed into one single p-value, which is likely to produce false positives and false negatives.

To enhance precision in some cases, choose a value greater than 1, resulting into an expansion of score ranges. Since floating point PSSMs must be rounded to integers for our dynamic programming method, a value greater than 1 can help getting better E- and p-values for PSSMs containing very small values.

This option should be used with great care. PSSMs with smaller score ranges are more prone to rounding errors than those with larger ranges. Larger score ranges result into considerably more space consumption by the probability distribution calculation.

`-qm`
: Do not print status messages to `stderr`.

`-qw`
: Do not print warnings to `stderr`.

`-q`
: Quiet, do not print anything to `stderr`. This is equivalent to specifying both `-qm` and `-qw`. Matches are still written to `stdout`.

`-version`
: Show program version. Option `--version` is a synonym for this option.

There is also a multithreaded version of *PoSSuMsearch*, usually called `possumsearch-mt`. It knows one additional option:

`-j` *jobs*
: Number of simultaneous jobs. By default, without this option the number of jobs is 1. Set the optional argument *jobs* to the number of physical CPUs inside your computer to get best performance or to a lower number to keep some CPUs free for other processes. If *jobs* exceeds the number of CPUs, performance may suffer badly.

  If `-j` is specified without *jobs*, *PoSSuMsearch* tries to ask the operation system for the number of CPUs installed. Note that this may lead to undesired results if Hyper-Threading is activated—measurements on a 2-processor machine with Hyper-Threading enabled showed drastically reduced performance when using four (virtual) CPUs, that was even slightly below the performance of a single CPU doing the same task.

Both programs will terminate silently with error code 0 if no error occured. On error, they will terminate with error code 1 and print the error to `stderr`.

Search directions

On any data, *PoSSuMsearch* supports searching with PSSMs in forward and reverse directions in order to find matches to the provided PSSMs and their reverse. Additionally, on DNA data, searching

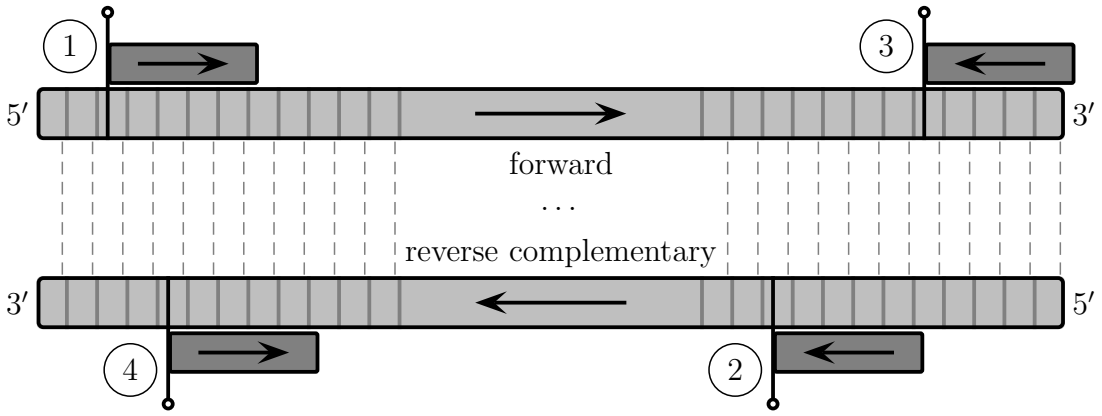


Figure A.2: The search directions supported by *PoSSuMsearch*. If the data is DNA, then there are four cases to consider, namely searching with a PSSM (1) in forward direction on forward strand (option `-fn`, default), (2) in reverse direction on reverse complementary strand (option `-rc`), (3) in reverse direction on forward strand (option `-rn`), and (4) in forward direction on reverse complementary strand (option `-fc`). Note that the arrows denote directions in a biologically correct sense since DNA is commonly read from 5′- to 3′-end. The lower strand in the figure is the *complement* to the upper strand, not the reverse. In case of non-DNA data, the lower strand does not exist, and so then do cases (2) and (4) not.

on the complementary strand is possible. This sums up to a total of four cases, see Figure A.2 for reference. *PoSSuMsearch* offers command line switches to choose any combination of these, the default is `-fn` (search for matches on the forward, non-complementary strand). Specifying one of the other options disables `-fn`, so `-fn` must be specified explicitly if this is also required. Cases (1) and (3) can be used independently of the alphabet, cases (1) and (2) are most commonly used on DNA data. Whether or not cases (3) and (4) are especially useful in practice is arguable, though, still we provide options for these since the user usually knows better what he wants than we do. For convenience, option `-2` can be used to search for matches falling into the categories of cases (1) and (2). Use option `-4` to search for matches in all four possible ways.

Within *PoSSuMsearch*, only the forward, non-complementary strand is known, as provided by the user, represented by the upper strand in Figure A.2. Searching in reverse direction is implemented by reversing the PSSM's row order, covering cases (2) and (3). Searching on the complementary strand is accomplished by alphabet transformation, i.e., by permuting the PSSM's columns according to Watson-Crick base pairing, covering cases (2) and (4). The sequence itself remains unchanged. Since a match does not imply the existence of corresponding matches on the complementary strand nor does it imply the existence of reverse matches, *PoSSuMsearch* must search explicitly for every possible case, hence passing option `-4` results in roughly a 4-fold time consumption for the search phase compared to when searching for a single case.

There are various possible ways one could think of to report reverse and/or complementary matches. In *PoSSuMsearch*, matches are always reported with respect to the forward strand since this is the only sequence that is explicitly represented in the computer and stored in the database. The left-

most position of a match relative to the beginning of the sequence the match occurs in is shown as the start position, starting with position 0, regardless of search direction. E.g., the matching positions for the instances of cases (1) to (4) in Figure A.2 would be reported as marked as 2, $n-10$, $n-5$, and 4, respectively, where n is the number of characters in the sequence. The matching sequence is printed in forward direction as occuring in the database, so in particular, not in reverse even if the match was found on a reverse strand. For matches on the complementary strand, the *complementary sequence* is printed as matching sequence. To get the matching sequences as they occure in the database (i.e., on the non-complementary forward strand), specify option `-ncompl`.

Tab delimited output format

This output format contains one single line per match, containing 18 entities, separated by tabulators, with the following meanings:

1. matched PSSM's identifier (`ID`),
2. matched PSSM's accession (`AC`),
3. matched PSSM's description (`DE`, multiple lines separated by "`. `"),
4. group identifier, which is the position of the group in the profile library file that the PSSM belongs to, starting at 0,
5. position of PSSM within its group, starting at 0,
6. start position of the match with respect to the beginning sequence the match occurs in, starting at 0 (see Section A.4.2 for more details on this),
7. length of the match,
8. search direction ("`fn`" for forward non-complementary, "`rc`" for reverse complementary, "`rn`" for reverse non-complementary, "`fc`" for forward complementary, see Section A.4.2),
9. threshold used for searching (value is useless if `-best` was specified since there is no specific predefined threshold in that case),
10. match score,
11. minimum score the PSSM can achieve,
12. maximum score the PSSM can achieve,
13. p-value,
14. E-value,
15. MSS,
16. matching sequence number, starting at 0,
17. matching sequence description (multiple lines separated by "`. `"), and

18. matching substring (see also Section A.4.2 and *PoSSuMsearch* option `-ncompl`).

Note that tabulators in string entities (descriptions, identifier, etc.) are *not* filtered and may therefore cause problems when parsing an output containing such a string.

If the probability distribution is not available during match evaluation (maybe because `-mssth` or `-rawth` was specified but neither `-pdis` nor `-freq`), the fields for E- and p-value will still be there, but left empty. The same is true for missing information due to lack of `AC` or `DE` tags in the PSSM specification or missing sequence description. A parser reading tab delimited *PoSSuMsearch* output should take this into account.

A.4.3 PoSSuMdist

Description

This is a supplementary program for *PoSSuMsearch*. It is used to precalculate the complete probability distribution which is used to derive PSSM thresholds from E- and p-values. This can be useful if the same PSSM library is searched multiple times and lazy evaluation within *PoSSuMsearch* should be circumvented. Note that a complete probability distribution may require a significant amount of space on file and can take a long time to calculate.

In *PoSSuMsearch*, it is not possible to use a generated probability distribution file with PSSM libraries different from that given to *PoSSuMdist* to generate the distribution. It is not even possible if PSSMs are only rearranged within, deleted from or inserted into that PSSM library. Hence the use of precalculated probability distributions decreases the grade of flexibility in favor of speed.

Command line options

The program for probability distribution calculation *PoSSuMdist* is called as follows:

`possumdist` [*options*]

Valid choices for *options* are

`-help`
: Show options and terminate with error code 0.

`-pr` *matrixfile*
: Name of a profile library file. A “library” here is a collection of one or more PSSMs stored in the format as described in Section A.4.1 on page 200. This option is mandatory.

`-protein`
: Generate a probability distribution file for an input sequence encoded by the standard protein alphabet as described in the description of *PoSSuMsearch* option `-protein`.

`-dna`
: Generate a probability distribution file for an input sequence encoded by the standard DNA alphabet as described in the description of *PoSSuMsearch* option `-dna`.

`-smap` *mapfile*
: Generate a probability distribution file for an input sequence encoded by the symbol mapping defined in *mapfile*. See the description of *PoSSuMsearch* option `-smap` for more details.

`-db` *dbfile*
: Generate a probability distribution file for the enhanced suffix array *dbfile*. This option is used for convenience to just read the symbol mapping from an enhanced suffix array. The enhanced suffix array itself is not read.

`-freq` *freqfile*
: Specify the file storing the relative frequencies of characters in the input sequence. See Section A.4.1 on page 201 for file format reference and Section A.4.4 on the next page for a description of *PoSSuMfreqs*, a simple program for generating frequency files from a database.

`-uniform`
: If no frequency file is available, this option can be specified to assume characters being distributed uniformly. Note that this option is not meant for regular use—for accurate results, determining the real character distribution and specifying it via `-freq` is mandatory.

`-pdis` *distfile*
: Specify the name of the output file storing the precalculated probability distribution. This file can be used later by *PoSSuMsearch* only in conjunction with the profile library specified by `-pr` and input sequences encoded by the specified alphabet. Note that the *factor* of `-mult` given to *PoSSuMdist* will be encoded into *distfile*, hence it can't be changed by a later invokation of *PoSSuMsearch*. See Section A.4.3 on the following page for format description and more information.

`-mult` *factor*
: The values of the scoring matrices are always scaled by the value of *factor*, which defaults to 1.0. See the description of the *PoSSuMsearch* option `-mult` on page 210 for more details.

`-qm`
: Do not print status messages to `stderr`.

`-qw`
: Do not print warnings to `stderr`.

`-q`
: Quiet, do not print anything to `stderr`. This is equivalent to specifying both `-qm` and `-qw`.

`-version`
: Show program version. Option `--version` is a synonym for this option.

Like for *PoSSuMsearch*, there is also a multithreaded version of *PoSSuMdist*, called `possumdist-mt`. For technical reasons, it needs more RAM than the single-threaded version (sometimes much more) because *all* probability distributions are kept in RAM and finally written to file as a whole after all calculations are finished. This version knows one additional option:

`-j` *jobs*

Number of simultaneous jobs. By default, without this options the number of jobs is 1. Set the optional argument *jobs* to the number of physical CPUs inside your computer to get best performance. See description for option `-j` of *PoSSuMsearch* in Section A.4.2 on page 211 for additional notes, also applying to *PoSSuMdist*.

Both programs will terminate silently with error code 0 if no error occured. On error, they will terminate with error code 1 and print the error to `stderr`.

Format of the probability distribution file

A probability distribution file contains the complete probability distributions of all PSSMs in the profile library in order of occurence, written as binary stream and compressed via *zlib* (see `http://www.gzip.org/zlib/`). This data is architecture dependent and can't be exchanged between different architectures because of different byte orders and eventually different sizes of data types. Exchanging probability distribution files between incompatible architectures will yield unpredictable results, from false matches to program crashes.

The distributions are written one after the other, containing

- minscore, the absolute of the minimum achievable score (unsigned integer),
- maxscore, the absolute of the maximum achievable score (unsigned integer),
- the absolute of the global matrix minimum multiplied by the matrix height (unsigned integer),
- the factor specified by `-mult` when the distribution was calculated (double), and
- an array of p-values of length maxscore − minscore +1, ranging from minscore to maxscore (doubles).

All scores are scaled by the factor specified by `-mult` and rounded to integers. So, the general file layout is simply

1. minscore	1. maxscore	1. global minimum	1. factor	1. array of p-values
2. minscore	2. maxscore	2. global minimum	2. factor	2. array of p-values
...	...	...	...	...

in uncompressed form. To save space, *zlib* is used to compress the output transparently. Use *gunzip* (see `http://www.gzip.org/`) for manual decompression if needed.

A.4.4 PoSSuMfreqs

Description

For accurate results in score threshold calculation from significance thresholds, the relative frequencies of characters in the database need to be known. *PoSSuMfreqs* is a simple program to determine those frequencies and write them to `stdout` in the format described in Section A.4.1 on page 201.

Command line options

The program for determining relative frequencies of characters *PoSSuMfreqs* is called as follows:

`possumfreqs` [*options*]

Valid choices for *options* are

`-help`
: Show options and terminate with error code 0.

`-db` *dbfile*
: Name of a database to determine the relative frequencies of characters from, which can be either an enhanced suffix array, or a FASTA, GenBank, or EMBL file. The sequence must consist of characters over the alphabet as specified by the options `-dna`, `-protein`, or `-smap`, see below. This option is mandatory.

`-protein`
: Generate a frequency file for an input sequence encoded by the standard protein alphabet as described in the description of *PoSSuMsearch* option `-protein`.

`-dna`
: Generate a frequency file for an input sequence encoded by the standard DNA alphabet as described in the description of *PoSSuMsearch* option `-dna`.

`-smap` *mapfile*
: Generate a frequency file for an input sequence encoded by the symbol mapping defined in *mapfile*. See the description of *PoSSuMsearch* option `-smap` for more details.

`-qm`
: Do not print status messages to `stderr`.

`-qw`
: Do not print warnings to `stderr`.

`-q`
: Quiet, do not print anything to `stderr`. This is equivalent to specifying both `-qm` and `-qw`.

`-version`
: Show program version. Option `--version` is a synonym for this option.

A.4.5 PSSM converters

The *PoSSuM* software distribution comes with two simple converters, `transfac2gen` and `prints-2gen`, to transform TRANSFAC and PRINTS PSSM libraries into *PoSSuM*-PSSM format, respectively. They both take a single command line argument, which is the name of a PSSM library to be converted. The result is printed to `stdout`, i.e., it must be redirected to some other file to be usable by *PoSSuMsearch* or *PoSSuMdist*.

A.4.6 Using the PoSSuM software distribution

Example 1: Basic operations

Build an enhanced suffix array `sprot` from FASTA file `sprot.fas` containing protein data, using the predefined protein alphabet. To save disk space, not all possible tables are built, only those required by *PoSSuMsearch*.

```
$ mkvtree -db sprot.fas -indexname sprot -protein -tis -suf -lcp -skp -v
```

Generate character distribution from the previously built enhanced suffix array `sprot` and write it to `frequencies.txt`, then search all PSSMs in `profiles.txt` in `sprot`, deriving PSSM thresholds via the lazy dynamic programming algorithm *LazyDistrib* using an E-value of 10^{-15} and the character distribution from `frequencies.txt`. The size of the database and its alphabet are known from the `sprot` project.

```
$ possumfreqs -db sprot > frequencies.txt
$ possumsearch -pr profiles.txt -db sprot -esa -eval 1e-15 -lazy\
 -freq frequencies.txt
```

Precalculate probability distribution of PSSM library file `profiles.txt`, write distribution to `dist.gz`, and search all PSSMs with a p-value cutoff of 10^{-20} or less in FASTA file `sprot.fas` via *LAsearch*, which contains protein data with a character distribution stored in `frequencies.txt`. The alphabet of the database must be explicitly specified and match the alphabet used when `dist.gz` was created (`-protein` here). Perform the same search again on the previously built enhanced suffix array via *ESAsearch* and then via simple search and observe the differences in running time.

```
$ possumdist -pr profiles.txt -protein -freq frequencies.txt -pdis dist.gz
$ possumsearch -pr profiles.txt -protein -db sprot.fas -pval 1e-20 -lahead\
 -pdis dist.gz
$ possumsearch -pr profiles.txt -db sprot -pval 1e-20 -esa -pdis dist.gz
$ possumsearch -pr profiles.txt -db sprot -pval 1e-20 -simple -pdis dist.gz
```

For a working example, take a look into the `share/PoSSuM/examples/` directory of the *PoSSuM* software distribution. Included are a Bourne shell script (`demo.sh`), a FASTA sequence file containing two sequences (`demo.fas`), and 17 PSSMs in *PoSSuM*-PSSM format (`demo.lib`). The shell script builds an enhanced suffix array from the FASTA file and performs some searches in the enhanced suffix array and in the FASTA file. The output of the shell script can be redirected to a file and compared to the included file `results.txt` found in the examples directory.

Example 2: Using reduced alphabets

Here is a complete example on how to use the `-realpha` option of *PoSSuMsearch* to speed up *ESAsearch*. We use a custom symbol map, called `prot8.map`, containing eight character classes to build an enhanced suffix array `sprot8` from the protein data in FASTA file `sprot.fas`. The content of `prot8.map` reads

```
G
ALM
VI
ND
P
YFWC
KRQE
STH
BXZ*
```

PoSSuMsearch is used to search the profiles in `profiles.txt` in `sprot8`. Note that the distribution data previously generated can be used here again. Remember that internally the PSSMs are treated as if the input sequence was encoded by the standard protein alphabet, so the probability distribution must be, too. Hence when a precalculated probability distribution should be used in conjunction with `-realpha`, it must always refer to the standard protein alphabet. All relevant commands are shown below.

```
$ mkvtree -db sprot.fas -indexname sprot8 -smap prot8.map -tis -ois -suf -lcp -skp -v
$ possumfreqs -db sprot.fas -protein > frequencies.txt
$ possumdist -pr profiles.txt -protein -freq frequencies.txt -pdis dist.gz
$ possumsearch -pr profiles.txt -db sprot8 -pval 1e-20 -esa -realpha -pdis dist.gz
```

The results should be the same as in the example before, except for the ordering.

Example 3: Computing scores for all substrings

If the scores of all substrings of a sequence need to be known, e.g. to determine a distribution of scores, the best method is to use simple search or *ESAsearch* in conjunction with setting the threshold to the PSSMs' minscore. This can be easily performed by using the normalized matrix similarity scoring shema, since $MSS = 0 \Leftrightarrow th =$ minscore. To achieve this, use something like the following.

```
$ possumsearch -pr profiles.txt -db sprot -esa -mssth 0
```

Do not use *LAsearch* in this case, since here every substring must be read to its full length anyways and simple search avoids the extra overhead of *LAsearch*. Still, *ESAsearch* is even more preferable.

A.4.7 Messages and warnings

Both programs, *PoSSuMsearch* and *PoSSuMdist*, print progress messages to inform the user about what the program is doing, and warnings if problems are detected. Messages and warnings are always printed to `stderr` and are therefore separated from the matches, which are always printed to `stdout`.

Messages can safely be ignored, but if any warnings occur, you should read them as they could point you to some undiscovered problem. Ignore them only if you know they are harmless.

Most warnings are self-explanatory, but there are some that can be confusing:

- Warnings concerning PSSM library files:

Character 'x' is undefined/defined as a wildcard/defined as the special separator character in the input alphabet.
A PSSM defines a column for some character which is not defined as a valid character in the database alphabet and will therefore never contribute to a match score. Those columns are ignored for probability distribution calculation and during matching. This is a common warning for e.g., protein PSSMs defining a column for 'B' which is a wildcard in the predefined protein alphabet.

Character 'x' may occur in the input sequence, but is not defined for the PSSM.
A character may occur in the database that no column is defined for in a PSSM. This is a bad warning because this means that the missing column is inserted and filled with a *very* low, negative score. No problem for the searching algorithms, but a big problem for probability distribution calculation—the score range is enlarged artificially and the calculation is likely to abort due to insufficient memory. Expect to see this warning when e.g., trying to search DNA PSSMs on protein data.

Character class {...} may occur in the input sequence, but no column for any of its representatives is defined in the PSSM.
This is just the same like above, but instead of a single character 'x', the inserted column stands for a set of characters. The PSSM is expected to define a column for exactly one of them, otherwise the same will happen as described above.

- Warnings concerning frequency files:

Character 'x' is undefined in the input alphabet.
The frequency file defines a frequency for some character which is not defined in the database alphabet and will therefore never occur in a sequence. This frequency is ignored then.

Character 'x' is defined as a wildcard in the input alphabet.
The frequency file defines a frequency for a character that is defined as wildcard in the database alphabet and will therefore never match. This frequency is still accounted for.

Character 'x' is defined as the special separator character in the input alphabet.
The frequency file defines a frequency for some character that is mapped internally to a special separator which will never occur in a sequence. This frequency is ignored then.

Sum of relative frequencies is not 1.0.
The sum of all frequencies should be exactly 1.0 such that they constitute a probability distribution. If the sum is not 1.0, this warning is issued, but the frequencies are accepted as provided. Note that this warning can be an artifact due to rounding errors.

  If any of the first three warnings occurs, probably the wrong frequency file for the database alphabet was specified.

- Searching on complementary strand with *PoSSuMsearch*:

Characters 'x' and 'X' are both undefined in the input alphabet.
Searching on the complementary strand, reverse or not, requires exchanging PSSM columns, in particular the "A"-column with "T"- or "U"-column and "C"-column with "G"-column. *PoSSuMsearch* tries to find these columns automatically, ignoring case (so the 'x' and 'X' above may stand for 'a' and 'A'). If there is no column for neither the lower nor the upper case letter of the to-be-exchanged columns, this warning will be issued, meaning that the search will still proceed but with the corresponding columns *unexchanged*.

A.5 Predefined Hit-set filters in the Genlight system

In addition to the user defined *Hit-set* filters, Genlight comes with several predefined filters. Table A.5 on the next page gives an overview of these filters.

Filter name	Arguments	Semantic/Explanation
Select only non-identical homolog pairs	-	With this filter you can select all pairs from a *Hit-set* that are not completely identical. Here completely identical means, that they have exactly the same sequence and annotation
Select full (query) length matches		This filter returns only sequence pairs, where the aligned region detected by the comparison method covers the complete query sequence
Select full (hit) length matches		This filter returns only sequence pairs, where the aligned region detected by the comparison method covers the complete database sequence
Select full (hit) length matches starting with PATTERN1 in matching query region and PATTERN2 in matching hit region	PATTERN1 (e.g. ATG), PATTERN2 (e.g. M)	This filter returns only sequence pairs, where the aligned region detected by the comparison method covers the complete database sequence and the aligned region in the query begins with PATTERN1 and the aligned region in the datbase sequence starts with PATTERN2
Select full length protein matches starting with ATG/M and ending with stop		Special filter for BLASTX, FASTX, FASTY based *Hit-sets*. Sequence pairs passing this filter, has to match the complete database sequence. Further the matching area has to start with the start codon ATG in the query, a methionine (M) in the datbase sequence and has to end with one of the three stop codons TAA,TAG or TGA.
Select almost full (query) length matches (except start offset, end offset)	START OFFSET, END OFFSET	This filter selects only pairs, where the aligned region covers the query sequence completely, except some allowed mismatches at the start and the end. The number of allowed non matching characters is given by the two parameters “START OFFSET” and “END OFFSET”.
Select almost full (hit) length matches (except start offset, end offset)	START OFFSET, END OFFSET	This filter selects only pairs, where the aligned region covers the database sequence completely, except some allowed mismatches at the start and the end. The number of allowed non matching characters is given by the two parameters “START OFFSET” and “END OFFSET”.
Filter by keyword contained in query sequence header	KEYWORD (e.g. transmembrane), OCCURENCE	Sequence pairs passing this filter have to contain the string specified by the KEYWORD parameter as a substring in the annotation (header) of the query sequence. Setting the OCCURENCE parameter to “NO” negates this filter
Filter by keyword contained in hit sequence header	KEYWORD (e.g. transmembrane), OCCURENCE	Sequence pairs passing this filter have to contain the string specified by the KEYWORD parameter as a substring in the annotation (header) in the annotation of the hit sequence. Setting the OCCURENCE parameter to “NO” negates this filter.

Table A.5: The predefined filters in Genlight, their parameters and semantics.

Bibliography

[ABB+81] S. Anderson, A.T. Bankier, B.G. Barrell, M.H. de Bruijn, A.R. Coulson, J. Drouin, I.C. Eperon, D.P. Nierlich, B.A. Roe, F. Sanger, P.H. Schreier, A.J. Smith, R. Staden, and I.G. Young. Sequence and organization of the human metochondrial genome. *Nature*, **290**(5806):457–465, 1981. 1.1

[ABF+03] T. K. Attwood, P. Bradley, D. R. Flower, A. Gaulton, N. Maudling, A. L. Mitchell, G. Moulton, A. Nordle, K. Paine, P. Taylor, A. Uddin, and C. Zygouri. PRINTS and its automatic supplement, prePRINTS. *Nucl. Acids Res.*, **31**(1):400–402, 2003. 3.1

[ABOH01] S.F. Altschul, R. Bundschuh, R. Olsen, and T. Hwa. The estimation of statistical parameters for local alignment score distributions. *Nucl. Acids Res.*, **29**(2):351–361, 2001. 4.1

[ACF+00] T.K. Attwood, M.D.R. Croning, D.R. Flower, A.P. Lewis, J.E. Mabey, P. Scordis, J.N. Selley, and W. Wright. PRINTS-S: The database formerly known as PRINTS. *Nucl. Acids Res.*, **28**(1):225–227, 2000. 2.5.1, 4.2

[ACH+00] M.D. Adams, S.E. Celniker, R.A. Holt, C.A. Evans, J.D. Gocayne, and J.C. Venter et al. The genome sequence of drosophila melanogaster. *Science*, **287**(5461):2185–2195, 2000. 1.1

[ADRF04] I. Alam, A. Dress, M. Rehmsmeier, and G. Fuellen. Comparative homology agreement search. An effective combination of homology search methods. *Proc. Nat. Acad. Sci. U.S.A.*, **101**(38):13814–13819, 2004. 4.5.3

[AGM+90] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, **215**(3):403–413, 1990. 1.1, 1.1, 2.5, 5.4.3

[AHB+04] A. Andreeva, D. Howorth, S.E. Brenner, T.J.P. Hubbard, C. Chothia, and A.G. Murzin. SCOP database in 2004: refinements integrate structure and sequence family data. *Nucl. Acids Res.*, **32**(1):D226–D229, 2004. 2.7.3, 4.5

[AKO02] M.I. Abouelhoda, S. Kurtz, and E. Ohlebusch. The enhanced suffix array and its applications to genome analysis. In *Proceedings of the 2nd Workshop on Algorithmns in Bioinformatics (WABI)*, pages 449–463, 2002. 3.5

[AKO04] M.I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, **2**:53–86, 2004. 1.1, 3.1, 3.5, 3.5.1

[ALM+99] R.A. Alm, L.S. Ling, D.T. Moir, B.L. King, and E.D. Brown. Genomic-sequence comparison of two unrelated isolates of the human gastric pathogen Helicobacter pylori. *Nature*, **397**(6715):176–180, 1999. 5.7.3

[Alt91] S.F. Altschul. Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.*, **219**(3):555–565, 1991. 2.5.5

[AMG+06] T.K. Attwood, A. Mitchell, A. Gaulton, G. Moulton, and L. Tabernero. The PRINTS protein fingerprint database: functional and evolutionary applications. In M. Dunn, L. Jorde, P. Little, and A. Subramaniam, editors, *Encyclopedia of Genetics, Genomics, Proteomics and Bioinformatics*. John Wiley & Sons, 2006. 2.5.1, 5.4.4

[AMS+97] S.F. Altschul, T.L. Madden, A.A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucl. Acids Res.*, **25**(17):3389–3402, 1997. 1.1, 1.1, 1.1, 2.5, 2.5.2, 5.4.3

[AO03a] M.I. Abouelhoda and E. Ohlebusch. A local chaining algorithm and its applications in comparative genomics. In *Proceedings of the 3rd Workshop on Algorithms in Bioinformatics (WABI)*, volume **2812**, pages 1–16, Springer Berlin/Heidelberg, 2003. Lecture Notes in Bioinformatics. 4.3

[AO03b] M.I. Abouelhoda and E. Ohlebusch. Multiple genome alignment: Chaining algorithms revisited. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume **2676**, pages 1–16, Springer Berlin/Heidelberg, 2003. Lecture Notes in Computer Science. 4.3, 4.3.1

[AO05] M.I. Abouelhoda and E. Ohlebusch. Chaining algorithms for multiple genome comparison. *Journal of Discrete Algorithms*, **3**(2-4):321–341, 2005. 4.3, 4.3.1, 4.3.1, 4.3.1, 4.4, 4.5.3, 4.7, 6.1, A.4.2, A.4.2

[Apo85] A. Apostolico. The myriad virtue of subword trees. In Apostolico, A. and Gali, Z., editor, *Combinatorial Algorithms on Words*, volume **F12**, pages 85–96. Springer, 1985. 1.1, 3.4

[BBC+06] J. Baumbach, K. Brinkrolf, L.F. Czaja, S. Rahmann, and A. Tauch. CoryneRegNet: An ontology-based data warehouse of corynebacterial transcription factors and regulatory networks. *BMC Genomics*, **7**(1), 2006. 3.12, 6.1

[BBD+00] A. Bateman, E. Birney, R. Durbin, S.R. Eddy, K.L. Howe, and E.L.L. Sonnhammer. The Pfam protein families database. *Nucl. Acids Res.*, **28**(1):263–266, 2000. 2.5.1, 2.7.2

[BCD+04] A. Bateman, L. Coin, R. Durbin, R.D. Finn, V. Hollich, S. Griffith-Jones, A. Khanna, M. Marshall, S. Moxon, E.L.L. Sonnhammer, D.J. Studholme, C. Yeats, and S.R. Eddy. The Pfam protein families database. *Nucl. Acids Res.*, **32**(1):D138–D141, 2004. 2.7, 2.7.3

[BCH98] S.E. Brenner, C. Chothia, and T.J.P. Hubbard. Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proc. Nat. Acad. Sci. U.S.A.*, **95**(11):6073–6078, 1998. 4.5

[BDD98] R.E. Bruccoleri, T.J. Dougherty, and D.B. Davidson. Concordance analysis of microbial genomes. *Nucl. Acids Res.*, **26**(19):4482–4486, 1998. 5.1.1

[BE95a] T. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning*, **21**(1-2):51–80, 1995. 2.3

[BE95b] T. Bailey and C. Elkan. The value of prior knowledge in discovering motifs with MEME. In *Proc. of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 21–29, Menlo Park, CA, 1995. AAI Press. 2.3

[BG98a] T.L. Bailey and M. Gribskov. Combining evidence using p-values: application to sequence homology searches. *Bioinformatics*, **14**(1):48–54, 1998. 4.5.4

[BG98b] T.L. Bailey and M. Gribskov. Methods and statistics for combining motif match scores. *J. Comput. Biol.*, **5**(2):211–221, 1998. 4.5.4, 4

[BH87] O.G. Berg and P.H. Hippel. Selection of DNA binding sites by regulatory proteins. Statistical mechanical theory to operators and promotors. *J. Mol. Biol.*, **193**(4):723–750, 1987. 2.5.4, 2.5.4

[BHGK06] M. Beckstette, R. Homann, R. Giegerich, and S. Kurtz. Fast index based algorithms and software for matching position specific scoring matrices. *BMC Bioinformatics*, **7**(389):, 2006. 6.1, A.4.2

[BHK+93] M. Brown, R. Hughey, A. Krogh, I.S. Mian, K. Sjölaender, and D. Haussler. Using Dirichlet mixture priors to derive hidden Markov models for protein families. In *Proc. of the First International Conference on Intelligent Systems in Molecular Biology*, pages 47–55, Menlo Park, CA, 1993. AAAI Press. 2.5.6

[BK97] C.B. Burge and S Karlin. Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.*, **268**(1):78–94, 1997. 2.7

[BKML+07] D.A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostel, and D.L. Wheeler. Genbank. *Nucl. Acids Res.*, **35**(1):D21–D25, 2007. 1.1

[BLB03] S. P. Bennett, L. Lu, and D. Brutlag. 3MATRIX and 3MOTIF: a protein structure visualization system for conserved sequence motifs. *Nucl. Acids Res.*, **31**(13):3328–3332, 2003. 2.5, 2.5

[BM77] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Communications of the ACM*, **20**(10):762–772, 1977. 3.4

[BMM+04] M. Beckstette, J.T. Mailänder, R.J. Marhöfer, A. Sczyrba, E. Ohlebusch, R. Giegerich, and P.M. Selzer. Genlight: Interactive high-throughput sequence analysis and comparative genomics. *Journal of Integrative Bioinformatics*, **1**(8):, 2004. 1.2, 2.1, 3.12, 4.4, 5.1.1, 5.7.3, 5.7.3, 6.1

[BMSLDR01] S. Bury-Mone, S. Skouloubris, A. Labigne, and H. De Reuse. The Helicobacter pylori UreI protein: role in adaptation to acidity and identification of residues essential for its activity and for acid activation. *Mol. Microbiol.*, **42**(4):1021–1034, 2001. 5.7.3

[Bra94] A. et al. Brazma. Approaches to the automatic discovery of patterns in biosequences. *J. Comput. Biol.*, **5**(2):279–305, 1994. 2.3

[BRT06] J. Baumbach, S. Rahmann, and A. Tauch. CoryneRegNet: An integrative bioinformatics platform for the analysis of transcription factors and regulatory networks. In *Proceedings of the European Conference on Computational Biology (ECCB)* , 2006. 3.12, 6.1

[BSH+04] M. Beckstette, D. Strothmann, R. Homann, R. Giegerich, and S. Kurtz. PoSSuMsearch: Fast and sensitive matching of position specific scoring matrices using enhanced suffix arrays. In *Proc. of the German Conference on Bioinformatics*, volume **P-53**, pages 53–64. GI Lecture Notes in Informatics, 2004. 3.10, 3.4, 6.1

[BSS04] M. Beckstette, A. Sczyrba, and P. M. Selzer. Genlight: An interactive system for high-throughput sequence analysis and comparative genomics. In *Proc. of the German Conference on Bioinformatics*, volume **P-53**. GI Lecture Notes in Informatics, 2004. 4.4, 5.1.1, 6.1

[BT04] M.A. Beer and S. Tavazoie. Predicting gene expression from sequence. *Cell*, **117**(2):185–198, 2004. 3.1

[Bur98] C.B. Burge. Modelling dependencies in pre-mRNA splicing signals. In S. Salzberg, D. Searls, and S. Kasif, editors, *Computational Methods in Molecular Biology*, pages 127–163, Amsterdam, 1998. Elsevier Science. 2.7

[BWO+96] C.J. Bult, O. White, G.J. Olson, L. Zhou, R.D. Fleischmann, G.G. Sutton, J.A. Blake, L.M. FitzGerald, R.A. Clayton, J.D. Gocayne, A.R. Kerlavage, B.A. Dougherty, J.F. Tomb, M.D. Adams, C.I. Reich, R. Overbeek, E.F. Kirkness, K.G. Weinstock, J.M. Merrick, A. Glodek, J.L. Scott, N.S. Geoghagen, and J.C. Venter. Complete genome sequence of the methanogenic archaeon, Methanococcus jannaschii. *Science*, **273**(5278):1058–1073, 1996. 1.1

[BYG89] R. A. Baeza-Yates and G. H. Gonnet. A new approach to text searching. In N. J. Belkin and C. J. van Rijsbergen, editors, *Proceedings of the 12th International Conference on Research and Development in Information Retrieval*, pages 168–175, Cambridge, MA, 1989. ACM Press. 2.4.2

[Cas88] G. Castillo. *Extreme Value Theory in Engineering.* Academic Press, 1988. 3.8.1

[CBK01] P. Cramer, D.A. Bushnell, and R.D. Kornberg. Structural basis of transcription. RNA polymerase II at 2.8 angstrom resolution. *Science*, **292**(5523):1863–76, 2001. 2.2, 2.2

[CCBH73] S.N. Cohen, A.C.Y. Chang, H.W. Boyer, and R.B. Helling. Construction of biologically functional bacterial plasmids in vitro. *Proc. Nat. Acad. Sci. U.S.A.*, **70**(11):3240–32444, 1973. 1.1

[CGK99] F. Corpet, J. Gouzy, and D. Kahn. Recent improvements of the ProDom database of protein familes. *Nucl. Acids Res.*, **27**(1):263–267, 1999. 2.5.1

[CGK01] F. Chetouani, P. Glaser, and F. Kunst. FindTarget: software for substractive genome analysis. *Microbiology*, **147**(10):2643–2649, 2001. 5.1.1

[CGK02] F. Chetouani, P. Glaser, and F. Kunst. DiffTool: building, visualizing and querying protein clusters. *Bioinformatics*, **18**(8):1143–1144, 2002. 5.1.1

[CHCB04] G.E. Crooks, G. Hon, J.M. Chandonia, and S.E. Brenner. WebLogo: A sequence logo generator. *Genome Research*, **14**(6):1188–1190, 2004. 2.4

[CI95] J. Cleveland and J.N. Ihle. Contenders in FasL/TNF death signaling. *Cell*, **81**(4):479–482, 1995. 5.5.3

[Cla94] J. M. Claverie. Some useful statistical properties of position-weight matrices. *Comput. Chem*, **18**(3):287–293, 1994. 2.5.6

[Con98] The C.elegans Sequencing Consortium. Genome sequence of the nematode C. elegans: a platform for investigating biology. *Science*, **282**(5396):2012–2018, 1998. 1.1

[Con00] The Gene Ontology Consortium. Gene Ontologies: tool for the unification of biology. *Nat. Genetics*, **25**(1):25–29, 2000. 5.4.6

[Con01] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, **409**(6915):860–921, 2001. 1.1, 5.8.1

[Con06] The Gene Ontology Consortium. The Gene Ontology (GO) project in 2006. *Nucl. Acids Res.*, **34**(1):D322–D326, 2006. 2.7.3

[dB46] N.G. de Bruijn. A combinatorial problem. In *Koninklijke Nederlands Akademie van Wetenschappen Proceedings*, volume **49**, pages 758–764, 1946. 3.5.1

[DDBB02] P. Datta, A. Dasgupta, S. Bhakta, and J. Basu. Interaction between FtsZ and FtsW of Mycobacterium tuberculosis. *J. Biol. Chem.*, **277**(28):24983–24987, 2002. 5.7.3

[DDSS01] T. Dandekar, F. Du, R.H. Schirmer, and S. Schmidt. Medical target prediction from genome sequence: combining different sequence analysis algorithms with expert knowledge and input from artificial intelligence approaches. *Computers and Chemistry*, **26**(1):15–21, 2001. 5.1

[DECS65] M.O. Dayhoff, R.V. Eck, M.A. Chang, and M.R. Sochard. Atlas of protein sequence and structure. National Biomedical Research Foundation, Silver Spring, MD, 1965. 1.1

[DEK98] R. Durbin, S. Eddy, and A. Krogh. *Biological sequence analysis. Probabilistic models of proteins and nucleic acids.* Cambridge University Press, New York, 1998. 2.7.2, 4.5

[Dep03] Logical Depth. LDhmmer. `http://logicaldepth.com/ldhmmer`, 2003. 4.7

[DNM00] B. Dorohonceanu and C.G. Nevill-Manning. Accelerating protein classification using suffix trees. In *in Proc. of the International Conference on Intelligent Systems for Molecular Biology*, pages 128–133, Menlo Park, CA, 2000. AAAI Press. 3.1, 3.4.1, 3.4.1, 3.5, 3.10, 3.10, 3.1, 3.2, 3.12

[Doo81] R.F. Doolittle. Similar amino acid sequences: chance or common ancestry? *Science*, **214**(4517):149–159, 1981. 1.1

[Doo86] R.F. Doolittle. *Of URFs and ORFs: a primer on how to analyze derived amino acid sequences*. University Science Books, Mill Valley, California, 1986. 2.2

[DRF+03] Q. Dong, L. Roy, M. Freeling, V. Walbot, and V. Brendel. ZmDB, an integrated database for maize genome research. *Nucl. Acids Res.*, **31**(1):244–247, 2003. 5.7.1

[ED94] S.R. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucl. Acids Res.*, **22**(11):2079–2088, 1994. 5.8.1

[Edd98] S. R. Eddy. Profile hidden Markov models. *Bioinformatics*, **14**(9):755–763, 1998. 1.1, 1.1, 2.7.2, 7, 2.7.2, 2.7.3, 4.5, 5.4.4

[Edd02] S.R. Eddy. A memory efficient dynamic programming algorithm for optimal structural alignment of a sequence to an rna secondary structure. *BMC Bioinformatics*, **3**(18), 2002. 5.8.1

[EJG+03] A.J. Enright, B. John, U. Gaul, T. Tuschl, C. Sander, and D.S. Marks. MicroRNA targets in Drosophila. *Genome Biol*, **5**(1):R1, 2003. 5.8.1

[EKM97] P. Embrechts, C. Klüppelberg, and T. Mikosch. *Modelling Extremal Events*. Springer, 1997. 3.8.1

[ENC04] ENCODE Project Consortium. The ENCODE (ENCyclopedia Of DNA Elements) Project. *Science*, **306**(5996):636–640, 2004. 5.8.1

[EPC+00] G. Emilien, M. Ponchon, C. Caldas, O. Isacson, and J.M. Maloteaux. Impact of genomics on drug discovery and clinical medicine. *Quarterly journal of medicine*, **93**(7):391–423, 2000. 1.1

[EUA96] T. Etzold, A. Ulyanov, and P. Argos. SRS: information retrieval system for molecular biology data banks. *Methods Enzymol.*, **266**:114–28, 1996. 5.1.1

[EvDO02] A.J. Enright, S. van Dongen, and C.A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucl. Acids Res.*, **30**(7):1575–1584, 2002. 5.4.3

[FAH+01] D. Frishman, K. Albermann, J. Hani, K. Heumann, A. Metanomski, A. Zollner, and H.W. Mewes. Functional and structural genomics using PEDANT. *Bioinformatics*, **17**(1):44–57, 2001. 5.1.1

[FAW+95] R.D. Fleischmann, M.D. Adams, O. White, R.A. Clayton, E.F. Kirkness, A.R. Kerlavage, C.J. Bult, J.F. Tomb, B.A. Dougherty, and J.M. Merrick. Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science*, **269**(5223):496–512, 1995. 1.1

[FB05] V. Freschi and A. Bogliolo. Using sequence compression to speedup probabilistic profile matching. *Bioinformatics*, **21**(10):2225–2229, 2005. 3.1

[FCD+76] W. Fiers, R. Contreras, F. Duerinck, G. Haegemann, D. Iserentant, J. Merregaert, W. Min Jou, F. Molemans, A. Raeymaekers, A. van den Berghe, G. Volckaert, and M. Ysebaert. Complete nucleotide sequence of bacteriophage MS2 RNA: primary and secondary structure of the replicase gene. *Nature*, **260**(5551):500–507, 1976. 1.1

[FGW+95] C.M. Fraser, J.D. Gocayne, O. White, M.D. Adams, R.A. Clayton, R.D. Fleischmann, C.J. Bult, A.R. Kerlavage, G. Sutton, J.M. Kelley, R.D. Fritchman, J.F. Weidman, K.V. Small, M. Sandusky, J. Fuhrmann, D. Nguyen, T.R. Utterback, D.M. Saudek, C.A. Phillips, J.M. Merrick, J.F. Tomb, B.A. Dougherty, K.F. Bott, P.C. Hu, T.S. Lucier, S.N. Peterson, H.O. Smith, C.A. Hutchison, and J.C. Venter. The minimal gene complement of Mycoplasma genitalium. *Science*, **270**(5235):397–403, 1995. 1.1

[FHSW04] M.C. Frith, U. Hansen, J.L. Spouge, and Z. Weng. Finding functional sequence elements by multiple local alignment. *Nucl. Acids Res.*, **32**(1):189–200, 2004. 2.3

[Fit70] W.M. Fitch. Distinguishing homologous from analogous proteins. *Syst. Zool.*, **19**(2):99–113, 1970. 1.1

[FMSB+06] R.D. Finn, J. Mistry, B. Schuster-Bockler, S. Griffith-Jones, V. Hollich, T. Lassmann, S. Moxon, M. Marshall, A. Khanna, R. Durbin, S.R. Eddy, E.L. Sonnhammer, and A. Bateman. Pfam: clans, web tools, and services. *Nucl. Acids Res.*, **34**(1):D247–D251, 2006. 1.1, 4.2, 4.5.3, 4.7, 5.4.4, 5.4.5

[FSD+05] K. Florquin, Y. Saeys, S. Degroeve, P. Rouze, and Y. Van de Peer. Large-scale structural analysis of the core promotor in mammalian and plant genomes. *Nucl. Acids Res.*, **33**(13):4255–4264, 2005. 3.12, 6.1

[Gal05] M. Y. Galperin. The molecular biology database collection: 2005 update. *Nucl. Acids Res.*, **33**(1):D5–D24, 2005. 2.5.1

[GBB+96] A. Goffeau, B.G. Barrell, H. Bussey, R.W. Davis, B. Dujon, H. Feldmann, F. Galibert, J.D. Hoheisel, C. Jacq, M. Johnston, E.J. Louis, H.W. Mewes, Y. Murakami, P. Philippsen, H. Tettelin, and S.G. Oliver. Life with 6000 genes. *Science*, **274**(5287):546–567, 1996. 1.1

[GBYS92] G. Gonnet, R.A. Baeza-Yates, and T. Snider. New indices for text: PAT trees and PAT arrays. In Frakes, W.B. and Baeza-Yates, R.A., editor, *Information Retrieval: Algorithms and Data Structures*, volume **132**, pages 66–82. Prentice-Hall, New Jersy, 1992. 3.5

[GFB+01] P. Glaser, L. Frangeul, C. Buchrieser, C. Rusniok, A. Amend, F. Baquero, P. Berche, H. Bloecker, P. Brandt, T. Chakraborty, A. Charbit, F. Chetouani, E. Couve, A. de Daruvar, P. Dehoux, E. Domann, G. Dominguez-Bernal, E. Duchaud, L. Durant, O. Dussurget, K.D. Entian, H. Fsihi, F. Garcia-del Portillo, P. Garrido, L. Gautier, W. Goebel, N. Gomez-Lopez, T. Hain, J. Hauf, D. Jackson, L.M. Jones, U. Kaerst, J. Kreft, M. Kuhn, F. Kunst, G. Kurapkat, E. Madueno, A. Maitournam, J.M. Vicente, E. Ng, H. Nedjari, G. Nordsiek, S. Novella, B. de Pablos, J.C. Perez-Diaz, R. Purcell, B. Remmel, M. Rose, T. Schlueter, N. Simoes, A. Tierrez,

J.A. Vazquez-Boland, H. Voss, J. Wehland, and P. Cossart. Comparative genomics of Listeria species. *Science*, **294**(5543):849–52, 2001. 5.1

[GJMM+05] S. Griffith-Jones, S. Moxon, M. Marshall, A. Khanna, S.R. Eddy, and A. Bateman. Rfam: annotating non-coding RNAs in complete genomes. *Nucl. Acids Res.*, **1**:121–124, 2005. 5.8.1

[GK95] R. Giegerich and S. Kurtz. A comparison of imperative and purely functional suffix tree constructions. *Science of Computer Programming*, **25**(2-3):187–218, 1995. 3.5

[GK99] M.Y. Galperin and E.V. Koonin. Searching for drug targets in microbial genomes. *Current Opinion in Biotechnology*, **10**(6):51–57, 1999. 5.7.3

[GKHC01] J. Gough, K. Karplus, R. Hughey, and C. Chlothia. Assignment of homology to genome sequences using a library of hidden Markov models that represent all proteins of known structure. *J. Mol. Biol.*, **313**(4):903–919, 2001. 2.7.3, 4.7, 5.4.4

[GLE90] M. Gribskov, R. Luethy, and D. Eisenberg. Profile analysis. *Meth. Enzymol.*, **183**:146–159, 1990. 2.6

[GME87] M. Gribskov, M. McLachlan, and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *PNAS*, **84**(13):4355–4358, 1987. 1.1, 1.1, 1.1, 2.5, 2.5.1, 2.5.5, 2.6, 3.1

[GMR+03] A. Gattiker, K. Michoud, C. Rivoire, A. H. Auchincloss, E. Coudert, T. Lima, P. Persey, M. Pagni, C.J.A. Sigrist, C. Lachaize, A.-L. Veuthey, and E. Gasteifer. Automatic annotation of microbial proteomes in Swiss-Prot. *Comput. Biol. Chem.*, **27**(1):49–58, 2003. 2.5.1, 2.6

[Gon04] H.G. Gonnet. Some string matching problems from Bioinformatics which still need better solutions. *J. Discrete Algorithms*, **2**(1):3–15, 2004. 3.1

[Got93] O. Gotho. Optimal alignment between groups of sequences and its application to multiple sequence alignment. *Comput. Appl. Biosci.*, **9**(3):361–370, 1993. 2.6

[GS96a] T. Gaasterland and C.W. Sensen. Fully automated genome analysis that reflects user needs and preferences. A detailed introduction to the MAGPIE system architecture. *Biochemie*, **78**(5):302–10, 1996. 5.1.1

[GS96b] T. Gaasterland and C.W. Sensen. MAGPIE: automated genome interpretation. *Trends in Genetics*, **12**(2):76–8, 1996. 5.1.1

[Gus97] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, New York, 1997. 3.4

[GV96] M. Gribskov and S. Veretnik. Identification of sequence patterns with profile analysis. *Methods Enzymol.*, **266**:198–212, 1996. 2.5.5

[GW94] L. Goldstein and M.S. Waterman. Approximations to profile score distributions. *J. Comput. Biol.*, **1**(2):93–104, 1994. 3.8.1

[HB01] J.Y. Huang and D.L. Brutlag. The EMOTIF Database. *Nucl. Acids Res.*, **29**(1):202 – 204, 2001. 2.4.2

[HBB+06] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P.S. Langendijk-Genevaux, M. Pagni, and C. J. A. Sigrist. The PROSITE database. *Nucl. Acids Res.*, **34**(1):D227–D230, 2006. 2.4.2

[HBFB99] K. Hofmann, P. Bucher, L. Falquet, and A. Bairoch. The PROSITE database, its status in 1999. *Nucl. Acids Res.*, **27**(1):215–9, 1999. 2.5.1, 2.6

[HDB98] M. Huynen, T. Dandekar, and P. Bork. Differential genome analysis applied to the species-specific features of Helicobacter pylori. *FEBS Letters*, **426**(1):1–5, 1998. 5.1

[HdlTV03] E. Herrero, M.A. de la Torre, and E. Valentin. Comparative genomics of yeast species: new insights into their biology. *Int. Microbiology*, **6**(3):183–90, 2003. 5.1

[HETC00] J.D. Hughes, P.W. Estep, S. Tavazoie, and G.M. Church. Computational identification of cis-regulatory elements associated with functionally coherent groups of genes in Saccharomyces cerevisiae. *J. Mol. Biol.*, **296**(5):1205–1214, 2000. 2.3

[HGPH00] J.G. Henikoff, E.A. Greene, S. Pietrokovski, and S. Henikoff. Increased coverage of protein families with the Blocks database servers. *Nucl. Acids Res.*, **28**(1):228–230, 2000. 2.5.1, 3.1, 3.2, 4.2, 5.4.4, A.4.1

[HGPM98] A.O. Henriques, P. Glaser, P.J. Piggot, and C.P. Moran. Control of cell shape and elongation by the rodA gene in Bacillus subtilis. *Molecular Microbiology*, **28**(2):235–247, 1998. 5.7.3

[HH89] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Nat. Acad. Sci. U.S.A.*, **89**(22):10915–10919, 1989. 2.5

[HH91] S. Henikoff and J.G. Henikoff. Automated assembly of protein blocks for database searching. *Nucl. Acids Res.*, **19**(23):6565–6572, 1991. 2.3, 2.5.1, A.4.2

[HH94] S. Henikoff and J.G. Henikoff. Position-based sequence weights. *J. Mol. Biol.*, **243**(4):574–578, 1994. 2.5.2

[HH96] J.G. Henikoff and S. Henikoff. Using substitution probabilities to improve position-specific scoring matrices. *Comput. Appl. Biosci.*, **12**(2):135–143, 1996. 2.5.1, 2.5.5, 2.5.5, 2.5.6, 2.5.6, 3.1

[HHAP95] S. Henikoff, J.G. Henikoff, W. Alford, and S. Pietrokovski. Automated construction and graphical presentation of protein blocks from unaligned sequences. *Gene*, **163**(2):17–26, 1995. 2.5.7

[HHG06] T. Höchsmann, M. Höchsmann, and R. Giegerich. Thermodynamic matchers: Strengthening the significance of RNA folding energies. In *Proceedings of the Computational Systems Conference (CSB)*, pages 111–121, 2006. 5.8.1

[HHS90] G. Hertz, G. III Hartzel, and G. Stormo. Identification of consensus patterns in unaligned DNA sequences known to be functionally related. *Comput. Appl. Biosci.*, **6**(2):81–92, 1990. 2.3

[HKB+93] D. Haussler, A. Krogh, M. Brown, I.S. Mian, and K. Sjölander. Protein modeling with hidden Markov models: an analysis of globins. In *Proc. of the 26th Hawaii International Conference on System Sciences*, pages 792–802, Washington, DC, USA, 1993. IEEE Computer Society. 1.1, 2.7.2, 2.7.2

[HKO02] M. Höhl, S. Kurtz, and E. Ohlebusch. Efficient multiple genome alignment. *Bioinformatics*, 18(1):312–320, 2002. 4.3.1

[HP99] J.G. Henikoff and S. Pietrokovski. Blocks+: A non-redundant database of protein alignment blocks derived from multiple compilations. *Bioinformatics*, **15**(6):471 – 479, 1999. 2.5.1, 4.2, A.4.1

[HRCV00] J.V. Helden, A.F. Rios, and J. Collado-Vidies. Discovering regulatory elements in non-coding sequences by analyses of spaced dyads. *Nucl. Acids Res.*, **28**(8):1808–1818, 2000. 2.3

[HS95] G.Z. Hertz and G.D. Stormo. Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps. In *Proc. of the Third International Conference on Bioinformatics and Genome Research*, pages 201–216, Singapore, 1995. World Scientific Publishing Co. 2.5.1

[HS99] G.Z. Hertz and G. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, **15**(7):563–577, 1999. 2.3, 2.3

[HSHA92] J. Heringa, H. Sommerfeldt, D. Higgins, and P. Argos. OBSTRUCT: a program to obtain largest cliques from a protein sequence set according to structural resolution and sequence similarity. *Comput. Appl. Biosci.*, **8**(6):599–600, 1992. 2.5.2

[HSL+04] N. Hulo, C.J.A. Sigrist, V. Le Saux, P. S. Langendijk-Genevaux, L. Bordoli, A. Gattiker, E. De Castro, P. Bucher, and A. Bairoch. Recent improvements to the PROSITE database. *Nucl. Acids Res.*, **32**(1):134–137, 2004. 1.1, 3.1

[HSW03] D. H. Haft, J. D. Selengut, and O. White. The TIGRFAMs database of protein families. *Nucl. Acids Res.*, **31**(1):371–373, 2003. 1.1, 2.7, 2.7.2, 2.7.3, 4.2, 4.7, 5.4.4

[HTG+94] D. Higgins, J. Thompson, T. Gibson, J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucl. Acids Res.*, **22**(22):4673–4680, 1994. 2.5.1, 4.5.2

[HW04] P.M. Haverty and Z. Weng. CisML: an XML-based format for sequence motif detection software. *Bioinformatics*, **20**(11):1815–1817, 2004. 3.11, A.4.2

[HWB90] S. Henikoff, J.C. Wallace, and J.P. Brown. Finding protein similarities with nucleotide sequence databases. *Methods Enzymol.*, **183**:111–132, 1990. 2.5.4

[IHS+01] V.R. Iyer, C.F. Horak, C.S. Scafe, D. Bolstein, M. Snyder, and P.Q. Brown. Genomic binding sites of the yeast cell-cycle transcription factor SBF and MBF. *Nature*, **409**(6819):533–538, 2001. 2.3

[IJI+89] F. Ishino, H.K. Jung, M. Ikeda, M. Doi, M. Wachi, and M. Matsuhashi. New mutations fts-36, lts-33, and ftsW clustered in the mra region of the Escherichia coli chromosome induce thermosensitive cell growth and division. *Journal of Bacteriology*, **171**(10):5523–5530, 1989. 5.7.3

[ISW+89] M. Ikeda, T. Sato, M. Wachi, H.K. Jung, F. Ishino, Y. Kobayashi, and M. Matsuhashi. Structural similarity among Escherichia coli FtsW and RodA proteins and Bacillus subtilis SpoVE protein, which function in cell division, cell elongation, and spore formation, respectively. *Journal of Bacteriology*, **171**(11):6375–6378, 1989. 5.7.3

[JCH95] I. Jonassen, J.F. Collins, and D.G. Higgins. Finding flexible patterns in unaligned protein sequences. *Protein Sci.*, **4**(8):1587–1595, 1995. 2.3

[KA03] P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. In R. Baeza-Yates, E. Chavez, and M. Chrochemore, editors, *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume **2676**, pages 200–210, Springer-Verlag, New York, 2003. Lecture Notes in Computer Science. 1.1, 3.5

[KBB+03] J. Kalinowski, B. Bathe, D. Bartels, N. Bischoff, M. Bott, A. Burkovski, N. Dusch, L. Eggeling, B.J. Eikmanns, L. Gaigalat, A. Goesmann, M. Hartmann, K. Huthmacher, R. Kramer, B. Linke, A.C. McHardy, F. Meyer, B. Mockel, W. Pfefferle, A. Pühler, D.A. Rey, C. Rückert, O. Rupp, H. Sahm, V.F. Wendisch, I. Wiegrabe, and A. Tauch. The complete corynebacterium glutamicum ATCC 13032 genome sequence and its impact on the production of l-aspartate-derived amino acids and vitamins. *J. Biotechnol.*, 104(1-3):5–25, 2003. 5.10

[KBD94] M.M. Khattar, K.J. Begg, and W.D Donachie. Identification of FtsW and characterization of a new FtsW division mutant of Escherichia coli. *Journal of Bacteriology*, **176**(23):7140–7147, 1994. 5.7.3

[KBH98] K. Karplus, C. Barret, and R. Hughey. Hidden Markov models for detecting remote protein homologies. *Bioinformatics*, **14**(10):846–856, 1998. 2.7.2, 4.5

[KDW+04] P.J. Kersey, J. Duarte, A. Williams, Y. Karavidopoulou, E. Birney, and R. Apweiler. The international protein index: An integrated database for proteomics experiments. *Proteomics*, **4**(7):1985–1988, 2004. 5.4.8

[Kei02] P.A. Keich, U. Pevzner. Finding motifs in the twilight zone. *Bioinformatics*, **18**(10):1374–1381, 2002. 2.3

[Ken83] W. Kent. A simple guide to five normal forms in relational database theory. *Communications of the ACM*, **26**(2):120–125, 1983. 5.5.4

[KGH+06] M. Kanehisa, S. Goto, M. Hattori, K.F. Aoki-Kinoshita, M. Itoh, S. Kawashima, T. Katayama, M. Araki, and M. Hirakawa. From genomics to chemical genomics: new developments in KEGG. *Nucl. Acids Res.*, **34**(1):D354–357, 2006. 5.8.1

[KGP+05] A. Krek, D. Grun, M.N. Poy, R. Wolf, L. Rosenberg, E.J. Epstein, P. MacMenamin, I. da Piedade, K.C. Gunsalus, M. Stoffel, and N. Rajewsky. Combinatorial microRNA target predictions. *Nat Genet*, **37**(5):495–500, 2005. 5.8.1

[KGR+03] A.E. Kel, E. Gößling, I. Reuter, E. Cheremushkin, O.V. Kel-Margoulis, and E. Wingender. MATCH: a tool for searching transcription factor binding sites in DNA sequences. *Nucl. Acids Res.*, **31**(13):3576–3579, 2003. 1.1, 2.5.9, 2.5.9, 3.1, 3.2, A.4.2

[KKMBW99] A. Kel, O. Kel-Margoulis, V. Babenko, and E. Wingender. Recognition of NFATp/AP-1 composite elements within genes induced upon the activation of immune cells. *J. Mol. Biol.*, **288**(3):353–376, 1999. 2.5.9

[KLA+01] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time Longest-Common-Prefix Computation in Suffix Arrays and its Applications. In *12th Annual Symposium on Combinatorial Pattern Matching (CPM2001)*, volume **2089**, pages 181–192, Springer-Verlag, New York, 2001. Lecture Notes in Computer Science. 3.5

[KLvHS01] B. Krogh, B. Larsson, G. von Heijne, and E.L.L. Sonnhammer. Predicting transmembrane protein topology with a hidden Markov model: Application to complete genome. *J. Mol. Biol.*, **305**(3):567–580, 2001. 2.7

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, **6**(2):323–350, 1977. 3.4

[KMSH94] A. Krogh, M. Mian, I.S. Sjölander, and D. Haussler. Hidden Markov Models in Computational Biology - Applications to Protein Modeling. *J. Mol. Biol.*, **235**(5):1501–1531, 1994. 1.1, 2.7.2, 2.7.2

[Koo03] E.V. Koonin. Comparative genomics, minimal gene-sets and the last universal common ancestor. *Nature Reviews Microbiology*, **1**(2):127–36, 2003. 5.1

[KS03] J. Kärkkäinen and P. Sanders. Simple Linear Work Suffix Array Construction. In *Proceedings of the 13th International Conference on Automata, Languges and Programming*. Springer, 2003. 1.1, 3.5

[KSPP03] D.K. Kim, S. Sim, J, H. Park, and K. Park. Linear-Time Construction of Suffix Arrays. In R. Baeza-Yates, E. Chavez, and M. Chrochemore, editors, *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM*, volume **2676**, pages 186–199, Springer-Verlag, New York, 2003. Lecture Notes in Computer Science. 1.1, 3.5

[KTP+04] M. G. Kann, P. A. Thiessen, A. R. Panchenko, A. A. Schaeffer, S. F. Altschul, and S. H. Bryant. A structure-based method for protein alignment. *Bioinformatics*, **21**(8):1451–1456, 2004. 2.5

[Kur99] S. Kurtz. Reducing the space requirement of suffix trees. *Software-Practice and Experience*, **29**(13):1149–1171, 1999. 3.5

[Kur05a] S. Kurtz. Lecture Notes for foundations of sequence analysis. `http://www.zbh.uni-hamburg.de/teaching/SS2005/00.914/scriptSommer2005.pdf`, 2005. 4.3

[Kur05b] S. Kurtz. The Vmatch large scale sequence analysis software. `http://www.vmatch.de/`, 2005. 3.10

[LA02] B. Lara and A. Ayala. Topological characterization of the essential Escherichia coli cell division protein FtsW. *FEMS Microbiol. Lett.*, **216**(1):23–32, 2002. 5.7.3

[LAB+93] C.E. Lawrence, S.F. Altschul, M.S. Bogouski, J.S. Liu, A.F. Neuwald, and J.C. Wooten. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, **262**(5131):208–214, 1993. 2.3, 2.5.4, 2.5.6, 2.5.6, 2.5.6

[LCP+06] I. Letunic, R.R. Copley, B. Pils, S. Pinkert, J. Schultz, and P. Bork. SMART 5: domains in the context of genomes and networks. *Nucl. Acids Res.*, **34**(1):D257–D260, 2006. 2.7.3, 4.2, 4.7, 5.4.4

[LFWW03] T. Li, K. Fan, J. Wang, and W. Wang. Reduction of protein sequence complexity by residue grouping. *Protein Engineering*, **16**(5):323–330, 2003. 3.6.1

[LGY+03] L.P. Lim, M.E. Glasner, S. Yekta, C.B. Burge, and D.B. Bartel. Vertebrate microRNA genes. *Science*, **299**(5612):1540, 2003. 5.8.1

[Low76] B. Lowerre. *The Harpy Speech Recognition System.* Carnegie-Mellon University, 1976. 4.7

[LP85] D.J. Lipman and W.R. Pearson. Rapid and sensitive protein similarity searches. *Science*, **227**(4693):1435–1441, 1985. 1.1, 1.1, 1.1

[LR90] C.E. Lawrence and A.A. Reilly. An expectation maximization algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, **7**(1):41–51, 1990. 2.3

[LSJR+03] B.P. Lewis, IH. Shih, M.W. Jones-Rhoades, D.P. Bartel, and C.B. Burge. Prediction of mammalian microRNA targets. *Cell*, **115**(7):787–98, 2003. 5.8.1

[LT06] N. Li and M. Tompa. Analysis of computational approaches for motif discovery. *Algorithms for Molecular Biology*, **1**(8), 2006. 2.3

[LTHK06] K. Liolios, N. Tavernarakis, P. Hugenholtz, and NC. Kyrpides. The Genomes On Line Database (GOLD) v.2: a monitor of genome projects worldwide. *Nucl. Acids Res.*, **34**(1):D332–D334, 2006. 1.1

[Lup96] A. Lupas. Prediction and analysis of coiled-coil structures. *Meth. Enzymology*, **266**:513–525, 1996. 5.4.3

[LVDS91] A. Lupas, M. Van Dyke, and J. Stock. Predicting coiled coils from protein sequences. *Science*, **252**(5010):1162–1164, 1991. 5.4.3

[LXB94] R. Luthy, I. Xenarios, and P. Bucher. Improving the sensitivity of the sequence profile method. *Protein Sci.*, **3**(1):139–146, 1994. 2.5.2

[MAA+07] Nicola J. Mulder, Rolf Apweiler, Teresa K. Attwood, Amos Bairoch, Alex Bateman, David Binns, Peer Bork, Virginie Buillard, Lorenzo Cerutti, Richard Copley, Emmanuel Courcelle, Ujjwal Das, Louise Daugherty, Mark Dibley, Robert Finn, Wolfgang Fleischmann, Julian Gough, Daniel Haft, Nicolas Hulo, Sarah Hunter, Daniel Kahn, Alexander Kanapin, Anish Kejariwal, Alberto Labarga, Petra S. Langendijk-Genevaux, David Lonsdale, Rodrigo Lopez, Ivica Letunic, Martin Madera, John Maslen, Craig McAnulla, Jennifer McDowall, Jaina Mistry, Alex Mitchell, Anastasia N. Nikolskaya, Sandra Orchard, Christine Orengo, Robert Petryszak, Jeremy D. Selengut, Christian J. A. Sigrist, Paul D. Thomas, Franck Valentin, Derek Wilson, Cathy H. Wu, and Corin Yeats. New developments in the InterPro database. *Nucl. Acids Res.*, **35**(1):D224–228, 2007. 4.7

[Mar02] B. Marshall. Helicobacter pylori: 20 years on. *Clinical medicine*, **2**(2):147–152, 2002. 5.7.3

[MBADS+05] A. Marchler-Bauer, J.B. Anderson, C. DeWeese-Scott, N.D. Fedorova, L.V. Geer, M. Gwadz, S. He, D.I. Hurwitz, J.D. Jackson, Z. Ke, C. Lanczycki, C.A. Liebert, C. Liu, F. Lu, G.H. Marchler, M. Mullokandov, B.A. Shoemaker, V. Simonyan, J.S. Song, P.A. Thiessen, R.A. Yamashita, J.J. Yin, D. Zhang, and S.H. Bryant. CDD: a Conserved Domain Database for protein classification. *Nucl. Acids Res.*, **33**(1):D192–196, 2005. 2.5, 5.4.4

[MBG+03] C.O. Marian, S.J. Bordoli, M. Goltz, R.A. Santarella, L.P. Jackson, O. Danilevskaya, M. Beckstette, R. Meeley, and H.W. Bass. The Maize *Single myb histone* 1 gene, *Smh1*, belongs to a novel gene family and encodes a protein that binds telomere DNA repeats in vitro. *Plant Physiology*, **133**(3):1336–1350, 2003. 1.2, 5.7.1, 5.8, 6.1

[MBPS+02] A. Marchler-Bauer, A.R. Panchenko, B.A. Shoemaker, P.A. Thiessen, L.Y. Geer, and S.H. Bryant. CDD: a database of conserved domain alignments with links to domain three-dimensional structure. *Nucl. Acids Res.*, **30**(1):281–3, 2002. 5.4.4

[McC76] E.M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, **23**(2):262–272, 1976. 1.1, 3.4, 3.4.1

[MEA+05] Marcel Margulies, Michael Egholm, William E Altman, Said Attiya, Joel S Bader, Lisa A Bemben, Jan Berka, Michael S Braverman, Yi-Ju Chen, Zhoutao Chen, Scott B Dewell, Lei Du, Joseph M Fierro, Xavier V Gomes, Brian C Godwin, Wen He, Scott Helgesen, Chun Heen Ho, Gerard P Irzyk, Szilveszter C Jando, Maria L I Alenquer, Thomas P Jarvie, Kshama B Jirage, Jong-Bum Kim, James R Knight, Janna R Lanza, John H Leamon, Steven M Lefkowitz, Ming Lei, Jing Li, Kenton L Lohman, Hong Lu, Vinod B Makhijani, Keith E McDade, Michael P McKenna, Eugene W Myers, Elizabeth Nickerson, John R Nobile, Ramona Plant, Bernard P Puc,

Michael T Ronan, George T Roth, Gary J Sarkis, Jan Fredrik Simons, John W Simpson, Maithreyan Srinivasan, Karrie R Tartaro, Alexander Tomasz, Kari A Vogt, Greg A Volkmer, Shally H Wang, Yong Wang, Michael P Weiner, Pengguang Yu, Richard F Begley, and Jonathan M Rothberg. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, **437**(7057):376–380, 2005. 1.1, 1.1, 4.7

[MFG+03] V. Matys, E. Fricke, R. Geffers, E. Gößling, M. Haubrock, R. Hehl, K. Hornischer, D. Karas, A. E. Kel, O. V. Kel-Margoulis, D.-U. Kloos, S. Land, B. Lewicki-Potapov, H. Michael, R. Munch, I. Reuter, S. Rotert, H. Saxel, M. Scheer, S. Thiele, and E. Wingender. TRANSFAC(R): transcriptional regulation, from patterns to profiles. *Nucl. Acids Res.*, **31**(1):374–378, 2003. 3.1, 3.10

[MG77] M. Maxam and W. Gilbert. A new method for sequencing DNA. *Proc. Nat. Acad. Sci. U.S.A.*, **74**(2):560–564, 1977. 1.1

[MG02] M. Madera and J. Gough. A comparison of profile hidden Markov model procedures for remote homology detection. *Nucl. Acids Res.*, **30**(19):4321–4328, 2002. 2.7.2

[MG06] K. Malde and R. Giegerich. Calculating PSSM probabilities with lazy dynamic programming. *J. Functional Programming*, **16**(1):75–81, 2006. 3.10

[MGM+03] F. Meyer, A. Goesmann, A.C. McHardy, D. Bartels, T. Bekel, J. Clausen, J. Kalinowski, B. Linke, O. Rupp, R. Giegerich, and A. Pühler. GenDB–an open source genome annotation system for prokaryote genomes. *Nucl. Acids Res.*, **31**(8):2187–95, 2003. 5.1.1

[MLUL+05] H. Mi, B. Lazareva-Ulitsky, R. Loo, A. Kejariwal, J. Vandergriff, S. Rabkin, N. Guo, A. Muruganujan, O. Doremieux, M.J. Campbell, H. Kitano, and P.D. Thomas. The PANTHER database of protein families, subfamilies, functions and pathways. *Nucl. Acids Res.*, **33**(1):D284–D288, 2005. 2.7.3, 4.7, 5.8.1

[MM90] U. Manber and E.W. Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 319–327, 1990. 1.1

[MM93] U. Manber and E.W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, **22**(5):935–948, 1993. 3.5

[MMBB05] P. Martinez, K. Malde, M. Beckstette, and J. Baguna. The origin of bilateral animals. A multigene approach. *Comparative Biochemistry and Physiology. Abstracts of the Society for Experimental Biology Annual Main Meeting*, **141**:S119–S120, 2005. 5.8

[MN61] J.H. Matthaei and M.W. Nirenberg. Characteristics and stabilization of DNAase-sensitive protein synthesis in E.coli extracts. *Proc. Nat. Acad. Sci. U.S.A.*, **15**(47):1580–1588, 1961. 1.1

[Mor68] D. R. Morrison. PATRICIA - practical algorithm to retrieve information coded in alphanumeric. *Journal of the Association of Computing Machinery*, **15**(4):514–534, 1968. 3.4

[Mou02] Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature*, **420**(6915):520–562, 2002. 5.8.1

[MRHSM02] M. Mollenhauer-Rektorschek, G. Hanauer, G. Sachs, and K. Melchers. Expression of UreI is required for intragastric transit and colonization of gerbil gastric mucosa by Helicobacter pylori. *Research in microbiology*, **153**(10):659–666, 2002. 5.7.3

[MS58] M. Meselson and F.W. Stahl. The replication of DNA in Escherichia coli. *Proc. Nat. Acad. Sci. U.S.A.*, **44**(7):671–682, 1958. 1.1

[MWL00] L. R. Murphy, A. Wallqvist, and R.M. Levy. Simplified amino acid alphabets for protein fold recognition and implications for folding. *Protein Engineering*, **13**(3):149–152, 2000. 3.6.1

[Mye99] G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, **46**(3):395–415, 1999. 2.4.2

[NHH00] C. Notredame, D. Higgins, and J. Heringa. T-Coffee: A novel method for multiple sequence alignments. *J. Mol. Biol.*, **302**(1):205–217, 2000. 2.5.1

[NKKZ06] J.W. Nam, J. Kim, S.K. Kim, and B.T. Zhang. ProMiR II: a web server for the probabilistic prediction of clustered, nonclustered, conserved and nonconserved microRNAs. *Nucl. Acids Res.*, **34**:W455–8, 2006. 5.8.1

[NLL95] A.F. Neuwald, J.S. Liu, and C.E. Lawrence. Gibbs motif sampling: Detection of bacterial outer membrane repeats. *Protein Sci.*, **4**:1618–1632, 1995. 2.3

[NLLL97] A.F. Neuwald, J.S. Liu, D.J. Lipman, and C.E. Lawrence. Extracting protein alignment models from the sequence database. *Nucl. Acids Res.*, **25**(9):1665–1677, 1997. 2.5.2

[NS76] K. Nishikawa and H.A. Scheraga. Geometrical criteria for formation of coiled-coil structures in polypeptide chains. *Macromolecules*, **9**(3):395–407, 1976. 5.4.3

[NW70] S.B. Needlemann and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**(3):443–453, 1970. 1.1, 1.1

[NWB98] C.G. Nevill-Manning, T.D. Wu, and D.L. Brutlag. Highly specific protein sequence motifs for genome analysis. *Proc. Nat. Acad. Sci. U.S.A.*, 95(11):5865 – 5871, 1998. 1.1

[Pea90] W.R. Pearson. Rapid and sensitive sequence comparison with FASTP and FASTA. In Doolittle, R., editor, *Methods Enzymol.*, volume **183**, pages 63–98. Academic Press, San Diego, CA, 1990. 5.4.3

[Pea94] W. R. Pearson. Using the FASTA program to search protein and DNA sequence databases. *Methods Mol. Biol.*, **24**:307–331, 1994. 5.4.3

[Pea99] W.R. Pearson. Flexible sequence similarity searching with the FASTA3 program package. In Misener, S. and Krawetz, S., editor, *Bioinformatics Methods and Protocols*, volume **132**, pages 185–219. Humana Press, Totowa, NJ, 1999. 5.4.3

[PH88] J.D. Palmer and L.A. Herborn. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *J. Mol. Evol.*, **28**(1-2):87–97, 1988. 5.5.2

[Plö05] T. Plötz. *Advanced Stochastic Protein Sequence Analysis.* Dissertation, Faculty of Technology, Bielefeld University, 2005. 4.7

[Pro07] Progeniq BioBoost accelerator boards. Performance benchmarks: BioBoost©HMMer. `http://www.progeniq.com/products/`, 2007. 4.7

[PSTB05] R. Pudimat, E. G. Schukat-Talamazzini, and R. Backofen. A multiple feature framework for modelling and predicting transcription factor binding sites. *Bioinformatics*, **21**(14):3082–3088, 2005. 3.1

[PTS+05] F. Pearl, A. Todd, I. Sillitoe, M. Dibley, O. Redfern, T. Lewis, C. Bennett, R. Marsden, A. Grant, D. Lee, A. Akpor, M. Maibaum, A. Harrison, T. Dallman, G. Reeves, I. Diboun, S. Addou, S. Lise, C. Johnston, A. Sillero, J. Thornton, and C. Orengo. The CATH domain structure database and related resources Gene3D and DHS provide comprehensive domain family information for genome analysis. *Nucl. Acids Res.*, **33**(1):D247–D251, 2005. 2.7.3, 5.4.4

[QFWW95] K. Quandt, K. Frech, E. Wingender, and T. Werner. MatInd and MatInspector: new fast and versatile tools for detection of consensus matches in nucleotide data. *Nucl. Acids Res.*, **23**(23):4878–4884, 1995. 3.1, 3.2

[QSP+05] E. Quevillon, V. Silventoinen, S. Pillai, N. Harte, N. Mulder, R. Apweiler, and R. Lopez. InterProScan: protein domains identifier. *Nucl. Acids Res.*, **33**:W116–W120, 2005. 4.7

[Rab90] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In Waibel, A. and Lee,K. F., editor, *Readings in speech recognition*, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, 1990. 2.7.1, 2.7.2

[Rah03] S. Rahmann. Dynamic programming algorithms for two statistical problems in computational biology. In *Proc. of the 3rd Workshop of Algorithms in Bioinformatics (WABI)*, pages 151–164. LNCS 2812, Springer Verlag, 2003. 3.1, 3.8.1, 3.8.2

[Rat04] Rat Genome Sequencing Project Consortium. Genome sequence of the brown norway rat yields insights into mammalian evolution. *Nature*, **428**(6982):493–521, 2004. 5.8.1

[RD04] M. Regnier and A. Denise. Rare events and conditional events on random strings. *Discrete Math. Theor. Comput. Sci.*, **6**(2):191–214, 2004. 2.3

[Reh02] M. Rehmsmeier. Automatic evaluation of database search methods. *Briefings in Bioinformatics*, **3**(4):342–352, 2002. 4.5

[Reh06] M. Rehmsmeier. Prediction of microRNA targets. *Methods Mol. Biol* , **342**:87–89, 2006. 5.8.1

[RHEC98] F.P. Roth, J.D. Hughes, P.W. Estep, and G.M. Church. Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nature Biotechnology*, **16**(10):939–945, 1998. 2.3

[RJS02] S. Rajasekaran, X. Jin, and J.L. Spouge. The efficient computation of position specific match scores with the fast fourier transformation. *J. Comput. Biol.*, **9**(1):23–33, 2002. 3.1

[RMV03] S. Rahmann, T. Müller, and M. Vingron. On the power of profiles for transcription factor binding site detection. *Statistical Applications in Genetics and Molecular Biology*, **2**(1), 2003. 3.1, 4.1

[RRW+00] B. Ren, F. Robert, J.J. Wyrick, O. Aparicio, E.G. Jennings, I. Simon, J. Zeitlinger, J. Schreiber, N. Hannet, E. Kanin, T.L. Volker, C.J. Wilson, S.P. Bell, and R.A. Young. Genome-wide location and function of DNA binding proteins. *Science*, **290**(5500):2306–2309, 2000. 2.3

[RSHG04] M. Rehmsmeier, P. Steffen, M. Höchsmann, and R. Giegerich. Fast and effective prediction of microRNA/target duplexes. *RNA*, **10**(10):1507–1517, 2004. 5.8.1

[Ruv01] G. Ruvkun. Molecular biology: Glimpses of a tiny RNA world. *Science*, **294**(5543):797–799, 2001. 5.8.1

[RV01] M. Rehmsmeier and M. Vingron. Phylogenetic information improves homology detection. *Proteins: Structure, Function, and Genetics*, **45**(4):360–371, 2001. 4.5

[RYW+00] G.M. Rubin, M.D. Yandell, J.R. Wortman, G.L. Gabor-Miklos, C.R. Nelson, I.K. Hariharan, M.E. Fortini, P.W. Li, R. Apweiler, W. Fleischmann, J.M. Cherry, S. Henikoff, M.P. Skupski, S. Misra, M. Ashburner, E. Birney, M.S. Boguski, T. Brody, P. Brokstein, S.E. Celniker, S.A. Chervitz, D. Coates, A. Cravchik, A. Gabrielian, R.F. Galle, W.M. Gelbart, R.A. George, L.S. Goldstein, F. Gong, P. Guan, N.L. Harris, B.A. Hay, R.A. Hoskins, J. Li, Z. Li, R.O. Hynes, S.J. Jones, P.M. Kuehl, B. Lemaitre, J.T. Littleton, D.K. Morrison, C. Mungall, P.H. O'Farrell, O.K. Pickeral, C. Shue, L.B. Vosshall, J. Zhang, Q. Zhao, X.H. Zheng, and S. Lewis. Comparative genomics of the eukaryotes. *Science*, **287**(5461):2204–15, 2000. 1.1

[RZG+02] J. Raymond, O. Zhaxybayeva, J.P. Gogarten, S.Y. Gerdes, and R.E. Blankenship. Whole-genome analysis of photosynthetic prokaryotes. *Science*, **298**(5598):1616–1620, 2002. 5.5.2

[SAC90] H.O. Smith, T.M. Annau, and S. Chandrasegaran. Finding sequence motifs in groups of functionally related proteins. *Proc. Nat. Acad. Sci. U.S.A.*, **87**(2):826–830, 1990. 2.3

[SAE+04] A. Sandelin, W. Alkema, P. Engstrom, W.W. Wasserman, and B. Lenhard. JASPAR: an open-access database for eukaryotic transcription factor binding profiles. *Nucl. Acids Res.*, **32**(1):D91–D94, 2004. 2.5.1, 3.1, 5.8.1

[SBB+] A. Sczyrba*, M. Beckstette*, A.H. Brivanlou, R. Giegerich, and C.R. Altmann. XenDB: Full length cDNA prediction and cross species mapping in Xenopus laevis. 1.2, 5.7.2, 5.8, 6.1

[SBGA04] A. Sczyrba, M. Beckstette, R. Giegerich, and C.A. Altmann. Identification of 10,500 Xenopus laevis full length clones through EST clustering and sequence analysis. In *Proceedings of the German Conference on Bioinformatics (GCB). Discovery Notes*, volume **P-53**, pages 6–7. Lecture Notes in Informatics, 2004. 5.8

[SC75] F. Sanger and A.R. Coulson. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *J. Mol. Biol.*, **54**(3):441–446, 1975. 1.1

[SCH+82] F. Sanger, A.R. Coulson, G.F. Hong, D.F. Hill, and G.B. Petersen. Nucleotide sequence of bacteriophage lambda DNA. *J. Mol. Biol.*, **162**(4):729–773, 1982. 1.1

[SCK00] C. Suter-Crazzolara and G. Kurapkat. An infrastructure for comparative genomics to functionally characterize genes and proteins. *Genome informatics*, **11**:24–32, 2000. 5.1.1

[SER+99] S.R. Sunyaev, F. Eisenhaber, I.V. Rodchenkov, B. Eisenhaber, V.G. Tumanyan, and E. Kuznetsov. PSIC: profile extraction from sequence alignments with position-specific counts of independent observations. *Protein Engineering*, **12**(5):387–394, 1999. 2.5.2

[SFA99] P. Scordis, D.R. Flower, and T.K. Attwood. FingerPRINTScan: intelligent searching of the PRINTS motif database. *Bioinformatics*, **15**(10):799–806, 1999. 3.1, 3.2, A.4.1

[SH90] G.D. Stormo and G.W. Hartzell. Identifying protein binding sites from unaligned DNA fragments. *PNAS*, **86**(4):1183–1187, 1990. 2.5.4

[SJ99] S. Suerbaum and C. Josenhans. Virulence factors of Helicobacter pylori: implications for vaccine development. *Mol. Med. Today*, **5**(1):32–39, 1999. 5.7.3

[SKB+96] K. Sjölander, K. Karplus, M. Brown, R. Hughey, A. Krogh, and D. Haussler. Dirichlet mixtures: A method for improved detection of weak but significant protein sequence homology. *Comput. Appl. Biosci.*, **12**(4):327–345, 1996. 2.5.1, 2.5.6

[SLSB05] Q.J. Su, L. Lu, S. Saxonov, and D.L. Brutlag. eBLOCKS: enumerating conserved protein blocks to achieve maximal sensitivity and specificity. *Nucl. Acids Res.*, **33**(1):D178–182, 2005. 2.5.1, 2.5.8

[SLZA+02] N. Sabarth, S. Lamer, U. Zimney-Arndt, P.R. Jungblut, T.F. Meyer, and D. Bumann. Identification of surface proteins of Helicobacter pylori by selective biotinylation, affinity purification, and two-dimensional gel electrophoresis. *Journal of Biological Chemistry*, **277**(31):27896–27902, 2002. 5.7.3

[SMD98] J. Stoye, V. Moulton, and A.W.M. Dress. DCA: An efficient implementation of the divide-and-conquer multiple sequence alignment algorithm. *Gene*, **211**(2):GC45–GC56, 1998. 2.5.1

[SSZ07] D.E. Schones, A.D. Smith, and M.Q. Zhang. Statistical significance of cis-regulatory modules. *BMC Bioinformatics*, **8**(19), 2007. 3.1, 3.4.1

[ST03] S. Sinha and M. Tompa. YMF: a program for discovery of novel transcription factor binding sites by statistical overrepresantion. *Nucl. Acids Res.*, **31**(13):3586–3588, 2003. 2.3

[Sta89] R. Staden. Methods for calculating the probabilities for finding patterns in sequences. *Comp. Appl. Biosci.*, **5**(2):89–96, 1989. 3.1, 3.8.1

[STLDR98] S. Skouloubris, J.M. Thiberge, A. Labigne, and H. De Reuse. The Helicobacter pylori UreI protein is not involved in urease activity but is essential for bacterial survival in vivo. *Infection and immunity*, **66**(9):4517–4521, 1998. 5.7.3

[Sto98] J. Stoye. Multiple sequence alignment with the divide-and-conquer method. *Gene*, **211**(2):GC45–GC56, 1998. 2.5.1

[SW81] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, **147**:195–197, 1981. 1.1, 1.1, 2.5, 5.4.3

[SWP+99] A. A. Schaeffer, Y. I. Wolf, C. P. Ponting, E. U. Koonin, L. Aravind, and S.F. Altschul. IMPALA: matching a protein sequence against a collection of PSI-BLAST constructed position specific scoring matrices. *Bioinformatics*, **15**(12):1000–1011, 1999. 2.5

[TAK94] R.L. Tatusov, S.F. Altschul, and E.V. Koonin. Detection of conserved segments in proteins: Iterative scanning of sequence databases with alignment blocks. *Proc. Nat. Acad. Sci. U.S.A.*, **91**(25):12091–12095, 1994. 2.5.1, 2.5.5, 2.5.6, 2.5.6, 2.5.6, 3.1

[TFJ+03] R.L. Tatusov, N.D. Fedorova, J.D. Jackson, A.R. Jacobs, B. Kiryutin, E.V. Koonin, D.M. Krylov, R. Mazumder, S.L. Mekhedov, A.N. Nikolskaya, B.S. Rao, S. Smirnov, A.V. Sverdlov, S. Vasudevan, Y.I. Wolf, J.J. Yin, and D.A. Natale. The COG database: an updated version includes eukaryotes. *BMC Bioinformatics*, **4**(41), 2003. 5.4.5

[THG94] J.D. Thompson, D.G Higgins, and T.J. Gibson. Improved sensitivity of profile searches through the use of sequence weights and gap excision. *Comput. Appl. Biosci.*, **10**(1):19–29, 1994. 2.5.2

[THHM92] G.R. Turbett, P.B. Hoj, R. Horne, and B.J. Mee. Purification and characterization of the urease enzymes of Helicobacter species from humans and animals. *Infection and immunity*, **12**(60):5259–5266, 1992. 5.7.3

[Tim06] TimeLogic biocomputing solution. Performance benchmarks: HMM performance. `http://www.timelogic.com/benchmark_hmm.html`, 2006. 4.6, 4.7

[TKL97] R.L. Tatusov, E.V. Koonin, and D.J. Lipman. A genomic perspective on protein families. *Science*, **278**(5338):631–637, 1997. 5.4.5

[TLB+05] M. Tompa, N. Li, T. L. Bailey, G.M. Church, B. De Moor, E. Eleazar, A.V. Favorov, M.C. Frith, Y. Fu, W.J. Kent, V.J. Makeev, A.A. Mironow, W.S. Noble, G. Pavesi, G. Pesole, M. Regnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, **23**(1):137–144, 2005. 2.3, 2.5

[TLM+01] G. Thijs, M. Lescot, K. Marchal, S. Rombauts, B. De Moor, P. Rouze, and Y. Moreau. Higher-order background model improves the detection of promotor regulatory elements by Gibbs sampling. *Bioinformatics*, **17**(12):1113–1122, 2001. 2.3

[TLRS01] Cormen T.H., C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms, second edition*. MIT Press and McGraw-Hill, 2001. 3.9

[TNG+01] R.L. Tatusov, D.A. Natale, I.V. Garkavtsev, T.A. Tatusova, U.T. Shankavaram, B.S. Rao, B. Kiryutin, M.Y. Galperin, N.D. Fedorova, and E.V. Koonin. The COG database: new developments in phylogenetic classification of proteins from complete genomes. *Nucl. Acids Res.*, **29**(1):22–28, 2001. 5.4.5

[TWK+97] J.F. Tomb, O. White, A.R. Kerlavage, R.A. Clayton, and G.G. Sutton. The complete genome sequence of the gastric pathogen Helicobacter pylori. *Nature*, **388**(6642):539–547, 1997. 5.7.3

[Ukk95] E. Ukkonen. On-line construction of suffix-trees. *Algorithmica*, **14**(3), 1995. 1.1, 3.4

[VA89] M. Vingron and P. Argos. A fast and sensitive multiple alignment algorithm. *Comput. Appl. Biosci.*, 5:115–121, 1989. 4.3.1

[vD00] S. van Dongen. Graph clustering by flow simulation. In *PhD Thesis*. University of Utrecht, The Netherlands, 2000. 5.4.3

[WAC+07] Y. Wang, K.J. Addess, J. Chen, L.Y. Geer, J. He, S. He, S. Lu, T. Madej, A. Marchler-Bauer, P. A. Thiessen, N. Zhang, and S.H. Bryant. MMDB: annotating protein sequences with entrez's 3D-structure database. *Nucl. Acids Res.*, **35**(1):298–300, 2007. 5.4.4

[WC53] J.D. Watson and F.H.C. Crick. A structure for desoxyribose nucleic acid. *Nature*, **171**(4356):737–738, 1953. 1.1

[WCF+98] E. Wingender, X. Chen, E. Fricke, R. Geffers, R. Hehl, I. Liebich, M. Krull, V. Matys, and H. Michael. Databases on transcriptional regulation: TRANSFAC, TRRD, and COMPEL. *Nucl. Acids Res.*, **26**(1):362–367, 1998. 1.1, 2.5.1

[Wei73] P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th IEEE Annual Symposium on Switching and Automata Theory*, pages 1–11, The University of Iowa, 1973. 1.1, 3.4, 3.4

[WESS00] D.L. Weeks, S. Eskandari, D.R. Scott, and G. Sachs. A H+-gated urea channel: the link between Helicobacter pylori urease and gastric colonization. *Science*, **287**(5452):482–485, 2000. 3.15

[WHS05] S. Washietl, I.L. Hofacker, and P.F. Stadler. Fast and reliable prediction of noncoding RNAs. *Proc. Nat. Acad. Sci. U.S.A.*, **102**(7):2454–2459, 2005. 5.8.1

[WJ94] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comput. Biol.*, **1**(4):337–338, 1994. 2.3

[WM92] S. Wu and U. Manber. Fast text searching allowing errors. In *Communications of the ACM*, volume **35**, pages 83–91. ACM Press, New York, NY, USA, 1992. 2.4.2

[WNB99] T.D. Wu, C.G. Nevill-Manning, and D.L. Brutlag. Minimal-risk scoring matrices for sequence analysis. *J. Comput. Biol.*, **6**(2):219–235, 1999. 2.5.1, 2.5.8, 2.5.8, 2.5.8, 3.1

[WNB00] T.D. Wu, C.G. Nevill-Manning, and D.L. Brutlag. Fast probabilistic analysis of sequence function using scoring matrices. *Bioinformatics*, **16**(3):233–244, 2000. 1.1, 3.1, 3.3, 3.3.1, 3.3.1, 3.8.1, 3.8.2, A.4.2

[WNH+04] C.H. Wu, A. Nikoloskaya, H. Huang, L.S.L. Yeh, D.A. Natale, Vinajaka C.R., Z.Z. Hu, R. Mazumder, S. Kumar, P. Kourtesis, R.S. Ledley, B.E. Suzek, L. Arminski, Y. Chen, J. Zhang, J.L. Cardenas, S. Chung, J. Castro-Alvear, G. Dinkov, and W.C. Barker. PIRSF: family classification system at the Protein Information Ressource. *Nucl. Acids Res.*, **32**(1):D112–D114, 2004. 4.7

[WQC06] J.P. Walters, B. Qudah, and V. Chaudhary. Accelerating HMMER sequence analysis suite using conventional processors. In *Proceedings of the 20th International Conference on Advanced Information Networking and Aplications (AINA06)*, volume **1**, pages 289–294, Washington, DC, USA, 2006. IEEE Computer Society. 4.7

[YH01] Y.K. Yu and T. Hwa. Statistical significance of probabilistic sequence alignment and related local hidden Markov models. *J. Comput. Biol.*, **8**(3):249–282, 2001. 4.1

[YMM+06] C. Yeats, M. Maibaum, R. Marsden, M. Dibley, D. Lee, S. Addou, and C.A. Orengo. Gene3D: Modelling protein structure, function and evolution. *Nucl. Acids Res.*, **34**(1):D281–D284, 2006. 4.7

[YTI+98] T. Yada, Y. Totoki, M. Ishikawa, K. Asai, and K. Nakai. Automatic extraction of motifs represented in the hidden Markov model from a number of DNA sequences. *Bioinformatics*, **14**(4):317–325, 1998. 2.3

[ZMD+96] E. Zaychikov, E. Martin, A. Denissova, M. Kozlov, V. Markovtsov, M. Kashlev, H. Heumann, V. Nikiforov, A. Goldfarb, and A. Mustaev. Mapping of catalytic residues in the RNA polymerase active center. *Science*, **273**(5271):107–109, 1996. 2.2

[ZP65] E. Zuckerkandl and L. Pauling. Molecules as documents of evolutionary history. *J. Theoretical Biology*, **8**(2):357–366, 1965. 1.1

[ZSWM00] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning DNA sequences. *J. Comput. Biol.*, **7**(1-2):203–214, 2000. 5.4.3