Dissertation zur Erlangung des Doktorgrades
der Fakultät für Chemie und Pharmazie
der Ludwig–Maximilians–Universität München

Fast, sensitive protein sequence searches using iterative pairwise comparison of hidden Markov models

Michael Remmert
aus Köln

2011

Erklärung:

Diese Dissertation wurde im Sinne von §13 Abs. 3 bzw. 4 der Promotionsordnung vom 29. Januar 1998 (in der Fassung der sechsten Änderungssatzung vom 16. August 2010) von Herrn Professor Dr. Patrick Cramer betreut.

Ehrenwörtliche Versicherung:

Diese Dissertation wurde selbstständig, ohne unerlaubte Hilfe erarbeitet.

München, am 16. September 2011

Michael Remmert

Dissertation eingereicht am: 16. September 2011

1. Gutachter: Prof. Dr. Patrick Cramer
2. Gutachter: Prof. Dr. Dmitrij Frishman

Mündliche Prüfung am: 23. November 2011

Acknowledgements

I owe my gratitude to all those people who have made this dissertation possible by supporting and encouraging me throughout the last years, and I can only try to acknowledge them here.

First and foremost, my deepest gratitude is to my advisor Dr. Johannes Söding for given me the opportunity to work in his group and to contribute to many fascinating projects. Furthermore, I would like to thank Johannes for all the fruitful discussions, the constant support, and for making the last years to such a great experience.

I would like to thank Prof. Dr. Patrick Cramer for being my doctoral supervisor, and Prof. Dr. Dmitrij Frishman for being my second PhD examiner. I am also very grateful to Prof. Dr. Klaus Förstemann, Prof. Dr. Roland Beckmann, Prof. Dr. Ulrike Gaul and Prof. Dr. Karl-Peter Hopfner for offering their time as members of my dissertation committee.

Furthermore, I deeply appreciate the critical reading of this thesis by Dr. Johannes Söding and Theresa Niederberger.

I also would like to thank all members of the Söding and Tresch group for the great atmosphere in our office, for all their help and discussions, and for all the fun at the social activities. Thanks to Andreas, Maria and Andy for the help in integrating their tools in this project.

Last but not least, I am deeply grateful to my parents, my sister Ulrike with Jürgen, my nieces Christiane and Caroline and all my good friends for all the support, patience, trust and encouragement.

Summary

Most sequence-based methods for protein structure or function prediction construct a multiple sequence alignment (MSA) of homologs as a first step. The standard search tool to generate multiple sequence alignments is PSI-BLAST ($>$ 25 000 citations), an extension of BLAST to profile-sequence comparison. PSI-BLAST owes its sensitivity to its use of sequence profiles and to its iterative search scheme. Significant sequence hits are added to the evolving multiple alignment from which a sequence profile is generated for the next search iteration. HMMER3 is similar to PSI-BLAST, but uses a profile hidden Markov model (HMM) instead of a sequence profile to represent the evolving query multiple alignment. The gain in sensitivity over PSI-BLAST is paid by a factor 3 to 4 reduction in speed.

In my thesis work, I have developed HHblits, the first method for iterative sequence searching based on the comparison of profile HMMs. It is faster than PSI-BLAST and HMMER3, more sensitive, and constructs multiple alignments of significantly better quality. The method builds on the HHsearch algorithm for pairwise comparison of profile HMMs, to which it owes its high sensitivity and alignment quality. In parallel to HHblits, our group developed a fast clustering method that can generate a covering set of HMMs for the entire UniProt database in a few weeks time.

To speed up the search by a factor $\sim$ 2000, I developed a prefilter that is based on a novel algorithm for profile-profile comparison designed for maximum speed. The algorithm effectively reduces profile-profile alignment to profile-sequence alignment by coding the database profiles by “column state sequences“ in which profile columns are represented by an alphabet of 219 discretized column states. This permits a fast implementation of the profile-profile comparison that employs the SIMD (single instruction multiple data) instruction sets available on modern CPUs. In this way, HHblits performs 16 byte operations in parallel in a single clock cycle. Several further filtering steps reduce the amount of slow, full-blown HMM-HMM comparisons to a fraction of $< 1/1000$. Our tests show that the loss of sensitivity due to the prefilter is negligible.

On a standard SCOP20 ROC benchmark (SCOP1.73 proteins filtered to 20% maximum sequence identity), HHblits detects twice as many true positives as PSI-BLAST and 54% more than HMMER3 at 1% error rate in the first iteration. Two search iterations HHblits detect significantly more true positives than five PSI-BLAST iterations. Alignment quality is likewise improved significantly. Furthermore, we are able to make confident fold predictions (E-value $< 10^{-3}$) and build structural models for 394 Pfam domains for which no fold prediction has been possible.

HHblits is a robust, general-purpose protein sequence search tool that has the potential to replace PSI-BLAST as state-of-the-art method for the generation of MSAs. Due to the high alignment quality, HHblits alignments are able to improve secondary structure prediction methods such as PSIPRED. Furthermore, these better alignments facilitates the function and structure prediction for proteins for which nothing is yet known.

In the second part of this thesis we study the evolution of outer membrane β-barrels (OMBBs). These proteins are the major class of outer membrane proteins (OMPs) from Gram-negative bacteria, mitochondria and plastids. Their transmembrane domains consist of 8 to 24 β-strands forming a closed, barrel-shaped β-sheet around a central pore. Despite their obvious structural regularity, evidence for an origin by duplication or for a common ancestry had not been found.

We use three complementary approaches to show that all OMBBs from Gram-negative bacteria evolved from a single, ancestral $\beta\beta$ hairpin. First, we link almost all families of known single-chain bacterial OMBBs with each other through transitive profile searches. Second, we identify a clear repeat signature in the sequences of many OMBBs in which the repeating sequence unit coincides with the structural $\beta\beta$ hairpin repeat. Third, we show that the observed sequence similarity between OMBB hairpins cannot be explained by structural or membrane constraints on their sequences. The third approach addresses a longstanding problem in protein evolution: how to distinguish between a very remotely homologous relationship and the opposing scenario of ”sequence convergence“. The origin of a diverse group of proteins from a single hairpin module supports the hypothesis that, around the time of transition from the RNA to the protein world, proteins arose by amplification and recombination of short peptide modules that had previously evolved as cofactors of RNAs. This research provides the basis for the identification and classification of outer membrane β-barrels and explains the evolutionary origin of this important bacterial protein class.

Contents

Acknowledgements iii

Summary iv

1. Motivation and overview 1

I. HHblits - Iterative HMM-based homology searches 3

2. Introduction to remote homology searches 4
- 2.1. Scoring models and gap penalties 4
- 2.2. Pairwise sequence alignment 6
- 2.3. Profile alignment 8
- 2.4. HMM-HMM alignment 9
  - 2.4.1. Log-sum-of-odds score 9
  - 2.4.2. Pairwise alignment of HMMs 10
  - 2.4.3. HHsearch 12
- 2.5. Iterative sequence search methods 12
  - 2.5.1. PSI-BLAST 13
  - 2.5.2. HMMER3 13

3. Material and methods 15
- 3.1. Workflow of HHblits 16
- 3.2. Generation of HHblits databases with kClust 18
- 3.3. Fast prefiltering of HHblits 20
  - 3.3.1. Column state alphabet for sequence profile encoding 20
  - 3.3.2. Generation of the column state alphabet 22
  - 3.3.3. Translation of sequence profiles in column state sequences 23
  - 3.3.4. SSE2 25
  - 3.3.5. Fast prefilter algorithms 25
  - 3.3.6. Gapless local alignment with SSE2 27
  - 3.3.7. Smith-Waterman with SSE2 28
  - 3.3.8. Smith-Waterman with backtrace 32

3.4. Additional filter steps 33
3.4.1. Early stopping 34
3.4.2. Tube shading 35
3.4.3. Avoid aligning the same templates in multiple iterations 36
3.5. *P*-value calculation with neural networks 36
3.6. Realignment of matches by MAC algorithm 39
3.6.1. Consecutive merging of best alignments 40
3.7. Multi-threading support 41
3.8. Compact database by FFINDEX 41

4. Results 43
4.1. Benchmarks 43
4.1.1. Homology detection benchmark 45
4.1.2. Homology detection benchmark with multi-domain proteins 47
4.1.3. Alignment quality 48
4.1.4. Runtime 49
4.1.5. Quality of *E*-values 53
4.1.6. Comparison of HHblits to HHsearch 54
4.1.7. Comparison of profiles built by HHblits or buildali.pl 55
4.1.8. UniProt20 versus NR20 database 57
4.1.9. Using HMMER3-formatted databases 58
4.1.10. Prefilter parameters 60
4.1.11. Parameter optimization 61
4.1.12. Things that did not work 62
4.2. Improved PSIPRED secondary structure prediction with HHblits profiles . . . 63
4.3. Structure prediction for selected Pfam domains 64
4.3.1. CTK3 - a possible new CID 65
4.3.2. PIP49 - a possible Ca^{2+}-activated protein kinase 67
4.3.3. The HAP complex 70
4.4. Web server 72

5. Conclusion and Outlook 74

II. Evolution of outer membrane β-barrels 79

6. Introduction to the evolution of outer membrane β-barrels 80
6.1. Cell envelope of Gram-negative bacteria 80
6.2. Outer membrane β-barrels 82
6.3. Protein evolution 84

7. Material and methods **86**
7.1. Exhaustive, transitive profile searches 86
7.2. Internal repeat detection 88
7.3. Analysis of structural convergence 89

8. Results **91**
8.1. Most OMBBs find each other using homology detection methods 91
8.2. Sequence repeats detected in majority of OMBBs 94
8.3. Sequence similarity not due to structural convergence 99

9. Discussion **103**

III. Appendix **105**

A. Characteristics & Definitions **106**
A.1. Length distributions of various databases 106
A.2. Effective number of sequences 106
A.3. False discovery rate 107

B. Quality of HHblits confidence values **108**

C. Pfam families with newly identified homologous structure **109**

D. HHblits Userguide **113**
D.1. Obtaining HHblits and the databases 114
D.2. Obtaining HHsearch and the databases 114
D.3. Getting started 115
  D.3.1. Overview of programs 115
  D.3.2. Generate a MSA by an iterative HHblits search 115
  D.3.3. Generate a MSA by buildali.pl 117
  D.3.4. Search with a profile HMM through a database 117
  D.3.5. Building customized databases 119
  D.3.6. Maximum Accuracy alignment algorithm 121
  D.3.7. How can I verify if a database match is homologous? 122
D.4. HHsearch/HHblits output: hit list and pairwise alignments 123
  D.4.1. Summary hit list 123
  D.4.2. HMM-HMM pairwise alignments 126
D.5. File formats 128
  D.5.1. Input alignment formats 128
  D.5.2. HHsearch/HHblits model format (hhm-format) 131

D.6. Summary of command-line parameters 134
  D.6.1. `hhmake` – build an HMM from an input MSA 134
  D.6.2. `hhblits` – iteratively search a database of HMMs with a query sequence or MSA 135
  D.6.3. `hhsearch` – search a database of HMMs with a query MSA or HMM 136
  D.6.4. `hhalign` – Align a query MSA/HMM to a template MSA/HMMt . . 138
  D.6.5. `hhfilter` – filter an MSA 139
  D.6.6. `reformat.pl` – reformat one or many alignments 140
  D.6.7. `buildali.pl` – build a PSI-BLAST alignment from a sequence or MSA 141
  D.6.8. `addss.pl` – add predicted secondary structure to an MSA or HMM 143
  D.6.9. `create_db.pl` – creates HHblits databases from HMMER-files, HHM-files or A3M-files 143
  D.6.10. `create_cs_db.pl` – creates HHblits prefilter database 144
  D.6.11. `alignhits.pl` – extract an alignment from a BLAST/PSI-BLAST output 145
  D.6.12. `hhmakemodel.pl` – generate MSAs or coarse 3D models from HH-search results file 146
D.7. Changes from previous versions 147
  D.7.1. 2.0.0 (June 2011) 147
  D.7.2. 1.6.0 (2010) 147
  D.7.3. 1.5.0 (August 2007) 148
D.8. License 151

Bibliography **152**

Lebenslauf **161**

List of Figures

2.1. Possibilities for Needleman-Wunsch alignment extension 6
2.2. Example histogram for a sequence profile 8
2.3. Pairwise alignment of HMMs 11
2.4. Iterative homology search strategy of PSI-BLAST 13
2.5. Iterative homology search strategy of HMMER3 14

3.1. Clustering procedure of the UniProt database 15
3.2. Schematic workflow of HHblits 16
3.3. Detailed workflow of HHblits 17
3.4. kDP algorithm in the kClust clustering 19
3.5. Overview of states in the column state alphabet 22
3.6. Translation of an alignment into a column state sequence 24
3.7. Prefiltering steps 26
3.8. Striped memory layout for SW algorithm 30
3.9. Dependencies in striped memory layout 31
3.10. Scheme of the early stopping filter 34
3.11. Scheme of the tube filter 35
3.12. Extreme value distribution 37
3.13. Neural network topology for training EVD parameters 38

4.1. Hierarchical organization of SCOP database 44
4.2. Homology detection sensitivity of HHblits on a SCOP dataset 46
4.3. ROC5 benchmark of iterative search methods 47
4.4. Homology detection performance of HHblits on multi-domain proteins . . . 48
4.5. Alignment quality of homology detection methods 50
4.6. Runtime performance of HHblits 51
4.7. Runtime dependency on query length and number of CPUs 52
4.8. E-value quality of HHblits 53
4.9. Performance comparison of HHblits and HHsearch 55
4.10. Increase of HHsearch performance by HHblits profiles compared to profiles built with buildali.pl 56
4.11. HHblits performance with different databases 58
4.12. HHblits performance with a database based on HMMER profiles 59

4.13. Improvement of HHblits performance with column state prefiltering 60
4.14. Performance of different prefilter algorithms 61
4.15. New homologous structures for Pfam families identified by HHblits 65
4.16. Structures of the known CIDs and Ctk3 67
4.17. Modeling of human PIP49/FAM69B 68
4.18. Multiple sequence alignment of human PIP49/FAM69B with important residues highlighted 69
4.19. Homologous structure for the HAP complex proteins 71
4.20. Screenshots of the HHblits web server 73

6.1. Structure of the cell envelope of Gram-negative bacteria 81
6.2. Gallery of OMBBs 83

7.1. Double hairpins of the outer membrane protein *OmpLA* 90

8.1. Representative OMBBs of known structure detect each other reliably in homology searches 92
8.2. Cluster map of 15,775 bacterial sequences found with HHsenser starting from *OmpT* 93
8.3. OMBB cluster map of 19 pooled HHsenser searches 95
8.4. 14 representative OMBBs with significant repeats found by HHrepID 96
8.5. Repeat analysis on the 474 cluster from the pooled OMBB cluster map . . . 97
8.6. Many OMBB sequences contain a hidden repeat pattern 98
8.7. Structural versus sequences similarity of double $\beta\beta$ hairpins 100
8.8. Structural versus sequences similarity between OMBBs and special proteins . 101

A.1. Length distribution of different databases 106

B.1. Quality of HHblits confidence values 108

D.1. Benchmark of HHsearch and HHblits on a SCOP20 dataset. 118

List of Tables

4.1. Quality of structural models based on profiles generated by buildali.pl and HHblits . . . 57
4.2. Quality of secondary structure prediction with PSIPRED on different input alignments . . . 63

7.1. Representative OMBBs for HMM-HMM comparisons . . . 87
7.2. Representative OMBBs for HHsenser runs . . . 88

8.1. Proteins from Gram-positive bacteria in the *OmpT* cluster map . . . 93
8.2. Sequence repeats are detected in 14 out of the 23 representative OMBBs . . 97

A.1. Length distribution of different databases . . . 107

C.1. List of 394 Pfam families with newly identified homologous structure 109

Abbreviations

AVX	advanced vector extensions
BLAST	basic local alignment search tool
BLOSUM	block substitution matrix
CASP	Critical Assessment of protein Structure Prediction
CLANS	cluster analysis of sequences
CPU	central processing unit
CS-BLAST	context-specific BLAST
CS62	column state alphabet with 62 states
DNA	deoxyribonucleic acid
DP	dynamic programming
EM	expectation maximization
FDR	false discovery rate
FP	false positive
FPGA	field-programmable gate array
GPU	graphics processor unit
HHblits	HMM-HMM based local iterative sequence search
HMM	hidden Markov model
IM	inner membrane
kDP	k-mer dynamic programming
MAC	maximum accuracy
MMX	multi media extension
MSA	multiple sequence alignment

NCBI	National Center for Biotechnology Information
OM	outer membrane
OMBB	outer membrane β-barrel
OMP	outer membrane protein
PAM	point accepted mutation
PDB	protein data bank
Pfam	protein family database
PPI	protein-protein interaction
PSI-BLAST	position-specific iterated BLAST
RMSD	root mean square deviation
RNA	ribonucleic acid
ROC	receiver operating characteristic
SCOP	structural classification of proteins
SIMD	single instruction, multiple data
SOV	segment overlap
SSE2	streaming SIMD extensions 2
SW	Smith-Waterman
TM	transmembrane
TMBB	transmembrane β-barrel
TP	true positive
UniProt	universal protein resource

1. Motivation and overview

HHblits - Iterative HMM-based homology searches

In the last years, due to progress in high-throughput technologies more and more genomes are being sequenced and the bioinformatic databases with protein sequences (i. e., UniProt) grow at an ever increasing rate. Therefore, fast computational methods for analyzing and interpreting this data are becoming very important. This holds especially for the generation of multiple sequence alignments (MSAs), which are a key intermediate step in the sequence-based prediction of evolutionarily conserved properties, such as secondary or tertiary structure, disorder, transmembrane regions, post-translational modifications, short linear motifs, or interaction interfaces. An MSA alignes homologous protein sequences, that means, sequences from proteins with a common evolutionary ancestor, in a way as to place related amino acids in the same column. Conserved residues in a MSA of homologous proteins often indicate structurally or functionally important regions, e. g., catalytic sites.

The performance of nearly all prediction methods, which use a multiple sequence alignment as input, depends to a large degree on the quality of this MSA. The standard search tool to generate MSAs is PSI-BLAST ($>$ 25 000 citations), an extension of BLAST to profile-sequence comparison. PSI-BLAST owes its sensitivity to its use of sequence profiles and to its iterative search scheme. Significant sequence hits are added to the evolving multiple alignment from which a sequence profile is generated for the next search iteration.

In the main part of this thesis (part I) we present HHblits ($\underline{\text{H}}$MM-$\underline{\text{H}}$MM-$\underline{\text{b}}$ased $\underline{\text{l}}$ightning-fast $\underline{\text{it}}$erative sequence $\underline{\text{s}}$earch), the first iterative sequence search method based on the pairwise comparison of Hidden Markov Models (HMMs). This method is based on the very sensitive HMM-HMM alignment method HHsearch and extends it to enable fast, iterative sequence searches through large HMM databases representing all the known sequence space. Its profile-profile alignment prefilter makes HHblits faster than PSI-BLAST yet as sensitive as HHsearch with a high alignment quality. Our method replaces PSI-BLAST in the MSA-generation step in our protein structure prediction server HHpred, the best-scoring server in template-based structure prediction during the last CASP9 blind structure prediction benchmark. HHblits will be published in Nature Methods (Remmert et al., 2011).

The first chapter of this part (chapter 2) gives an introduction to remote homology searches with the widely used scoring models and protein alignment algorithms, sequence profiles and profile HMMs, the pairwise alignment of HMMs, and the state-of-the-art iterative sequence search methods PSI-BLAST and HMMER3. In chapter 3, we present the workflow of

HHblits and the methods we used to achieve a fast and sensitive sequence searching method. This includes the fast clustering of the protein sequence database, the fast prefiltering procedure with a column-state alphabet, various further filter steps and improvements in the core code of HHsearch and HHblits. Chapter 4 describes various benchmarks to test the performance of HHblits in comparison with the other iterative sequence search methods PSI-BLAST and HMMER3. In addition, we demonstrate the utility of HHblits by predicting new structures for Pfam families, for which no protein structure is known. In the last chapter of this part (chapter 5) we discuss the performance and usability of our method and give an outlook to further possible improvements to HHblits.

Evolution of outer membrane β-barrels

The second part of this thesis (part II) deals with the evolution of outer membrane β-barrel proteins (OMBBs). These proteins are the major class of outer membrane proteins (OMPs) from Gram-negative bacteria, mitochondria and plastids. They are composed of a closed, barrel-shaped β-sheet. Until this work, a common ancestry for this class with a high structural regularity could not be identified. We used three complementary approaches based on our HMM-comparison methods to show that all OMBBs from Gram-negative bacteria descended from a single, ancestral $\beta\beta$ hairpin. The main part of this analysis was published in 2010 in Molecular Biology and Evolution (Remmert et al., 2010).

Chapter 6 gives an overview about outer membrane β-barrels and the evolution of proteins in general. The methods we used to investigate the evolution of the OMBBs are explained in chapter 7. In chapter 8 we show the results of our three approaches. First, we could link almost all families of known single-chain bacterial OMBBs with each other through transitive profile searches. Second, we could identify a clear repeat signature in the sequences of many OMBBs in which the repeating sequence unit coincides with the structural $\beta\beta$ hairpin repeat. Third, we show that the observed sequence similarity between OMBB hairpins cannot be explained by structural or membrane constraints on their sequences. The final chapter 9 discusses the analyses carried out in this study and gives an outlook for further use of these analyses.

Part I.

HHblits - Iterative HMM-based homology searches

2. Introduction to remote homology searches

The function of a protein depends on its 3-dimensional structure and this structure is dependent on the amino acid sequence. But from all possible protein sequences - a protein with 100 amino acids can have 20^{100} possible sequences - only a negligible fraction of them are able to fold into a stable structure (Söding and Lupas, 2003). Hence it is very unlikely that a new protein develops *de novo* from a random sequence of amino acids. Rather, protein sequences evolve from pre-existing ancestral sequences. By recognizing similarities between the sequence of an uncharacterized protein and sequences of already characterized proteins, it is therefore often possible to infer the function or structure of the new protein. Two proteins with a common ancestor are said to be homologous. In computational sequence analysis, homology detection has become a basic procedure in areas such as protein evolution and structure and function prediction of new proteins. Likewise, homology information is used to explore cellular and developmental processes based on the study of homologous proteins in simple and well studied model organisms.

The functionally or structurally important regions of proteins such as catalytic sites are assumed to be highly conserved in homologous sequences. Alignments of protein sequences are a way to identify such regions. These alignments are computed by the insertion of gaps in order to place conserved and homologous residues in the same column. Alignment methods use a scoring scheme to align the protein sequences. This scheme ideally assigns the highest score to the true alignment. The right choice of such a scoring scheme and the development of more sensitive schemes is important in protein sequence analysis, because the accuracy of an alignment method often depends on the underlying scoring scheme. In the next sections the scoring procedure of alignments is described.

2.1. Scoring models and gap penalties

The aim of comparing protein sequences is to check whether there is evidence for a common ancestor from which they have diverged by mutation and selection. Thus, a scoring scheme has to take into account that in nature some amino acid mutations may arise more often than others, for example the mutation from a hydrophobic amino acid to another hydrophobic one is more likely than to a charged amino acid. It is also commonly assumed that mutations at different sites in the sequence occur independently. Then the score of an alignment is computed by a sum of terms for each pair of aligned amino acids and a penalty score for each gap. The score for a pair of aligned amino acids a and b is defined as ratio of the probability of a and b under a match-model M $(P(a,b|M))$ and the probability of a and b under the null- or random-model R $(P(a,b|R))$. In the null- or random-model R each amino acid a occurs independently with background probability $f(a)$ and the probability of an alignment of two unrelated residues a and b is simply the product of their background frequencies: $f(a) \cdot f(b)$. In the match-model M two residues occur with the joint probability $p(a,b)$ which is usually derived from the observed numbers of amino acid substitutions in a large set of training alignments. Such a scoring scheme can be described by the *log-odds-ratio*

$$s(a,b) = \log\left(\frac{P(a,b|M)}{P(a,b|R)}\right) = \log\left(\frac{p(a,b)}{f(a)f(b)}\right) \tag{2.1}$$

These scores are positive for conservative substitutions, i. e., substitutions between amino acids with similar properties, whereas non-conservative substitutions will results in a negative score because they are expected to occur less frequently in real alignments. Substitution scores $s(a,b)$ are arranged in *substitution matrices* of dimension 20×20.

One of the first substitution matrices was the PAM (Point Accepted Mutation) matrix published by Dayhoff *et al.* in 1978 (Dayhoff et al., 1978). The authors set up the PAM1 matrix based on very similar sequences. This matrix estimates what rate of substitution would be expected for 1% accepted mutations. This matrix is then extrapolated to matrices for more diverse sequences up to matrix PAM250, which corresponds to a sequence identity of roughly 20%. But it showed up that these matrices can not well approximate changes over long evolutionary time scales. To solve this problem, Henikoff and Henikoff developed a new family of substitution matrices, the BLOSUM (BLOck SUbstitution Matrix) family in 1992 (Henikoff and Henikoff, 1992). These matrices are based on the BLOCKS database and count substitutions in ungapped regions of alignments which have a predefined maximal sequence similarity. So the BLOSUM50 matrix is calculated from sequences with 50% or less sequence identity. The most widely used substitution matrix in alignment applications is the BLOSUM62.

The other part of the alignment score is the gap penalty, a negative score for each insertion of a gap into a sequence. There are two ways for penalizing gaps: (1) The linear gap penalty of length g is defined as

$$\gamma(g) = -gd \tag{2.2}$$

with the *gap-open* penalty d. (2) The affine gap penalty uses in addition to the *gap-open* penalty d a *gap-extension* penalty e

$$\gamma(g) = -d - (g-1)e \tag{2.3}$$

Usually, the *gap-extension* penalty e is set to something smaller than the *gap-open* penalty d to penalize one long insertion or deletion less than several small ones. This corresponds to the observation that gaps occur preferably in consecutive stretches.

2.2. Pairwise sequence alignment

Pairwise alignment methods try to efficiently compute an optimal alignment for a pair of sequences $x = x_1 \dots x_n$ and $y = y_1 \dots y_m$ based on a given scoring scheme. In 1970, Needleman and Wunsch developed a *dynamic programming* (DP) algorithm for building an optimal *global alignment* (Needleman and Wunsch, 1970) which was later revised by Gotoh to a more efficient version (Gotoh, 1982). In a global alignment, the sequences are aligned end to end and this algorithm guarantees to find the optimal solution by using previous calculated optimal alignments of smaller subsequences. It computes a $n \times m$ matrix S, where $S(i,j)$ denotes the score of the optimal alignment between the subsequences $x_1 \dots x_i$ and $y_1 \dots y_j$. In the case of a pairwise alignment, there are three ways to extend an alignment: (1) a pair of aligned residues, (2) a residue in sequence x aligned to a gap in sequence y, or (3) a gap in x aligned to a residue in y (see Fig. 2.1B). This allows to recursively calculate the best score at position (i,j) with the following formula:

$$S(i,j) = \max \begin{cases} S(i-1,j-1) + s(x_i,y_j), \\ S(i-1,j) - d, \\ S(i,j-1) - d. \end{cases} \tag{2.4}$$

$s(x_i,y_j)$ is the substitution score between the two amino acids x_i and y_j derived from a substitution matrix such as BLOSUM62. The algorithm fills the matrix from the upper left

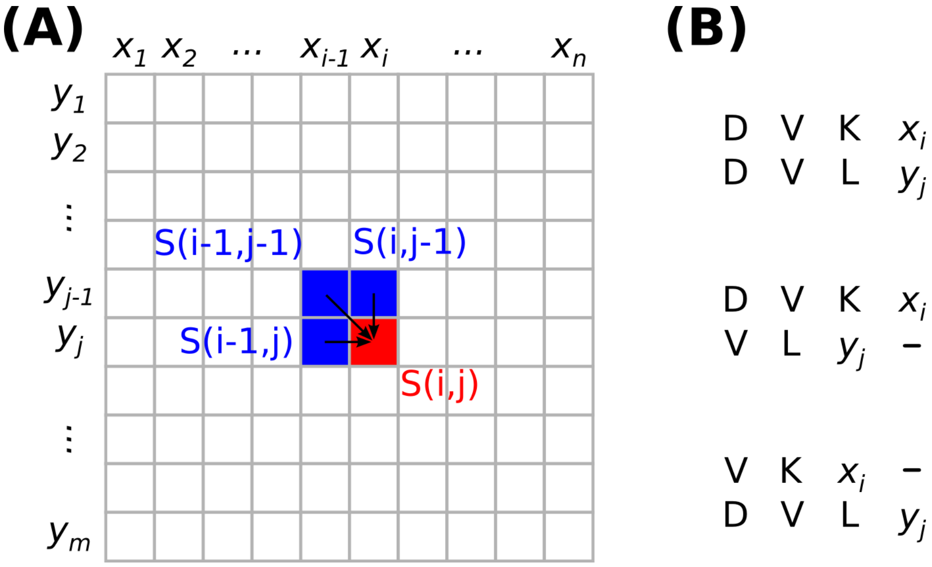


Figure 2.1.: (A) Example matrix for a pairwise alignment between two subsequences $x_1 \dots x_n$ and $y_1 \dots y_m$ with the Needleman-Wunsch algorithm. This matrix is filled starting in the upper left corner and repeatedly calculating the best score $S(i,j)$ up to position (i,j) by applying the recursive formula 2.4. The score $S(i,j)$ of the red cell is computed from the above-left, left, or above cell (blue). (B) The three possible ways of how an alignment can be extended up to positions (i,j).

corner to the lower right corner by applying formula 2.4 to calculate each $S(i,j)$ value from the above-left, left, or above cell as shown in figure 2.1A.

After filling this matrix the value in the last cell $S(n,m)$ is the score of the best global alignment. To identify the alignment itself, a traceback method is used starting in the last cell and identifying for each cell which of the three choices contributed to its value. Since the traceback starts at $S(n,m)$, the alignment is actually built in reverse.

In nature, there are often proteins which share one domain but differ in the rest of the protein sequence. In this case the assumption for *global alignments*, that there exists a biologically meaningful alignment between the entire sequences, fails and a *local alignment* is the better choice. Local alignment methods compute an optimal alignment between two subsequences of x and y and are usually the most sensitive method for comparing highly diverged sequences, even if they have a common ancestor over the entire length. In 1981, Smith and Waterman published an algorithm for local alignments (Smith and Waterman, 1981) which is closely related to the Needleman-Wunsch algorithm for global alignments. To address the problem that an alignment can start at every position in the matrix, the formula 2.4 is updated to allow $S(i,j)$ to become 0 if all other options have a value less than 0:

$$S(i,j) = \max \begin{cases} 0, \\ S(i-1,j-1) + s(x_i,y_j), \\ S(i-1,j) - d, \\ S(i,j-1) - d. \end{cases} \tag{2.5}$$

Here, the score of the best alignment is given by the cell with the highest value, which is not necessarily the cell (n,m). The traceback starts at this position and ends if one cell has the value 0.

Both pairwise alignment methods, as described above, use the linear gap cost structure as in equation 2.2. They can be extended to use the affine gap cost (see equation 2.3) by calculating three instead of one matrix $S(i,j)$. Such methods are described in detail in (Durbin et al., 1998) whereat the general concept of dynamic programming and complexity stay the same.

Pairwise alignments calculated by such methods can be used to detect homology between different proteins. One can easily see that the above described Needleman-Wunsch and Smith-Waterman algorithms have a time complexity of $O(nm)$, the product of the lengths of the two sequences being compared. It is obvious that methods with such a runtime can not be used for searching current databases (e.g. the UniProt database [1] with more than 14 million sequences (04_2011)). In practice heuristic methods are used that are much faster but miss possibly significant alignments.

---

[1] `http://www.uniprot.org/`

BLAST

The most used heuristic method is BLAST (Basic Local Alignment Search Tool) (Altschul et al., 1990) that employs a *seed and extend* heuristic in which first small exact matches are found which are further extended to longer inexact ones. This algorithm starts with generating a list of k-mers (usually $k = 3$ for proteins) which have a similarity bigger than a given threshold T to some k-mer in the query sequence. Then it scans the database for all occurrences of these k-mers and every hit (seed) is extended until the score drops a certain distance below the best score computed so far for the seed. Finally, all hits with a score greater than a given threshold are reported. But this method has a dilemma regarding sensitivity and speed; a k-mer must be sensitive enough to find as many homologs as possible, but at the same time it should give as few chance matches as possible, which in turn triggers numerous unneeded extensions. A new version of BLAST (Altschul et al., 1997) softens this problem by using a 2-hit strategy. In this approach extensions are generated for consecutive matches of two short k-mers on the same diagonal within a window of 40 residues. The 2-hit approach reduces the number of chance matches and increases the sensitivity by lowering the similarity threshold of the 1-hit approach.

2.3. Profile alignment

Sequence-sequence comparison methods as described before work well for closely related sequences. But they are not sensitive enough for the identification of homologous proteins that are strongly diverged. Such homologs are important for 3D structure prediction, protein function prediction and for the analysis of protein evolution. Extending the limits of sensitivity is therefore of great practical importance.

A great improvement in sensitivity was achieved by developing *profile-sequence* methods like PSI-BLAST (see 2.5.1) in 1997 (Altschul et al., 1997). A profile is built from an alignment of homologous sequences. Such an alignment contains more evolutionary information about the sequence family than a single protein sequence. From the alignment with length L one can compute the amino acid frequencies in each column and generate the profile, which is simply a $20 \times L$ matrix recording the amino acid frequencies.

In this work, profiles are visualized as histograms, in which the bar heights are proportional to the corresponding amino acid probabilities in the sequence profile and colors indicate amino acid properties. (Figure 2.2). The power of a profile is given by the ability

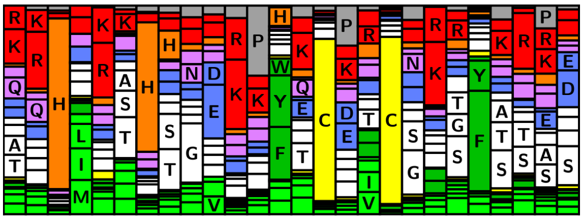


Figure 2.2: In this work sequence profiles are shown as histograms, in which the bar heights are proportional to the corresponding amino acid probabilities in the sequence profile. The different amino acids are colored according to their properties (e.g. hydrophobic amino acids in green).

to distinguish between conserved residues, which indicates functional or structural important regions, and residues that vary a lot in the family. The algorithm for computing an alignment between a profile and a sequence is almost completely analogous to the algorithm for sequence-sequence alignments. The only difference is that the profile matrix is used for the substitution score of two positions i and j and not a substitution matrix.

The next improvement in sensitivity was made possible by comparing profiles with profiles, thus using evolutionary information on the query and the template side. Examples for early profile-profile comparison programs are LAMA (Pietrokovski, 1996), PROF_SIM (Yona and Levitt, 2002) and COMPASS (Sadreyev and Grishin, 2003).

2.4. HMM-HMM alignment

In 2005, Söding published a new alignment method (Söding, 2005). This method is based on the comparison of profile HMMs (Hidden Markov Models) and it has proven to be the most powerful method for remote homology detection in proteins. A profile HMM contains, additionally to the amino acid frequencies given in sequence profiles, position specific probabilities for insertions and deletions along the alignment. Thus, an insertion or deletion that occurs at the same position as found in many sequences in the sequence family is less penalized than an insertion at another position. Each column of a profile HMM contains a match state M, a delete state D and an insert state I. In the following, we show how the log-odds score is generalized to the case of HMM-HMM comparison and explain a method to efficiently compute such an alignment.

2.4.1. Log-sum-of-odds score

In the case of sequence-HMM comparison, the log-odds score can be written as

$$S_{LO} = \log \frac{P(x_1, \ldots, x_L|\text{emission on path})}{P(x_1, \ldots, x_L|\text{Null})} \tag{2.6}$$

This log-odds score is a measure for how much more probable it is that a sequence is emitted by the HMM rather than by a random null-model. The probability for the null-model is simply the product of the amino acid background frequencies.

In the case of HMM-HMM comparison, an alignment between two profile HMMs corresponds to a certain path through the two HMMs. The generalization of the log-odds score to such a case is the so called *log-sum-of-odds score* (LSO):

$$S_{LSO} = \log \sum_{x_1,\ldots,x_L} \frac{P(x_1, \ldots, x_L|\text{co-emission on path})}{P(x_1, \ldots, x_L|\text{Null})} \tag{2.7}$$

The sum runs over all sequences with length L that can be emitted along the alignment path through the HMMs. One can easily see that this log-sum-of-odds score is indeed a generalization of the log-odds score, because in the case of sequence-HMM comparison only one term in the sum can contribute and equation 2.7 can be reduced to equation 2.6.

To apply standard dynamic programming algorithms (e. g., the Viterbi algorithm) in order to find the HMM-HMM alignment with the highest log-sum-of-odds score, a similarity measure is needed that allows to compare the amino acid distributions of two columns i and j in the HMMs p and q. Therefore, equation 2.7 can be rewritten as

$$S_{LSO} = \sum_{k:X_k Y_k = MM} S_{aa}\left(q_{i(k)}, p_{j(k)}\right) + \log P_{tr} \tag{2.8}$$

X_k,$Y_k \in \{M,I,D\}$ are the states in the HMMs p and q in the kth column of the pairwise alignment of these HMMs. P_{tr} is the product of all transition probabilities for the path through p and q and the column score $S_{aa}(q_i, p_j)$ is given by

$$S_{aa}(q_i, p_j) = \log \sum_{a=1}^{20} \frac{q_i(a) p_j(a)}{f(a)} \tag{2.9}$$

in which $q_i(a)$ and $p_j(a)$ denotes the emission probability of amino acid a in the columns i and j of the HMMs and $f(a)$ id the fixed background frequency of the amino acid a. The division by $f(a)$ can be seen as a weighting factor increasing the weight of rare amino acids compared to more common ones. For two similar amino acid distributions this column score is positive, for dissimilar distributions it is negative. This is important for the computation of local alignments.

2.4.2. Pairwise alignment of HMMs

Figure 2.3a shows a profile HMM with each column containing a match state M, a delete state D and an insert state I. Because only match and insert states can emit amino acids, a match state in one HMM can only be aligned with a match state (MM) or an insert state (MI) in the other HMM. A delete state can only be aligned to a gap (DG) where the interpretation of a gap is analogous to that of a gap in a pairwise sequence alignment, meaning that for the column aligned to the gap there exists no column in the other HMM that evolved from the same ancestor. Thus, five possible states, MM, MI, IM, DG and GD, exists in an HMM-HMM alignment (shown with the allowed pair state transitions in Fig. 2.3c).

A path with the maximal log-sum-of-odds score (equation 2.8) can be calculated with a variant of the Viterbi algorithm that maintains five dynamical programming matrices S_{XY}, one for each pair state $XY \in \{MM, MI, IM, DG, GD\}$. These matrices are calculated

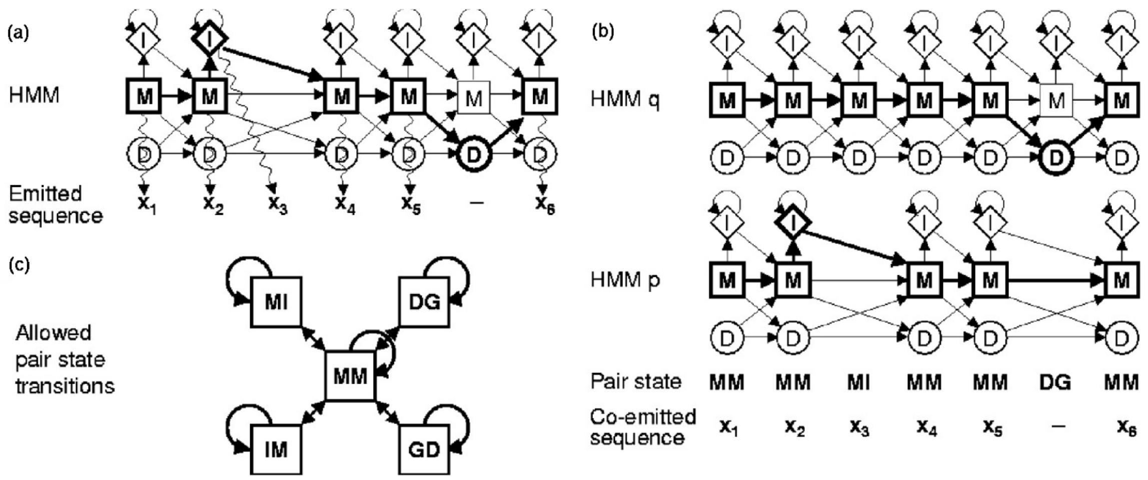


Figure 2.3.: (a) The alignment of a sequence to a profile HMM can be represented by a path through the HMM (bold arrows). Each column in the HMM contains a matching state M, a delete state D and an insert state I. (b) Alignment of two HMMs with a path through the HMMs (bold arrows) corresponding to a sequence co-emitted by both HMMs. (c) Allowed transitions between pair states. (Figure taken from Söding (2005))

recursively (similar to the Needleman-Wunsch algorithm) by

$$S_{MM}(i,j) = S_{\mathrm{aa}}(q_i,p_j) + \max \begin{cases} S_{MM}(i-1,j-1) + \log\left(q_{i-1}(M,M)\, p_{j-1}(M,M)\right) \\ S_{MI}(i-1,j-1) + \log\left(q_{i-1}(M,M)\, p_{j-1}(I,M)\right) \\ S_{IM}(i-1,j-1) + \log\left(q_{i-1}(I,M)\, p_{j-1}(M,M)\right) \\ S_{DG}(i-1,j-1) + \log\left(q_{i-1}(D,M)\, p_{j-1}(M,M)\right) \\ S_{GD}(i-1,j-1) + \log\left(q_{i-1}(M,M)\, p_{j-1}(D,M)\right) \end{cases} \tag{2.10}$$

$$S_{MI}(i,j) = \max \begin{cases} S_{MM}(i-1,j) + \log\left(q_{i-1}(M,M)\, p_j(M,I)\right) \\ S_{MI}(i-1,j) + \log\left(q_{i-1}(M,M)\, p_j(I,I)\right) \end{cases} \tag{2.11}$$

$$S_{DG}(i,j) = \max \begin{cases} S_{MM}(i-1,j) + \log\left(q_{i-1}(M,D)\right) \\ S_{DG}(i-1,j) + \log\left(q_{i-1}(D,D)\right) \end{cases} \tag{2.12}$$

and similar for $S_{IM}(i,j)$ and $S_{GD}(i,j)$. $q_i(X,X')$ and $p_j(Y,Y')$ denote the transition probabilities to go from state $X,Y \in \{M,I,D\}$ in column i or j to a state $X',Y' \in \{M,I,D\}$, and $S_{XY}(i,j)$ is the score of the best partial alignment which ends in column i of HMM q and column j of HMM p in state XY. Equation 2.10 shows the maximization of a global alignment. For the calculation of a local alignment, a zero has to be added as a sixth case to this maximization equation. The optimal alignment is constructed as usual by backtracking from the cell with maximum score.

2.4.3. HHsearch

The pairwise alignment of HMMs is implemented in the homology search method HHsearch. This tool further increases its sensitivity by using additional secondary structure information. Therefore, a secondary structure score $S_{ss}(q_i,p_j)$ based on the predicted secondary structure and its confidence is added to the amino acid column score $S_{aa}(q_i,p_j)$ in equation 2.10.

The background frequencies of the amino acids, used by the null model, can be chosen by the user, e. g., the background frequencies calculated from a dataset or the average frequencies from the query or template HMM. It seems that the average amino acid distribution from query and template HMM ($f_{null}(aa) = 0.5 \cdot (q.f(aa) + t.f(aa))$) works best.

In 2001, Pei et al. have shown that in alignments of homologous sequences conserved columns tend to occur in clusters along the sequence (Pei and Grishin, 2001). So in an alignment of two homologous HMMs it could be expected that conserved columns also occur in clusters. Therefore, a correlation score S_{corr} is added to the Viterbi score of the HMM-HMM comparison.

$$S_{corr} = w_{corr} \sum_{d=1}^{4} g(d) \tag{2.13}$$

$g(d)$ is a correlation function which describes the correlation of the score S_{aa} in column l and in a column with a fixed sequence separation d:

$$g(d) = \sum_{l=1}^{L-d} S_l S_{l+d} \qquad \text{with} \quad S_l = S_{aa}(q_{i(l)}, p_{j(l)}) \tag{2.14}$$

The weight parameter w_{corr} was optimized to 0.1 on a test set of 317 × 317 pairwise alignments.

2.5. Iterative sequence search methods

A further increase in sensitivity of remote homology detection in proteins could be reached by the use of *iterative* search methods. Iterative search methods perform multiple searches of a database. After each iteration the information of homologous sequences is incorporated into a model for the query sequence and this model is used for re-searching the database. The first widely used iterative searching method was PSI-BLAST (Altschul et al., 1997) (with more than 25000 citations), which builds upon BLAST and performs a profile-sequence search strategy. A more sensitive iterative method was recently developed in the new version 3 of the HMMER package (Eddy, 2009) that consists of programs for HMM-sequence comparison. In the following we give a brief introduction into these widely used iterative sequence search methods with which we compare our new method HHblits. HHblits itself will be explained in detail in chapter 3.

2.5.1. PSI-BLAST

PSI-BLAST is an iterative extension of the widely used BLAST program (see 2.2) developed by Altschul *et. al* in 1997 (Altschul et al., 1997). The first iteration of PSI-BLAST is identical to a normal BLAST search. After the first iteration, all putative homologous hits, that means all hits with an E-value below a given threshold, are added to the query sequence and a new query profile is generated. The next search iteration is then performed with this new profile. This scheme is given in figure 2.4 and it is repeated in all following iterations, so the query profile is refined with new homologs after each iteration. This iterative scheme is performed until no new sequences are found or the maximum number of iterations is reached. Thus, by incorporating distantly homologous sequences, PSI-BLAST is able to generate a more informative, diverse model for the query and so this method is more sensitive than BLAST for detecting remotely homologous proteins.

2.5.2. HMMER3

HMMER3 is a software for HMM-sequence comparison (Eddy, 2009). The iterative search method in HMMER3 is called JACKHMMER. In the first iteration, a profile HMM is build from the query sequence using a simple scoring scheme with BLOSUM62 scores converted to probabilities plus gap-open and gap-extend probabilities. With this profile HMM a sequence database is searched and all hits that pass the inclusion threshold are added to the query

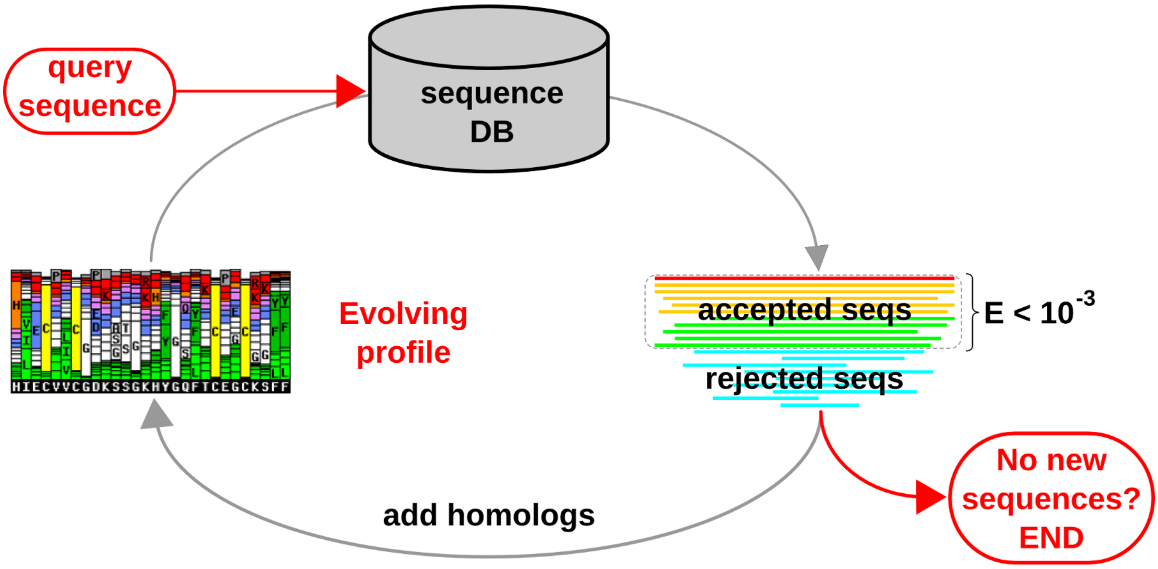


Figure 2.4.: Iterative homology search strategy of PSI-BLAST. In the first iteration, the profile is derived from the mutation probabilities in the BLOSUM substitution matrix. At the beginning of all following iterations, the profile is refined with all hits from the previous iteration that have an E-value below a given threshold. This iteration is performed until no new sequences are found or the maximum number of iterations is reached.

sequence in a multiple alignment and a profile is made from these results. This new profile is then used as input for the next search iteration against the sequence database (see figure 2.5). Iterations continue until no new sequences are found or the maximum number of iterations is reached. The match columns are always defined by the query sequence, so the length of the profile HMM doesn't change in later iterations.

HMMER3 uses internally a new heuristic acceleration algorithm called *MSV* (Multiple ungapped Segment Viterbi) for reducing the search time and so having a runtime only about 3-fold slower than PSI-BLAST (Eddy, 2009). By removing insertion and deletion states and setting the match-match transition probabilities to 1, this algorithm creates a simplified version of its local alignment model which generates multiple ungapped local alignment segments. A fast runtime is reached by using SSE2 instructions which allow to parallelize simple instructions on modern processors. All high-scoring sequences in the first filter step are then passed on to slower, more sensitive filters and finally to the full HMM-sequence comparison of HMMER3. A further increase of speed was obtained by using the SSE2 instructions in the all steps of the HMMER3 sequence processing pipeline including the Forward/Backward algorithm.

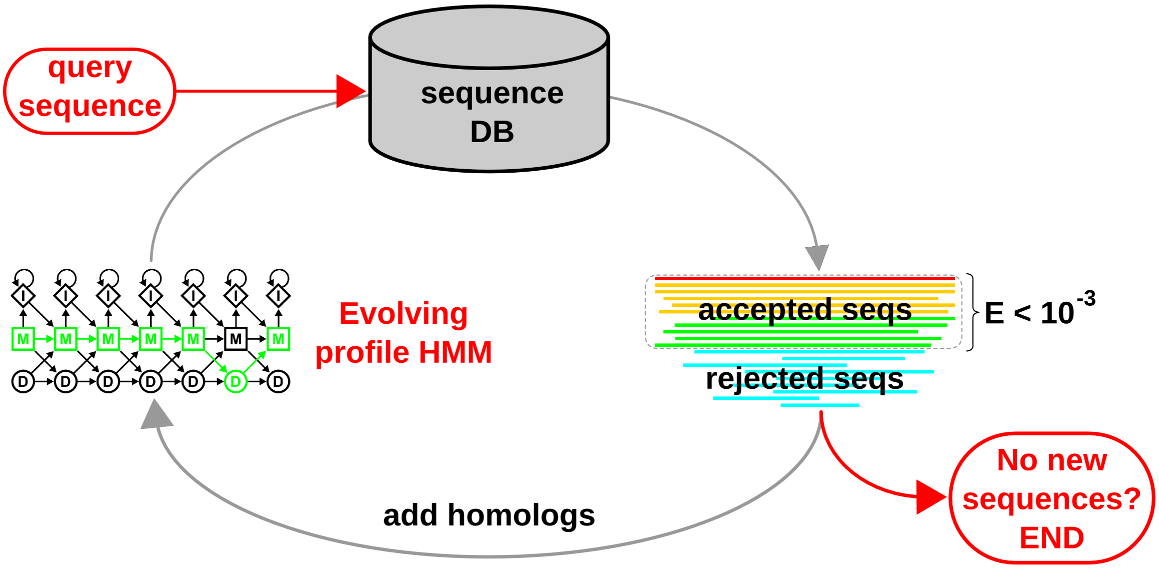


Figure 2.5.: Iterative homology search strategy of HMMER3. In the first iteration, HMMER3 builds a profile HMM from the query sequence using probabilities from the BLOSUM62 matrix and gap-open and gap-extend probabilities. In all further iterations, a profile HMM is build from the alignment of all matches that pass the inclusion threshold in the previous iteration. This iteration is performed until no new sequences are found or the maximum number of iterations is reached.

3. Material and methods

HHblits is the first iterative HMM-HMM comparison method. This method gains its sensitivity by two devices. Firstly, it compares profile HMMs on the query and on the database side, which has been shown to be much more sensitive than sequence-sequence or profile-sequence search methods (Söding and Lupas, 2003). Secondly, it uses an iterative searching strategy similar to PSI-BLAST and HMMER3 (see 2.5). In an iterative search, all significant hits of one iteration are used to expand the query profile HMM with which the next search iteration is started.

A problem for this kind of method is the prohibitively slow runtime of HMM-HMM comparison. Furthermore, an iterative searching method needs a database which covers the whole protein sequence space and for HHblits we have to represent every sequence in the database by profile HMMs. To solve this problem, we use the method kClust (Hauser and Söding, 2011) previously developed in our group to cluster the UniProt database[1] down to 20% sequence identity (UniProt20). This reduces the number of database entries from $\sim$ 14.5 million sequences to $\sim$ 2.6 million clusters (Figure 3.1, UniProt from 29. March 2011). This clustered database is still to big for a complete HMM-HMM search and so we prefilter the UniProt20 database in each iteration with a novel algorithm for profile-profile comparison designed for maximum speed. The algorithm effectively reduces profile-profile alignment to profile-sequence alignment by coding the database profiles by *c*olumn state sequences in which profile columns are represented by an alphabet of 219 discretized column states. This column state alphabet was developed together with Andreas Biegert.

For the calculation of the profile-sequence alignment we use a fast implementation that employs the SIMD (Single Instruction Multiple Data) instruction sets available on modern

---

[1] `http://www.uniprot.org/`

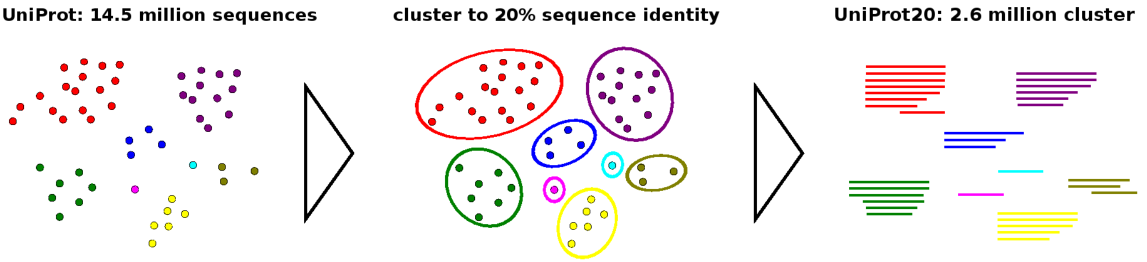

UniProt: 14.5 million sequences
cluster to 20% sequence identity
UniProt20: 2.6 million cluster

Figure 3.1.: Clustering the UniProt database to 20% maximum sequence identity reduces the number of database entries from $\sim$ 14.5 million sequences to $\sim$ 2.6 million clusters.

CPUs. In this way, HHblits performs 16 byte operations in parallel in a single clock cycle. Several further filtering steps reduce the amount of slow, full-blown HMM-HMM comparisons to a fraction of $< 1/1000$.

This chapter gives a detailed overview of the fast clustering by kClust, the column state alphabet and the fast prefiltering based on this alphabet, and several additional filter steps which decrease the runtime of HHblits. In addition, improvements in the core code for HHblits and HHsearch as well as a compact database packing for the HHblits databases are described.

3.1. Workflow of HHblits

The schematic workflow of HHblits is shown in figure 3.2. In the first iteration, a query profile HMM is built from the query sequence using context-specific pseudocounts (Biegert and Söding, 2009). This profile HMM is firstly used for the fast prefiltering step against the column state database of the UniProt20. The time-consuming HMM-HMM comparison is then performed only on the best matches of this prefiltering. All profile HMMs that pass the

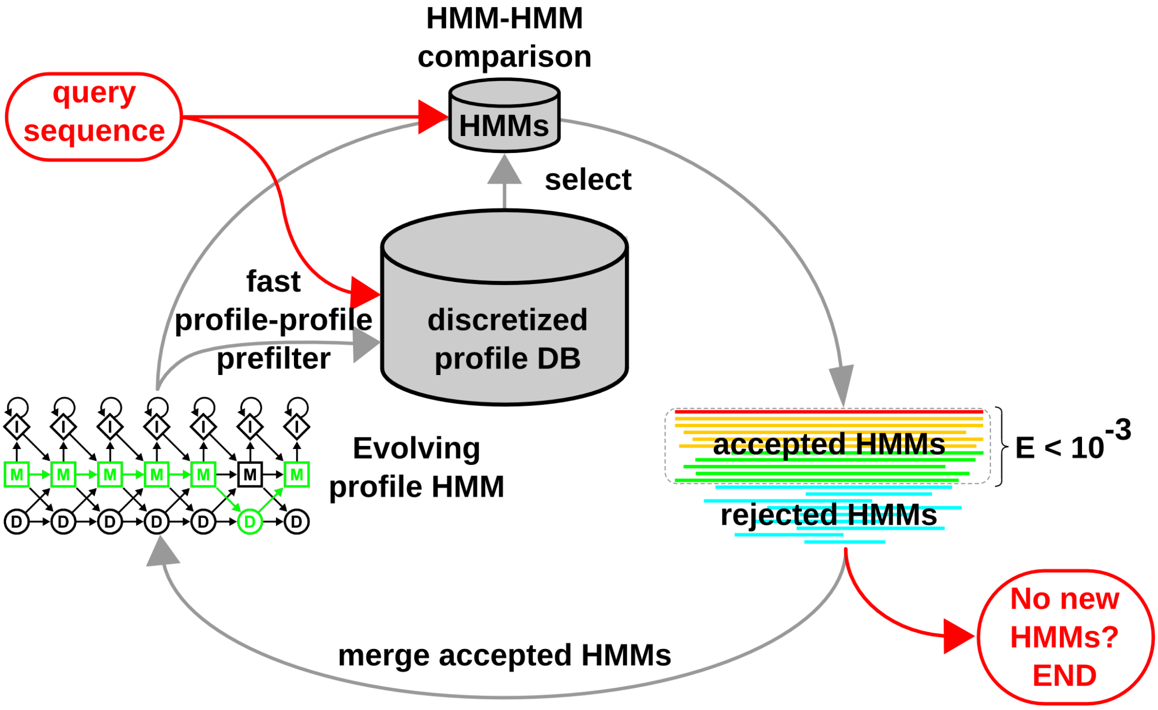


Figure 3.2.: Schematic workflow of HHblits. Each search iteration contains two main steps. First, a fast profile-profile prefiltering with the column state database of the UniProt20 is performed. In the second step, the HMM-HMM comparison is executed only against the best matches of the prefiltering step. All HMMs that have an E-value below a threshold of 10^{-3} are likely to be homologous to the query and are merged to the evolving query profile HMM, which is used as input for the next search iteration. This scheme is repeated until no new HMMs are found or a maximum number of iterations is reached.

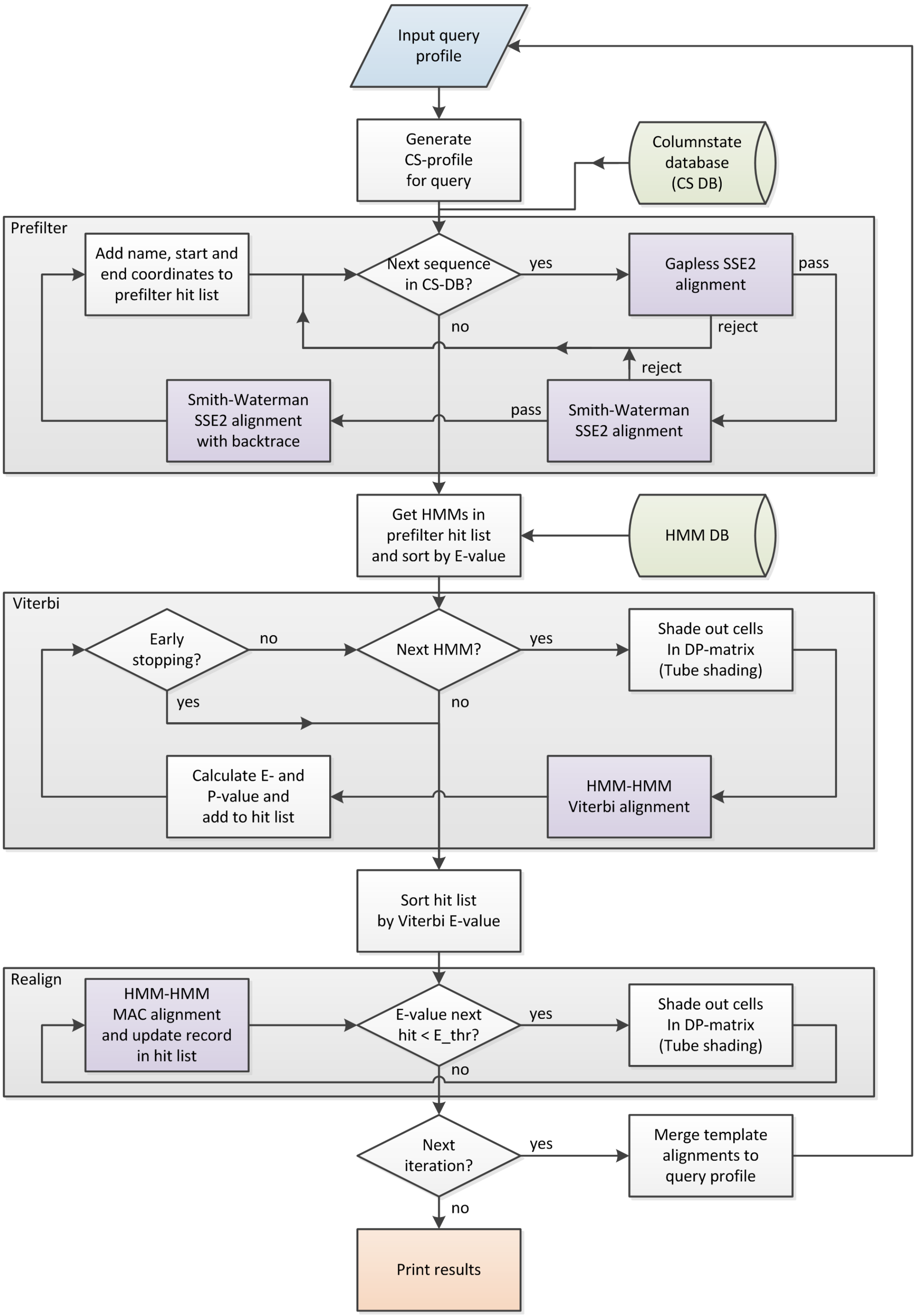

Input query profile
Generate CS-profile for query
Columnstate database (CS DB)
Prefilter
Add name, start and end coordinates to prefilter hit list
Next sequence in CS-DB?
yes
Gapless SSE2 alignment
pass
no
reject
reject
pass
Smith-Waterman SSE2 alignment with backtrace
Smith-Waterman SSE2 alignment
Get HMMs in prefilter hit list and sort by E-value
HMM DB
Viterbi
Early stopping?
no
Next HMM?
yes
Shade out cells In DP-matrix (Tube shading)
yes
no
Calculate E- and P-value and add to hit list
HMM-HMM Viterbi alignment
Sort hit list by Viterbi E-value
Realign
HMM-HMM MAC alignment and update record in hit list
E-value next hit < E_thr?
yes
Shade out cells In DP-matrix (Tube shading)
no
Next iteration?
yes
Merge template alignments to query profile
no
Print results

Figure 3.3.: Detailed workflow of HHblits. The prefilter starts with a column state (CS) database and a CS-profile for the query. The different steps of the prefiltering are described in section 3.3. A list of HMMs that pass the prefilter, sorted by their prefilter E-values, is given to the HMM-HMM comparison part. In the Viterbi algorithm, several additional filters are applied (see section 3.4) and the E-values and P-values of all hits are calculated (see section 3.5). Afterwards, the alignments of the best hits are realigned by the Maximum Accuracy (MAC) algorithm (see section 3.6). If a further iteration will be performed, the template alignments are merged to the query profile and the scheme is repeated. DP: dynamic programming.

inclusion E-value threshold of 10^{-3} in this comparison are merged with the query profile HMM. This new profile is then used as input for the next prefiltering and search iteration. Iterations continue until no new HMMs are found or the maximum number of iterations is reached.

Figure 3.2 gives a more detailed workflow of the prefilter steps and the main parts of the HMM-HMM comparison with the Viterbi algorithm and the Maximum Accuracy (MAC) algorithm in the realign step. The single steps are described in the next sections.

3.2. Generation of HHblits databases with kClust

Essential for an iterative HMM-HMM comparison is a procedure to cluster the sequence databases, because we cannot generate HMMs for over 14 million protein sequences in a reasonable time, not to mention the memory usage. We use kClust, an algorithm for fast clustering of protein sequence databases. It was initially developed in the diploma thesis of Christian Mayer at the Max-Planck institute in Tübingen and further improved by Maria Hauser in our group (Hauser and Söding, in preparation).

In kClust the database sequences are processed sequentially and clustered by their similarity. This algorithm consists of two modules: a prefilter step (k-mer similarity scoring) and a fast sequence alignment step (k-mer dynamic programming, kDP). At first, all database sequences are sorted according to their length. For each sequence the k-mer similarity scoring, which discards sequences clearly not similar enough for one cluster, is performed with all cluster representatives that are found so far. The kDP algorithm, which is a fast heuristic approximating the optimal local alignments obtained by the Smith-Waterman algorithm, is then performed on all representatives that pass the filter step. If the kDP score between the new sequence and a representative is above a specified threshold, the new sequence becomes a member of the cluster of this representative. If no representative has a score above the threshold, a new cluster is generated and the new sequence becomes a representative. Finally, a refinement step improves the clustering to reassigning members to new clusters if they have a greater similarity score.

The k-mer similarity scoring is a fast way to identify sequences that are clearly not similar and where a full alignment between the query and database sequence would be too costly. The idea of this filter is that a pair of homologous sequences with a certain sequence identity should have more similar k-mer pairs, i. e., k-mers with an ungapped alignment score above a threshold, than unrelated sequences. The property that similar sequences have many similar k-mers, is used to filter out unrelated sequences without computing a full alignment.

For the k-mer similarity scoring an index table of all exact k-mers is generated for all database sequences. The index table stores for each k-mer a pointer to a list of database sequences that contain this k-mer. During the prefiltering, a list of similar k-mers is generated for each overlapping k-mer of the query together with the corresponding similarity scores. Each list is then compared to the k-mers of the database sequences by exact matching with the index table. The similarity score of the best match for each matched database sequence is added to the overall prefiltering score for this database sequence. Finally, all database sequences with a prefiltering score above a given threshold are passed to the kDP algorithm. This prefilter step uses by default 6-mers.

The kDP algorithm is a fast heuristic for the Smith-Waterman local alignment algorithm (see 2.4.2) and operates on a so-called *sparse k-mer matrix* (see figure 3.4), where the dots are the k-mer matches between the sequences. The aim of the kDP algorithm is to identify the optimal path (highest scoring) through this sparse k-mer matrix. Various tricks in this algorithm reduce the time complexity and result in a fast runtime. Compared to the prefilter step, the k-mer size is reduced to 4 by default to improve the sensitivity of this algorithm. After identifying the optimal path, the regions between the matched k-mers are explicitly computed with the original Smith-Waterman algorithm.

Maria Hauser now has improved kClust by performing a second iteration, where the previously identified clusters are further merged. For all clusters obtained in the first iteration alignments are built with the global alignment program KALIGN (Lassmann and Sonnhammer, 2005), and sequence profiles and consensus sequences are generated with HHmake. The

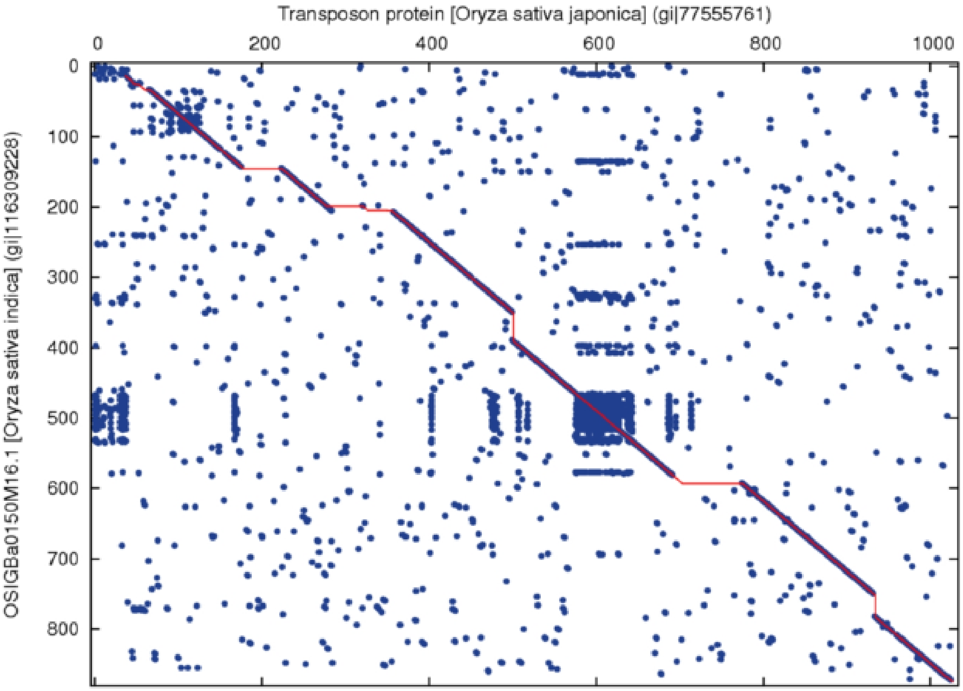


Figure 3.4.: Sparse k-mer matrix between two proteins with a high sequence similarity. The blue dots are the similar 4-mers of the two sequences and the red line indicated the highest scoring path through this matrix. (Figure taken from the diploma thesis of Christian Mayer)

sequence profiles are used as queries in the second iteration. The use of sequence profiles as representatives would result in profile-profile alignments for the cluster comparisons that is too slow for large-scale applications. Hence, consensus sequences are used for the cluster representatives. The workflow of the clustering method is the same as in the first iteration. The second iteration benefits from the additional, family-specific information provided in the sequence profiles and consensus sequences.

For the final HHblits database, an alignment is generated for every cluster by KALIGN and for all alignments with more than 50 sequences an HMM is created. We cannot build HMMs for all alignments, because this would increase the HMM database size from currently $\sim$ 2GB to $\sim$ 150GB. The other HMMs will be generated on the fly in HHblits, which takes about 10^{-4} seconds for alignments with less than 50 sequences. These A3M- and HMM-files are than packed with FFINDEX (see section 3.8) into two database files. Finally, the column state database (see section 3.3.1) needed for the HHblits prefiltering is generated from the A3M-alignments.

3.3. Fast prefiltering of HHblits

The prefiltering in HHblits is the crucial step for achieving a runtime similar to PSI-BLAST. It needs to be as fast as possible, because it has to be performed on more than 2.5 million database entries. But it is also the step which could limit the sensitivity of HHblits. Hence, it has to be sensitive enough to separate possible true homologs from non-homologous alignments, such that only a small amount of putative homologs has to be compared by the time-consuming HMM-HMM comparison.

In the following section, we demonstrate how we integrate profile-profile comparison based on a specific column state alphabet in combination with fast algorithms based on the SSE2 technology in order to achieve a very fast, but sensitive prefilter.

3.3.1. Column state alphabet for sequence profile encoding

HHblits uses a very fast implementation of the Smith-Waterman algorithm in its second prefilter to balance speed and sensitivity to pick up even very distant evolutionary relationships. This algorithm is guaranteed to find an optimal local alignment using affine gap penalties between a query and a database sequence in contrast to BLAST and PSI-BLAST, which use heuristic prefilter themselves. The Smith-Waterman algorithm uses a matrix of substitution scores $S(x_i,y_j)$ that describes the similarity between amino acids x_i in sequence x and y_j in sequence y (see 2.4.2).

In HHblits, we have more evolutionary information on the query and the database side through the sequence profile representation. Thus, we want replace the substitution score

$S(x_i,y_j)$ by the standard profile-profile co-emission score between two profiles p and q

$$S\left(p(i,\cdot),q(j,\cdot)\right) = \log \sum_{a=1}^{20} \frac{p(i,a)q(j,a)}{P(a)} \tag{3.1}$$

$p(i,a)$ is the frequency of amino acid a in column i of the sequence profile p and $P(a)$ is the background probability of a. The problem is that this equation consists of a sum over all 20 amino acids and is therefore too slow to replace the substitution score in the HHblits prefilter.

But we can calculate a profile-to-sequence score $S\left(p(i,\cdot),y_j\right)$ fast. The substitution scores can be seen as a special case of profile-to-sequence scores, where the profile is generated from one of the sequences by using substitution matrix pseudocounts (Henikoff and Henikoff, 1996; Tatusov et al., 1994): $p(i,y_j) = P(y_j|x_i)$. $P(y_j|x_i)$ is the conditional probability of the amino acid y_j given the amino acid x_i in the sequence x. Therefore, the substitution score $S(x_i,y_j)$ used within the Smith-Waterman algorithm can be expressed as standard profile-to-sequence score of profile column i of the query sequence profile p with residue y_j of the database sequence y:

$$S\left(p(i,\cdot),y_j\right) = \log\left(\frac{p(i,y_j)}{P(y_j)}\right) = \log\left(\frac{P(y_j|x_i)}{P(y_j)}\right) = S(x_i,y_j) \tag{3.2}$$

Thus, we can use the profile-to-sequence score without an increase in runtime. But we have to compress all evolutionary information on the database side into one single sequence. For that reason, we used consensus sequences of the database profiles in the first implementation of this prefiltering. But during the reduction of a sequence profile into a consensus sequence a lot of information is lost.

Our solution is to encode each alignment column in the database profiles by an one-letter character from a pre-computed alphabet of column states. This alphabet was developed together with Andreas Biegert based on his context profile library (Biegert and Söding, 2009). We tested alphabets with 62 column states (all states can be encoded by a letter or digit) and with 219 column states (states are encoded by all printable ASCII characters with few exceptions). For the following explanations we use the CS62 alphabet, which can be visualized more easily, but everything works the same for the CS219 alphabet, which we used in the end.

Each column state $k \in \{1,\ldots,62\}$ of the CS62 alphabet is characterized by a specific amino acid profile vector $cs_k(\cdot)$ and a representing character $cs_k \in \{\text{0-9}, \text{A-Z}, \text{a-z}\}$. Figure 3.5 shows the 62 column states with their profile vectors and one-letter codes. This alphabet was generated by clustering a large representative set of training profile columns (see 3.3.2 for details). Every profile y in the HHblits database could now be replaced by a sequence of column states such that $y_j \in \{1,\ldots,62\}$. With this column state database, a new profile-to-column-state score can be defined for a query profile p and a database column state

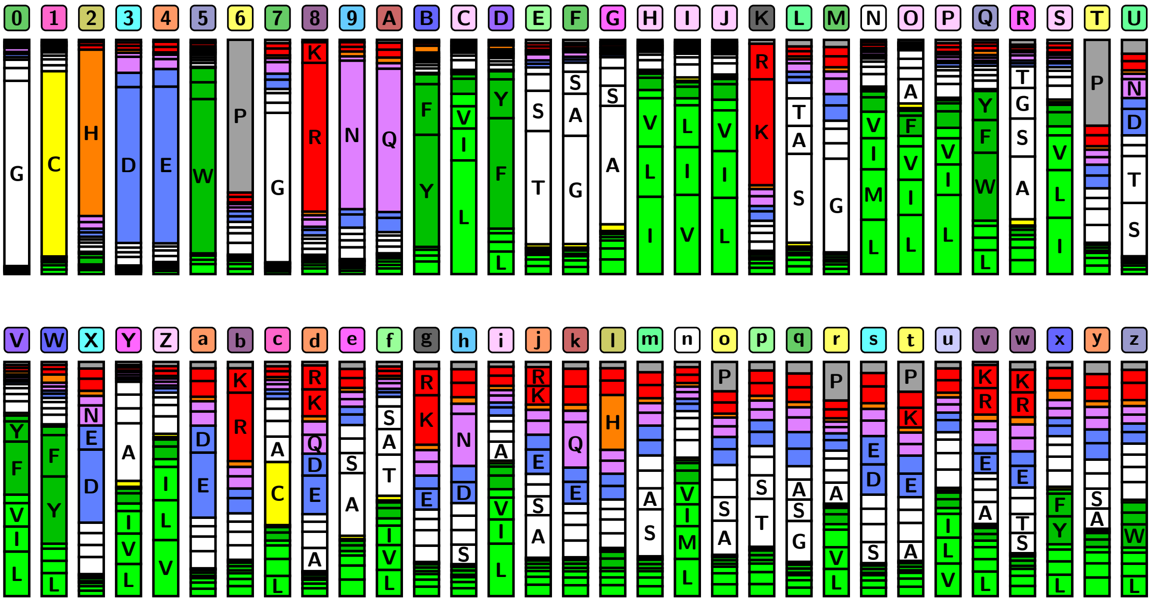


Figure 3.5.: Overview of all 62 column states in the CS62 alphabet (For visualization a 62 state alphabet is shown instead of the alphabet with 219 states, which is currently used in HHblits). Each column state is defined by a one-letter code (colored boxes with rounded corners) and a corresponding amino acid profile vector (histogram). The first column states are more or less pure and encode the amino acids itself, whereas other non-pure column states contain groups of amino acids that readily substitute each other, e.g. hydrophobic or small amino acids. Column states at the very end (x,y,z) represent highly diverged alignment columns with slightly biased background distributions.

sequence y with profile information on both sides:

$$S\left(p(i,\cdot), y_j\right) = \log \sum_{a=1}^{20} \frac{p(i,a)cs_{y_j}(a)}{P(a)} \tag{3.3}$$

This equation still contains a sum over all amino acids, but we can easily pre-calculate all scores $S\left(q(i,\cdot),k\right)$ for $i \in \{1,\ldots,L_q\}, k \in \{1,\ldots,62\}$. Thus, we can perform a look-up of pre-calculated scores $S\left(q(i,\cdot),k\right)$ during dynamic programming and do not have to recalculate the alignment match scores repeatedly. This procedure is completely analogous to the profile-to-sequence comparison with the exception of a larger alphabet size.

3.3.2. Generation of the column state alphabet

The improvement of the new profile-to-column-state score (equation 3.3) over the profile-to-sequence score based on consensus sequences depends to a large extent on the column state alphabet. The clustering procedure for generating this CS62 alphabet is similar to the generation of the context profile library for CS-BLAST (Biegert and Söding, 2009) (see supplement information of this paper for more details). As a training set we use $N = 10$ million profile vectors randomly sampled from NR20 alignments. These alignments are built by clustering the NCBI NR database[2] down to 20% maximum pairwise sequence identity and adding context-specific pseudocounts to the resulting clusters. The 10 million profile vectors are translated to count vectors by multiplying the profile vector $t_n(a)$ by the effective number of sequences Neff_n (see appendix A.2 for details) in the alignment from which the training profile vector was calculated: $c_n(a) = \mathrm{Neff}_n t_n(a)$. The N training vectors are then clustered in order to obtain a set of $K = 62$ column states which together can approximately describe all training vectors. More precisely, we seek to determine column state profiles $cs = (cs_1, \ldots, cs_K)$ and their prior probabilities $\alpha = (\alpha_1, \ldots, \alpha_K)$ that maximize the likelihood $P(c|cs, \alpha)$ that each of the training profile count vectors $c = (c_1, \ldots, c_N)$ was generated by exact one column state profile. An Expectation Maximization (EM) algorithm is used to find parameters (cs^*, α^*) which optimize the likelihood $P(c|cs, \alpha)$. In order to make this problem traceable, hidden variables $z = (z_1, \ldots, z_N)$ were used, where $z_n \in \{1, \ldots, K\}$ indicates which column state profile cs_k has emitted training profile count vectors c_n. The probability distribution over z can now be calculated by:

$$P(z_n = k|c, cs, \alpha) = \left(\prod_{x=1}^{20} cs_k(x)^{c_n(x)} \right)^w \alpha_k \tag{3.4}$$

More details can be found in the supplementary information (equation 6-8) in Biegert and Söding (2009). The main differences to the case of the context profile library are on the one hand the number of clusters ($K = 62$ instead of $K = 4000$) and the length of the profiles (1 column instead of 13), on the other hand a hard clustering is performed where each training profile column is generated by only one column state instead of the soft clustering in the case of the context profile library. The hard clustering is achieved by using a high weight $w = 1000$ (instead of $w = 1.6$), because than for each hidden variable z_n, the term for one column state k dominates all other and each training vector is assigned to exact one column state profile. Because of this hard clustering, the CS62 alphabet is optimized in such a way that the observed counts in an alignment column are described as well as possible by a single column state, rather than through a mixture of column states.

3.3.3. Translation of sequence profiles in column state sequences

For using the fast profile-to-column-state score (equation 3.3) in the prefiltering, the HHblits database (e.g. UniProt20) needs to be translated into a CS62 column state sequence database. This process is illustrated in figure 3.6. In a first step, a sequence profile is generated from a given sequence alignment by adding a small amount of context-specific pseudocounts (histogram below alignment). Afterwards, each profile column is translated into the one column state that best describes the observed counts (colored boxes with rounded corners). More precisely, each profile column i is replaced with the CS62 letter cs_k

---

[2] `http://www.ncbi.nlm.nih.gov/`

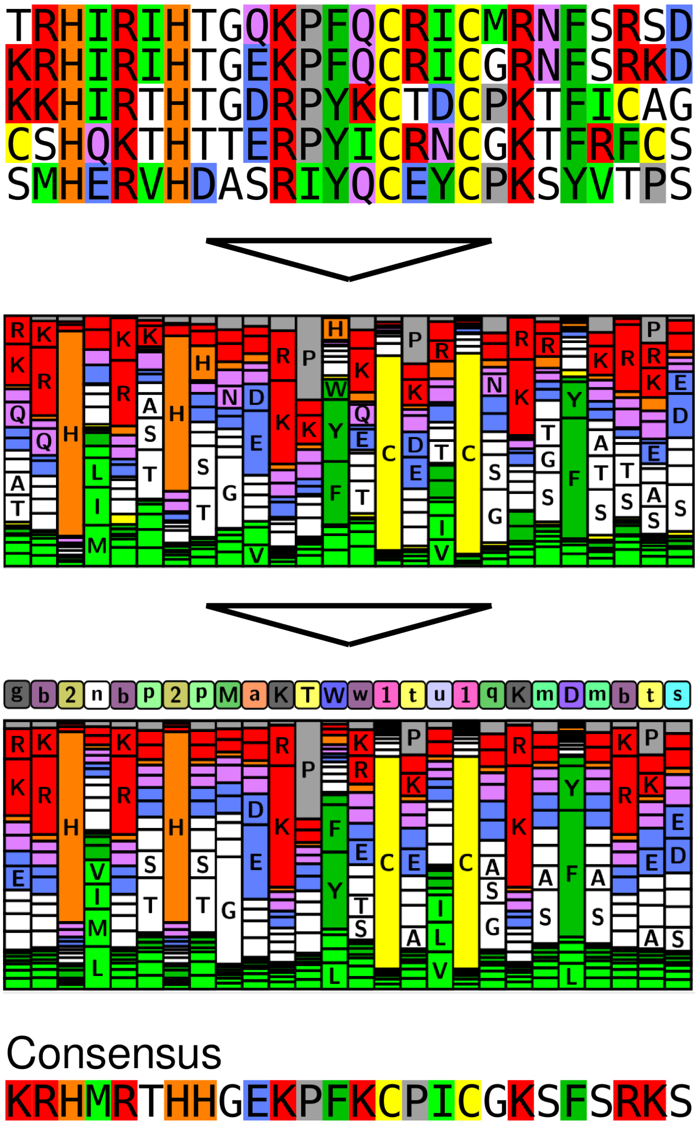

TRHIRIHTGQKPFQCRICMRNFSRSD
KRHIRIHTGEKPFQCRICGRNFSRKD
KKHIRTHTGDRPYKCTDCPKTFICAG
CSHQKTHTTERPYICRNCGKTFRFCS
SMHERVHDASRIYQCEYCPKSYVTPS
gb2nbP2PMaKTWw1tu1qKmDmbts
Consensus
KRHMRTHHGEKPFKCPICGKSFSRKS

Figure 3.6: Translation of an alignment into a sequence of column states. In a first step, a sequence profile is generated from a given sequence alignment by adding a small amount of context-specific pseudocounts (histogram below alignment). Afterwards, each profile column is translated into the one column state that best describes the observed counts (colored boxes with rounded corners). Below each column state letter its characteristic profile column is shown in order to facilitate the comparison between the translated and the original profile column. At the bottom there is the consensus sequence generated from the sequence alignment. Clearly, the column state sequence represents the given sequence alignment with all observed counts very well, regardless of highly conserved or less conserved profile columns. The consensus sequence on the other hand cannot distinguish between highly conserved columns and more diverged columns.

which has the maximum probability to generate the observed amino acid counts $c(i,a)$:

$$k = \operatorname*{argmax}_k \alpha_k \left(\sum_{a=1}^{20} cs_k(a)^{c(i,a)} \right)^w \tag{3.5}$$

α_k is the Bayesian prior probability for column state cs_k, determined in the process of generating the column states alphabet (section 3.3.2). It quantifies the probability that an alignment column is emitted by profile vector $cs_k(\cdot)$ prior to knowing that alignment column. The counts $c(i,a)$ of amino acid a at position i of the input alignment are modeled with a multinomial distribution. This translation procedure is implemented into a small binary CSTRANSLATE, which takes as input an alignment and the CS62 library and outputs the generated CS62 column state sequence in FASTA format.

3.3.4. SSE2

In the last years people have put a lot of effort into the speed optimization of sequence search tools such as PSI-BLAST and HMMER3 (Chaudhary et al., 2006; Sun and Buhler, 2007; Walters et al., 2006), but most of these efforts have had little impact. The only accelerations that seem to work are based on implementing the code on a specialized hardware. Such specialized hardware includes FPGAs (field-programmable gate arrays) (Derrien and Quinton, 2007, 2010; Oliver et al., 2007; Sotiriades and Dollas, 2007; Takagi and Maruyama, 2009), VLSI ASICs (special-purpose chips), large multiprocessor clusters (Chukkapalli et al., 2004; Rekapalli et al., 2009; Wang et al., 2003), GPUs (graphics processor units) (Horn et al., 2005; Lican and Ya, 2010; Ligowski and Rudnicki, 2009; Ling and Benkrid, 2010; Liu et al., 2009; Manavski and Valle, 2008; Vouzis and Sahinidis, 2011; Walters et al., 2009; Yao et al., 2010), special processor supplementary instruction sets such as SIMD (Single Instruction, Multiple Data) (Edgar, 2010; Farrar, 2007; Rognes, 2011; Rognes and Seeberg, 2000; Szalkowski et al., 2008; Wozniak, 1997), and combinations of these hardwares (Lavenier and Nguyen, 2010; Nguyen and Lavenier, 2009).

For prefiltering more than 2.5 million sequence alignments our algorithms must be as fast as possible to achieve a runtime similar to PSI-BLAST, but on the other hand our method should be applicable on a wide range of personal computers without the need of additional hardware like FPGAs. Therefore, we compared the runtimes when using GPUs and SSE2 (Streaming SIMD Extensions 2) instructions, because they are available on most actual personal computers. Based on some initial tests we decided to use SSE2 which gave us better results than GPUs.

Streaming SIMD Extensions 2 was first introduced by Intel in 2001 with the initial version of Pentium 4 CPUs (Central Processing Unit). It is a special processor supplementary instruction set and extends the earlier instruction sets SSE and MMX. In the meantime, Intel has updated this instruction set to SSE3 and SSE4 with newer CPU-families. The newest extension is called AVX (Advanced Vector Extensions) and was released this year with the new Sandy Bridge Processor family of Intel. The supplementary instruction sets work on special 128 bit registers on the CPU and can perform simple instructions on complete registers. In one 128 bit register one can store 16 chars (one char is 8 bit) and because of that the addition of two 128 bit registers performs 16 additions of chars in parallel in one clock cycle. This parallelization leads to an enormous speed-up in our algorithms. SSE2 is supported by all Intel CPUs since the Pentium 4 (2001) and by all AMD CPUs since the Athlon64 (2003).

3.3.5. Fast prefilter algorithms

For obtaining a very fast and sensitive prefiltering we perform a multi-step prefiltering. The different steps are shown in figure 3.7. First, we perform a gapless alignment algorithm

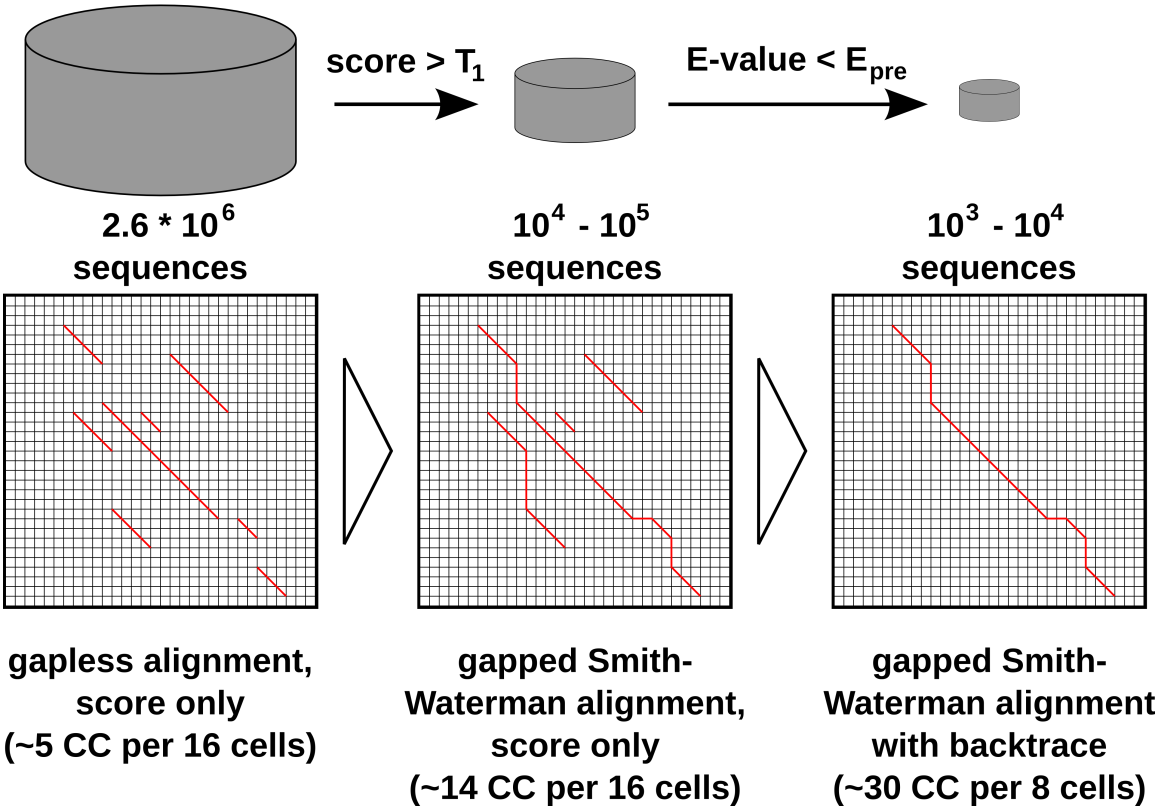

score > T_1
E-value < E_{pre}
2.6 * 10^6 sequences
10^4 - 10^5 sequences
10^3 - 10^4 sequences
gapless alignment, score only (~5 CC per 16 cells)
gapped Smith-Waterman alignment, score only (~14 CC per 16 cells)
gapped Smith-Waterman alignment with backtrace (~30 CC per 8 cells)

Figure 3.7.: Different steps of the HHblits prefiltering. Firstly, a very efficient gapless alignment score is calculated for all column state sequences in the database (2.6 million for UniProt20). Only sequences with a score above a given threshold in this gapless alignment ($\sim$ 10%) are transfered to the next prefilter step. In this step a standard Smith-Waterman score is calculated and only sequences with a prefilter E-value below the threshold E_{pre} are passed to a last prefilter step. Again, the number of sequences is reduced by about 90%. On the final 10^3 - 10^4 sequences a full Smith-Waterman algorithm with backtrace is performed to identify the exact positions of the best matches. Below each alignment the expected number of clock cycles (CC) to compute cells in the dynamic programming matrix are given.

between the query profile and the whole column state database (in case of the UniProt20 about 2.6 million sequences). In each alignment we calculate the maximum match score of an ungapped region in a very efficient way by using SSE2 instructions (details see below). All database column state sequences are passed to the second filter step, if their match score is above a given threshold $T_1 = 2.5 + \log_2(mn)$ bits, where m and n are the lengths of the query profile and database sequence. This gapless alignment algorithm is sensitive enough to filter out more than 90% of non-homologous sequences. In the second filter step, we perform a Smith-Waterman alignment implemented with SSE2 instructions (based on an algorithm by Farrar (2007), details see below). In order to minimize the runtime, we again calculate only the score of the best match in this algorithm. From this bit score S, an approximate E-value is calculated: $E = N_{DB} mn 2^{-S}$, where N_{DB} is the number of sequences/HMMs in the database, and sequences pass if $E < E_{pre} = 1000$. Again, the number of database sequences is reduced by about 90%. On the final 10^3 - 10^4 sequences a full Smith-Waterman algorithm with backtrace (implemented with SSE2) is performed to identify the exact positions of the best matches. The knowledge of the exact positions is important for further filter steps explained in section 3.4.2. These final sequences with the exact positions of the best matches are then passed to the main HMM-HMM comparison of HHblits.

The SSE2 instruction set gives us the opportunity to parallelize simple instructions such as maximization or addition. To maximize the number of operations per clock cycle, the special 128 bit registers are divided into 16 8-bit elements (unsigned chars). This results in 16 operations in parallel in one clock cycle, but it limits the range from 0 to 255. Each unsigned char holds the score in units of 1/4 bits plus an offset of 50, allowing to represent a score range between -12.5 and +51.5 bits. The algorithms are programmed such that the scores will saturate at 255 upon overflow. Since any score larger than 51 bits will pass the filter anyway (51 bits corresponds approximately to an E-value of 0.1 for average protein lengths), this range is sufficient for prefiltering. But in the final step we are interested in the exact start and end positions of the best alignments and thus we need to prevent the score from saturating. Therefore, we use for the last filter step 8 16-bit elements (unsigned short). With these elements, we can perform only 8 operations in parallel, but we obtain a score range of -12.5 to 16371.5 bits which avoids saturation of the score for most proteins. Only when aligning proteins with a length above $\sim$ 8,000 the score could saturate. But for these few cases the identified positions are good enough, because usually the correct alignment should be the longest and in this case we could identify the correct alignment.

For the efficient use of SSE2 commands, it is important to have all needed data 16-byte aligned in the memory, because then the values are read with a single aligned load instruction. We achieve this requirement by using the `memalign(16,...)` command to allocate 16-byte aligned memory when loading all column state sequences (translated into unsigned chars) into the memory. Furthermore, we pre-calculate all profile-to-column-state scores $S(q_i,k)$ (see equation 3.3) for the query profile q and all $k \in \{1,\ldots,62\}$ and store them also 16-byte aligned.

3.3.6. Gapless local alignment with SSE2

In the first prefilter step, the maximum match score of an ungapped region between the query profile q with length m and a database column state sequence y with length n is searched. This can be done by filling an $m \times n$ score-matrix S with a dynamic programming algorithm similar to the Smith-Waterman algorithm described in 2.4.2. The difference is that we update each cell in this matrix by only looking at the above-left cell, because we do not allow gaps. Therefore, the score in each cell $S(i,j)$ will be the maximum of 0 (a new alignment that starts at this position) and $S(i-1,j-1) + s(q_i,y_j)$. $s(q_i,y_j)$ is the pre-calculated profile-to-column-state score between column i in the query profile q and the column state letter at position j in the database sequence y. For efficiency, we do not store the full S matrix but use only two vectors, because for the calculation of a new column only the scores of the previous column are needed. Based on this the algorithm can be written with only 5 SSE2 commands in the inner loop (see algorithm 1), resulting in a performance of 16 cells in only 5 clock cycles. Here, the command *_mm_subs_epu8(S, offset)* for example means, that all 16 8bit segments in register *offset* are subtracted from the corresponding 16 segments in register *S*. In addition, this *subs* command saturates, that means that all results < 0 are set to 0. Finally, the maximum match score of an ungapped local alignment between the query profile q and the column state sequence y is the maximum value within the $Smax$ register.

Algorithm 1: Inner loop of the gapless alignment algorithm with SSE2 commands

```
for $j \leftarrow 0$ to lenT do
    for $i \leftarrow 0$ to lenQ do
        S = _mm_adds_epu8(S, *(qji++));        // $S(i,j) = S(i-1,j-1)$
                                               //     $+(s(q_i,y_j) + offset)$
        S = _mm_subs_epu8(S, offset);          // $S(i,j) = \max(0, S(i,j) - offset)$
        _mm_store_si128(s_curr_it++, S);       // store register S
        Smax = _mm_max_epu8(Smax, S);          // $Smax = max(Smax, S(i,j))$
        S = _mm_load_si128(s_prev_it++);       // load next register S
```

3.3.7. Smith-Waterman with SSE2

In the second prefilter step we want to achieve a higher selectivity at the same sensitivity than in the first gapless alignment algorithm. Because of that we perform a Smith-Waterman alignment between the query profile and the database column state sequence. We use an adjusted version of the Farrar implementation which is freely availably[3] and is faster than other SIMD implementations. In the following we give a brief description of this algorithm and the changes and improvements we have made (for more details see Farrar (2007)).

The basic version of the Smith-Waterman algorithm is described in 2.4.2. The Farrar implementation uses the Gotoh improvements for affine gap penalties (Gotoh, 1982) with three instead of one matrices. There are two matrices E and F for alignment scores ending with a gap in the query or database sequence, respectively:

$$E(i,j) = \max \begin{cases} E(i,j-1) - G_{ext}, \\ H(i,j-1) - G_{open} \end{cases} \tag{3.6}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - G_{ext}, \\ H(i-1,j) - G_{open} \end{cases} \tag{3.7}$$

---

[3] `http://sites.google.com/site/farrarmichael/smith-waterman`

G_{open} and G_{ext} are the gap-open and gap-extension penalties. The alignment scores are calculated in $H(i,j)$:

$$H(i,j) = \max \begin{cases} 0, \\ E(i,j), \\ F(i,j), \\ H(i-1,j-1) + s(q_i,y_j) \end{cases} \tag{3.8}$$

$s(q_i,y_j)$ is the pre-calculated profile-to-column-score between the query profile q and the database column state sequence y.

For an efficient memory usage, the Farrar implementation uses a special memory layout for the query profile q with length m. This layout is a striped access parallel to the query profile (see figure 3.8). The query is divided into t equal length segments $S = \{S_1, \ldots, S_t\}$, where the length of each segment p is equal to the number of elements in one SIMD register ($p = 16$ in the case of unsigned chars). The number t of segments is defined as $t = (m+p-1)/p$ and the query segments itself are defined as $S_n = q_{0*t+n}, q_{1*t+n}, \ldots, q_{(p-1)*t+n}$ where $1 \leq n \leq t$. If the query is not long enough to fill all segments, the segments are padded with null entries. For example, when we have a query q with length $m = 21$ and $p = 4$ elements in one register, then is $t = (21 + 4 - 1)/4 = 24/4 = 6$ and the segments have the following layout:

$$\begin{aligned} S_1 &= q_1, q_7, q_{13}, q_{19} \\ S_2 &= q_2, q_8, q_{14}, q_{20} \\ S_3 &= q_3, q_9, q_{15}, q_{21} \\ S_4 &= q_4, q_{10}, q_{16}, 0 \\ S_5 &= q_5, q_{11}, q_{17}, 0 \\ S_6 &= q_6, q_{12}, q_{18}, 0 \end{aligned}$$

When calculating the match scores $H(i,j)$ in segment S_n in one column of the score matrix, all needed values $H(i-1,j-1)$ are located in segment S_{n-1} of the previous column because of this striped scheme (red arrows in figure 3.8). This reduces the number of needed time-consuming shift-operations.

The pseudocode of the striped SW-alignment algorithm is given in algorithm 2. The calculation of $H(i,j)$ is dependent on the previous value on the major diagonal $H(i-1,j-1)$. Therefore, two buffers (*HStore* and *HLoad*) are generated to store the H values. One buffer is used to read the previous H values and the other stores the current ones. At the beginning of a new column, these buffers are swapped and the data can be reused. The algorithm starts with the calculation of the number of segments into which the data is divided. In the main loop of the algorithm over all database sequence positions $j \in \{0, \ldots, len_T - 1\}$

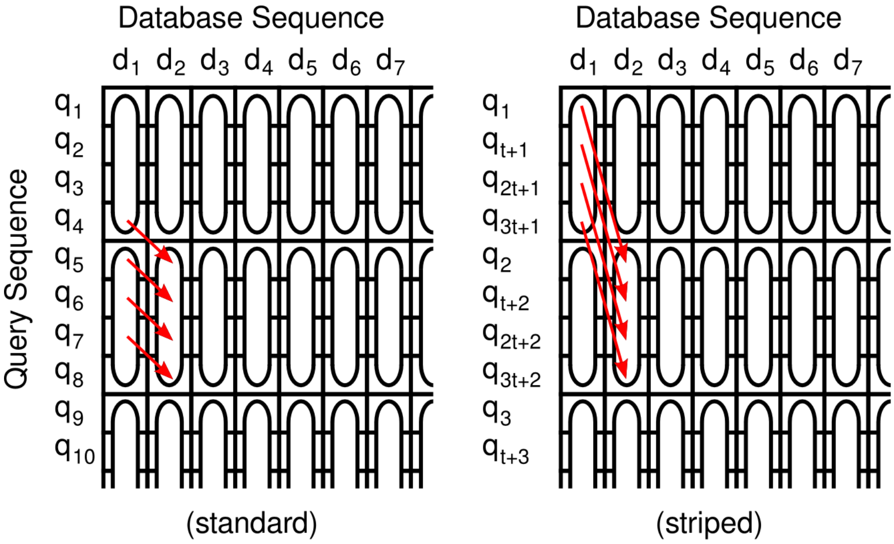

Database Sequence
d_1 d_2 d_3 d_4 d_5 d_6 d_7
Query Sequence
q_1 q_2 q_3 q_4 q_5 q_6 q_7 q_8 q_9 q_{10}
(standard)
Database Sequence
d_1 d_2 d_3 d_4 d_5 d_6 d_7
q_1 q_{t+1} q_{2t+1} q_{3t+1} q_2 q_{t+2} q_{2t+2} q_{3t+2} q_3 q_{t+3}
(striped)

Figure 3.8.: Memory layout for the query profile used by a standard and by the striped implementation. The segments in both layouts run parallel to the query, but the striped implementation accesses the query in a striped pattern. Because of this striped implementation, the cells needed for the calculation of the match scores $S(i,j) = S(i-1,j-1) + s(q_i,y_j)$ (shown as red arrows in this figure) in one segment are completely located in another segment of the previous row. In the standard implementation, the cells are located in different segments and so time-consuming shift operations have to be done.

the last H segment of the previous column is loaded and the values in this segment are shifted to the left. This is important because the values in the first segment of each new database column depends on the last segment of the previous column shifted to the left (see figure 3.9A). Then the two H buffers are swapped. The inner loop iterates over all segments in one database column. The new value $H(i,j)$ is calculated by adding the pre-calculated profile-to-column-state score $s[i][j]$ to $H(i-1,j-1)$, the highest score encountered so far is saved, and the H values are adjusted with greater E or F values, if there are any. The computation of $E(i,j)$ is the maximum of $H(i,j-1) - G_{open}$ and $E(i,j-1) - G_{ext}$, where $E(i,j-1)$ are the values from the cells left of the current cells. In a similar way, $F(i,j)$ is computed with the maximum of $H(i-1,j) - G_{open}$ and $F(i-1,j) - G_{ext}$.

One problem occurs in the first segment of each column by the calculation of the new F values. These values depends on the cells above in the matrix and due to the striped layout the values of the cells in the first segment of each column depends on the values of the last segment in the same column, which are unknown yet (see figure 3.9B). Therefore, in cases where F influence the value of H, a special *F loop* is performed after the inner loop to correct these errors and to calculate the correct value of H. In the most cases, F remains zero and this additional correction step is not necessary.

Algorithm 2: Pseudocode of the SW-alignment algorithm with SSE2 commands

```
// calculate number of segments for query profiles
// (Here we have 16 elements in SIMD register)
iter = (lenQ + 15) / 16;
// loop over database sequence
for $j \leftarrow 0$ to lenT do
    // Load last vH segment and shift for use in next iteration
    vH=pvHStore[iter-1]<< 1;
    // Swap the two H buffers
    swap (pvHStore, pvHLoad);
    // loop over query segments
    for $i \leftarrow 0$ to iter do
        // add score s(i,j) to vH
        vH = vH + s[i][j];
        // Update highest score encountered this far
        vMaxScore = max(vMaxScore, vH);
        // Adjust vH with any greater vE or vF values
        vH = max(vH, vE);
        vH = max(vH, vF);
        // save vH values
        pvHStore[j] = vH;
        // Calculate new vE and vF based on gap penalties
        vH = vH - vGapOpen;
        vE = vE - vGapExtend;
        vE = max(vE, vH);
        vF = vF - vGapExtend;
        vF = max(vF, vH);
        // load the next h value
        vH = pvHLoad[j];
    // F loop to correct vH values based on possible vF influence
```

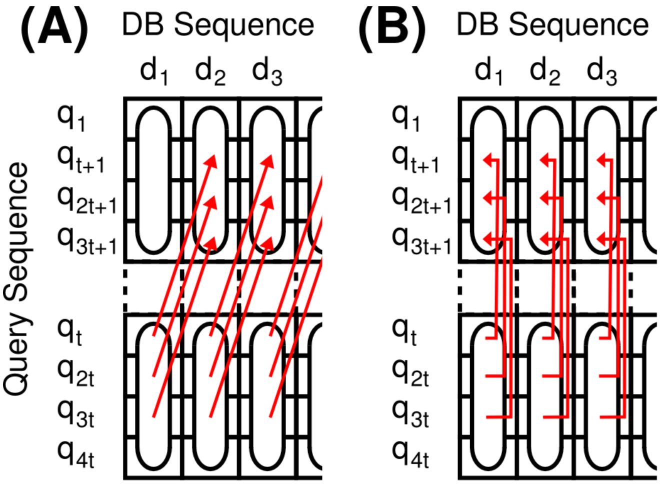


Figure 3.9: (A) Dependencies between the last H segment and the first H segment of the next column. By shifting the values in the last H segment they are aligned with the next segment over. (B) Dependencies between the last F segment and the first. The values in the last F segment are shifted to the left so the values are aligned with the next segment over. (Figures are based on figures from Farrar (2007))

3.3.8. Smith-Waterman with backtrace

In the final step of the prefiltering, we again perform a Smith-Waterman algorithm based on the Farrar implementation, but this time we also use a backtrace algorithm to identify the exact start and end positions of the best alignments. These exact positions are needed for an additional filter step in the HMM-HMM comparison where only the region around the here identified alignments is activated in the dynamic programming matrix (Tube shading, see section 3.4.2). In this Smith-Waterman algorithm we cannot allow for score saturation and so we split the special 128 bit register for the SSE2 instructions in 8 16bit segments (unsigned short). This reduces the number of parallel instructions from 16 to 8, but it increases the score range to -12.5 to 16371.5 bits thus avoiding score saturation.

To perform the backtrace calculation, we need to store two $n \times m$ matrices (n is the query length, m is the DB-sequence length). The *Hmatrix* stores the alignment scores of an alignment that ends in this column ($Hmatrix[i][j] = H(i,j)$) and the *Btmatrix* saves in a special encoding the backtrace of the calculated cells. The *Btmatrix* stores which term in the maximum formulas for $H(i,j)$, $E(i,j)$ and $F(i,j)$ (equations 3.8, 3.6, 3.7, respectively) has lead to the new value, or, in other words, from which cell in the dynamic programming matrix the new value was derived. In this special encoding, we use two half-bytes to encode the backtrace of H and one half-byte to encode the backtrace of E and F, respectively. In the case of H, the encoding 00 indicates that $H(i,j)$ was generated from the calculation $H(i-1,j-1)+s(i,j)$, the encoding 01 means that $H(i,j)$ was generated from $E(i,j)$ and the encoding 10 indicates a generation from $F(i,j)$. For the E-matrix, an encoding of 1 means that $E(i,j) = H(i,j-1) - G_{open}$ and an encoding of 0 stands for $E(i,j) = E(i,j-1) - G_{ext}$. The encoding for F is analogous to that of E.

Algorithm 3: Pseudocode for filling the backtrace-matrix *Btmatrix* in the inner loop of the Smith-Waterman algorithm by using SSE2 instructions. Shown are the important steps for encoding the backtrace of the H-matrix, the encoding for the E- and F-matrix has similar steps

```
// add score $H(i,j) = H(i-1,j-1) + s(i,j)$
// build maximum of H, E and F values:
// $H = \max\{H,E,F\}$
// Set backtrace register for H-matrix
  Bttmp = _mm_cmpeq_epi16(vH, vE);            (1)
  Bt = _mm_and_si128(Bttmp, 0x0001);          (2)
  Bttmp = _mm_cmpeq_epi16(vH, vF);            (3)
  Bttmp = _mm_and_si128(Bttmp, 0x0010);       (4)
  Bt = _mm_or_si128(Bt,Bttmp);                (5)
// Set backtrace register for E- and F-matrix
⋮
// Store Bt value
```

Now these new matrices *Hmatrix* and *Btmatrix* need to be integrated into the Smith-Waterman implementation with the SSE2 instructions. The *Hmatrix* simply stores the H-values which can be performed by a single store-instruction. The pseudocode for filling the *Btmatrix* with the backtrace-encoding of the H-matrix is shown in algorithm 3. In the inner loop the new H register is set by maximizing over H, E and F. Then we compare the segments in the H and E register (line (1) in algorithm 3). This command sets all bits in one segment of the register *Bttmp* to 1, if the two corresponding segments in H and E are equal, otherwise to 0. So all bits equal 1 in one segment of *Bttmp* indicates that the value of the H-value of this segment comes from the corresponding E-value. The *_mm_and_si128* command (line (2)) performs a bitwise “AND” between all bits in the two registers and sets all bits to 0 except the last half-byte of each segment, which describes this encoding. The lines (3)-(5) perform the same for the H and F registers and store the encoding in the second to last half-byte. In a similar way the encoding for the E- and F-matrix is generated.

After the matrices are filled by this algorithm, the maximum score in the *Hmatrix* is searched and from this point a backtrace is started. The backtrace ends when the algorithm reaches a cell with a score of 0 and the start and end positions of this alignment are saved. To eliminate this alignment from further searches we inactive the cells in the *Hmatrix* around this alignment by looping over all cells contained in the alignment and setting the cells in the same row and column within a distance of 150 to zero. This distance is approximately 2/3 of the distance parameter for the tube shading (see section 3.4.2) to ensure that no short domains will be missed in the HMM-HMM comparison by inactivating to much cells. Then, a suboptimal alignment is searched by identifying the next maximum score. If the score is also above the given threshold, a backtrace for this suboptimal alignment is performed and the new positions are stored. This is repeated until no further suboptimal alignment with a score above the threshold is found or until a maximum of 10 suboptimal alignments is extracted. We use this approach instead of the Waterman-Eggert algorithm (Waterman and Eggert, 1987) which computes suboptimal and non-overlapping alignments, because in the Waterman-Eggert algorithm the complete dynamic programming matrix have to be recalculated after each suboptimal alignment and this is too slow for our prefiltering.

3.4. Additional filter steps

The main part in decreasing the runtime of HHblits and so the possibility to perform an iterative HMM-HMM comparison was achieved by the fast prefilter explained in 3.3. To further reduce the runtime of HHblits and to reach runtimes similar to PSI-BLAST, additional filter steps are integrated in the HMM-HMM comparison step of our method. In this section we give a detailed overview of these additional filters: a filter for skipping further HMM-HMM comparisons (“early stopping“) if there are no good matches in the 200 previously aligned HMMs, a filter for reducing the search space in the dynamic programming matrix by only searching in a tube around the prefilter matches and a filter which prevents realigning matches that had already been identified in a previous iteration. All these additional filters can be separately disabled by command-line parameters, but this is not recommended as it results in a longer runtime at an only very slightly increased sensitivity.

3.4.1. Early stopping

The *early stopping* filter further reduces the number of HMMs that will be compared by the time-consuming HMM-HMM comparison. This filter is based on the assumption that the prefilter is good in dividing the homologous and non-homologous proteins. The HMM-HMM comparison runs over the list of prefiltered HMMs sorted by their prefilter scores. Therefore, most of the homologous matches should be found at the beginning of this comparison. If we identify only non-homologous matches over a long range, the probability to find a homologous match is very low and the runtime costs for searching the remaining HMMs do not further justify a possible gain in sensitivity for finding a few more homologous matches. Furthermore, homologs that are missed due to the early stopping might still be found in the following search iterations.

The scheme of this early stopping filter is shown in figure 3.10. A coarse estimate for the probability for a match to be a true homolog is $1/(1+E)$ for a Viterby E-value E. Before starting the HMM-HMM comparison between a new database HMM and the query HMM, we average $1/(1+E)$ over the last 200 processed Viterbi alignments:

$$\frac{1}{200} \sum_{k \in \{\text{last } 200 \text{ matches}\}} \frac{1}{1+E(k)} \tag{3.9}$$

If this average drops below a given threshold (default is 0.01), the HMM-HMM comparison

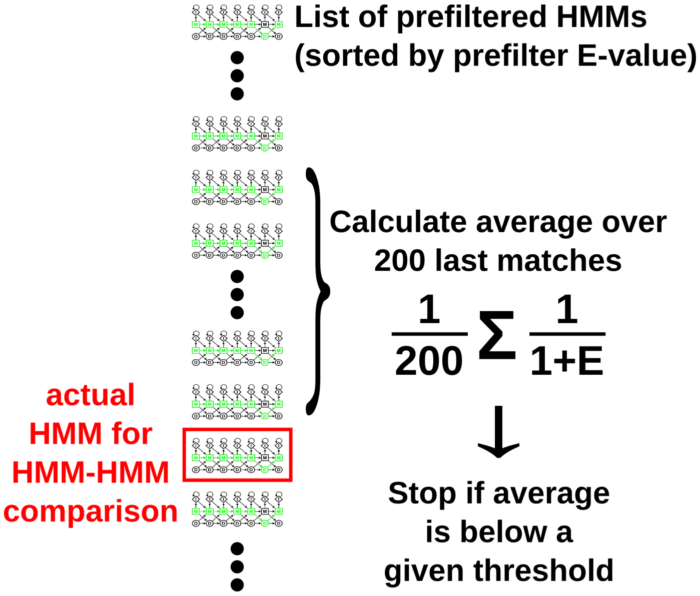


Figure 3.10: Scheme of the *early stopping* filter step. The HMM-HMM comparison of HHblits runs over a list of all HMMs that have passed the prefilter, sorted by their prefilter scores. Before starting the HMM-HMM comparison of the query and the current HMM (red box), an average over the E-values of the last 200 matches $\frac{1}{200}\sum_{n=1}^{200}\frac{1}{1+E(n)}$ is calculated. If this average drops below a given threshold, the HMM-HMM comparison stops and discards all further HMMs. For the first 200 HMMs the sum is filled up with artificial E-values of 0.

stops and all further HMMs in the prefilter list will be discarded. For an efficient calculation, the E-values of the last 200 matches are stored in an array and after each HMM-HMM comparison the oldest entry in this array will be replaced with the new E-value and the sum is updated. At the beginning, this array is initialized with artificial E-values of 0 to guarantee that at least 200 HMMs will be compared. Note that, when performing HHblits with more than one thread, the results between several runs could be slightly different, because the order in which alignments are processed by different CPU-cores can change and hence the threshold may be reached at another position in the list.

3.4.2. Tube shading

The *tube shading* filter reduces the search space in the dynamic programming matrix of the HMM-HMM comparison. In this main comparison, a Viterbi alignment is calculated for each database HMM (see section 2.4). For the Viterbi algorithm, a dynamic programming $m \times n$ matrix (m is the query length, n is the length of the database HMM) must be filled by running over all cells. But we already have the information from the last prefilter step in which regions of this matrix the best matches are likely to be located according to the profile-profile Smith-Waterman algorithm. In most cases, the final alignment is located in the same region. Therefore, we constrain the Viterbi search to only the regions around the prefilter matches.

The scheme of this tube shading is given in figure 3.11. At first, all cells in the dynamic

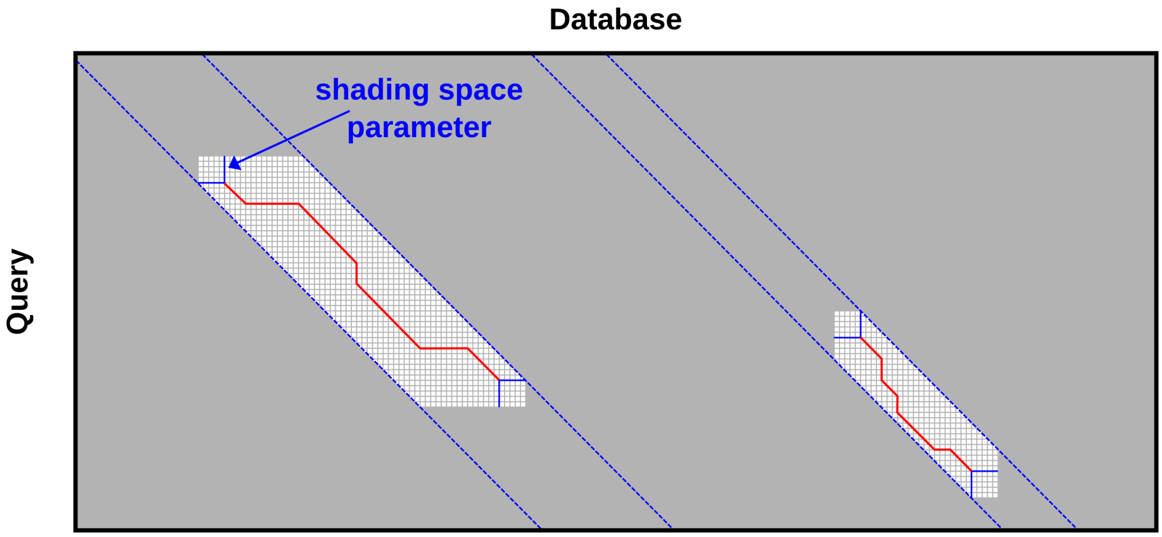


Figure 3.11.: Scheme of the *tube* filter step. This filter reduces the search space in the dynamic programming matrix of the Viterbi algorithm when comparing the query and database HMM. This is done by crossing out all cells in the matrix and activate only the cells in a tube around the alignments identified in the last prefiltering step. There can be more than one alignment if some of the suboptimal alignments also have a score above the prefilter threshold. The size of the tube can be specified by a *shading space* parameter. The Viterbi algorithm now runs only on the activated cells.

programming matrix are crossed out. Then, we get the start- and end-positions of all alignments that had a score above the threshold in the prefiltering. For each alignment we calculate the diagonals which limit the alignment at the bottom and at the top and add an additional shading space to these diagonals (default is 200, see blue lines in figure 3.11). These diagonals and the shading space left and right of the alignment define the region in which the cells of the matrix are activated. For average proteins with a length between 200 and 400 amino acids nearly the full matrix is activated. But especially for long proteins, which result in long runtimes, the search space of this time-consuming comparison can be reduced dramatically. A similar tube shading is used in the realign step, where all good matches are realigned with the more accurate MAC (Maximum ACcurancy) algorithm. Here, the activated cells in the dynamic programming matrix are defined by the previously calculated Viterbi alignments.

3.4.3. Avoid aligning the same templates in multiple iterations

The last additional filter prevents HMMs that have been found in one iteration of HHblits with an E-value below the inclusion threshold, to be aligned again in later iterations. Therefore, the names of all identified matches in one iteration are stored in a list of previous hits. If the same HMM passes the prefilter in a later iteration, it won't be aligned again by the HMM-HMM comparison. This reduces the runtime and it prevents the alignment from deteriorating it in the cases of a too diverse query profile, i. e., if the database match is closely related to the query sequence and the query profile contains the information of evolutionary more distantly related sequences. But on the other hand the scores and therefore the E-values of such matches that were found in previous iterations could improve by a better query profile. For that reason, we re-score in the last iteration all matches that were identified in the previous iterations, but we keep the first alignment.

3.5. P-value calculation with neural networks

The interpretation of the results of an HMM-HMM comparison depends strongly on the statistics of the similarity scores. The scores are calculated during the Viterbi algorithm, but for a meaningful interpretation of the data, measures of the confidence for homology are needed. Most methods report E-values or P-values for all of their matches. The P-value of a match with score S is defined as the probability to obtain a chance hit with score $\geq S$ in a pairwise comparison. Therefore, the score distribution for non-homologous sequences is needed. In the case of HHsearch and HHblits the score distribution of non-homologous sequences follows an EVD (extreme value distribution) as shown in figure 3.12.

If such a score distribution is given, the P-value can be estimated by calculating the area under the curve to the right of S. From this P-value P, an E-value E can be computed. The E-value is defined as the expected number of chance hits with a score $\geq S$ in a search

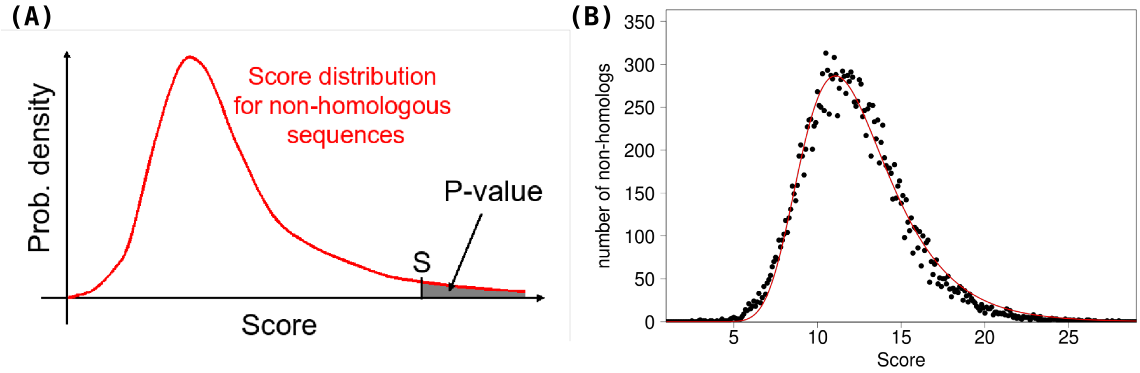


Figure 3.12.: **(A)** The score distribution of non-homologous sequences follows an EVD (extreme value distribution). The P-value of a score S is defined as the area under the curve to the right of this score. **(B)** Example of the score distribution of non-homologous sequences of an HHblits search. The red line is the fitted EVD probability density function.

of a database with N_{DB} sequences: $E = N_{DB}P$. In HHblits we have the problem that there are two search steps (the prefiltering and the Viterbi alignment) which are partially correlated and influence the E-value calculation. We can easily define the E-value for the two extreme cases: (1) If the prefiltering and Viterbi scores are perfectly correlated the E-value can be calculated by $E = N_{DB}P$. (2) If the scores are completely uncorrelated the E-value can be calculated by $E = N_{pre}P$, where N_{pre} is the number of HMMs that pass the prefilter. We handle the partially correlated case by introducing a empirical correlation factor $(E_{pre}/N_{DB})^{\alpha}$ in the E-value formula:

$$E = N_{DB}P \times \left(\frac{E_{pre}}{N_{DB}}\right)^{\alpha} \qquad (3.10)$$

E_{pre} is the E-value-threshold of the prefilter. α is a measure for the degree of correlation ($\alpha = 0$: perfect correlation, $\alpha = 1$: no correlation) and is defined as $\alpha = 0.4 + 0.02 \times (\mathrm{Neff}_T - 1) \times (1 - 0.1 \times (\mathrm{Neff}_Q - 1))$. Here, Neff_Q and Neff_T are the numbers of effective sequences (definition see appendix A.2) in the query and template HMMs, respectively. The three coefficients were optimized to yield accurate E-values (see chapter 4.1.5).

The E-value is the most widely used significance measure. Also, some thresholds in HHblits are given as E-values, e. g., the inclusion threshold with a value of 10^{-3} means that by chance one non-homologous protein passes this threshold in 1000 HHblits runs. In HHsearch and HHblits a *probability* for a homologous relationship is given to the user as an additional significance measure. It includes the secondary structure score and is based on the score distribution for non-homologs and homologs in an all-against-all comparison of the SCOP database:

$$P(\mathrm{pos}|S) = \frac{P(S|\mathrm{pos})P(\mathrm{pos})}{P(S|\mathrm{pos})P(\mathrm{pos}) + P(S|\mathrm{neg})P(\mathrm{neg})} \qquad (3.11)$$

In the previous version of HHsearch, the parameters of the EVD were estimated for each query in an additional calculation step. Therefore, an HHsearch run was performed against a calibration database which consists of one HMM for every fold in SCOP, hence all HMMs in this calibration database should be unrelated. Then, the score distribution of all non-homologous proteins was calculated by excluding the three best-scoring folds. This was done to be sure that only non-homologous matches contributed to the score distribution. The λ and μ parameters for this EVD were determined by Maximum Likelihood and stored in the query HMM. These parameters was then used in the actual search for the P-value calculation. For HHblits we must find another way to determine the parameters, because an additional calibration search in every search iteration is too slow.

We now determine the parameters λ and μ by a machine learning approach and train two neural networks, one for the calculation of λ and one for the calculation of μ (Sadreyev and Grishin, 2008). A neural network can model complex relationships between the inputs and the output. It is an interconnected network of nodes, usually arranged in three layers: the input nodes, the hidden nodes and the output nodes. Each node has an activation function, which generates the output of the node from the weighted input. In a learning phase, the weights w of all edges in the network are changed to model the given training data (input and corresponding output) with a minimal error.

In our case, we use for both parameters the same neural network topology (see figure 3.13): 4 input nodes $k \in \{1,2,3,4\}$, that are fully connected with 4 hidden nodes $l \in \{5,6,7,8\}$ and one output node $o = 9$. As inputs we take the length of the query and database HMM and the diversities of these HMMs. All inputs are normalized to a value between 0 and 1 which results in a slightly better performance of the neural network. In the hidden layer we use a logistic function to generate the output H_l of the hidden node l:

$$H_l = \frac{1}{1 + \exp\left(-\sum_{k=1}^{4} w_{kl} I_k\right)} \tag{3.12}$$

I_k is the input value of the input node k and w_{kl} is the weight of the edge connecting

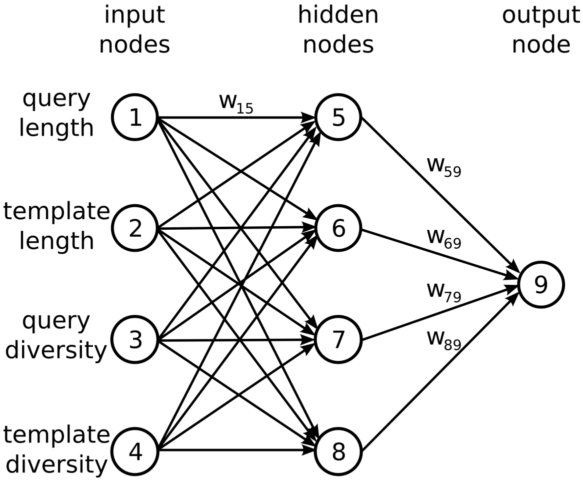


Figure 3.13: Neural network topology for the EVD parameter λ and μ. The network consists of 4 input nodes, 4 hidden nodes and one output node. Inputs are the length of the query and the database HMM and their diversities. All edges in this network have individual weights, which were trained with the backpropagation learning algorithm.

input node k and hidden node l. In the output layer, we simply use the sum of weighted hidden node outputs H_l to generate the final output O:

$$O = \sum_{l=5}^{8} w_{lo} H_l \tag{3.13}$$

The update of the weights in the neural network is performed by a backpropagation algorithm. It is a supervised learning method, which means, that for any input in the trainings dataset the output is known. The backpropagation algorithm updates the weights by a simple gradient descent approach, which means that it calculates the gradient of the error of the network regarding the weights and the weights are then updated in the direction of the negative gradient. This is repeated until the performance of the neural network is good enough.

To generate a training set for these neural networks, we create HMMs for several combinations of length and diversity and use the calibration step of the previous HHsearch version to calculation the λ and μ parameters. For the final learning of the neural networks our training dataset consists of more than 1 million combinations of input values and the corresponding parameters. 1% of these training data is taken from this training set and used as a validation set to check for overfitting. Overfitting can happen if you have too few training data for the free parameters of your neural network. In this case a neural network simply learns the input data, but could not generalize for slightly different inputs. By using an additional validation set while learning the risk of overfitting could be minimized.

We generate these neural networks with the SNNS (Stuttgart Neural Network Simulator[4]), a graphical application for generating, training and analyzing neural networks. The neural networks for λ and μ were built in this application and trained with our training data and the backpropagation learning algorithm. The resulting weights are extracted and implemented in our HHblits and HHsearch code. For each comparison between a query and a database HMM, the λ and μ parameter can now be calculated by using the weights and the activation functions in equation 3.12 and 3.13. Using the cumulative distribution function of the EVD ($F = e^{-e^{-(x-\mu)\lambda}}$) with these parameters, the P-value can now be calculated by

$$Pvalue(score) = \left(1 - e^{-e^{-h}}\right) \qquad \text{with} \qquad h = \lambda \cdot (score - \mu) \tag{3.14}$$

3.6. Realignment of matches by MAC algorithm

HHblits and HHsearch use by default the Viterbi algorithm for the comparison of profile HMMs. The scores generated by the Viterbi algorithm are well suited for the calculation of P-values (see section 3.5). But the alignment quality of the generated alignments could sometimes be increased by the use of the maximum accuracy (MAC) algorithm, which was

---

[4] `http://www.ra.cs.uni-tuebingen.de/SNNS/`

first proposed by Holmes and Durbin (1998) for global HMM-to-sequence alignments. A version adapted to the case of local HMM-HMM comparison was described by Biegert and Söding (2008). The MAC algorithm derives an alignment from a posterior probability matrix which is computed with the Forward/Backward algorithm. Based on these posterior probabilities, a MAC alignment σ maximizes the expected number of correctly aligned pairs of residues:

$$A(\sigma) = \sum_{(i,j)\in\sigma} P\left(M_i^q \diamond M_j^p\right) \rightarrow max \tag{3.15}$$

$P\left(M_i^q \diamond M_j^p\right)$ is the posterior probability of match state i in HMM q to be aligned to match state j in HMM p. For the case of local alignments a greediness parameter *mact* is used that is subtracted from the posterior probabilities for every aligned pair along the alignment path:

$$A(\sigma) = \sum_{(i,j)\in\sigma} \left(P\left(M_i^q \diamond M_j^p\right) - mact\right) \rightarrow max \tag{3.16}$$

This local MAC alignment maximizes the sum of probabilities for each residue pair to be correctly aligned minus a penalty (*mact*). With the *mact* parameter, the alignment greediness can be controlled, from nearly global, long, greedy alignments (*mact* near 0) to very precise and short (*mact* near 1). The posterior probabilities of every aligned pair are shown as confidence values in the resulting alignments. In appendix B we demonstrate the excellent correlation between the confidence values and the fraction of correctly aligned pairs.

Although the alignments generated by the Viterbi and by the MAC algorithm are often similar, there are cases where they differ completely. But unlike the Viterbi scores MAC scores are not guaranteed to follow an EVD and therefore the Viterbi scores are better suited for the P-value calculation. Thus, we use the Viterbi algorithm for the HMM-HMM comparison, but afterwards we realign the best matches with the MAC algorithm and save the new calculated alignments with the previously calculated Viterbi scores as results.

3.6.1. Consecutive merging of best alignments

This realign-procedure is parallelized for a small runtime (see section 3.7). But especially when the query profile in HHblits consists of only one sequence, the alignment quality could be increased by merging the information of the best matches to the query profile before realigning further database matches. Therefore, we consecutively merge the alignments of the first n matches (controlled by the parameter *premerge*) to the query profile and perform the parallelized realignment of all other matches with this evolved query profile.

3.7. Multi-threading support

Most of the current computers contain so called *multi-core* processors, which are single computing components with two ore more independent processor cores. Today, even laptops or desktop-computers contain dual- or quad-core processors (2 and 4 cores, respectively) and computing nodes in compute clusters often contains even more cores. Therefore, it is important for current software to be able to parallelize their computation on the cores to efficiently use all available power of the central processing unit. Since the previous version of HHsearch, the main HMM-HMM comparison and the realigning of the best matches by the MAC-algorithm are already parallelized. For each available core a single thread is generated and each of these threads performs the HMM-HMM comparison between the query HMM and one database HMM. One master thread controls the loading of new data to each thread. Here, it is important to lock the access to variables that are read and modified by several threads, so that only one thread at a time can write into such a variable.

This parallelization by threads of the HMM-HMM comparison and the realignment step is also used in HHblits. But for a fast runtime, the prefilter must also be parallelized on the available cores additionally to the SSE2 parallelization. We used the *OpenMP* (Open Multi-Processing) API which supports multi-platform shared memory multi-processing programming of various programming languages such as C and C^{++}. The API is managed by a non-profit consortium defined by many hard- and software vendors. With OpenMP a loop over the database sequences in the prefilter can simply be parallelized by adding the following line before the loop:

```
#pragma omp parallel for schedule(static) private(score, thread_id)
```

With this, the loop iterations will be split up among different threads and can be executed in parallel on the available cores. The private statement in the above code line defines which variables should be private for each thread and do not influence the variable value in other threads. The statement `#pragma omp critical` before a variable-assignment prevents different threads from writing to the same variable at the same time. In summary, the prefilter algorithm steps are parallelized on all available cores with OpenMP and on each core they are in addition parallelized by the SSE2 instructions.

3.8. Compact database by FFINDEX

HHblits performs an HMM-HMM comparison between a query and a database HMM profile. We therefore need to store HMMs for all database entries. In addition, if the user performs more than one search iteration or if he is interested in the output alignment, A3M-alignment files are needed for all entries. This means for the UniProt20 database that for a standard HHblits run with more than one iteration we have to store over 5 million files on the disk. This is a problem, since most filesystems cannot handle this amount of files in one directory and we have to split up all files in different sub-folders. In addition, the access time increases with the number of single files in one directory. A first step to reduce this number of files is to store HMM-files only for those alignments that consist of more than 50 sequences (see section 3.2).

In a second step, Andreas Hauser in our group wrote a small helper tool FFINDEX. It concatenates single files in one large database file, separated by null-characters, and creates a so called *index-file*. This index-file stores for every single file contained in the large database file its offset and length. With this information, each single file in the large one can be accessed very fast. Using a binary search in the index-file, the access time for getting the needed files out of the database is very fast with a guaranteed complexity of $O(\log(N))$. In addition, we save the disk space for the meta data for all single files. For generating an HHblits database, a binary called FFINDEX_BUILD is provided with the HHblits package. This binary reads all files in a directory or a list of files and generates one large database file together with a sorted index.

4. Results

The performance of HHblits is analyzed in various approaches, which will be described in this chapter. The first section includes benchmarks on the SCOP database that compare the sensitivity, the alignment quality and the runtime of HHblits with the ones of the iterative sequence search methods PSI-BLAST and HMMER3. This benchmark setup is also used for parameter optimization on a special optimization dataset. Furthermore, the improved quality of profiles generated by HHblits compared to those generated by buildali.pl is shown and we compare HHblits with HHsearch. In the second section, we show that the prediction of secondary structure with PSIPRED can be improved by using alignments generated by HHblits.

The third section demonstrates the advantage of HHblits by predicting new structures in Pfam families, for which until now no structure is known. For three of such cases we give a detailed overview of our results and show how these predictions could help biologists to elucidate the function of these proteins. Finally, we give an overview of our HHblits webserver, which is integrated in our Bioinformatics Toolkit[1] (Biegert et al., 2006).

4.1. Benchmarks

We evaluated the remote homology detection performance of HHblits and compared it to other iterative homology detection methods, PSI-BLAST (version 2.2.24) (Altschul et al., 1997) and HMMER3 (version 3.0 rc1) (Eddy, 2009). The benchmark dataset is derived from SCOP 1.73[2] (Murzin et al., 1995), filtered to a maximum pairwise sequence identity of 20% (SCOP20, 6616 domains). SCOP is a database of protein domains with known structure, hierarchically ordered by class, fold, superfamily, and family (see figure 4.1). Following a common procedure, we define all protein domains from the same fold as true positives (TPs, homologs) and all domains from different folds as false positives (FPs, non-homologs). Exceptions are the members of Rossman-like folds (c.2-c.5, c.27 and 28, c.30 and 31) and the four- to eight-bladed β-propellers (b.66-b.70). Pairs from these groups of folds should be treated as unknown.

We have to made sure that we never optimize any parameter on the proteins in the benchmark dataset, because ROCplot analyses are particularly prone to overtraining parameters

---

[1] `http://toolkit.lmb.uni-muenchen.de/`

[2] `http://scop.mrc-lmb.cam.ac.uk/scop/`

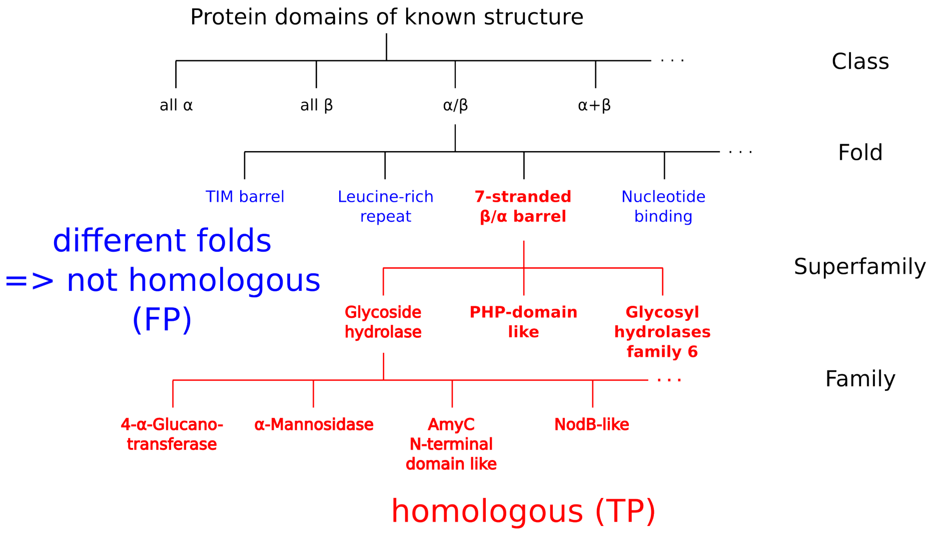


Figure 4.1.: Hierarchical organization of the SCOP database consisting of protein domains with known structure. The domains are hierarchically ordered from class (secondary structure composition) over fold and superfamily down to the family level (close homologs). Domains from the same fold are defined as true positives (red) and all pairs from different folds as false positives (blue). Exceptions are the the members of Rossman-like folds and the four- to eight-bladed β-propellers, which should be treated as unknown.

on the benchmark set, since the ROCplot depends on only a few tens or hundreds of high-scoring false positives in the relevant range. Furthermore, training and test proteins must not be too similar, since even for a maximum pairwise sequence identity of 25% between training and test *sequences*, the *profiles* built from these sequences can be very similar. For that purpose, we divide the SCOP20 dataset into an *optimization set* for parameter optimization and a *test set* for evaluating the performance by assigning the domains of every fifth fold in SCOP20 to the optimization set (1329 domains), the others to the test set (5287 domains). Because iterative search tools usually need large databases such as the UniProt database to build diverse profiles or profile HMMs, all but the last search iteration are performed against the UniProt database in the case of PSI-BLAST and HMMER3 and the profile HMM UniProt20 database in the case of HHblits. The last search iteration uses a combined database of the UniProt database and the SCOP set.

For HHblits, profile HMMs for all SCOP sequences are needed. But we could not use profile HMMs built by several iterations of HHblits or buildali.pl, because these profiles would be much more diverse than what is typical for UniProt20 clusters and this would lead to an unfair advantage over the other methods. To simulate realistic searches in our benchmark, our SCOP HMMs should represent realistic UniProt20 clusters of typical sizes. We therefore assigned each SCOP sequence to the best matching cluster in the UniProt20 database. To assign a SCOP sequence to a UniProt20 cluster, a BLAST search is performed with each SCOP sequence against a consensus database of all UniProt20 clusters. If a UniProt20 cluster match in this search has a sequence identity above 30% and a coverage with the SCOP sequence above 90%, the SCOP sequence is added to this cluster alignment. Then, a new HMM from the SCOP sequence and all sequences in this cluster is generated with the match states defined by the SCOP sequence. If no UniProt20 cluster is found that fulfills these requirements, a singleton cluster is created for the SCOP sequence.

4.1.1. Homology detection benchmark

We performed an all-against-all comparison of the test set domains with the three iterative homology detection methods and show their performances by ROCplots (see Figure 4.2). In a ROCplot (receiver operating characteristic plot) analysis the true and false positive hits at different E-value thresholds are counted and the cumulative values are plotted. To avoid a few large folds from dominating the benchmark, we weight each hit with 1/(number of members in query SCOP fold). The fold-weighted FPs on the x-axis are shown in a logarithmic scale to highlight the important region with low false discovery rates ($FDR = \frac{FP}{FP+TP}$, see appendix A.3). In all search iterations, HHblits detects significantly more TPs than PSI-BLAST and HMMER3. More precisely, HHblits detects twice as many TPs than PSI-BLAST and 54% more TPs than HMMER3 at 1% FDR in the first iteration (Figure 4.2A). At a 10% FDR, HHblits detects 112% more than PSI-BLAST and 68% more than HMMER3. For an iterative search strategy the performance at this low FDRs is important, because even a single non-homologous protein could corrupt the resulting profile. At a FDR of 10%, even two iterations of HHblits detect significantly more true positives than PSI-BLAST with three or even eight (data not shown) iterations and approximately the same number of TPs than HMMER3 with 3 iterations (light dashed red line in figure 4.2C).

High-scoring false positives

We further analyzed the high-scoring false positive matches in the third iteration of the previous benchmark to see, if we could identify a systematic error. HHblits has 90 false positives with an E-value better than 10^{-3}. But for many of them we could identify a similar structure or function and it is doubtful if these are real false positives. For example, 10 of the highest-scoring false positives (the best one has an E-value of 10^{-33}) are between members of the SCOP families d.211.1.1 and a.118.24.1. These families are annotated as *Ankyrin* and *Pseudo-Ankyrin* and the SCOP annotation for Pseudo-Ankyrin explains that there are similarities in the repeat sequence and assembly with the ankyrin repeat. Another big group of high-scoring false positives (15 matches with an E-value better than 10^{-3}) indicates a relationship between the SCOP families a.1.2.1 and d.58.1.5. The first family belongs to the superfamily of *alpha-helical ferredoxin* and the other family is annotated as *Ferredoxin domains from multidomain proteins*. The detailed annotation describes that members of this family may be more closely related to other ferredoxins than to each other. Further matches are between the SCOP families c.2.1.2 and c.72.3.1, the first one belongs to

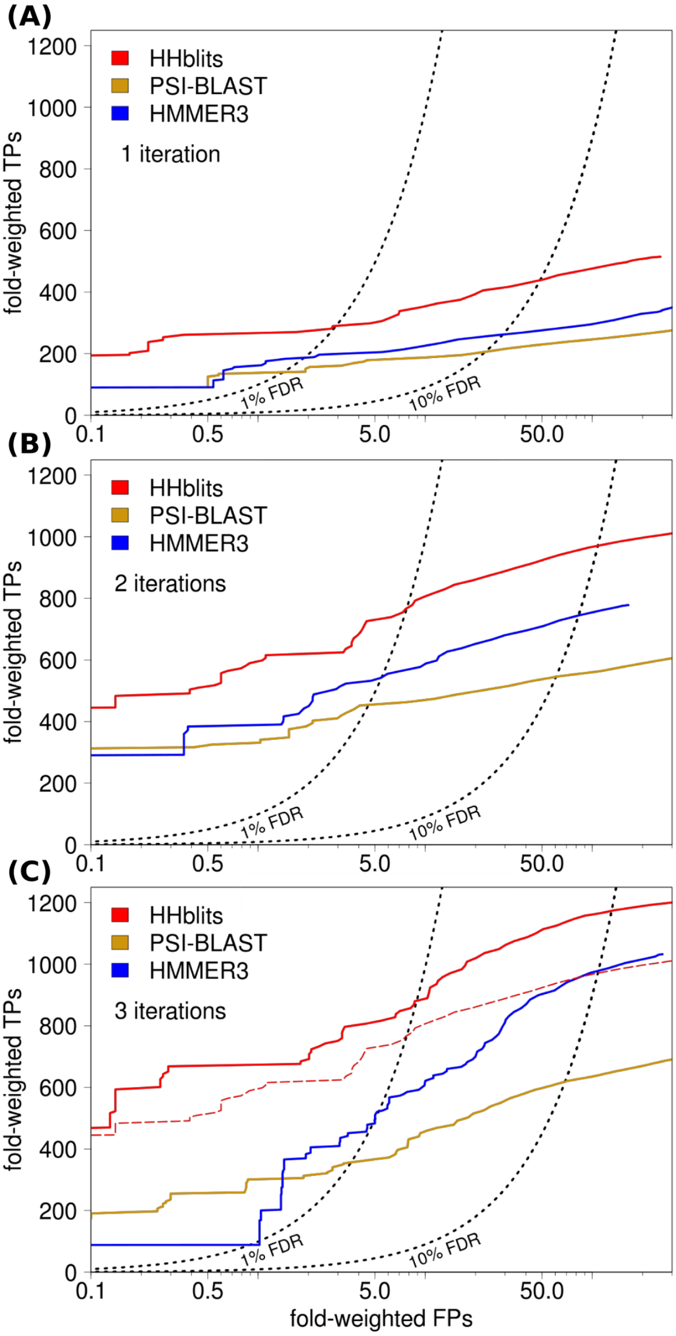


Figure 4.2: Homology detection sensitivity of iterative search methods. (A)-(C) show ROCplots for different number of iterations (1, 2 and 3, respectively) on the SCOP20 test set. All but the last search iteration are performed against the UniProt to build a profile as input for the last search iteration against a combined database containing the UniProt database and the SCOP20 dataset. TPs are defined as pairs from the same SCOP fold, FPs as pairs from different folds. At a false discovery rate (FDR) of 10% HHblits detects significantly more TPs than the other methods, for example in the first iteration twice as many as PSI-BLAST and 68% more than HMMER3. In (C), the light red curve shows the performance of 2 iterations of HHblits and it demonstrates the clear improvement to 3 iterations of PSI-BLAST.

the large group of *Rossmann-fold domains* and the second one is annotated as combination of the *Rossmann-like* and *Ribokinase-like* topologies.

The analysis of high-scoring false positive matches of the other methods shows that these tools identify much more false positives with an E-value better than 10^{-3}. HMMER3 has 1809 such false positives and 95 have an E-value better than 10^{-9}. PSI-BLAST results more than 15900 false positive matches with an E-value better than 10^{-3}, but only 32 with an E-value better than 10^{-9}. Some of these false positives belong to the same families mentioned before. Many of the highest-scoring false positives in PSI-BLAST stand out, because these are matches between two different SCOP families located on the same protein (e. g., *d1vgya2* and *d1vgya1*). Hence these false positive matches might be appear due to homologous over-extension (Gonzalez and Pearson, 2010), where alignments of homologous domains extend into neighboring non-homologous regions and unrelated information is included in the query profile for the next iteration.

Roc5 benchmark

Figure 4.3B gives the ROC5 plot for this benchmark, which assesses how well a method ranks the matched proteins within each search, thus E-values doesn't need to be comparable between searches. From the TP and FP hits at various E-value thresholds we can infer a ROC5 score ($\in [0,1]$) for each query. This score is defined as the area under the TP-versus-FP ROC curve up to the fifth false positive hit, divided by the area under the optimal ROC curve (see example in figure 4.3A). To assess the overall homology detection performance, we plot the fraction of queries with ROC5 scores above a variable ROC threshold ($\in [0,1]$). Again, HHblits outperforms PSI-BLAST and HMMER3 in all search iterations. After two search iterations, for example, HHblits achieves a ROC5 score above 0.2 for 38% of all queries, whereas HMMER3 has 29% and PSI-BLAST only 27% of queries with a ROC5 score above 0.2. For 20% of the queries, HHblits achieves a ROC5 score above 0.51, HMMER3 above 0.39 and PSI-BLAST above 0.33.

4.1.2. Homology detection benchmark with multi-domain proteins

A common problem of iterative methods is homologous over-extension (Gonzalez and Pearson, 2010), where alignments of homologous domains extend into neighboring non-homologous regions and unrelated information is included in the query profile for the next iterations,

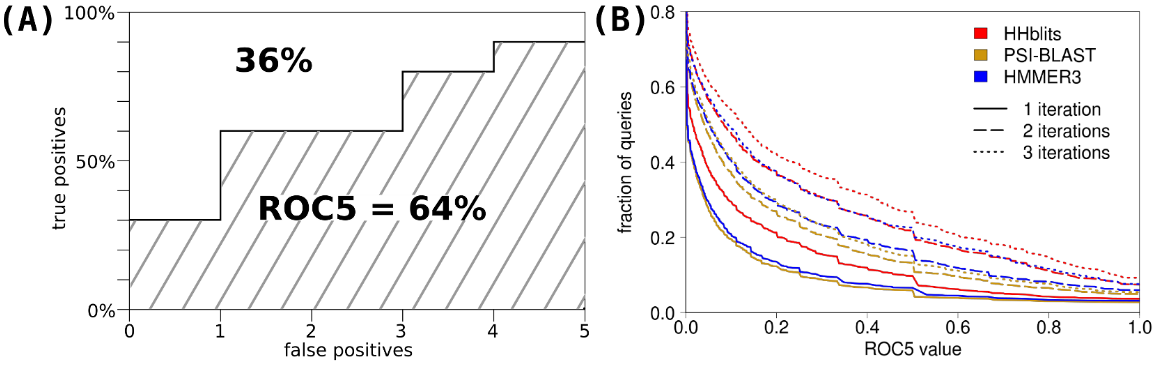


Figure 4.3.: ROC5 benchmark of iterative search methods. (A) Example for the calculation of a ROC5 score. This score is defined as the area under the TP-versus-FP ROC curve (gray hatched area) up to the fifth false positive hit, divided by the area under the optimal ROC curve. (B) This ROC5 plot shows the fraction of queries with an ROC5 value above the threshold on the x-axis. For all iterations, the curve of HHblits dominates the curves of the other methods.

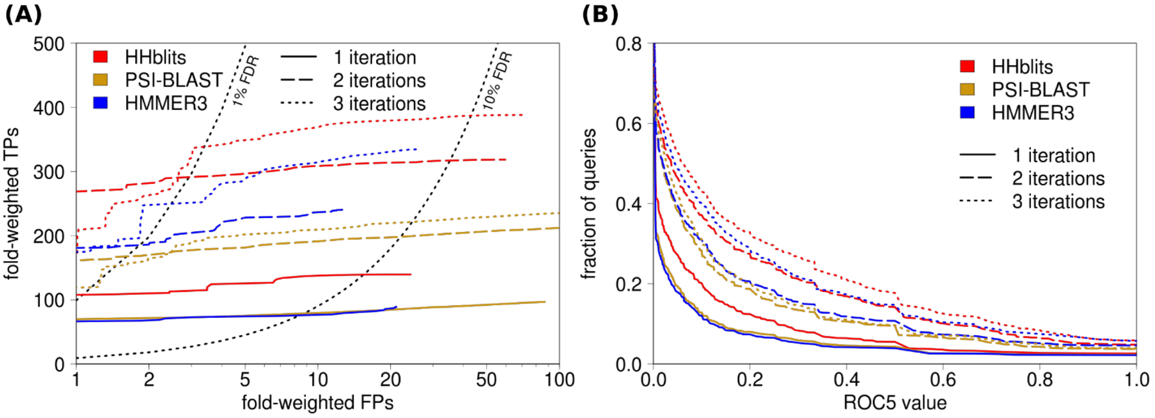


Figure 4.4.: Homology detection benchmark on multi-domain proteins. True and false positives are counted only if they cover the region of the SCOP domain in the query NR protein. (A) HHblits detects significantly more TPs than PSI-BLAST and HMMER3 in the ROCplot, for example 67% more TPs than PSI-BLAST and HMMER3 at 1% error rate in the first iteration. Due to the prefilter steps, HHblits and HMMER3 detect only a small fraction of false positives and their curves end early. (B) For all iterations in this ROC5 plot, HHblits clearly outperforms the other methods.

which will lead to high-scoring false positives in the next search iterations. To check for this type of error, we have created a benchmark set of multi-domain proteins. This set consists of proteins from the non-redundant (NR) database from NCBI[3] that contain a SCOP domain from our SCOP20 test set. The hit between the NR-sequence and the SCOP domain must fulfill the following conditions: PSI-BLAST E-value better than 10^{-40}, coverage of more than 95%, sequence identity above 60% and the NR-sequence must contain at least 100 additional residues. Because of these additional residues the NR-sequence contains with a high probability at least one further domain. This procedure leads to 2343 multi-domain proteins from the NR.

We perform the same comparisons as in section 4.1.1 for all extracted multi-domain proteins. We count TP and FP pairs only if the corresponding alignment covers the SCOP domain in the query NR protein by at least 50 residues. The resulting ROCplot is shown in figure 4.4A. Again, HHblits detects significantly more TPs in all search iterations compared to PSI-BLAST and HMMER3. In detail, HHblits detects 67% more TPs than PSI-BLAST and HMMER3 at 1% FDR in the first iteration and 77% more at 10% FDR. A similar performance is shown in the ROC5 plot in figure 4.4B. Again, HHblits outperforms PSI-BLAST and HMMER3 in all search iterations.

4.1.3. Alignment quality

In another benchmark, we evaluated the alignment quality for the iterative homology detection methods. We compared the predicted sequence alignments between SCOP domains with a gold standard structural alignment generated by TM-ALIGN (Zhang and Skolnick,

---

[3] `http://www.ncbi.nlm.nih.gov/`

2005). To create a benchmark data set, we randomly picked up to 10 domain pairs from each family in SCOP 1.73, requiring a maximum sequence identity of 30% and a minimum TM-SCORE of 0.6 when aligning the sequences with TM-ALIGN. The latter criterion ensures that the proteins have a similar structure and a high-confidence structural alignment. The resulting set consists of 5904 domain pairs from 2895 different domains. For each of these 2895 protein domains we performed 2 search iterations against the UniProt to build a profile, and a last iteration against a database containing all proteins belonging to the same family as the query domain. The alignment quality for those pairs with a structural reference alignment is assessed by 2 standard performance measures: *Alignment sensitivity* is the fraction of structurally aligned residues pairs that are correctly predicted (pairs correctly aligned/pairs structurally alignable). *Alignment precision* is defined as the fraction of aligned residues pairs in the predicted alignment that are correct (pairs correctly aligned/pairs aligned).

Figure 4.5A shows the average residue-based sensitivity and precision of the created alignments. In HHblits a special parameter (*mact*) allows to choose the trade-off between sensitivity and precision (see section 3.6). With the default value for *mact* (0.5), HHblits alignments have an increased precision of 15% compared to HMMER3 at a slightly better sensitivity, the precision and sensitivity compared to PSI-BLAST alignments could be increased by 26% and 25%, respectively. The additional use of predicted secondary structure in HHblits further increases the alignment quality (dark red curve). In all other benchmarks we do not use predicted secondary structure, since this would be an unfair advantage to the other methods which did not use this information. The alignment sensitivity and precision for various sequence identity bins is given in Figure 4.5B. HHblits outperforms PSI-BLAST and HMMER3 both in sensitivity and precision over the entire range of sequence identities, especially for the difficult alignments with a sequence identity below 20%.

4.1.4. Runtime

An important consideration when deciding between alternative software packages such as PSI-BLAST and HHblits is the runtime requirement of the program. Our goal was to have a runtime similar to the widely used PSI-BLAST method, in order to have all arguments for switching to our new method on our side. Therefore, we measured the runtimes of the three iterative homology detection methods for various number of iterations.

The test set for this benchmark was generated by randomly picking 100 proteins from the NCBI NR (non-redundant) database. This set has an average length of 354 residues, the shortest sequence has 30 and the longest 2102 residues (see length distribution in appendix A.1). For all sequences in the test set a homology search is performed against the UniProt database with the default parameters of each program. Every run is performed without multi-threading on a Intel Xeon X5570 processor with 2.93GHz and 24GB RAM. To reduce the influence of the computer network, all data was initially copied to a local disk. All runs

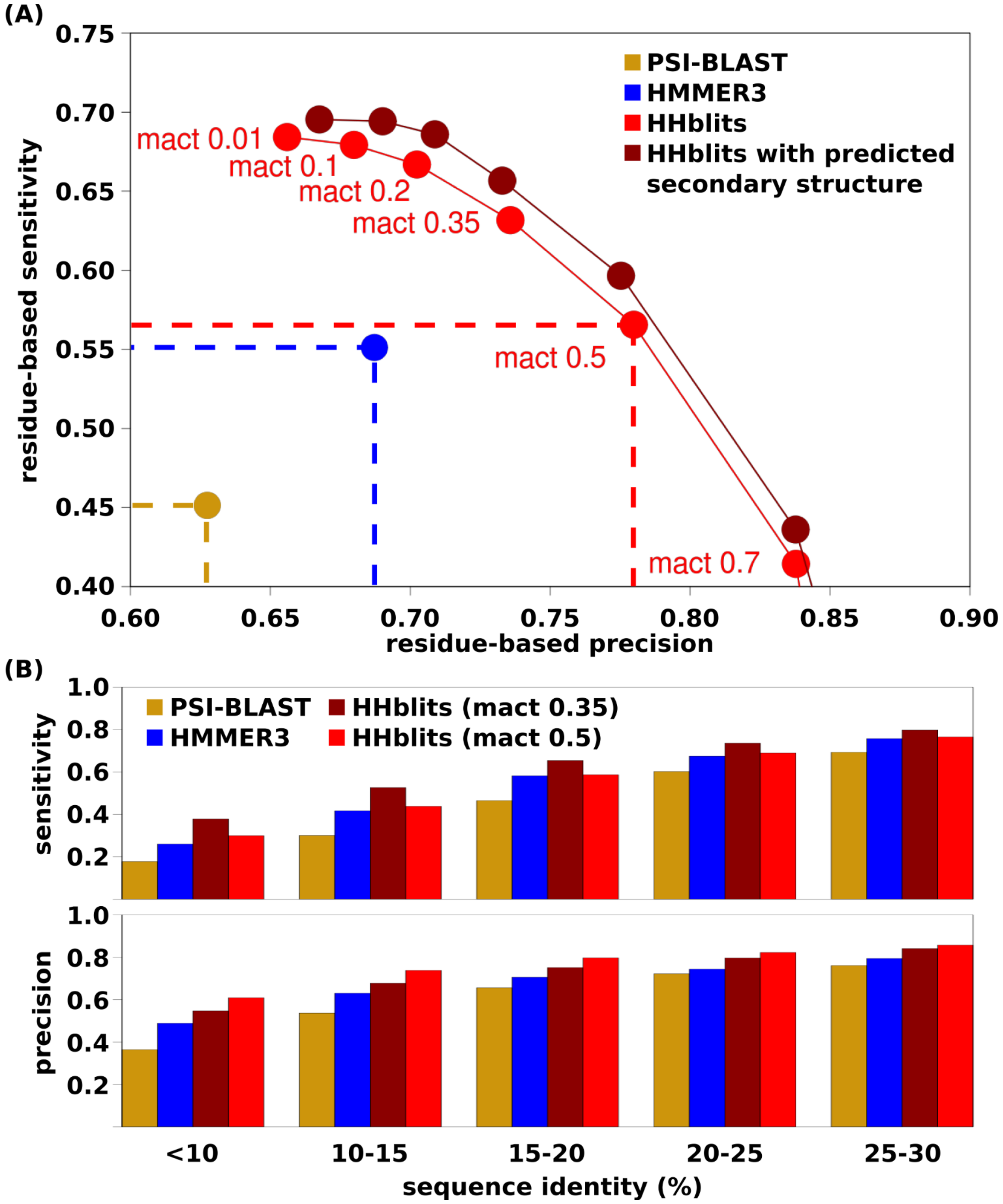


Figure 4.5.: Alignment quality of remote homology detection methods. (A) Average alignment residue-based sensitivity and precision for the iterative search methods compared with a gold-standard structural alignment generated by TM-ALIGN. HHblits generates alignments with a significant better quality than the other detection methods and in addition, the trade-off between sensitivity and precision can be chosen in HHblits with the mact-parameter. (B) Alignment sensitivity and precision for various sequence identity bins. HHblits outperforms PSI-BLAST and HMMER3 in both measurements, especially in the difficult alignments with a sequence identity below 20%.

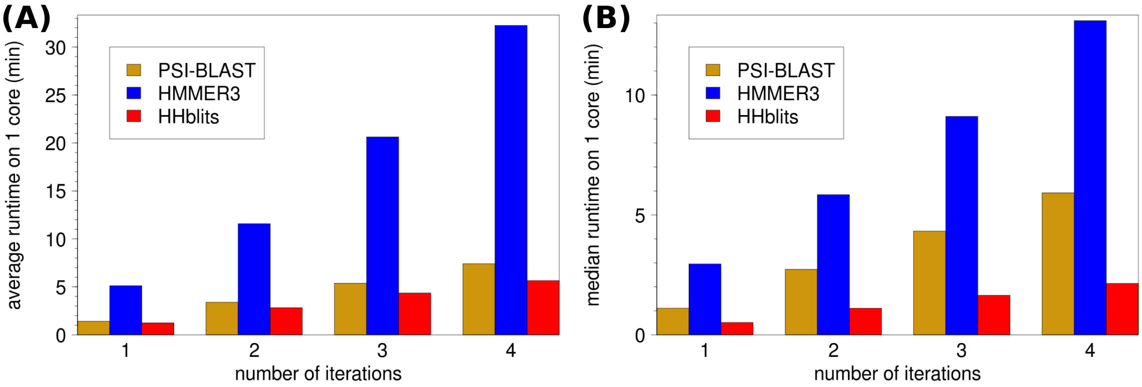


Figure 4.6.: (A) Average and (B) median runtimes of the homology detection methods for random proteins. The runtime is computed on 100 random protein sequences from the NCBI NR database with default parameters for each method. HHblits is slightly faster than PSI-BLAST, HMMER3 shows a significantly longer runtime.

are repeated 3 times and the average runtime is calculated.

Figure 4.6 shows the average and median runtimes for the three homology detection methods for 1, 2, 3, and 4 iterations, respectively. The average runtime of HHblits is slightly better than the one of PSI-BLAST for all numbers of iterations. One has to keep in mind that the sensitivity of 2 iterations of HHblits significantly outperforms the sensitivity of 8 PSI-BLAST iterations, so the HHblits runtime for similar results is clearly better. HMMER3 shows for all iterations a significantly longer runtime by a factor 3 to 5. In the case of median runtimes, the differences between HHblits and PSI-BLAST are even bigger.

Additional runtime measurements are shown in figure 4.7. Here we check the influence of the query length on the runtime ((A) - (D) for different numbers of iterations) and the scaling of the runtime when using multi-threading on multi-core processors ((E) and (F)). HHblits has a very good runtime for queries with a sequence length below 400 residues and clearly outperforms PSI-BLAST in this range of query lengths ((A)-(D)). This range covers more than 73% of all sequences (see length distributions in appendix A.1). In the range of 400 to 800 residues both methods have a similar runtime and only for proteins with a length $>$ 800 residues the runtime of HHblits is slightly worse than that of PSI-BLAST. HMMER3 scales in a similar way as HHblits with the query length, always by a factor 3 to 5 slower. Figure 4.7E gives the runtimes of 2 iterations against the UniProt database when various numbers of CPUs are used. The same results are shown in figure 4.7F, but this time, the total CPU time is given on the y-axis. The total CPU time is calculated by multiplying the real runtime with the number of used CPUs. PSI-BLAST scales very well with the number of CPUs, the total CPU time is nearly the same for all number of CPUs. The scaling of HHblits and HMMER3 is a little bit worse, especially when using more than 4 CPUs. For HHblits the reason is that some code parts of our implementation are still single-threaded, but we are planning to parallelize these parts (see outlook in section 5).

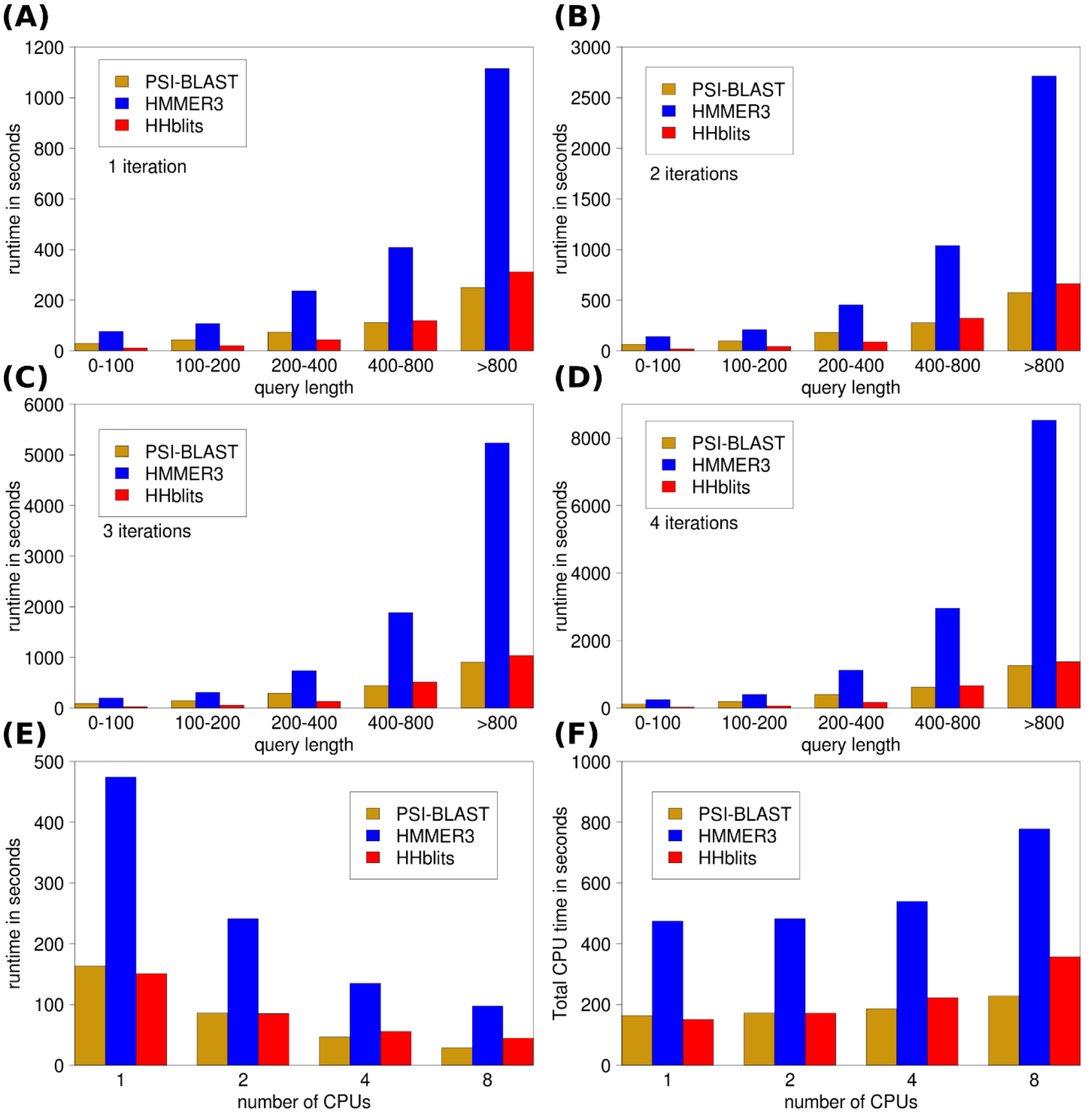


Figure 4.7.: (A)-(D) Runtime dependency of the benchmarked methods on the query length for 1, 2, 3 and 4 iterations, respectively. HHblits has a very good runtime for queries with a sequence length below 400 residues and clearly outperforms PSI-BLAST in this range of query lengths. In the range of 400 to 800 residues both methods have a similar runtime and only for proteins with a length > 800 residues the runtime of HHblits is slightly worse to that of PSI-BLAST. HMMER3 scales in a similar way as HHblits with the query length, always by a factor 3 to 5 slower. (E)-(F) Runtime dependency for 2 iterations on the used number of CPUs. (E) shows the real runtime, whereas (F) gives the total CPU time of the runs. The total CPU time is defined as the real runtime times the number of CPUs. PSI-BLAST scales very well with the number of CPUs, the total CPU time is nearly the same for all number of CPUs. The scaling of HHblits and HMMER3 is a little bit worse, especially when using more than 4 CPUs.

4.1.5. Quality of E-values

Another critical aspect for homology detection methods is the reliability of the reported P-values and E-values. The E-value of a match is an estimate of the number of chance hits to be expected with a score better than that of the database match. It can be calculated from the P-value, which is the probability to obtain a chance hit with score $\geq S$, by the following formula: $E = N_{DB}P \times (E_{pre}/N_{DB})^{\alpha}$ (for details see section 3.5). α is defined as $\alpha = 0.4 + 0.02 \times (\mathrm{Neff}_T - 1) \times (1 - 0.1 \times (\mathrm{Neff}_Q - 1))$. The three coefficients were optimized by testing several combinations and choosing the best performing one.

The database for this benchmark is generated from the NCBI NR (non-redundant) database by reversing all sequences in this database and shuffling the residues in each sequence. For the HHblits NR20 database we reverse and shuffle the alignment columns instead of the single residues in the sequences. As query set we take 20 000 random sequences from the NR. Because of the reversed and shuffled database there should be no homologous pairs between

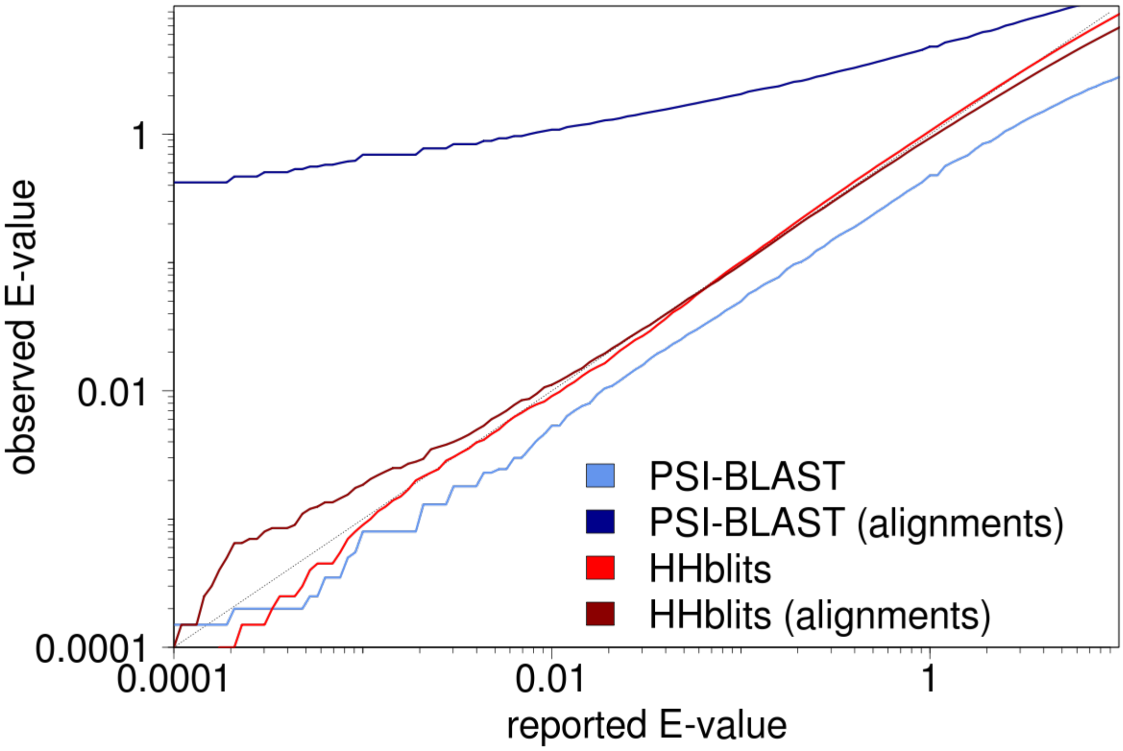


Figure 4.8.: E-value quality of HHblits and PSI-BLAST. The number of chance matches in a search with random NR proteins against a database of shuffled sequences is used to derive the observed E-value for each reported one. The light colored curves show the results of a default search with HHblits (red) and PSI-BLAST (blue). Both curves indicate reliable E-values, the ones of PSI-BLAST are slightly too pessimistic by a factor of ~ 2. The dark colored curves demonstrate the E-value quality when starting the searches with a query alignment instead of a single query sequence. HHblits (dark red) again produce reliable E-values, whereas the PSI-BLAST E-values (dark blue) are way too optimistic, i. e., a reported E-value of 10^{-3} corresponds to an observe one of 0.6.

any query sequence and a database sequence and so any match between two sequences corresponds to a chance hit. We perform a search with each sequence in the query set and count the number of matches at a given (*reported*) E-value threshold, which, together with the size of our query set, allows us to derive the *observed* E-value. Figure 4.8 plots the observed against the reported E-value of HHblits and PSI-BLAST. Desirable is a curve near the main diagonal, which means that the reported E-values are nearly identical to the observed ones. The curves in light colors show the results for a standard HHblits (red) or PSI-BLAST (blue) run started with a single sequence. The reported E-values of HHblits are nearly identical to the observed ones, only near an E-value of 10^{-4} the reported E-value is a little bit too pessimistic. The observed E-values of PSI-BLAST are slightly too pessimistic by a factor of ~ 2, i. e., an observed E-value of 10^{-2} corresponds to an reported E-value of 2×10^{-2}.

We are also interested if the E-value quality changes in later iterations when the input is not a single sequence but a sequence alignment instead. Therefore, we generate sequence alignments for all sequences in the query set by running 2 iterations HHblits and perform the benchmark with these alignments as input. The results are shown as dark colored lines in figure 4.8. The curve of HHblits (dark red) indicates again a good E-value quality, the reported E-values are nearly identical with the observed ones. But the results of the PSI-BLAST run (dark blue curve) show a significant difference. These E-values are way too optimistic, i. e., a reported E-value of 10^{-3} corresponds to an observed one of 0.6. This seems to be related to an internal low-complexity filter in PSI-BLAST that is deactivated when starting with a profile instead of a sequence. If we, for example, perform 2 iterations PSI-BLAST with a query protein that contains a low-complexity region against a database of shuffled sequences, we get in the first iteration no results below the E-value threshold. But in the second iteration, we get many false positive matches with good E-values due to the low-complexity region, although the input profile still consists of only one sequence.

4.1.6. Comparison of HHblits to HHsearch

HHblits is the iterative version of the HMM-HMM comparison method HHsearch. When performing only 1 search iteration of HHblits, the only difference to HHsearch is the prefiltering of the database. In order to check the loss of sensitivity through the prefilter, we perform an all-against-all search on SCOP profiles with HHsearch and HHblits with the same options. The SCOP profiles were generated by buildali.pl with default options. Figure 4.9 shows the ROCplot of this benchmark. Both methods perform very similarly, they detect the same amount of TPs at a FDR of 1%. Only at a FDR of 10%, the sensitivity of HHsearch is slightly better by $\sim 1.5\%$. But in return the runtime of HHblits is shorter by a factor of ~ 20 on a single CPU for this test set.

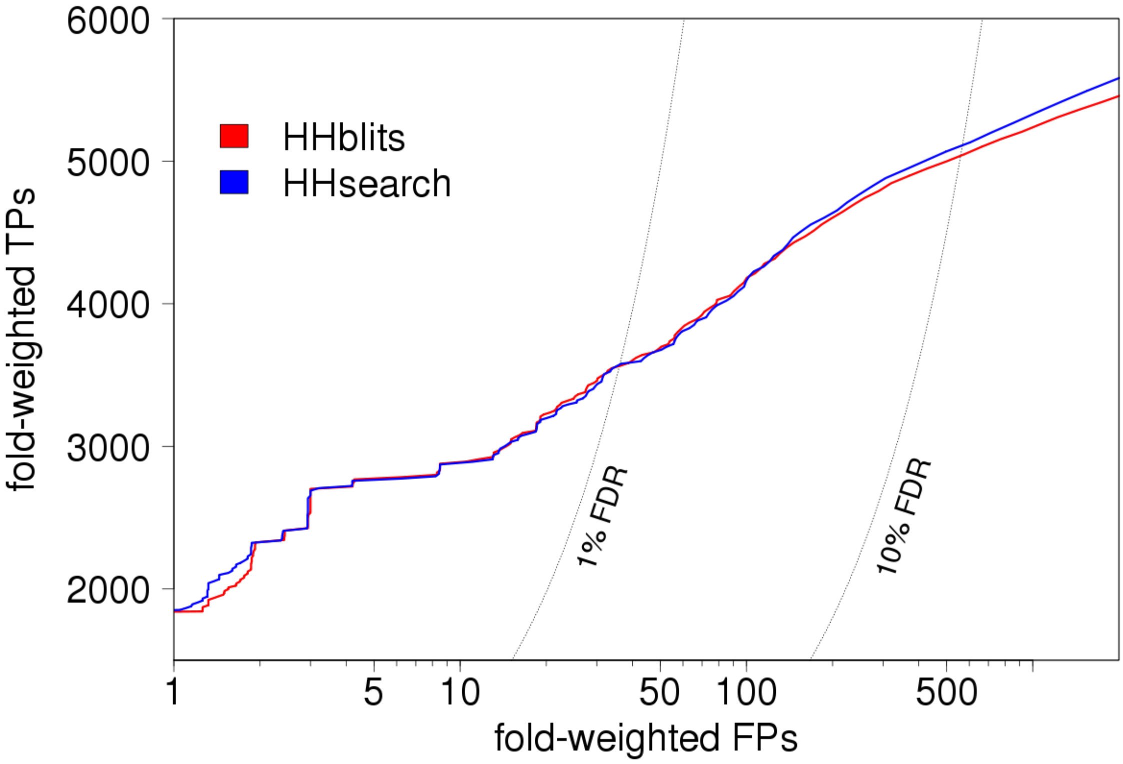


Figure 4.9.: Performance comparison of HHblits and HHsearch. Both methods are benchmarked with the same parameters (HHblits only 1 iteration), so the only difference is the prefilter step of HHblits. Both methods show a very similar performance, at a FDR of 10% the sensitivity of HHsearch is only slightly better (∼ 1.5%). But in return, the prefilter of HHblits makes sure that the runtime is significant shorter.

4.1.7. Comparison of profiles built by HHblits or buildali.pl

Important for the performance of HMM comparison methods is the quality of the input profile HMMs. A corrupted profile that contains information from non-homologous sequences will lead to high-scoring false positives. An input profile with a low diversity on the other hand may contain too little information to identify very remote homologs. Up to now, HHsearch uses the buildali.pl script to generate the profile HMMs for the query sequence and for all database sequences. buildali.pl performs up to 8 iterations of PSI-BLAST and in addition to a simple PSI-BLAST search, it runs an edge-pruning algorithm to reduce the risk of corrupted alignments by homologous over-extension. This can occur in multi-domain proteins by a homologous match which is extended into a non-homologous region.

Here, we check the performance of HHsearch with profiles built by HHblits and by buildali.pl with their default parameters. We generate different sets of profile HMMs for each sequence in our SCOP20 benchmark set. These sets consist of 1 and 3 iterations of HHblits and of 1, 3 and 8 iterations of buildali.pl. As database we use the sets with the most diverse profile HMMs, that means, for HHblits the set with 3 iterations and for buildali.pl the set with 8 iterations. We perform an HHsearch run on all generated sets with their corresponding database. To check the influence of the profile HMM only on the query side, we also run HHsearch with the HHblits profile HMMs on the query side and the buildali.pl database.

Figure 4.10 shows the HHsearch performance with the different profile HMMs. In all cases the profiles generated by a higher number of iterations lead to a higher sensitivity at a FDR of 10%. The profiles built by HHblits increase the sensitivity of HHsearch significantly compared to the profiles built by buildali.pl. In more detail, HHsearch detects 45% more TPs at a FDR of 1% when using profiles built by 3 iterations HHblits compared to them built with 3 iterations buildali.pl, 17% more at a FDR of 10%. Further iterations of buildali.pl (dotted blue line for 8 iterations) gives only very little impact. When using HHblits profiles only on the query side (orange curves), the improvement of HHsearch is about the half of the improvement when using HHblits profiles on both sides.

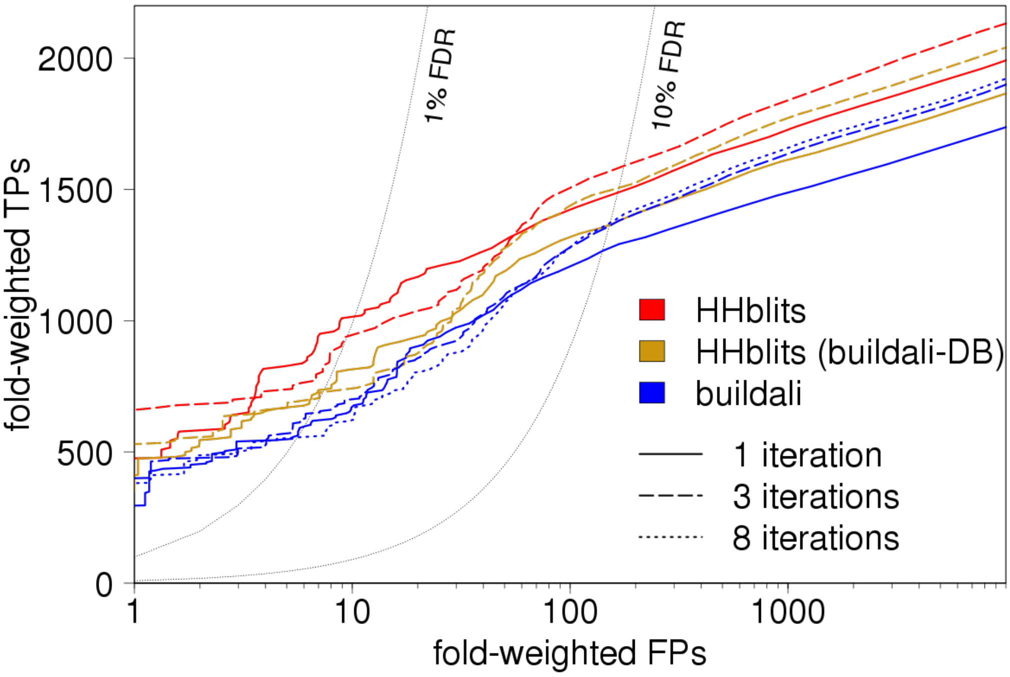


Figure 4.10.: HHsearch performance with different profile HMMs as input. The red curves demonstrate the performance, when the query profiles are built with 1 (solid) or 3 (dashed) iterations of HHblits. The database in these cases contains the 3 iteration profiles. In blue, the performance for using buildali.pl profiles is shown. Here, the database contains the profiles from 8 iterations buildali.pl. For the orange curves, we use HHblits profiles on the query side, but the database contains the profiles from 8 iterations buildali.pl. The HHsearch sensitivity is clearly increased when using HHblits profiles.

Using HHblits profiles for comparative modelling

The quality of the input profiles is not only important for the sensitivity of HHsearch, it is also important for the quality of homology models generated with HHsearch and MODELLER (Sali and Blundell, 1993). Therefore, we used our SCOP benchmark set from the alignment quality benchmark (see section 4.1.3) with 2895 protein domains. We generate a profile set with 8 iterations buildali.pl, and with 3 iterations HHblits with a mact of 0.5 and 0.35. On these three test sets, we run our CASP pipeline to build homology models (Hildebrand et al., 2009). The generated models are compared with the original structures and the TM-SCORE (Zhang and Skolnick, 2004) is calculated for each pair. This score indicates the difference between two structures by a score between (0,1], where 1 indicates a perfect match.

The results are given in table 4.1. The models generated with HHblits profiles outperform the models built with buildali.pl profiles for nearly all sequence identity ranges by ∼ 1.4%. The best models are generated when using more specific profiles created with a higher mact-value, but the differences are rather low.

4.1.8. UniProt20 versus NR20 database

We are interested in the influence of the protein database on the performance of HHblits. There are two widely used databases which cover the whole protein sequence space. One is the NR (non-redundant) database from the NCBI [4] with 14 605 097 sequences in the actual version (August 12, 2011), the other database is the UniProt (Universal Protein Resource) database [5] with 14 423 061 sequences (March 29, 2011). The length distribution of these databases is given in appendix A.1. The UniProt database consists of two sections, the Swiss-Prot DB, which is manually annotated and reviewed (∼ 500 000 sequences), and the automatically annotated TrEMBL database. We cluster both databases down to a maximum pairwise sequence identity of ∼ 20% and generate the database files for HHblits.

Figure 4.11 shows the results for different iterations of both databases when searching with

---

[4] `http://www.ncbi.nlm.nih.gov/`

[5] `http://www.uniprot.org/`

Table 4.1.: Quality of structural models based on profiles generated by 8 iterations of buildali.pl and 3 iterations of HHblits. The quality is measured as TM-SCORE between the structural model and the original structure.

Sequence ID range	BUILDALI.PL	HHblits mact 0.35	HHblits mact 0.5
< 10%	0.532836	**0.546671**	0.542546
10% - 15%	0.583984	0.592679	**0.593901**
15% - 20%	0.635397	0.643247	**0.644214**
20% - 25%	0.660866	0.669225	**0.670757**
25% - 30%	0.681375	0.682534	**0.684878**
total	0.623852	0.631827	**0.632812**

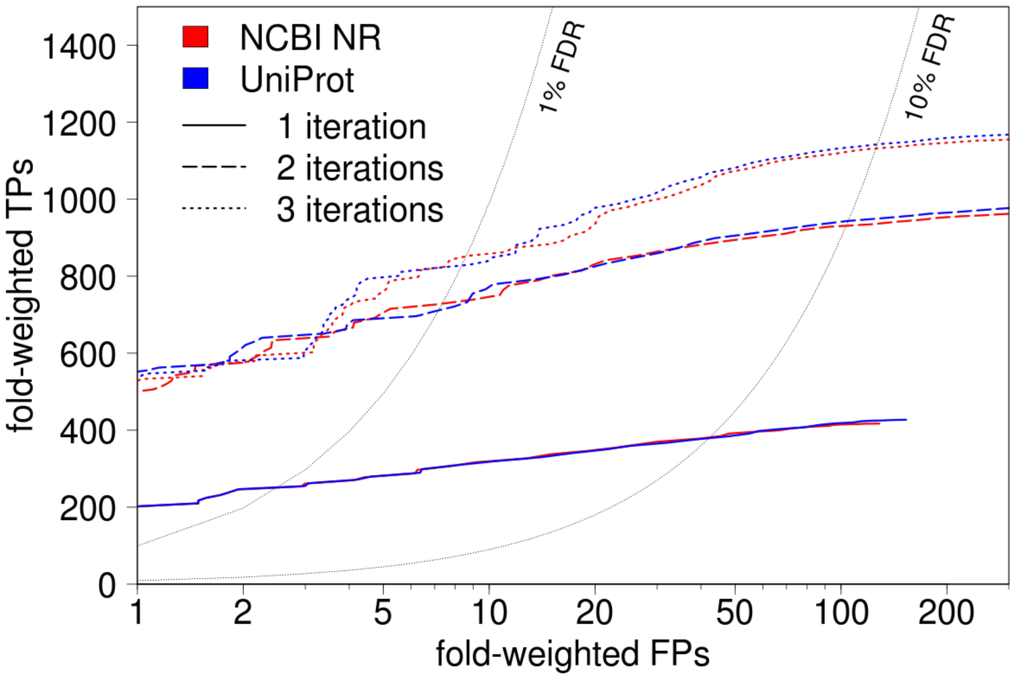


Figure 4.11.: Performance of HHblits when using the cluster version NR20 of the NR database from NCBI or the UniProt20 database. In all search runs, we perform the last iteration against a combination of the given database and the SCOP20 test set. Both databases perform very similarly with only a small advantage for the UniProt database.

all queries in the SCOP20 test set. In all cases, the database for the last search iteration is combined with the SCOP20 test set to obtain true and false positive hits. Like in the other benchmarks shown before, the true and false positive matches are weighted by the fold-size of the query fold to avoid a few large folds from dominating the benchmark. Both databases perform very similar for all number of search iterations. We see only a very small increase when using the UniProt database. For the final release of HHblits we decided to use the UniProt database as default database because of the good manual annotation of the Swiss-Prot entries.

4.1.9. Using HMMER3-formatted databases

Some databases in the bioinformatics field are available as HMMER2- or HMMER3-formatted databases only, without the multiple sequence alignments that gave rise to the HMMs. An example of such a database is the CATH database[6], a hierarchical domain classification of protein structures in the PDB. HHblits can read profile HMMs in HMMER3 format, but we also have to generate a prefilter database of column state sequences. This means that we have to built a column state sequence from an HMMER3 profile HMM. A problem with HMMER3 profile HMMs is that these profiles already contains pseudocounts.

---

[6] `http://www.cathdb.info/`

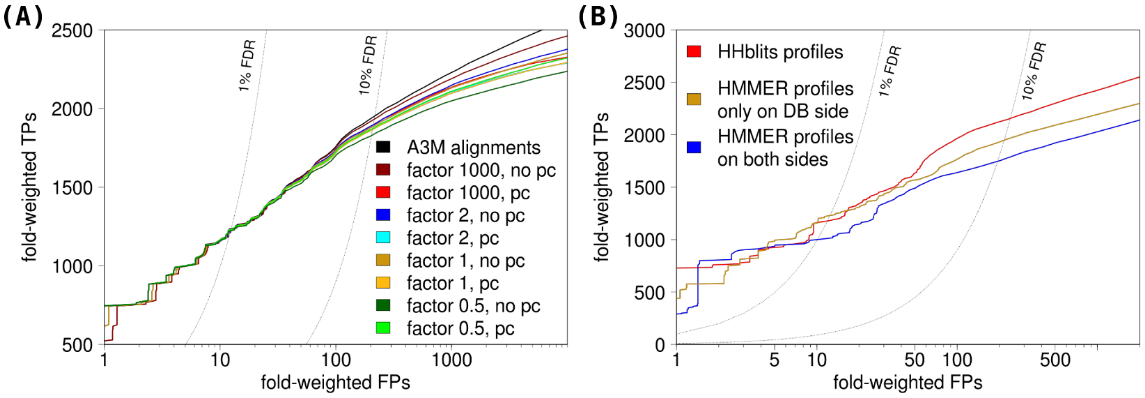


Figure 4.12.: HHblits performance with a database based on HMMER3 profiles. (A) The column state database for the prefilter step is built from A3M alignments (black curve) or from mixtures of the HMMER3 profiles and the corresponding original database sequences. A factor indicates the ratio of the HMMER3 profile to the original sequence, i. e., a factor of 1 means that the HMMER3 profile and the original sequence are used in a ratio of 50/50 and a factor of 2 corresponds to a ratio of 66/33. 4 different factors are checked, each one time with additional pseudocounts and one time without. The best performance is reached with a column state database generated from A3M-alignments. (B) The performance of HHblits decreases significantly when using HMMER3 profiles on both the query and the database side.

Figure 4.12A demonstrates the performance of HHblits with HMMER3 profile HMMs of a SCOP25 test set. The HMMER3 profile HMMs were generated with HMMBUILD from the SCOP25 A3M-alignments. The difference between the curves is the method how the column state sequences are generated. The best way to generate these sequences is to use A3M-alignments (black curve), if they are available. The other curves show the cases, where for each database entry a temporary alignment is built from the original database sequence and the corresponding HMMER3 profile HMM in different ratios and this alignment is used to generate the column state sequence. A factor indicates the ratio of the HMMER3 profile to the original sequence, i. e., a factor of 1 means that the HMMER3 profile and the original sequence are used in a ratio of 50/50 and a factor of 2 corresponds to a ratio of 66/33. We have checked 4 different factors (0.5, 1, 2, 1000) and for each factor, we built the column state database in one case with additional pseudocounts, in another case without. A factor of 1000 and no additional pseudocounts gives the best results of these cases (dark red curves). This means that the best way is to use mainly the HMMER3 profile. Additional pseudocounts are counterproductive, because the HMMER3 profile already contains pseudocounts.

In another benchmark we compare the performance of HHblits with HMMER3 profiles compared to HHblits profiles. We used SCOP25 A3M alignments and generated HMMER3 profiles with HMMBUILD and HHblits profiles with HHmake. The column state database is generated by using the A3M alignments. Figure 4.12B shows the performance of HHblits when using different combinations of these profile sets. The use of HMMER3 profiles on the database side leads to a significantly decreased sensitivity of HHblits. The sensitivity decreases by 11% at a FDR of 10%. This effect is even more drastic when using HMMER3 profiles on both the query and the database side (blue curve). Here, the sensitivity of HHblits drops by 24% at a FDR of 10%. In summary, whenever possible HHblits formatted HMMs should be used instead of HMMER3-formatted files.

4.1.10. Prefilter parameters

The prefiltering step in HHblits is crucial for the performance. Too strict prefiltering would lead to a decreased sensitivity, while on the other side too lax prefiltering would increase the runtime. As described in section 3.3, we cannot use a normal profile-profile alignment algorithm due to its slow runtime. Here, we compare the performance of using a prefilter based on consensus sequences and prefilters based on column state sequences. All these prefilters have a similar runtime.

Figure 4.13 gives the results of this benchmark for 1 and 2 iterations of HHblits. The prefilters based on column state sequences (red and green) clearly outperform the prefilter approach based on consensus sequences. For 2 iterations there is an improvement in sensitivity of more than 20% at a FDR of 10%. When increasing the number of column states from 62 (green) to 219 (red) states, we obtain a further slight improvement.

Furthermore, we are interested in the sensitivity of our fast prefilter algorithms (see sections 3.3.5-3.3.8) without downstream HMM-HMM comparison. We perform an all-against-all

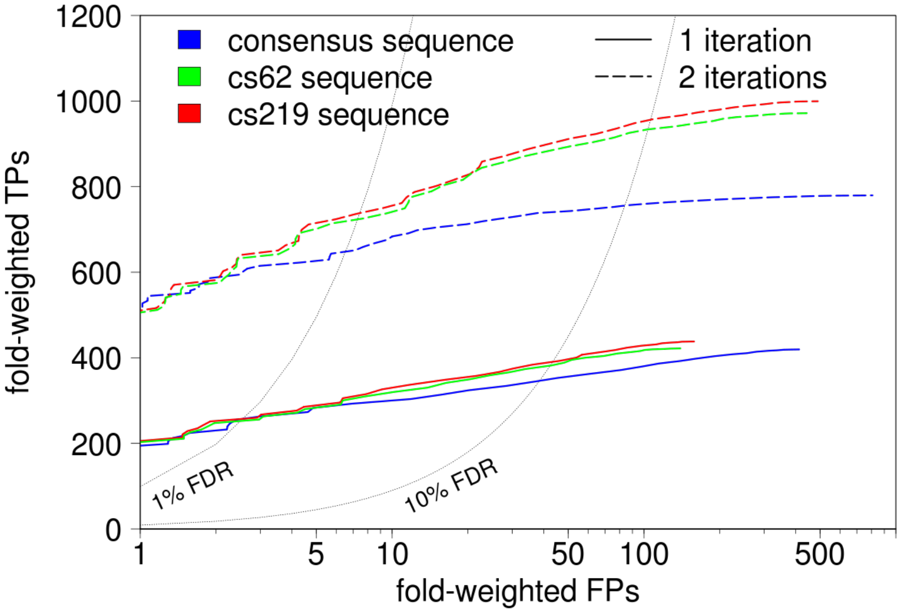


Figure 4.13.: Improvement of HHblits performance when using a column state database for prefiltering. Column states improve the sensitivity of 1 and 2 iterations HHblits compared to the prefiltering based on consensus sequences (blue). Extending the number of column states from 62 (green) to 219 (red) states gains a slightly further improvement.

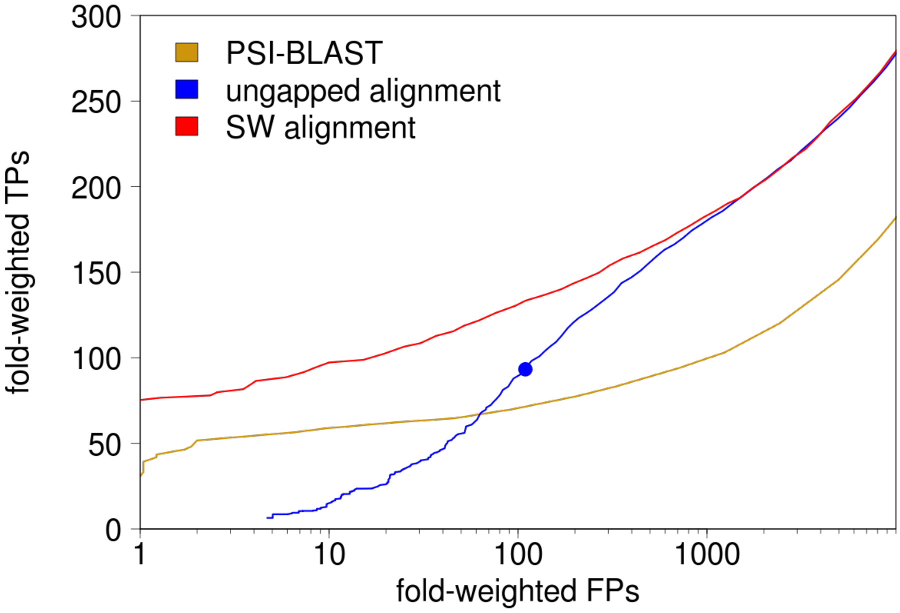


Figure 4.14.: Performance of different prefilter algorithms. The Smith-Waterman algorithm (red curve) shows a significant improvement over the heuristic method PSI-BLAST (orange). The ungapped alignment approach (blue) has a poor sensitivity for the best soring FPs, but for higher numbers of FPs it reaches the same sensitivity as the SW algorithm. The blue dot shows roughly the point of the score threshold for the ungapped alignment filter.

benchmark on the SCOP20 optimization set with only the prefilter algorithms and create the ROCplot shown in figure 4.14. As expected, the Smith-Waterman algorithm (red curve) shows a significant improvement over the heuristic method PSI-BLAST (orange). The ungapped alignment approach (blue) has a poor sensitivity for the best scoring false positives, but for higher numbers of FPs it reaches the same sensitivity as the SW algorithm. Because of that, the ungapped alignment is a good choice for the first prefilter step, because it is very fast and filters out many FPs, but it is not specific enough. This specificity comes with the second filter step with the SW algorithm.

4.1.11. Parameter optimization

Our iterative HMM-HMM comparison method HHblits consists of various parameters which need to be optimized for an optimal performance. In the following, we describe the choice of some of these parameters. The optimizations are performed on the small SCOP20 *optimization* set (1329 protein domains) that is distinct from the comprehensive test set (5287 domains) with no optimization set member sharing a common fold with any test set member.

HHblits contains several adjustable parameters: an inclusion E-value threshold e that defines up to which E-value matches will be merged to the query profile for the next iteration, parameters for the pseudocount admixture (*pca* and *pcb*), and an threshold *neffmax* for skipping further iterations of HHblits when the evolved query profile HMM has a diversity (Neff, see appendix A.2) above this threshold. These parameters are optimized on the SCOP20 optimization set with 1329 protein domains. For a given parameter configuration, we perform an all-against-all comparison of the protein domains and count the true positive (TP) and false positive (FP) hits at various E-value thresholds. Based of these data a ROC5 score ($\in [0,1]$) could be calculated for each query (see section 4.1.1). The mean ROC5 score conveniently captures the homology detection performance in a single value and is therefore used as performance index in the parameter optimization. Optimization results by mean ROC5 scores proved to be more robust and less noisy than results obtained by optimizing the overall sensitivity at a given FDR rate.

An optimization script tries to optimize each parameter by generating the ROC5 scores for different parameter values. If more than one parameter corresponds to a similar property of HHblits, the values of these parameters may influence each other, e. g., the parameters *pca* and *pcb* for the pseudocount admixture. In such cases, the parameter values are optimized in turns in a circular order. The optimization runs reveal that parameter settings $e = 0.001$, $pca = 1$, $pcb = 1.5$, and $neffmax = 10$ gives the best mean ROC5 values. The default values of the most important parameters can be found in the userguide (appendix D.6.2), a full list can be retrieved by starting HHblits on the command line with the parameter "*-h all*".

A number of parameters increases the performance of HHblits with increasing parameter values, but on the other hand increases the runtime. Examples are all prefilter thresholds. These parameters are optimized by finding a good balance between sensitivity and runtime.

4.1.12. Things that did not work

In some cases our ideas to improve the performance of HHblits were not successful. We tried to replace the KALIGN alignment software, which is used to generate alignments for the clusters of our database, with a new extension of the CLUSTAL alignment software (Higgins and Sharp, 1988), called CLUSTAL OMEGA (Sievers et al., 2011). But this replacement gave no performance improvement.

Another idea was to improve the alignment quality by using a higher secondary structure score weight in the MAC realignment. The realignment is only performed for the best hits of the Viterbi search and so we would expect mainly homologous sequences in this step. We thought that the alignment quality of two homologous sequences could be improved by increasing the score of aligning secondary structure elements. But also in this case we did not see any significant improvement with respect to the default case.

Lucia Puchbauer has tried in her bachelor thesis to improve the performance of HHblits by using network-based information in the database. The idea is that if a query protein has many highly ranked matches to related protein domains, i. e., belonging to a single family in SCOP, this gives support to the predicted homology of the query protein with this family and makes it less likely that all these matches are chance hits. A few methods exists that use the similarity among the database proteins to exploit the idea that similar matches support each other (Melvin et al., 2009; Noble et al., 2005; Weston et al., 2006), but they have some drawbacks, i. e., they cannot produce statistical significance estimates such as E-values. Our implementation of this idea calculates a new network-based E-value based on a transitive scoring algorithm and re-rank all HHblits matches by this new E-value - for more details see the bachelor thesis of Lucia Puchbauer. But unfortunately we obtained only a slightly increased sensitivity in HHblits and because of the needed additional precalculations of the database this method is not included in the standard HHblits method.

4.2. Improved PSIPRED secondary structure prediction with HHblits profiles

Multiple sequence alignments are the basis for most bioinformatic analyses of proteins, such as the prediction of secondary structure, solvent accessibility, transmembrane regions or protein-protein interactions. Furthermore, these multiple sequence alignments can be used to generate a 3-dimensional structure models by homology modeling. Important for all these predictions and analyses is the quality of the sequence alignment used.

In this section, we compare the quality of alignments generated by HHblits with alignments built by the state-of-the-art method PSI-BLAST. As prediction method we use PSIPRED (Jones, 1999), the most widely used tool for the prediction of secondary structure. PSIPRED uses a two-stage neural network to predict protein secondary structure based on the profile information from the given alignment. Even though this method is more than 10 years old, there has been no significant improvements in the prediction of secondary structure in the last years (Aydin et al., 2011). One possible way to improve these predictions might be achieved by increasing the quality of the input alignments.

We benchmarked the performance of PSIPRED with different input alignments generated

Table 4.2.: Quality of secondary structure prediction with PSIPRED on different input alignments. The best results are achieved with HHblits alignments filtered to a diversity of 7.

Input alignment	SOV	Q3
2 iterations PSI-BLAST	74.64%	77.31%
3 iterations PSI-BLAST (PSIPRED default)	77.52%	80.38%
1 iteration HHblits	77.87%	80.71%
2 iterations HHblits	78.31%	80.99%
3 iterations HHblits	78.12%	80.83%
HHblits diversity 4	77.42%	80.20%
HHblits diversity 5	78.26%	81.05%
HHblits diversity 6	78.51%	81.29%
HHblits diversity 7	**78.62%**	**81.31%**
HHblits diversity 8	78.49%	81.17%

by PSI-BLAST and HHblits. The benchmark dataset is a selection of representative PDB chains generated with PDBselect (Griep and Hobohm, 2010) 2007. It contains 3649 sequences ranging from 30 to 1040 amino acids. Based on this dataset, we generated different sets of input alignments for PSIPRED: 2 and 3 iterations PSI-BLAST, the latter one is the default case of PSIPRED, and 1, 2 and 3 iterations of HHblits. In addition, we created 5 sets of HHblits alignments with a fixed diversity (effective number of sequences, see appendix A.2) by filtering the HHblits alignments with the HHfilter method using the option *-neff*. For all alignments, we run PSIPRED with the default parameters and calculate the SOV and Q3 scores. The SOV (Segment OVerlap) score (defined by Rost et al. (1994)) measures the overlap between the observed and the predicted secondary structure segments. The Q3 score gives the overall per-residue identity in the three predicted states (helix, strand, loop).

The results of this benchmark are shown in Table 4.2. Alignments built with 3 iterations of PSI-BLAST (PSIPRED default case) achieve a SOV score of 77.52% and a Q3 score of 80.38%. Even with alignments built with one iteration of HHblits we obtain a better performance of PSIPRED (SOV 77.87%, Q3 80.71%) and with 2 iterations we see a clear improvement with a SOV score of 78.31% and a Q3 score of 80.99%. The best performance can be achieved by filtering the HHblits alignments to an effective number of sequences of 7. In this case, the performance of PSIPRED can be increased by more than 1% up to a SOV score of 78.62% and a Q3 score of 81.31% by simply using better input alignments. One has to keep in mind that the neural network of PSIPRED is trained with alignments generated by PSI-BLAST. Hence there might be further space for improvements by retraining this network with HHblits alignments. We have integrated the prediction of secondary structure in the perl-script ADDSS.pl. This script filters the HHblits alignments to a diversity of 7, predicts the secondary structure with PSIPRED and writes this prediction as an annotation line into the A3M-formatted alignment file.

4.3. Structure prediction for selected Pfam domains

Pfam is a database of protein domain families with multiple sequence alignments and HMMER3 profiles for each family (Sonnhammer et al., 1997). It can be used to identify possible domains of new protein sequences. The Pfam database consists of two parts: PfamA is a manually annotated database containing sequences from the UniProt database. For each PfamA family there exists a *seed* alignment, which is a small set of representative sequences for this family that was manually created. PfamB consists of automatically generated families with no annotation or literature reference. Some PfamA families are grouped together into *clans*. A clan is a collection of protein families for which there is bioinformatic evidence that they have a common evolutionary origin. Clans are built by a similarity in tertiary structure or, in the absence of a structure, by a related function, by a significant match of the same sequence to HMMs from different families, or by profile-profile comparisons such as PRC or HHsearch (Finn et al., 2006).

The actual version of Pfam (version 24.0) contains 11913 families in the PfamA database. For nearly half of them (6785) no tertiary structure is known. If we further exclude all families which are part of a clan with a known 3D structure, we end up with 5716 families, for which the tertiary structure cannot be inferred by homology. The identification of homologous structures and the generation of comparative 3D models for these families would allow biologist to elucidate the function of these protein families.

We used HHblits and HMMER3 to scan the PDB70 database (PDB from August 24, 2010, filtered to 70% maximum pairwise sequence identity) with these 5716 protein families and to identify new homologies. In the case of HHblits we used the seed-alignments of the Pfam families and created profile HMMs with 2 iterations of HHblits. These profile HMMs are then used for the final search against the PDB70 database. For HMMER3, we used the provided HMMER3 profiles from the Pfam FTP-server and performed a final search against the PDB70 sequence database. The results when using an E-value cutoff of 10^{-3} are shown in figure 4.15. HHblits is able to identify new homologies for 621 Pfam families ($\sim$ 11%). For 227 families out of these, HMMER3 is also able to find a homolog, but for the other 394 families (see list in appendix C) HMMER3 could not identify a homolog protein structure with an E-value better than 10^{-3} and for 176 families out of these HMMER3 has no matched protein structure up to an E-value of 10. On the other hand HMMER3 could identify homologies for 44 Pfam families with an E-value better than 10^{-3} where HHblits has no matches up to this E-value cutoff. In the following subsections we present the details of three cases where HMMER3 has no matches and HHblits identifies a clear homology and a structural model could be generated.

4.3.1. CTK3 - a possible new CID

The synthesis of new proteins starts with the transcription of the genomes to mRNA (messenger RNA) by RNA polymerase (pol) II. Pol II contains a special C-terminal domain (CTD) which consists of up to 52 repeats of the sequence `YSPTSPS` (Tyr1-Ser2-Pro3-Thr4-

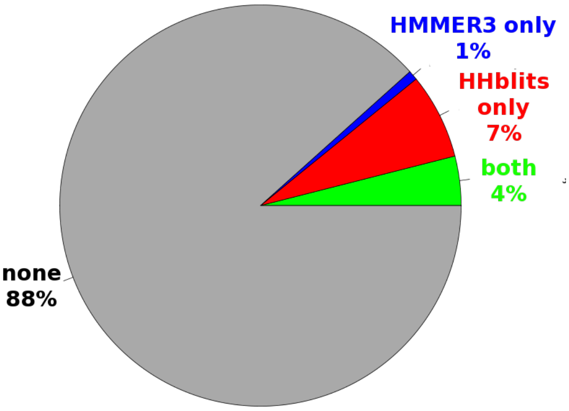


Figure 4.15: Identification of new homologous structures for Pfam families by HHblits and HMMER3. For 5716 families in the PfamA (version 24.0) there exists no structure information. Using an E-value cutoff of 10^{-3} HHblits is able to identify a homologous structure in about 11% of these cases. In two third of the families with HHblits matches, HMMER3 could not find a homologous structure. On the other hand, in a further 1% HMMER3 detects a homology where HHblits has no matches up to the E-value threshold.

Ser5-Pro6-Ser7) (Corden, 1990). Ser2, Ser5, and Ser7 are dynamically phosphorylated and dephosphorylated during the transcription cycle by special kinases. This phosphorylation pattern controls the co-transcriptional assembly of different transcription initiation, elongation and termination factors, as well as RNA processing factors. The different CTD phosphorylation modifications can be recognized by special peptide binding domains, the so called CTD-interacting domains (CIDs). Currently, three proteins containing a CID are known in yeast: (1) *Nrd1* preferentially binds to CTD phosphorylated at Ser5 (Vasiljeva et al., 2008), but a strong correlation between Nrd1 and CTD phosphorylated at Ser7 was also found (Kim et al., 2010). (2) *Pcf11* and (3) *Rtt103* both binds to Ser2-phosphorylated CTDs (Kim et al., 2010; Meinhart and Cramer, 2004), but they show differences in the affinities for different phosphorylated diheptad repeat CTD peptides (Lunde et al., 2010).

The main CTD Ser2 kinase in yeast is *Ctk1* (Jones et al., 2004). This kinase is part of the protein kinase complex *CTDK-I*, which is composed of three subunits of 58, 28, and 32 kDa (Hautbergue and Goguel, 2001; Sterner et al., 1995). *Ctk1* is the largest subunit of this complex and it is important in vivo for normal growth and differentiation. *Ctk1* deletion results in an increase in phosphorylation of CTD Ser5 and eliminates the transient increase in CTD Ser2 (Hautbergue and Goguel, 2001). *Ctk2* is a cyclin-related protein with homology to cyclin C. However, *Ctk3* does not exhibit any similarity to other proteins. It is an unstable protein with a relative short half-life (30min) (Hautbergue and Goguel, 2001), which is processed through a ubiquitin-proteasome pathway. But both *Ctk2* and *Ctk3* are required for *Ctk1* kinase activation.

PSI-BLAST and HMMER3 are not able to identify any proteins homologous to *Ctk3*, but in an HHblits search the best three matches are the other known CIDs *Rtt103*, *Pcf11*, and *Nrd1* with probabilities of 98%, 94%, and 93%, respectively. The matched alignment covers the whole CID, whereat the alignment confidence values (see section 3.6 and appendix B) for the last helix are relative small. Figure 4.16 shows the structures of the three known CIDs *Rtt103* (PDB-ID 2L0I), *Pcf11* (PDB-ID 1SZ9), *Nrd1* (PDB-ID 3CLJ) and the structural model of *Ctk3*. *Rtt103* is shown with a Ser2 phosphorylated CTD peptide (magenta). The *Ctk3* model was generated with the MODELLER software (Sali and Blundell, 1993) based on the alignment between *Ctk3* and the known CIDs given by HHblits. It is colored according to the posterior probabilities of each aligned column in the HHblits alignment with *Rtt103* where the color indicates the alignment quality from red (highly confidence) to blue (low confidence). We see a high conservation in the first five helices of this domain, whereas the model of *Ctk3* has a insertion between the first and the second helix with a length of approximately 20 residues (blue dashed line). This homology on the sequence level supports a possible CTD-interaction function of *Ctk3*, which would also make sense in the biological context. The *CTDK-I* complex with *Ctk3* must be located near the RNA pol-II CTD, because the largest subunit of this complex phosphorylates the Ser2 in the CTD repeats. The CTD shows a high Ser5 and Ser7 phosphorylation near the transcription start site

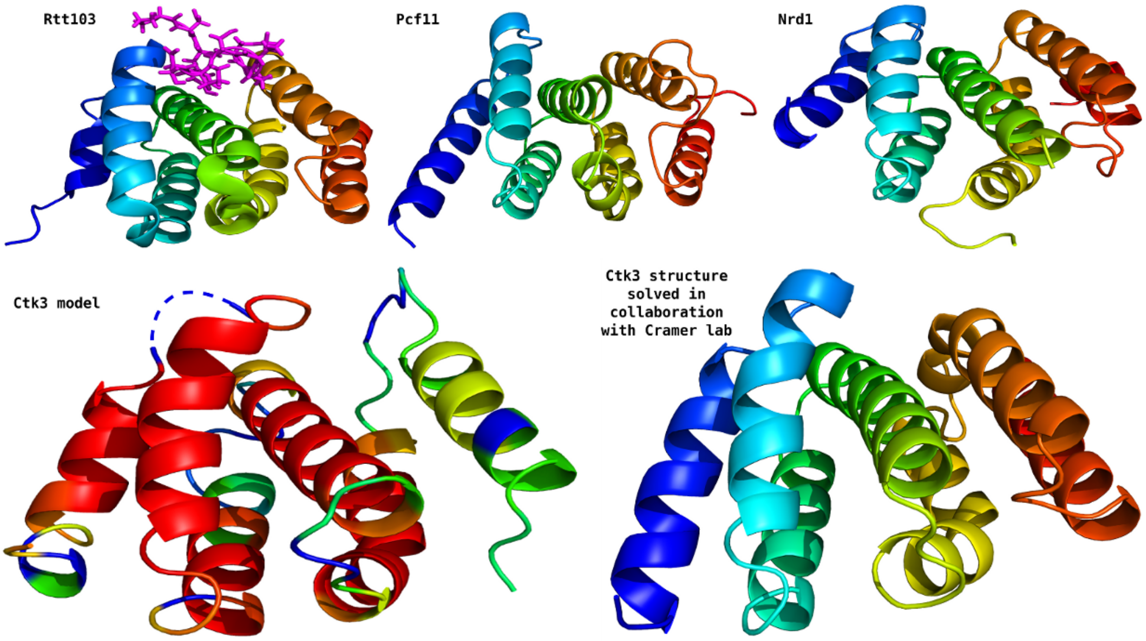


Figure 4.16.: Structures of the known CTD-interacting domains (CIDs) *Rtt103* (PDB-ID 2L0I), *Pcf11* (PDB-ID 1SZ9), and *Nrd1* (PDB-ID 3CLJ), a structural model of *Ctk3*, whose structure and function was unknown so far, and the newly solved structure of *Ctk3*. *Rtt103* is shown with a Ser2 phosphorylated CTD peptide (magenta). The *Ctk3* model is colored according to the alignment posterior probabilities given by the HHblits matches to *Rtt103* from red (highly confidence) to blue (low confidence).

and with decreasing values of these phosphorylations towards the polyA-site the number of Ser2 phosphorylations increase (Mayer et al., 2010). It is hence possible that the *CTDK-I* complex with *Ctk3* binds to CTD phosphorylated at Ser5 or Ser7 and triggers the Ser2 phosphorylations further downstream.

Based on our prediction and the model for *Ctk3*, the structure for *Ctk3* from *Saccharomyces Pombe* was solved by Wolfgang Mühlbacher and Andreas Mayer from the Cramer group here at the Gene Center Munich. This new structure is also given in figure 4.16. As predicted, it shows a high similarity to the known CTD-interacting domains and represents the fourth classical CID in yeast. At the moment, further binding studies are performed to identify a possible target peptide for *Ctk3* within the C-terminal domain.

4.3.2. PIP49 - a possible Ca^{2+}-activated protein kinase

PIP49 (Pancreatitis Induced Protein 49) is a putative transmembrane protein which is activated in pancreas in response to acute pancreatitis (Samir et al., 2000). In 2000, Samir et al. computationally predicted a transmembrane helix near the N-terminus (Y31 to Y53) and a weak relationship to the gastric inhibitory polypeptide receptor precursor (GIP-R) using PROPSEARCH (Samir et al., 2000). A signal peptide or other homologies are not known and neither PSI-BLAST nor HMMER3 could identify any match in the PDB database.

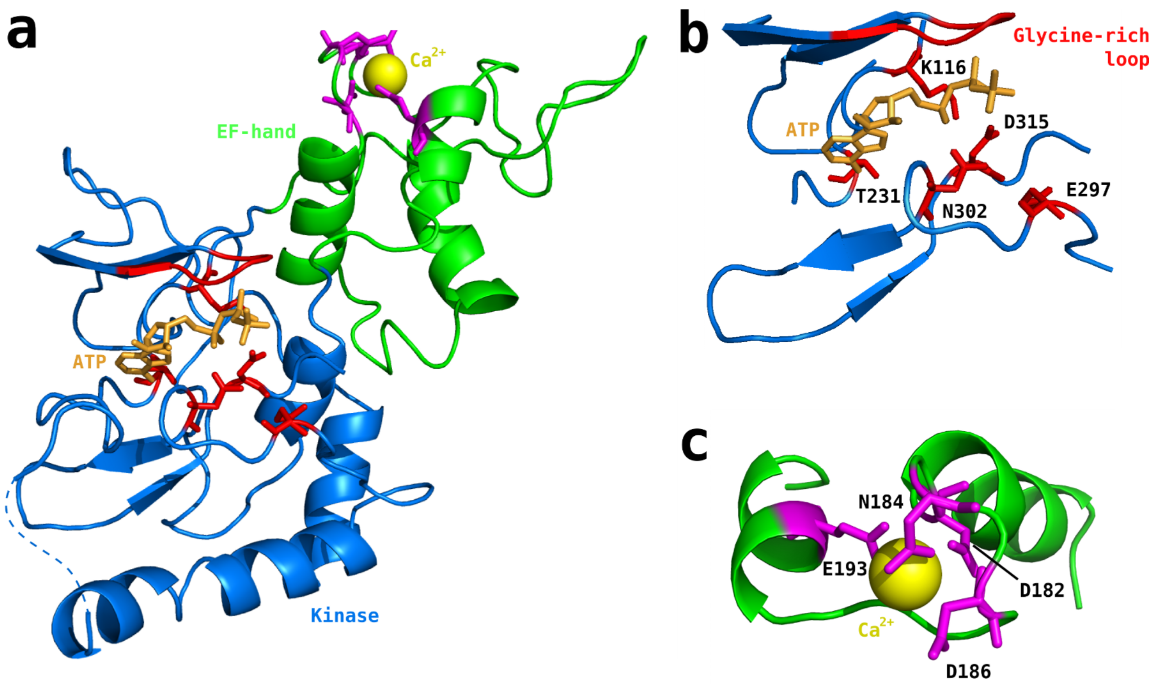


Figure 4.17.: (a) Homology model of the previously unknown human *PIP49/FAM69B* kinase domain (blue) with inserted EF hand (green). In the catalytic center of the protein kinase domain inportant conserved residues are highlighted in red and a bound ATP-molecule is shown in orange. The conserved residues in the EF-hand, which are important for Ca^{2+}-binding, are highlighted in magenta. These two regions are given with more detail in (b) and (c), respectively.

We started our search with the human *PIP49/FAM69B* (UniProt -ID: Q5VUD6), because the Pfam MSA is missing the N-terminal part. The 100 best HHblits matches in the PDB database are all with protein kinases with high confidence (probability $> 99\%$, E-value $< e^{-14}$) with HHblits. Interestingly, a tandem Ca^{2+}-binding EF hand could be identified (E-value 0.09), which is inserted after the small N-terminal β sheet of the kinase domain. Although many protein kinases contain EF hands downstream of their kinase domain (Finn et al., 2010), *PIP49/FAM69B* is the first case where an EF hand is inserted within the kinase domain.

From the list of PDB matches we chose a protein kinase with bound ATP (PDB-ID: 1RDQ) and a Ca^{2}-bound EF hand (PDB-ID: 3C1V) as templates and used the corresponding HHblits alignments to create a homology model with MODELLER. Figure 4.17a shows the resulting homology model with the protein kinase domain in blue, the inserted EF hand in green, and the modelled ATP-molecule in orange. The active center of the kinase domain with the bound ATP-molecule is accentuated in 4.17b with the critical residues for the kinase function highlighted in red. A high conservation of these residues can be seen in the MSA of *PIP49/FAM69B* (see figure 4.18). In the histogram view of the MSA, the catalytic center of the kinase domain is highlighted by the blue bars, and the red stars indicate the critical residues according to Wang et al. (2006) such as the catalytic lysine at position

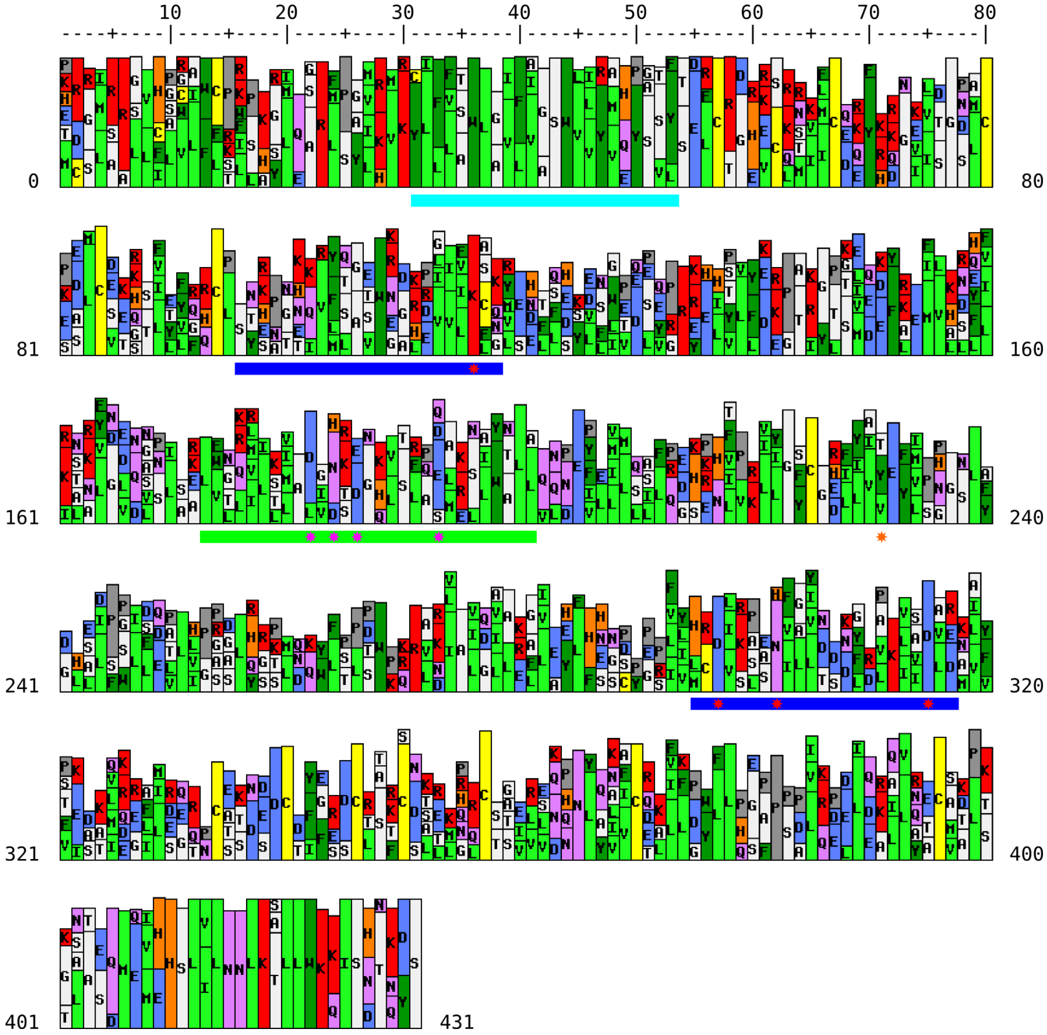


Figure 4.18.: Multiple sequence alignment of human *PIP49/FAM69B* shown as histogram view. Highlighted is the catalytic center of the protein kinase domain (blue bars) with the conserved important residues (red stars). The green bar indicates the EF hand, where the magenta stars identify the important residues for the Ca^{2+}-binding. The orange star at position 231 gives the corresponding position to the gatekeeper in other kinases. Thy cyan bar shows the predicted transmembrane helix.

116 (K116) that coordinates the α- and β-phosphates of ATP, and E297, N302 and D315 in the catalytic loop. Another important residue in the kinase domain is the so-called gatekeeper (T321), which controls access to a pre-existing internal hydrophobic pocket at the back of the ATP-binding site. Nearly all kinases in the human kinome have a tyrosin or threonine residue as a gatekeeper (Wang et al., 2006). The corresponding position in the *PIP49/FAM69B* MSA is indicated by the orange star at position 231 in figure 4.18 with a high probability for a tyrosin or threonine residue. Figure 4.17c shows the modelled EF hand of *PIP49/FAM69B*. The EF-hand motif contains a helix-loop-helix topology, in which the short loop region (about 12 amino acids) usually binds calcium ions. The critical residues for the Ca^{2+}-binding are highlighted in magenta (D182, N184, D186 and E193).

These residues also show a high conservation in the MSA of *PIP49/FAM69B* (magenta stars in figure 4.18).

Furthermore, the kinase domain is framed by two short domains with five or more highly conserved cysteines each that are likely to form disulfide bonds (see conserved Cys-columns in figure 4.18). We could confirm the prediction of the transmembrane α-helix near the N-terminus (cyan bar in figure 4.18) by different transmembrane prediction methods such as HMMTOP, MEMSAT-SVM and PHOBIUS, but a relationship to GIP-R could not be found and is highly unlikely, since GIP-R consists of 7 transmembrane helices which would contradict our predictions. Based on our homology models and the conservation of critical residues, we predict that *PIP49/FAM69B* and *FAM69A* are ER membrane bound protein kinases in the ER lumen that are activated by Ca^{2+} through structural rearrangement of their EF hand. Residue conservation suggests that metazoan *FAM69C* will also possess protein kinase activity.

4.3.3. The HAP complex

The HAP complex (HAT (Histone AcetylTransferases) Associated Protein) is composed of three subunits *Hap1*, *Hap2* and *Hap3* (also named *Elp4*, *Elp5* and *Elp6*, respectively) (Li et al., 2001). These *HAP* proteins can be isolated as a complex of the three subunits and as a six-subunit complex with *Elongator*. The *Elongator* is a three-subunit protein complex containing HAT activity and has been found associated with elongating forms of RNA polymerase II (Wittschieben et al., 1999). The role of the HAP complex is unknown. It might allow histone acetylation only in the presence of transcribing polymerase, keep the HAT activity of free *Elongator* in check, or it might modulate *Elongator* activity through physical interaction.

With HHblits we could identify a homologous relationship between all three *HAP* proteins and the family of *RecA*-like motor ATPases (probabilities of 99%, 96% and 95%, respectively). Neither PSI-BLAST nor HMMER3 could identify any homologous structures. One of the best HHblits matches in all searches is the circadian clock protein *KaiC* which consists of two *RecA*-like domains containing shared regions including Walker A (P loop) and B motifs involved in ATP-binding and hydrolysis. *HAP1* and *HAP3* are completely covered by the alignments to *RecA*-like domains, *HAP2* has an additional C-terminal domain with ~ 130 amino acids. For this additional domain we couldn't predict any homologous structure.

Figure 4.19 shows the three modeled structures of the Hap proteins and one domain of *KaiC* with the Walker A and B motif highlighted in magenta and purple, respectively, and an ATP molecule (gray) in the ATP-binding pocket. The models are created with MODELLER based on the alignment with *KaiC* given by HHblits. They are colored according to the posterior probabilities of each aligned column in the HHblits alignments where the color indicates the alignment quality from red (highly confidence) to blue (low confidence). Especially the model of *Elp6* (*Hap3*) shows a high confidence in the alignment of the ATP-

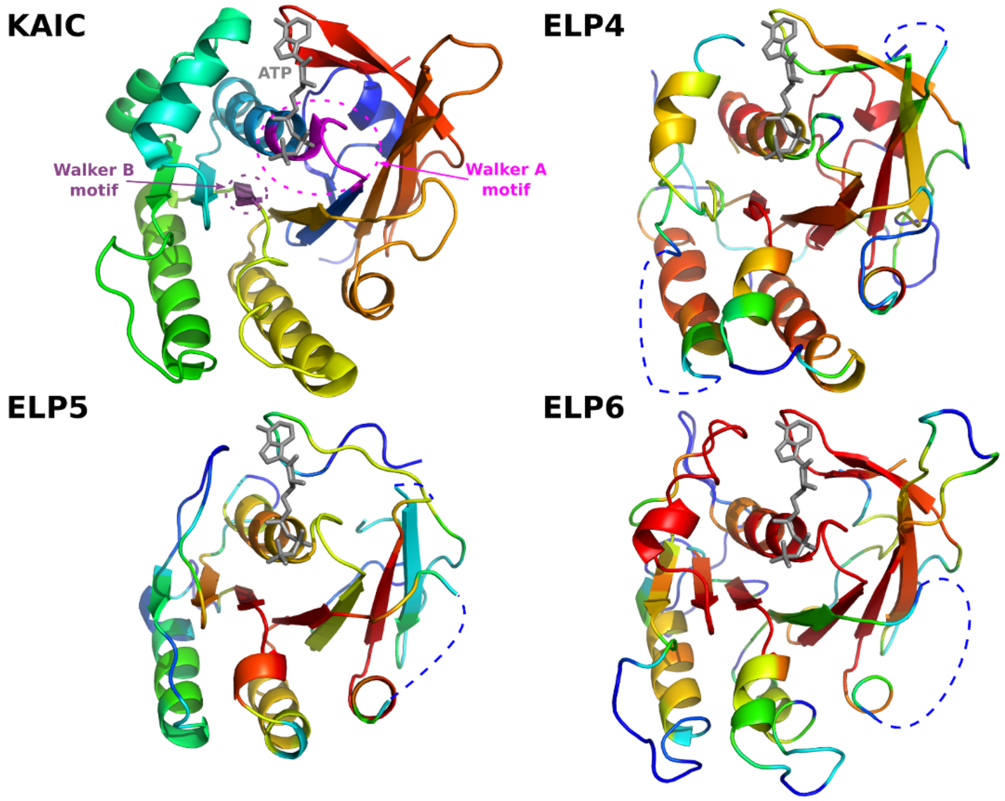


Figure 4.19.: Structure of the circadian clock protein *KaiC* and structural models of the three *HAP*-complex proteins whose structure and function is unknown until now. All structures are shown with a bound ATP-molecule (gray structure). In the structure of *KaiC*, the Walker A (P loop, magenta) and Walker B (purple) motif are highlighted. The models are colored according to the alignment posterior probabilities given by their HHblits matches to *KaiC* from red (highly confidence) to blue (low confidence).

binding pocket with the Walker A and B motifs which might indicate an ATPase activity. On the sequence level we see a high conservation in the Walker B motif (ΦΦΦD, of which Φ is a hydrophobic residue) in all three *HAP* proteins, only in *Elp4* the aspartic acid is not well conserved. But the Walker A motif (GXXXXGK(T/S)) with the important lysine (K) residue, which is crucial for nucleotide-binding, is not conserved in the *HAP* proteins. Even though all *HAP* proteins show a clear homology to the RecA-like ATPases, an active ATPase function of these proteins is uncertain.

HHblits predicts new folds for some hundred Pfam families where neither PSI-BLAST nor HMMER3 found any relationship in the PDB database. As in the examples shown above, homology models could be generated for many matches and first assumptions about possible functions could be drawn. The next step would be the experimental validation of the prediction in the laboratory by e.g. measures of ATPase activity or binding studies, as it is done in the case of *Ctk3* in cooperation with Wolfgang Mühlbacher and Andreas Mayer. Therefore, HHblits is a powerful method to get an idea of the possible structure and function of a new protein and to suggest validation experiments.

4.4. Web server

We have integrated the HHblits software into the Bioinformatics Toolkit (Biegert et al., 2006), our webserver with various bioinformatic tools. It can be accessed at `http://toolkit.lmb.uni-muenchen.de` or at `http://toolkit.tuebingen.mpg.de`. This Toolkit provides a user-friendly platform to work with our methods developed in-house and many of the state-of-the-art tools that are freely available. The users can manage their submitted jobs on a sidebar on the left that holds a status and color-coded list of all recent jobs in the current session. By clicking on previously submitted jobs their status can be checked and their results can be viewed. Most of the tools are interconnected, allowing job results of one tool to be forwarded as input to others. The jobs are performed on a large compute cluster with more than 20 compute nodes.

Figure 4.20 shows some screenshots of the HHblits method. On top, the input page with the possibility to paste an input sequence or multiple sequence alignment is shown. Alternatively, the user can upload a file with the input data. The most important parameters such as the number of iterations or the used database can be choosen directly on the input page, further parameters can be modified by pressing the “Show” link in the “more options” section. All parameters are directly linked to online help pages which describe the parameters and the result pages in more detail. By pressing the “Submit” button, the input data is processed and the HHblits method is run on the compute cluster. After a waiting screen showing log messages, the user is forwarded to the result page (bottom half of figure 4.20). The output is organized into three sections: (1) A bar graph summarizing the positions and color-coded significances of the database matches with more than 40% probability. (2) A tabular hit list with probabilities, E-values, scores, and match regions in query and templates. (3) The pairwise query-template alignments with annotation for secondary structure (if available), consensus sequence and column-column match quality. In addition, a profile histogram view is provided, where the pairwise query-template alignments of the profile HMMs are depicted as histograms over the amino acid distributions in the columns of the underlying multiple alignments. In the “Show alignment” section, the user retrieves the generated multiple sequence alignment, can save it or forward it to start another tool in our Toolkit with this alignment. More details of the webserver results pages are given in the online help pages and in the userguide (see appendix D).

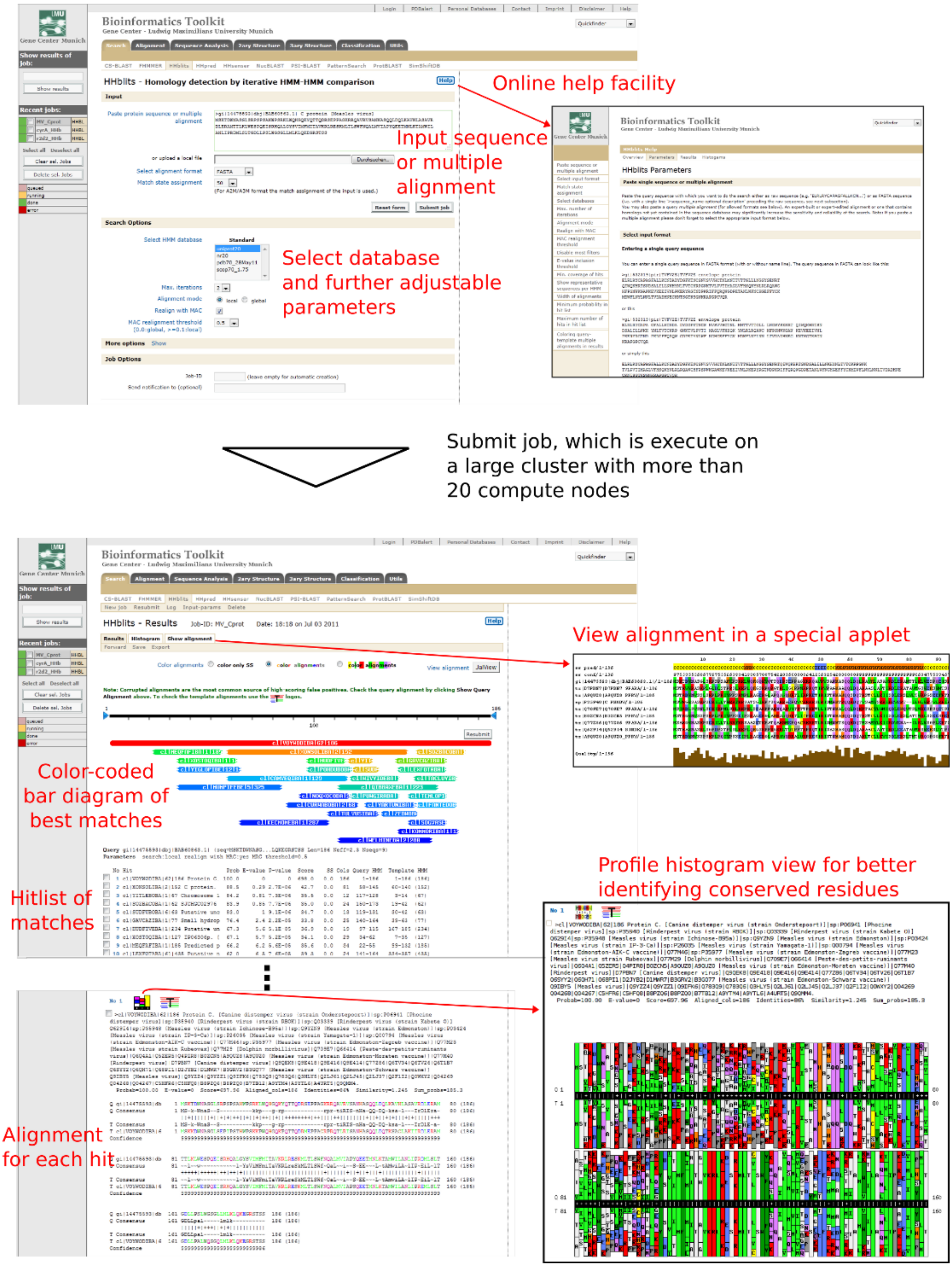

Online help facility
Input sequence or multiple alignment
Select database and further adjustable parameters
Submit job, which is execute on a large cluster with more than 20 compute nodes
View alignment in a special applet
Color-coded bar diagram of best matches
Hitlist of matches
Profile histogram view for better identifying conserved residues
Alignment for each hit

Figure 4.20.: Screenshots of the HHblits web server with the input page and the corresponding result pages.

5. Conclusion and Outlook

Most sequence-based methods for protein structure or function prediction construct a multiple sequence alignment of homologs as a first step. The best methods for constructing such alignments use an iterative search scheme, where significant matches are added to the evolving multiple alignment. The standard iterative search method is PSI-BLAST, an extension of BLAST to profile-sequence comparison. An increase in sensitivity could be gained by HMMER3 using profile HMMs on the query side, but this gain is paid by a three- to four-fold reduction in speed.

We have developed HHblits, the first iterative sequence searching method based on the pairwise comparison of HMMs. HHblits builds on the HHsearch algorithm for pairwise HMM-HMM comparison, to which it owes its high sensitivity and alignment quality. We achieve a runtime similar to PSI-BLAST by two steps: Firstly, we cluster the database, which covers the whole sequence space, with our method kClust down to 20% maximum pairwise sequence identity. This reduces the number of database entries in the case of the UniProt database from over 14 million down to 2.6 million entries, which are represented as profile HMMs in our HHblits database. Secondly, we perform different fast filter steps to reduce the amount of time-consuming HMM-HMM comparisons to a minimum.

We must reduce the information contained in our database profile HMMs into a single sequence in order to apply very fast sequence comparison methods in the prefilter step. When representing a sequence profile as a consensus sequence, a lot of information is lost, because diverse profile columns cannot adequately be modeled by one single amino acid. We use an approach that encodes profile columns by a special column state alphabet over 219 states. Each column state is associated with a characteristic profile vector and because of that the evolutionary information in diverse profile columns can be better represented. In addition to the column state alphabet, which increases the sensitivity of the prefilter algorithms, we use “single instruction multiple data“ (SIMD) instructions to increase the speed of our prefilter algorithms. With these instructions, several operations can be calculated in parallel. Our prefilter algorithms - including a gapless alignment and a Smith-Waterman algorithm - run on the whole database within seconds. Several further filter steps reduce the time spent on the slow HMM-HMM comparisons to a minimum.

We demonstrate the performance of HHblits not only on different benchmark sets, but we also show the quality in different biological cases. Most of our benchmarks were performed on a SCOP20 dataset which is completely disjunct from the set used for the parameter optimization. HHblits shows a significant increase in sensitivity, i. e., it finds nearly twice as many true positives as PSI-BLAST and over 50% more than HMMER3 in the first iteration. Besides the sensitivity in identifying homologs, the alignment quality is also drastically improved. Based on a gold standard set with structural alignments HHblits has an increased precision and sensitivity compared to the other methods, i. e., more than 20% in each case compared to PSI-BLAST. The high alignment quality is also demonstrated by an increased PSIPRED performance by using HHblits alignments as input compared to the default PSI-BLAST alignments. The HHblits performance shown in these benchmarks is achieved by a runtime slightly better than PSI-BLAST and by a factor 3-5 better than HMMER3.

On the Pfam domain database we could show that HHblits is able to make confident (E-value $< 10^{-3}$) fold predictions and to build structural models for some hundred domain families for which no fold prediction has been possible (see list in appendix C). We demonstrate with three example cases, how the HHblits results can be used for homology modeling and for drawing first assumptions about possible functions of this domain family. These are the basis for focussing the experimental investigations. Therefore, HHblits is a very good method to get a first idea of the possible structure and function of a new protein and to make a detailed plan of the further work in the laboratory.

In summary, we have shown that the iterative HMM-HMM comparison method HHblits has the potential to replace the state-of-the-art methods PSI-BLAST and HMMER3 in the field of homology searching and multiple sequence alignment generation. Our new method has a similar runtime compared to PSI-BLAST by using a fast prefilter based on profile-profile comparison. Furthermore, HHblits greatly improves upon PSI-BLAST and HMMER3 in terms of sensitivity/selectivity and alignment quality. The performance of HHblits decreases when using HMMER3 databases, the best results will be achieved by using the provided HHblits databases (scripts for the generation of own databases are provided). HHblits is integrated in our automatic protein structure and function prediction server HHpred, which was assessed to be one of the best out of the 81 servers in the last (2010) blind protein structure prediction competition CASP (Critical Assessment of Techniques for Protein Structure Prediction)[1].

Outlook

HHblits is very powerful sequence searching and homology detection method and it is able to replace PSI-BLAST with a similar runtime and a clearly better performance. We already have some ideas to further improve the performance of HHblits.

Integrating taxnonomic information

Lorenz Maier tries in his diploma thesis to improve the performance of HHblits by the additional use of taxonomic information. Homologous proteins share a common ancestor and so one can expect that homologous proteins come to lie together within a taxonomic tree.

---

[1] `http://predictioncenter.org/casp9/groups_analysis.cgi?type=server&tbm=on&submit=Filter`

In other words, matched database sequences whose taxa are distant with respect to the taxa in the query profile are more likely to be false positives than sequences from closely related taxa (Abeln et al., 2007). We integrate this information by calculating a special E-value which does not depend on the full database size but on the weighted number of sequences below the last common ancestor of the query and the database cluster in a taxonomic tree (for more details see diploma thesis of Lorenz Maier). It is intuitively clear that there should be information contained in the taxonomic relationship between proteins, but we could only slightly improve the performance of HHblits. One must say that it is difficult to design a good benchmark to test the improvement of this taxonomy information, because usually the performance is checked on databases such as SCOP, but these domains often comes from well studied protein families. We expect the most improvement in cases where only very little is known about a protein family and only very few members are sequenced by now.

Further optimization of runtime and memory requirements

We plan to further optimize the runtime and memory requirements of HHblits, especially for using the method with multiple CPU cores. At the moment, the merging of HMMs below the inclusion threshold with the evolving query profile HMM is only single-threaded, so this could be optimized by parallelizing the merging procedure. Another option is to further increase the speed of the prefilter step. A recent paper of Rognes (2011) states an increase in speed of a factor of 2-6 of the Smith-Waterman algorithm based of SIMD instructions compared to the Farrar approach implemented in our prefilter. This increased speed is achieved by an inter-task parallelization, where the parallelization is carried out across multiple database sequences. Furthermore, we are planning to optimize the reading of the needed databases into the memory by using memory mapping, in which the operating system manages reading data into memory from disk concurrently with program execution. The memory imprint of the forward-backward algorithm could also be reduced by representing the current and previous rows of the dynamic programming matrices by just two vectors instead of whole matrices. This is primarily important when aligning extremely long sequences where the memory requirements could achieve the dimension of gigabytes.

Optimize homology probabilities

Another idea is to optimize the homology probabilities provided by HHblits and HHsearch. At the moment, the probability is obtained by fitting in a large benchmark set the ratio of true positive (i.e., homologous) pairs to all pairs as a function of the total score (see equation 3.11 in section 3.5). The new idea is to employ a neural network (NN) and use more features than only the score such as the diversities and length of the two HMMs, posterior probabilities or the relative entropy over aligned profile columns. The NN is then trained to distinguish between homologs and false positives, e. g., on a SCOP dataset, and implemented in the C^{++} code. Based on this approach the probabilities should be more meaningful for discriminating between homologous and non-related proteins.

Multi-component mixture score

A further plan is to improve the score of the HMM-HMM comparison. Up to now, the total score is calculated by adding the profile column score and secondary structure score, if secondary structure predictions are available, independently of each other, using constant weights. A better way would be to combine the score from different per-residue features in a nonlinear fashion (Söding and Remmert, 2011). CONTRalign for example models the probability of an alignment as a conditional random field (CRF) (Do et al., 2006). But this method has a fixed score for each combination of features and adds these scores linearly. A further improvement is possible by formulating the score as a nonlinear function of the sequence features as it is done in the RAPTOR servers (Peng and Xu, 2009). We are planning to combine different scores such as profile-profile score, sequence-sequence score, profile-sequence score, secondary structure score, and scores based of the context or tertiary structure angles as a linear mixture with weights depending non-linearly on various inputs. The nonlinear weight functions could be learned by neural networks with inputs which are mostly position-dependent alignment and structure features. Possible input features could be the diversity or the conservation in the HMMs, the residue-residue similarity, or structural features such as a coordination number, which is the number of C-β atoms in a sphere around the C-β atom at a given position, or the sequence conservation in a structural neighborhood. When updating the HMM comparison score we also could check if we get a further increase in performance when using the score generated by the Forward algorithm instead of the score of the Viterbi algorithm.

Abstract state profile comparison

Secondary structure and solvent accessibility have two important characteristics that make it useful for remote homology detection and alignment and hence also 3D struture prediction: (1) The state of a residue can be predicted from local windows of sequence profiles centered around the residue, typically of size 13 or 15 columns. (2) The states evolve slowly in time and hence are relatively similar even between remote homologs.

The idea is to generate an alphabet of abstract states that has the two characteristics above. This alphabet should ideally contain almost all the information from the profile windows that obeys the two characteristics. We could then expect this alphabet to contain secondary structure and solvent accessability information implicitly, among other information. Hence it should be even more powerful for remote homology detection than those two alphabets (secondary structure states and solvent accessibility states) together. For example predicted secondary structure and solvent accessibility do not improve alignment quality a lot since neighboring states are mostly identical. In contrast, we can expect that the register of an amphipathic helix or a beta strand is quite well conserved in evolution and naturally differs between neighboring residues. By using a relatively small number of abstract states (64 or 256) we are able to construct a substitution matrix for the abstract states, just as for amino acids. One can thereby statistically describe the evolutionary behaviour of these states. The main improvement to the case of comparing amino acid distributions is that these abstract states depend on a whole window in the profile, whereas the statistical model of similarity between amino acid distributions is always column-by-column and does not merge together information over several neighboring columns.

Part II.

Evolution of outer membrane β-barrels

6. Introduction to the evolution of outer membrane β-barrels

Outer membrane β-barrel proteins (OMBBs) are the major class of outer membrane proteins (OMPs) from Gram-negative bacteria, mitochondria and plastids. They are composed of a closed, barrel-shaped β-sheet around a central pore. Until this work, a common ancestry for this class with a high structural regularity could not be identified. In the following chapters we use three complementary approaches to show that all OMBBs from Gram-negative bacteria evolved from a single, ancestral $\beta\beta$ hairpin.

6.1. Cell envelope of Gram-negative bacteria

There exists three domains of life, the bacteria, the archaea and the eucarya. Bacteria can be separated into two groups, the Gram-positive and the Gram-negative ones. The differences between these groups lie in the cell envelope, and it is possible to distinguish between these groups by a special staining method, the **Gram staining**. Gram-positive bacteria have a cytoplasmic membrane and a cell wall which contains up to 40 layers of peptidoglycan. Gram-negative bacteria also have a cytoplasmic membrane, but there are only 1-2 layers of peptidoglycan. Additionally these bacteria have an outer membrane which separates the periplasm with the peptidoglycan from the external environment. Because of the outer membrane these cells can not be stained with the Gram method.

The inner membrane

The inner membrane (or cytoplasmic membrane) is a lipid bilayer composed of phospholipids, integral proteins and lipoproteins, which are located at the periplasmatic side of the membrane. The integral proteins have an α-helical transmembrane domain and are mainly appropriate for the transport of molecules. Some transport proteins need ATP (Adenosin-5'-triphosphat) for activity such as in the case of the ABC-transporters (ATP-binging cassette). At the inner membrane biochemical processes like the synthesis of lipid molecules and the oxidative phosphorylation are taking place.

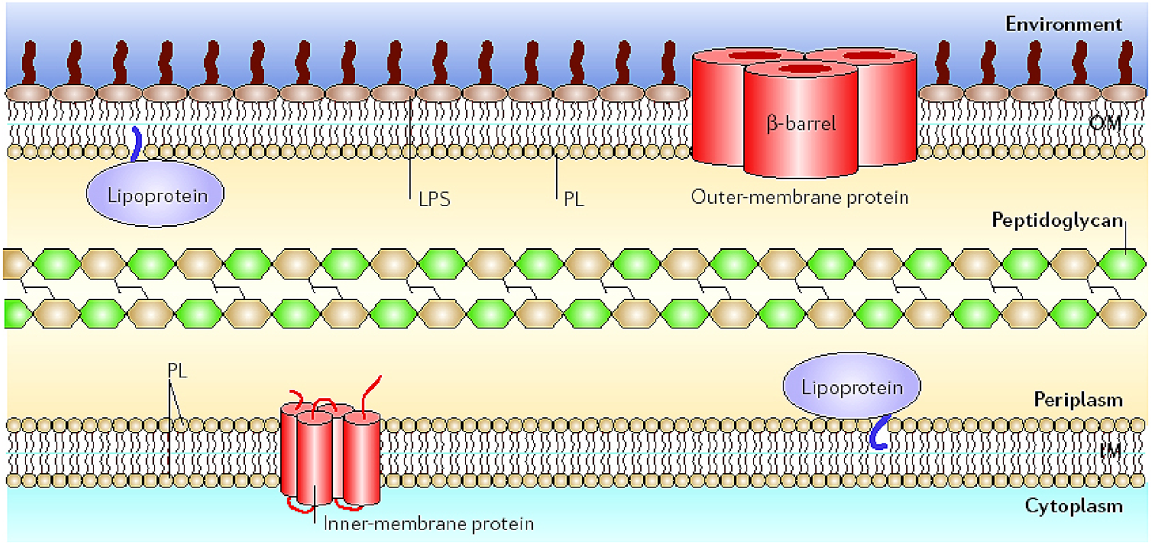


Figure 6.1.: Structure of the cell envelope of Gram-negative bacteria: The cell envelope of Gram-negative bacteria is composed of the inner membrane (IM), the periplasm and the outer membrane (OM). The inner membrane is a lipid bilayer, which contains lipoproteins and integral membrane proteins with a α-helical transmembrane domain. The outer membrane is an asymmetric lipid bilayer composed of lipopolysaccharides (LPS) at the outside and phospholipids (PL) at the periplasmatic side. Integral membrane proteins in the outer membrane typically have β-barrels as their transmembrane domain. The periplasm is the space between these two membranes where the peptidoglycan cell wall is located. (Figure taken from (Ruiz et al., 2006))

The periplasm

The periplasm is the compartment between the two membranes and usually 12-15 nm wide. It contains the peptidoglycan cell wall and soluble proteins like binding proteins which guide molecules to transport proteins into the inner membrane (e.g. ABC-transporters). Another big group of periplasmatic proteins are enzymes which are involved in degradation of nutrients. Because of the absence of ATP all reactions in the periplasm occur without an energy source.

The outer membrane

The outer membrane is an asymmetric lipid bilayer composed of lipopolysaccharides (LPS) at the outer leaflet and phospholipids at the inner. LPS consists of Lipid A, a disaccharide with bounded fatty acids, and a polysaccharide composed of a core-polysaccharide and an O-specific polysaccharide with differing lengths. Lipid A is embedded into the outer membrane while the rest of the LPS projects from the surface. LPS is an endotoxin and induces a strong response from eukaryotic immune systems. The strong lateral interaction between LPS molecules allow LPS to be highly compacted (Takeuchi and Nikaido, 1981), and so the outer membrane acts as a barrier.

Besides integral outer membrane proteins (see 6.2) there are lipoproteins in the outer membrane. Additionally the outer membrane acts as an anchor for surface organelles such as pili that have a crucial role in pathogenesis (Remaut and Waksman, 2004).

6.2. Outer membrane β-barrels

Outer membrane β-barrels (OMBBs) are the major class of outer membrane proteins (OMPs) in Gram-negative bacteria (Koebnik et al., 2000). They also occur in the outer membranes of mitochondria and chloroplasts (Duy et al., 2007; Paschen et al., 2003), according to the endosymbiont theory. Only a single bacterial OMP with an α-helical topology has been described (Wza) (Dong et al., 2006). OMBBs are functionally very diverse, ranging from the transport of molecules to enzymatic and structural functions, and constitute major bacterial antigens. They are synthesized in the cytoplasm and translocated across the inner membrane by the Sec-transport mechanism. In the periplasm the N-terminal signal peptide is cleaved by a signal peptidase and chaperones bind the unfolded OMBBs. The mechanism of insertion into the outer membrane is still unknown. Voulhoux and Tommassen developed a new model based on the outer membrane protein *Omp85* (Voulhoux and Tommassen, 2004), where the chaperone Skp binds in the periplasm to an OMBB and prevents the aggregation of the unfolded protein. Then this OMBB interacts with the periplasmatic domain of *Omp85*, the folding takes place with the presence of periplasmatic folding catalysts, and the protein inserts in a channel formed by *Omp85* which opens to allow the insertion of this OMP into the outer membrane.

OMBBs consist of a closed β-sheet with 8 to 24 strands forming the central pore. Prototypical OMBBs are built from a single polypeptide chain and are composed of repeating $\beta\beta$ hairpins with very short periplasmic loops (Fig. 6.2 left), linked by loops of variable length, many of which are crucial for their protein's functions. Two single H-bonded β-strands at the N- and C-termini close the barrel. A few OMBBs from Gram-negative bacteria form their transmembrane (TM) barrel from homotrimers (Fig. 6.2 middle), such as TolC or the trimeric autotransporter Hia from *H. influenzae*. In OMBBs, the C- and N-termini of the TM domain are located in the periplasm and the chain runs clockwise around the pore when viewed from the outside.

Two atypical trans-membrane β-barrels (TMBBs) with known structure occur in Gram-positive bacteria: MspA and α-hemolysin. Their barrels are very well structurally superposable with the OMBBs once they are flipped by 180 degrees. The N- and C-termini of their single $\beta\beta$ hairpins are outside the cell and their chains run in an *anti*-clockwise direction around the central pore. MspA (Faller et al., 2004) is the main porin from mycobacteria. Even though classified as Gram-positive, mycobacteria possess an atypical outer lipid membrane (Brennan and Nikaido, 1995). α-hemolysin is a toxin from pathogenic Gram-positive bacteria (Song et al., 1996) that lyses host cells by integrating into their plasma membranes. MspA and α-hemolysin form their β-barrels from 8 and 7 single $\beta\beta$ hairpins, respectively,

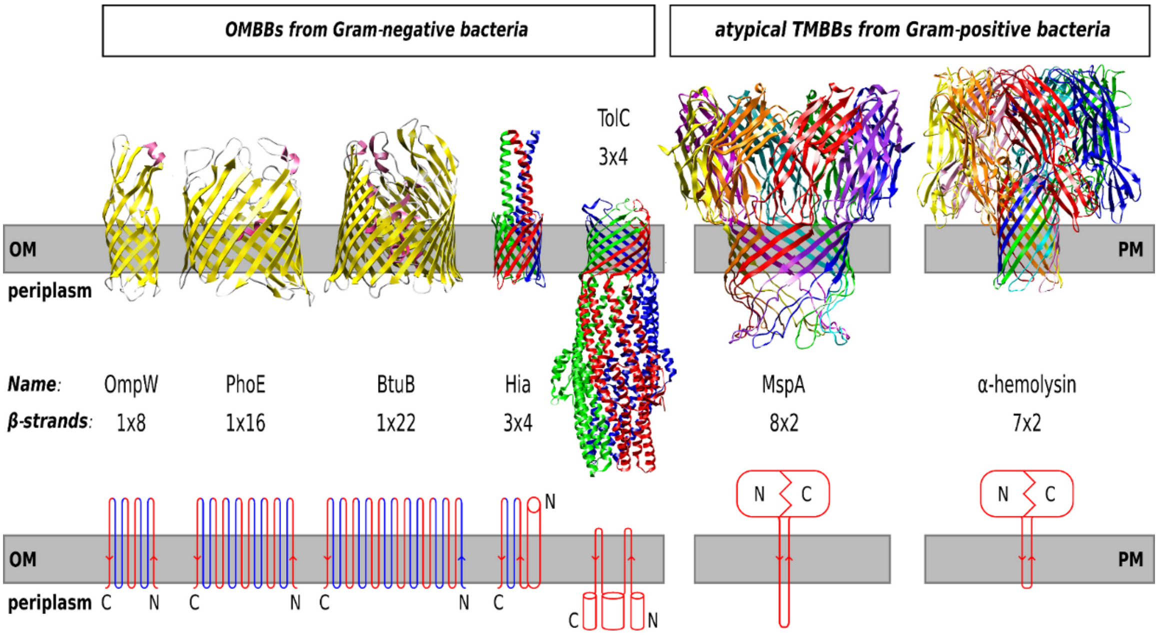


Figure 6.2.: Gallery with three examples of typical, single-chain OMBBs (left) and two multi-chain OMBBs from Gram-negative bacteria (middle), and two atypical transmembrane β-barrels (TMBBs) from Gram-positive bacteria. The topologies are schematically shown in the lower part. All OMBBs from Gram-negative bacteria have their N- and C-terminus in the periplasm, whereas the termini of the two atypical TMBBs from Gram-positive bacteria are outside the cell. Also, the peptide chains of OMBBs wind around the central pore in a clockwise sense when viewed from outside, whereas the $\beta\beta$ hairpins of the atypical TMBBs are traversed in counterclockwise direction. The structural hairpin repeats are colored in red and blue. OM: outer membrane; PM: host cell plasma membrane. PDB structures: OmpW (2F1C), PhoE (1PHO), BtuB (1NQE), Hia (2GR7), TolC (1EK9), α-hemolysin (7AHL), MspA (1UUN).

protruding from their large, extracellular domains, which are necessary for oligomerization (Fig. 6.2 right). The unusual topology of the atypical TMBBs as well as their singular phylogenetic representation among specific groups of Gram-positive bacteria makes it unlikely that they are related with the prototypical, single-chain OMBBs from Gram-negative bacteria. They can thus serve as analogous reference structures.

In OMBBs, the TM β-strand residues facing outside are in contact with the membrane and are mostly hydrophobic, those facing inside into the typically solvent-filled interior are mostly small and hydrophilic. This imparts a relatively regular, amphipathic character to the TM β-strands (Neuwald et al., 1995). Efforts to detect OMBBs from their sequence have focused on this generic sequence pattern (Bagos et al., 2004; Berven et al., 2004; Bigelow et al., 2004), but since it is not perfectly regular, total hydrophobicity is similar to cytosolic proteins, and β-strands make up only $\sim 50\%$ of the barrel sequence, the prediction of OMBBs remains challenging. Also, a similarity in sequence between OMBBs from different groups is hardly detectable. Neuwald *et al.* applied a Gibbs-sampling strategy to 32 OMBBs presumed to be involved in substrate uptake, and identified a significant, 11-residue sequence pattern that coincided with the first β-strand of the $\beta\beta$ hairpin repeats (Neuwald et al., 1995). It is unclear whether the motif resulted from the amphipathic character of the β-strands or whether it reflects a common evolutionary origin of the porins, the major class of proteins in their set. Recently, a monophyletic relationship was postulated for the 16-stranded bacterial porins (Nguyen et al., 2006) as well as for the presumably 16-stranded Omp85-like proteins (Moslavac et al., 2005). In support of the proposed common origin of OMBBs through oligomerization and fusion of shorter modules, Arnold *et al.* (Arnold et al., 2007) found that the duplicated and fused sequence of OmpX, an eight-stranded β-barrel, dimerized into a stable, 16-stranded TM β-barrel with a single pore.

6.3. Protein evolution

One of the major transitions in evolution is marked by the end of the RNA-world, when most enzymatic, structural, and regulatory functions of RNA were taken over by proteins (Orgel, 2004). This transition probably coincides with the advent of small protein proto-domains capable of folding into relatively stable, functional structures independent of their former RNA partners. The elevated error rates of the early replication machinery would initially have severely limited the length of single-gene mini-chromosomes (‘Eigen limit’) (Eigen and Schuster, 1977; Jeffares et al., 1998). Therefore, many of these proto-domains were probably formed by oligomerization from smaller peptide modules. It is plausible to assume that later, when lowered error rates allowed for longer mini-chromosomes, the genes of many of these peptide modules were fused together into genes encoding entire single-chain proto-domains (Lupas et al., 2001; Söding and Lupas, 2003). These later became the conserved cores of larger fold families, which evolved from the proto-domains by “piecemeal growth”, i.e., by multiple additions of structural elements, and, to a lesser extent, deletions and rearrangements (Fetrow and Godzik, 1998; McLachlan, 1972).

This scenario of the origin of proteins from ancient peptide modules was suggested based on the observation that a number of recurring fragments exist that display similarity both in structure and sequence (Alva et al., 2007, 2008; Coles et al., 2006; Copley et al., 2001; Friedberg and Godzik, 2005; Grishin, 2000, 2001; Krishna et al., 2006; Lupas et al., 2001; Shao and Grishin, 2000; Söding and Lupas, 2003). If this scenario is true, we would expect many domains to have formed by the amplification of a single peptide unit: Replication slippage provides a simple mechanism for repeat amplification. Also, for symmetry-related reasons stable protein complexes evolve more readily from identical units than from heterologous ones (Lukatsky et al., 2007). Indeed, of the ten most populated folds, six are composed of structural repeats (Söding and Lupas, 2003). Whether the numerous structurally repetitive folds evolved by amplification of an ancestral single module or whether their repeat structure is the result of evolution converging onto similar, stably folding substructures has been intensely investigated (Biegert and Söding, 2008; Chen et al., 1997; McLachlan, 1972, 1987; Nagano et al., 1999; Söding et al., 2006a).

The structural similarity between proteins cannot be considered proof of common ancestry, because structure space is relatively small with its limited number of arrangements of secondary structure elements and many examples of *structural convergence* have been described (Finkelstein and Ptitsyn, 1987; Krishna and Grishin, 2004). In practice, a homologous relationship is often accepted when the sequences are significantly similar (Doolittle, 1994; Murzin, 1998; Pearson, 1996), when both sequences and structures are sufficiently similar (Cheng et al., 2008; Holm and Sander, 1997; Madej et al., 2007; Murzin, 1993; Russell et al., 1997), or when, in addition to sequence or structure similarity, other information such as the co-occurrence of rare structural or functional features, functional annotations, or sequence motifs hint at a homologous relationship (Dietmann and Holm, 2001; Gewehr et al., 2007; Holm and Sander, 1997; Murzin, 1998; Nagano et al., 2002).

Despite the usefulness of these criteria, the degree of sequence similarity remains the most important criterion for common ancestry in practice. However, a significant but weak sequence similarity might be the result of constraints that similar structures impose on their sequences. The structural similarity in turn could have evolved convergently due to functional or biophysical constraints. Although the problem of how to distinguish between a similarity by structurally induced *sequence convergence* (Doolittle, 1994) and a very remotely homologous relationship has often been noted, few studies have tackled it directly. Theobald and Wuttke analyzed the evolutionary relationships among representatives of three similar, small, all-β folds: OB-fold, SH3, and PDZ domains (Theobald and Wuttke, 2005). They built sequence profiles for representative sequences from these folds and calculated profile-profile similarity scores. Since the inter-fold similarity scores can be considered as representative for relationships between *analogous* structures, the intra-fold scores that significantly exceeded the inter-fold scores were interpreted as indicating homologous relationships.

Here, we propose that the $\beta\beta$ hairpins of which OMBBs are composed, are homologous to each other, presenting an extreme example of divergent evolution. We follow three approaches to investigate the evolution of OMBBs. First, we multiply link most representative OMBBs with each other through significant sequence similarity. Second, we demonstrate that many OMBBs possess a clear and significant repeat signature on the sequence level. Both these approaches rely on detecting sequence similarities and could be misled by sequence convergence. In our third approach, we carry the idea of analogous relationships as reference distribution further. Using two atypical transmembrane β-barrels from Gram-positive bacteria as analogous reference structures, we argue that the similarities are unlikely to be the result of sequence convergence.

7. Material and methods

7.1. Exhaustive, transitive profile searches

We employ two sequence search methods to multiply link OMBBs from known groups with each other. The first method, HHsearch (see section 2.4.3), is based on the pairwise comparison of profile hidden Markov models (HMMs) and has been successfully applied by many groups for protein function and structure predictions. The dataset is derived from SCOP 1.73[1] (Murzin et al., 1995), filtered to a maximum pairwise sequence identity of 25% (SCOP25). SCOP is a database of protein domains with known structure, hierarchically ordered by class, fold, superfamily, and family. We added the sequences of the OMBBs *VDAC1* (Ujwal et al., 2008), *FhaC* (Clantin et al., 2007), and *PapC* (Remaut et al., 2008), which were not yet contained in SCOP v1.73. The resulting "SCOP25+OMBB" database contains 7767 proteins, out of which 23 are single-chain OMBB sequences (Table 7.1). A profile HMM was built for each of the SCOP25+OMBB sequences using the BUILDALI.PL script with default parameters. The 23 representative OMBBs were compared with the SCOP25+OMBB database using HHsearch (v1.5.0). We used default parameters but switched off the secondary structure scoring to ensure that the matches are not ranked according to a superficial similarity of the predicted secondary structure. We then analyzed which representative OMBB sequences were found before the first to fifth non-OMBB sequence.

The second method, HHsenser (Söding et al., 2006b), which relies on exhaustive, transitive profile searches, starts with each of 19 representative OMBBs in SCOP25 (v1.71) (Table 7.2) as a query. It first searches with PSI-BLAST for homologs of the query sequence. Since the exhaustive sequence comparisons are quite CPU time-intensive, we search a reduced database, consisting of all bacterial sequences plus environmental sequences from the NCBI, filtered with a maximum pairwise sequence identity of 90% ("nr_bac90" and "env90"). Significant matches with E-value$< 10^{-3}$ are aligned to the query sequence and a profile HMM is computed. Insignificant matches with E-values between 10^{-3} and 1.0 (the "trailing end") are considered as potential homologs and kept in a list for later verification. For this purpose, an iterative PSI-BLAST search is started with the potential homolog, a profile HMM is computed from the resulting alignment, and the HMM is compared with the evolving profile HMM of the query sequence using HHsearch version 1.5.0 (Söding et al., 2005). If the homologous relationship is rejected, the next potential homolog is taken from the list to

---

[1] `http://scop.mrc-lmb.cam.ac.uk/scop/`

Table 7.1.: 23 representative single-chain OMBBs used for the direct HMM-HMM comparisons with HHsearch. We obtained the representative sequences from the SCOP database (version 1.73, filtered for 25% maximum pairwise sequence identity), and added the sequences of *VDAC1*, *FhaC* and *PapC* that were not yet contained in SCOP v1.73. The last column gives the number of β-strands forming the β-barrel.

PDB-ID	protein name	Organism	β-strands
1AF6	Maltoporin Sucrose Complex (LamB)	*Escherichia coli*	18
1FEP	Ferric Enterobactin Receptor (FepA)	*Escherichia coli*	22
1HXX	Porin OmpF	*Escherichia coli*	16
1I78	Outer Membrane Protease (Ompt)	*Escherichia coli*	10
1K24	Outer Membrane Adhesin/Invasin (OpcA)	*Neisseria meningitidis*	10
1KMO	Outer Membrane Transporter (FecA)	*Escherichia coli*	22
1OH2	Sucrose-specific Porin	*Salmonella typhimurium*	18
1P4T	Neisserial surface protein (NspA)	*Neisseria Meningitidis*	8
1QD6	Outer Membrane Phospholipase A (OmpLA)	*Escherichia coli*	12
1QJ8	Outer Membrane Protein X (OmpX)	*Escherichia coli*	8
1QJP	Outer Membrane Protein A (OmpA)	*Escherichia coli*	8
1T16	Fatty Acid Transporter (FadL)	*Escherichia coli*	14
1THQ	Outer Membrane Enzyme (PagP)	*Escherichia coli*	8
1TLY	Bacterial nucleoside transporter (Tsx)	*Escherichia coli*	12
1UYN	Translocator Domain Of Autotransporter (Nalp)	*Neisseria Meningitidis*	12
2FCP	Ferric Hydroxamate Uptake Receptor (FhuA)	*Escherichia coli*	22
2FGQ	Porin Omp32	*Comamonas acidovorans*	16
2GUF	Cobalamin Transporter (BtuB)	*Escherichia coli*	22
2POR	Porin	*Rhodobacter capsulatus*	16
2QDZ	Omp85/TPSB transporter family protein (FhaC)	*Bordetella pertussis*	16
2VQI	P Pilus Usher Translocation Pore (PapC)	*Escherichia col*i	24
3EMN	VDAC1	*Mus musculus*	19
3PRN	Porin E1M	*Rhodopseudomonas blastica*	16

start an iterative PSI-BLAST search. If the homology is confirmed, the sequences in the new alignment are added to the evolving query sequence alignment. In addition, all sequences in the trailing end of the last PSI-BLAST search are added to the list of potential homologs. The exhaustive search continues until all potential homologs have been validated or rejected.

To cluster the obtained sequences in 2D, the sequence fragments returned by the HHsenser runs were extended to full length (using the command FASTACMD from NCBI's BLAST package) and clustered with the program CLANS (Frickey and Lupas, 2004). CLANS represents each sequence by a point in 2D (optionally also in 3D) and moves it in this space according to the forces exerted by all other points. These forces are calculated from the matrix of pairwise BLAST log-E-values. Very significant E-values result in large attractive forces, insignificant E-values give repulsive forces. In this way, after sufficient relaxation times, similar sequences come to lie closely together. We used default CLANS parameters with an E-value cutoff of 0.1 for the clustering procedure.

We use CLANS to generate a pooled cluster map from the resulting sequences of all 19 HHsenser runs. The false positive rate of this map (Fig. 8.3) is estimated by counting the proteins from Gram-positive bacteria. Usually, Gram-positive bacteria do not have an outer membrane and their proteins are likely to be false positives. The map contains 83 proteins

Table 7.2.: 19 representative OMBBs used as starting point for transitive profile searches with HHsenser. We obtained the sequences by filtering the sequences of all bacterial OMBBs in the SCOP database (version 1.71) for a maximum of 25% pairwise sequence identity. The last column gives the number of β-strands forming the β-barrel.

PDB-ID	protein name	Organism	β-strands
1AF6	Maltoporin Sucrose Complex (LamB)	*Escherichia coli*	18
1E54	Anion-selective porin (Omp32)	*Comamonas acidovorans*	16
1FW3	Outer Membrane Phospholipase A (OmpLA)	*Escherichia coli*	12
1I78	Outer Membrane Protease (Ompt)	*Escherichia coli*	10
1K24	Outer Membrane Adhesin/Invasin (OpcA)	*Neisseria meningitidis*	10
1KMO	Outer Membrane Transporter (FecA)	*Escherichia coli*	22
1NQE	Outer Membrane Cobalamin Transporter (Btub)	*Escherichia coli*	22
1P4T	Neisserial surface protein (NspA)	*Neisseria Meningitidis*	8
1PHO	Phosphoporin (Phoe)	*Escherichia coli*	16
1Q9F	Outer membrane protein (OmpX)	*Escherichia coli*	8
1T1L	Long-Chain Fatty Acid Transporter (Fadl)	*Escherichia coli*	14
1THQ	Outer Membrane Enzyme (PagP)	*Escherichia coli*	8
1TLY	Bacterial nucleoside transporter (Tsx)	*Escherichia coli*	12
1UYN	Translocator Domain Of Autotransporter (Nalp)	*Neisseria Meningitidis*	12
2ERV	Outer Membrane Enzyme (Pagl)	*Pseudomonas aeruginosa*	8
2F1V	Outer membrane protein (OmpW)	*Escherichia coli*	8
2GE4	Outer Membrane Protein A TM Domain (OmpA)	*Escherichia coli*	8
2O4V	Porin P (OprP)	*Pseudomonas aeruginosa*	16
2POR	Porin	*Rhodobacter capsulatus*	16

from Gram-positive bacteria, but most of them are likely to be true OMBBs. 32 are from *Mycobacterium*, a Gram-positive bacterium with an outer membrane composed of mycolic acids at the inner leaflet and lipooligosaccharides and glycolipids at the outer leaflet. 13 are from *Thermosinus carboxydivorans* and 12 from *Halothermothrix orenii*, two firmicute bacterial species in the genus *Clostridia* which are known from electron micrographs to possess outer membranes (Cayol et al., 1994; Sokolova et al., 2004). Two other proteins, one from *Selenomonas ruminantium* and one from *Clostridium bifermentans*, have more than 60% sequence identity to OMBBs with known structure from *E. coli.* 24 proteins remain as false positives from Gram-positive bacteria. To estimate the false positive proteins from Gram-negative bacteria, we build HMMs from the sequences of all clusters in the map (Fig. 8.3) and searched through the SCOP database without OMBBs using HHsearch. A total of 46 false-positive sequences from Gram-negative bacteria could be identified. All together, the false positive rate can be estimated to be around $(24 + 46)/21856 = 0.3\%$.

7.2. Internal repeat detection

To support our hypothesis that OMBBs evolved by amplification of $\beta\beta$ hairpin modules, we employed HHrepID, a fully automated method for the *de novo* identification of highly diverged protein repeats by probabilistic consistency (Biegert and Söding, 2008). To analyze the 23 representative OMBBs in Table 7.1 for sequence repeats, we ran the BUILDALI.PL script from the HHsearch 1.5.0 package with default parameters. HHrepID was then started with these multiple sequence alignments.

HHrepID converts the input alignment into a profile HMM. This query HMM is then repeatedly aligned to itself by means of HMM-HMM comparison (Söding, 2005) in order to detect internal sequence symmetries. The quality of predicted repeat alignments is improved by an algorithm that maximizes the expected accuracy (MAC, see section 3.6). Furthermore, the repeat detection algorithm makes use of the transitive nature of homology through a novel alignment merging procedure reinforcing the consistency of suboptimal self-alignments. This consistency reinforcement boosts the sequence signature of even highly diverged repeat units and at the same time suppresses traces of spurious alignments.

We performed a *de novo* repeat analysis on the 474 clusters of putative OMBBs from the pooled cluster map (Fig. 8.3). For this purpose, we constructed a multiple sequence alignment for each cluster using KALIGN2 (Lassmann and Sonnhammer, 2005), jump-started PSI-BLAST with three iterations to add further homologs and extracted the resulting sequence alignment from the results using the ALIGNHITS.PL script in the HHsearch package. HHrepID was run on these alignment.

7.3. Analysis of structural convergence

To investigate whether the observed sequence similarities among the OMBB hairpins could be explained by convergence, we performed a combined sequence-structure analysis of the 23 OMBBs from SCOP25 (v1.73, see Table 7.1). The OMBBs were divided into overlapping double hairpins, using the DSSP secondary structure assignment. Starts and ends were placed at periplasmic side, starting with the first and ending with the last barrel strand. We used double instead of single hairpins to increase the signal-to-noise ratio. For example, the 12-stranded OMBB *OmpLA* has 5 double hairpins with ends at the periplasmic side. 3 of these double hairpins are colored in Figure 7.1 in red, blue and green, respectively. The other two double hairpins are comprised by a green and a blue hairpin and by a blue and a red hairpin, respectively.

We used TM-ALIGN (Zhang and Skolnick, 2005) to search with these double hairpins for structurally similar fragments in six sets of protein structures: the SCOP database with all OMBB sequences removed (SCOP folds f.4, f.5, f.6), the set of 23 OMBBs, the set of two non-canonical OMBBs *Hia* and *TolC*, the set with two OMBBs from Gram-positives, *α-hemolysin* and *MspA*, the set with analogous, structural similar proteins (lipocalin-like and streptavidin-like β-barrels, SCOP IDs b.60 and b.61.1), and the set of only the C-terminal double hairpins from OMBBs. The TM-SCORE was normalized using the length of the OMBB double hairpins. For each matched pair, we also calculated the profile-profile similarity score with HHsearch, but based on the fixed alignment from TM-ALIGN and with zero gap penalties. Each pair of structure and sequence similarity scores was plotted in a scatter plot.

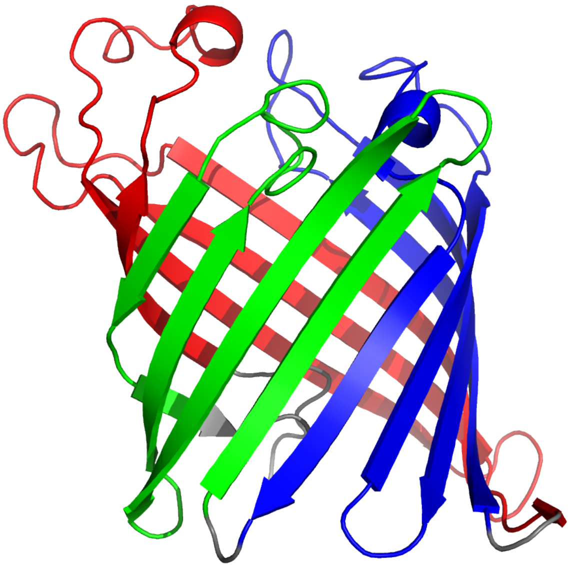


Figure 7.1: Double hairpins of the outer membrane protein *OmpLA* with ends at the periplasmic side. Three out of the five possible double hairpins are colored in red, blue and green, respectively. The other two double hairpins are comprised by a green and a blue hairpin and by a blue and a red hairpin, respectively.

For the plots with non-canonical OMBBs, a double hairpin must be generated for each of the four non-canonical OMBBs. *Hia* and *TolC* consists of three chains each with four β-strands. Here, full-length PDB chains were used. In *α-hemolysin* and *MspA*, each chain consist of only two transmembrane β-strands. A double hairpin was generated by cutting two single hairpins out of two neighboring chains and combining these single hairpins to a double hairpin. By their construction, these double hairpins are very well structurally superposable with the OMBBs when turned upside-down, as can be seen from the distribution of structural similarity scores in Fig. 8.8B.

8. Results

8.1. Most OMBBs find each other using homology detection methods

For remote relationships that date back to the origin of protein domains some four billion years ago, each amino acid has mutated several times on average (Doolittle et al., 1996), and sequence similarities have decayed to levels far below the noise threshold. Instead of directly comparing amino acid sequences, we therefore employ a highly sensitive homology detection method, HHsearch (see section 2.4.3), that compares *profile hidden Markov models (HMMs)* with each other (Söding, 2005). Profile HMMs encode position-specific amino acid (and gap) preferences that are conserved for much longer than amino acid identities.

The PDB database contains single-chain OMBBs from 12 families and 6 superfamilies according to the SCOP classification (v1.73) (Murzin, 1998) with between 8 and 24 β-strands. We show first that most of these can be linked with each other through homology searches based on HMM-HMM comparison. The "SCOP25+OMBB" was constructed as explained in 7.3. It contains 7767 representative single-domain sequences of known structure, out of which 23 are single-chain OMBB sequences (Table 7.1).

The 23 representative OMBBs were compared with the 7767 proteins in the SCOP25+OMBB database using HHsearch. Fig. 8.1A shows which of the 23 OMBBs (top row) detects the other 22 OMBBs (left) with higher significance than the first non-OMBB match in the database search. OMBBs found before the first to fifth non-OMBB match are marked in five shades of blue. 20 out of the 23 OMBBs are multiply connected with each other through the pairwise similarities of their HMMs. Notably, this includes the mitochondrial *VDAC1*. The other three OMBBs have two (*Tsx*, *PapC*) and a single connection (*PagP*) to the rest.

Next, we asked whether a transitive homology detection method would be able to provide significant links between the weakly connected OMBBs and the core group (Fig. 8.1B). We started exhaustive intermediate profile searches from 19 representative OMBBs (top row, Table 7.2) using the tool HHsenser (Söding et al., 2006b). We took all bacterial sequences from NCBI's non-redundant database plus the environmental sequences and filtered them for a maximum pairwise sequence identity of 90% ("nr_bac90" and "env90"). Each of the searches returned between a few hundred and $\sim$ 17,000 sequences. We checked which of the 23 representative OMBBs in the SCOP25+OMBB set (left column) was either found directly or had a BLAST E-value of 10^{-5} or better with one of the returned sequences. Most of

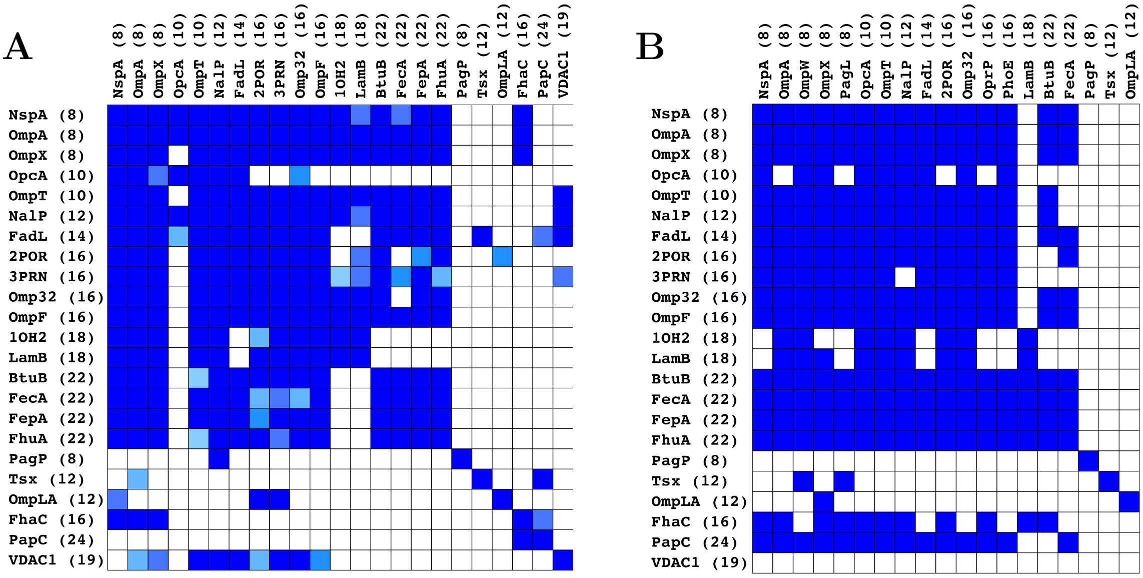


Figure 8.1.: Representative OMBBs of known structure detect each other reliably in homology searches. (A) Direct pairwise HMM comparisons by HHsearch: Each matrix column shows the results of an HMM-HMM search with the query proteins in the top row (with the number of barrel β-strands in parentheses) through the SCOP25 database (v.1.73) including *VDAC1*, *FhaC* and *PapC* (PDB codes: Table 7.1). The color shade indicates how many false positive (FP) matches were detected with a probability score higher than that of the database OMBB on the left, from blue (no FPs) to light blue (4 FPs). (B) Exhaustive transitive HMM-based homology searches: Each matrix column shows the results of an exhaustive transitive search started from the OMBB in the top row (Table 7.2). Blue indicates which OMBBs on the left were detected in the exhaustive search or have a BLAST E-value better than 10^{-5} with one of these sequences. The mitochondrial OMBB *VDAC1* can not be identified here, since only bacterial and environmental sequences were searched.

the OMBBs that are linked to the others through direct HMM-HMM comparison also find each other in transitive searches. In addition, *FhaC* and *PapC* are found in 11 and 14 of the transitive searches, and *Tsx* and *OmpLA* are detected by two and one of the transitive searches, respectively. Together with the direct HMM-HMM comparison, all but *PagP* could be multiply linked to the main group of OMBBs.

Fig. 8.2 shows a cluster map resulting from a single HHsenser search, started with the sequence of *OmpT* (red cross). The 15775 sequence fragments were extended to full length and clustered based on the matrix of all pairwise BLAST log-P-values using the program CLANS (Frickey and Lupas, 2004). Every point represents a sequence. They attract each other with a strength proportional to their pairwise log-P-values, so that similar sequences come to lie together closely. 19 out of the 23 representative OMBBs containing between 8 and 24 β-barrel strands were identified in this single search.

What is the fraction of false positive sequences in the transitive searches? Since Gram-positive bacteria do not typically have an outer membrane, their sequences are likely to be false positive matches in the map. The map in Fig. 8.2 contains only nine sequences from bacteria classified as Gram-positives (Table 8.1). All of these sequences belong to species in the genus *Clostridiae* which possess outer membranes (see 7.1) and are therefore likely to be

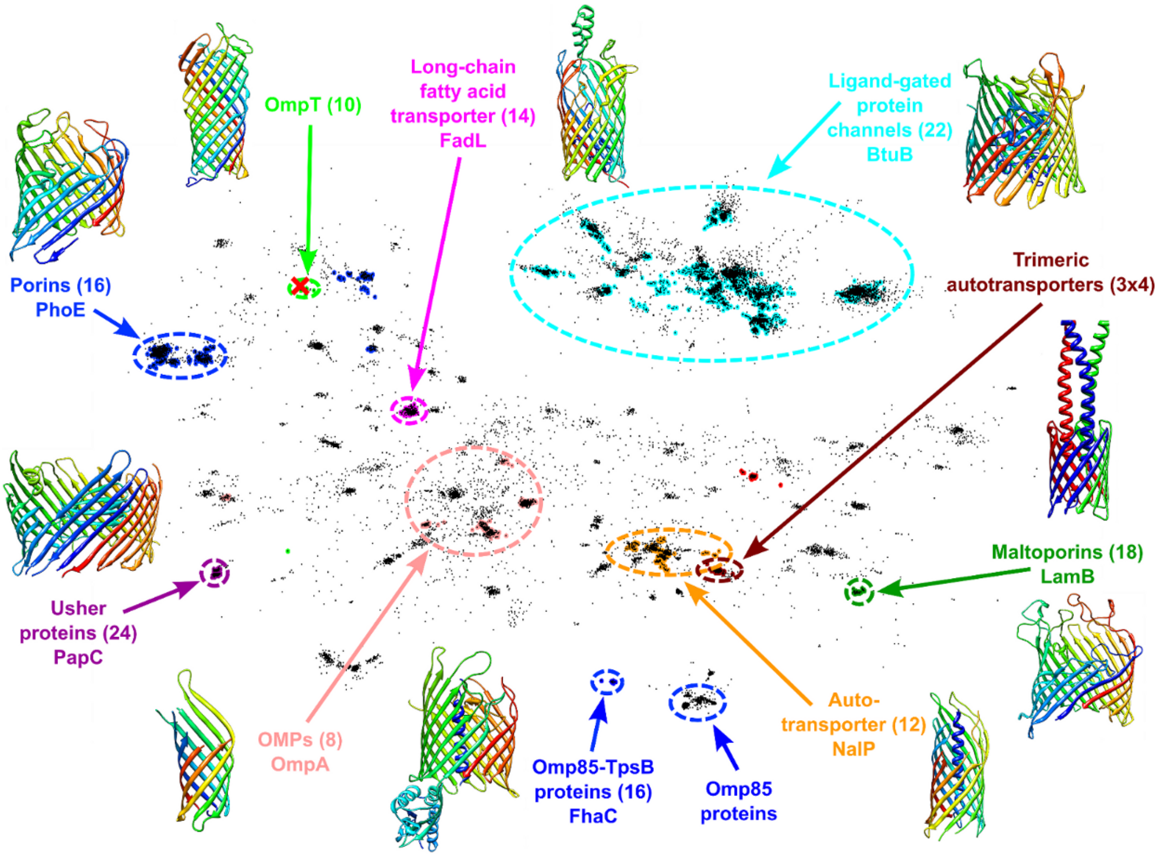

OmpT (10)
Long-chain fatty acid transporter (14) FadL
Ligand-gated protein channels (22) BtuB
Porins (16) PhoE
Trimeric autotransporters (3x4)
Maltoporins (18) LamB
Usher proteins (24) PapC
OMPs (8) OmpA
Omp85-TpsB proteins (16) FhaC
Omp85 proteins
Auto-transporter (12) NalP

Figure 8.2.: Cluster map of 15775 bacterial sequences found with the exhaustive transitive HHsenser search starting from *OmpT* (red cross). 19 out of the 23 *bona fide* OMBBs were identified in this single search with a false positive rate of only ∼ 0.03%. The highlighted clusters and OMBB structures illustrate the diversity of OMBB groups in the map.

Table 8.1.: Proteins from Gram-positive bacteria in the *OmpT* cluster map. All proteins belong to species in the genus *Clostridiae*, the first two of which are known to have an outer membrane (Cayol et al., 1994; Sokolova et al., 2004)

GI	Protein name	Organism
121533976	TonB-dependent receptor	*Thermosinus carboxydivorans*
121534322	TonB-dependent receptor	*Thermosinus carboxydivorans*
121534812	TonB-dependent receptor	*Thermosinus carboxydivorans*
121534955	surface antigin (D15)	*Thermosinus carboxydivorans*
121534975	hypothetical protein TcarDRAFT_1330	*Thermosinus carboxydivorans*
89210169	S-layer-like region	*Halothermothrix orenii H 168*
89210191	surface antigen (D15)	*Halothermothrix orenii H 168*
89211242	hypothetical protein HoreDRAFT_0748	*Halothermothrix orenii H 168*
13940157	tetrachloroethylene dehalogenase	*Clostridium bifermentans*

true OMBBs. We broadened our search for false positives and built HMMs from the sequences of all clusters in the map. We searched through the SCOP database using HHsearch and looked at all significant hits to non-OMBB domains. We identified 5 false positive sequences in a single cluster, which were included due to corrupted PSI-BLAST alignments from which their profile HMMs were built. We therefore estimate the fraction of false positives per search to be on the order of $5/15773 = 0.03\%$.

The results from the single cluster map are born out when the sequences of all 19 HHsenser searches are pooled (Fig. 8.3A): The majority of OMBBs are found in at least half of the HHsenser runs (Fig. 8.3B). The protein sequences in this figure are colored according to the number of HHsenser searches which could detect the respective protein, dark blue means detected by all 19 searches, blue to orange by some searches and brown for proteins detected by only one search. The pie charts on the right show the fraction of proteins detected by the specific number of searches. The upper chart refers to all 21856 proteins in the cluster map, the lower chart only to 12015 proteins annotated as OMPs. These pie charts demonstrate that about two thirds of all proteins and more than 75% of all known OMPs in this cluster map can be found in more than half of all HHsenser searches.

The fraction of false positives in the pooled map can be estimated to about 0.3% (see 7.1). Since about half of the detected sequences in the pooled cluster map belong to hypothetical proteins, our transitive searches roughly double the number of proteins that are confidently predicted to be OMBBs, from 12015 to 21850 proteins out of the 1,97 million proteins in the "nr_bac90 + env90" database used (Remmert et al., 2009). Furthermore, the pooled map is likely to be nearly complete: 55 out of 57 proteins annotated as OMBBs in *E. coli* have a confident match ($> 94\%$ probability) to a cluster in the map when using HHsearch / HHomp (Remmert et al., 2009). It is therefore not unlikely that most or all major groups of bacterial OMBBs have been found in one of our transitive searches. Taken together, all of the 23 representative OMBBs, including almost all annotated OMBBs in *E. coli*, can be linked with each other through direct HMM-HMM searches or through transitive homology searches at low error rate.

8.2. Sequence repeats detected in majority of OMBBs

If OMBBs originated by amplification of a single $\beta\beta$ hairpin, we might still find residual sequence similarities between the $\beta\beta$ repeats. We use the *de novo* repeat detection method HHrepID (Biegert and Söding, 2008), which was developed in our group and generates a profile HMM from the input sequence and looks for non-trivial, off-diagonal alignments of the HMM with itself (see 7.2). For N repeat units, we would ideally find $2N-1$ alignments, where each alignment corresponds to a register shift in the pairing of the repeat units. HHrepID gains further sensitivity by iteratively refining the suboptimal alignments for maximum consistency.

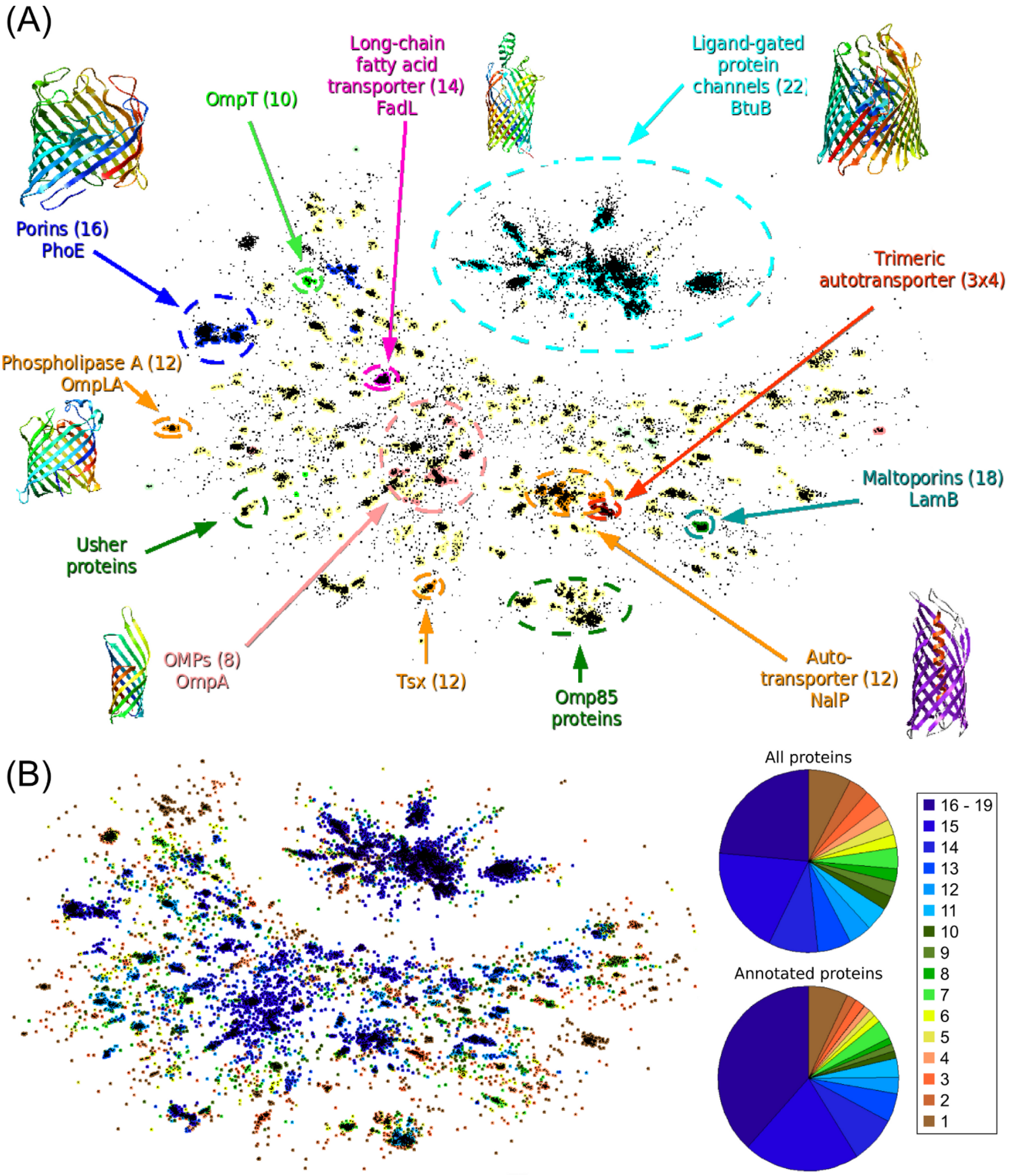
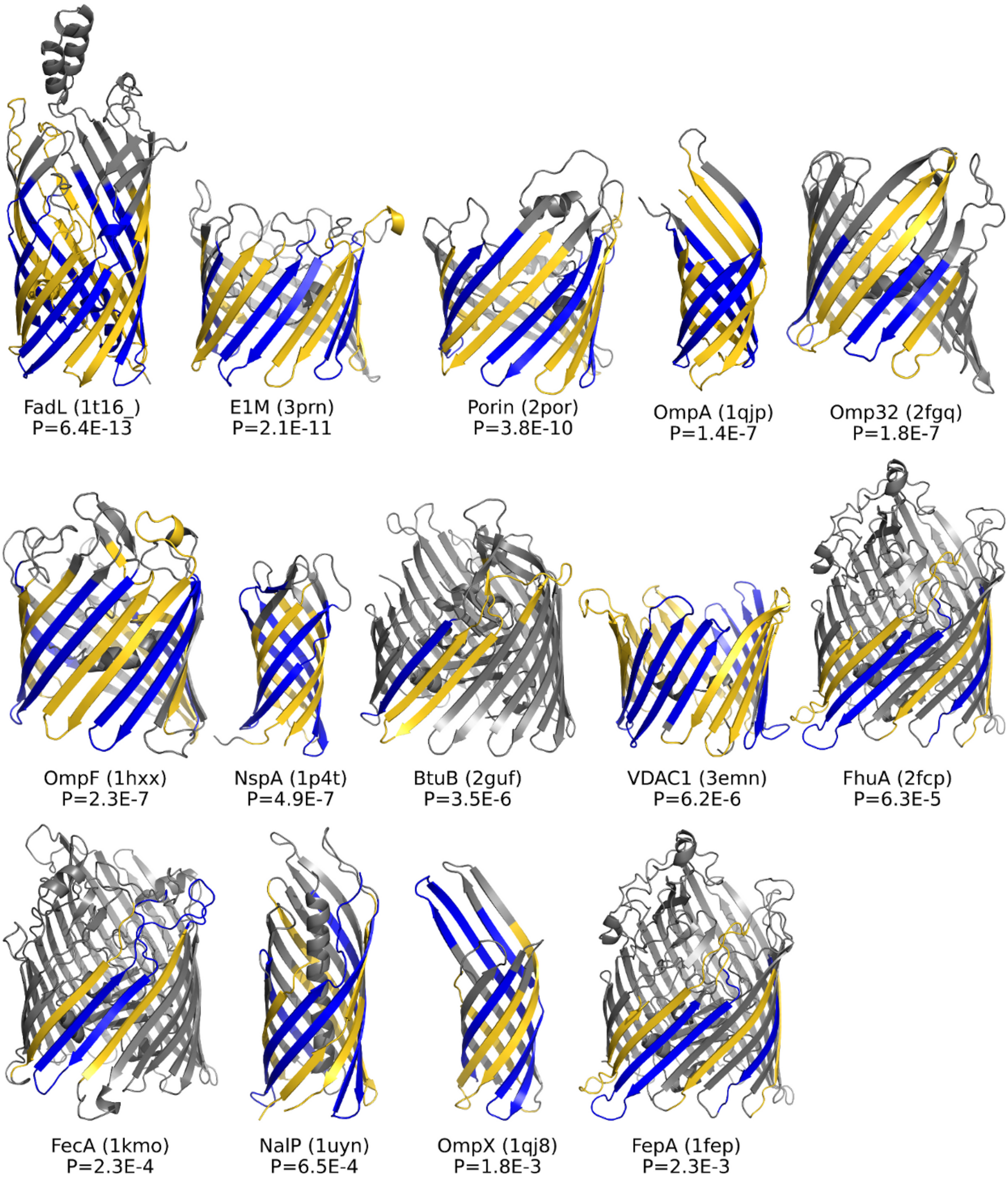


Figure 8.3.: (A) Pooled cluster map of 21856 protein sequences found with the exhaustive transitive HHsenser search starting from 19 OMBBs with known structure (Table 7.2). The different colors indicate the different numbers of β-strands. (B) Most sequence clusters in our map of putative OMBBs are reliably linked to known OMBBs by multiple HHsenser searches. This map is colored according to the number of transitive searches detecting each protein. The pie charts on the right show the fractions of proteins that were detected by all 19 searches (dark blue), by some of the searches (blue to orange), down to only one search (brown). The upper chart refers to all 21856 proteins in the map, the lower chart only to the 12015 protein sequences annotated as OMPs. About two thirds of all proteins and more than 75% of all known OMPs on the map can be found in more than half of all transitive searches (blue to cyan).



FadL (1t16_)
P=6.4E-13

E1M (3prn)
P=2.1E-11

Porin (2por)
P=3.8E-10

OmpA (1qjp)
P=1.4E-7

Omp32 (2fgq)
P=1.8E-7

OmpF (1hxx)
P=2.3E-7

NspA (1p4t)
P=4.9E-7

BtuB (2guf)
P=3.5E-6

VDAC1 (3emn)
P=6.2E-6

FhuA (2fcp)
P=6.3E-5

FecA (1kmo)
P=2.3E-4

NalP (1uyn)
P=6.5E-4

OmpX (1qj8)
P=1.8E-3

FepA (1fep)
P=2.3E-3

Figure 8.4.: 3D structures of the 14 representative OMBBs for which HHrepID detects a significant off-diagonal alignment with P-value better than 10^{-2} (see Table 8.2). Predicted repeat units are highlighted in blue and yellow. Almost all predicted repeats coincide with a structural $\beta\beta$-hairpin. Note that the conservation of sequence repeats seems to be highest near the C-terminal, in accord with functional restraints related to membrane insertion (Robert et al., 2006).

Table 8.2.: Sequence repeats are detected in 14 out of the 23 representative OMBBs. Repeats: detected (excluding the single end strands) and actual number of $\beta\beta$ hairpins. RMSD: median of the median RMSD of each detected repeat unit with all other detected repeats. P-value: significance for detection of sequence repeat pattern.

Name	PDB-ID	Repeats	RMSD	P-value
FadL	1t16	6/6	3.9	6.4×10^{-13}
E1M	3prn	6/7	1.1	2.1×10^{-11}
Porin	2por	6/7	1.5	3.8×10^{-10}
OmpA	1qjp	3/3	2.2	1.4×10^{-7}
Omp32	2fgq	5/7	1.8	1.8×10^{-7}
OmpF	1hxx	7/7	2.3	2.3×10^{-7}
NspA	1p4t	3/3	1.0	4.9×10^{-7}
BtuB	2guf	2/10	0.9	3.5×10^{-6}
VDAC1	3emn	5/9	4.2	6.2×10^{-6}
FhuA	2fcp	5/10	2.3	6.3×10^{-5}
FecA	1kmo	3/10	4.7	2.3×10^{-4}
NalP	1uyn	5/5	1.7	6.5×10^{-4}
OmpX	1qj8	3/3	2.0	1.8×10^{-3}
FepA	1fep	4/10	2.3	2.3×10^{-3}

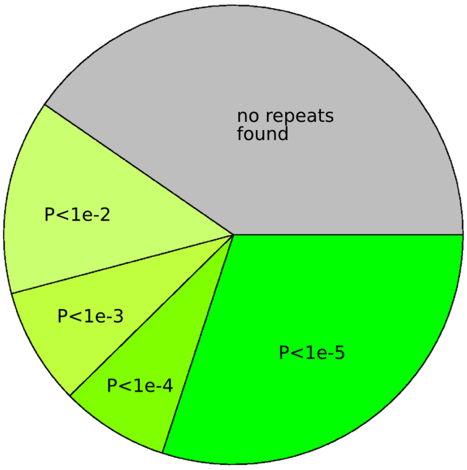


Figure 8.5: De novo repeat analysis using HHrepID on the 474 clusters of putative bacterial OMBBs from the pooled cluster map (Fig. 8.3). We detect repeats with a P-value better than 10^{-2} in 281 (59%) clusters (green sections), showing that internal sequence symmetry is a general property of the majority of bacterial OMBBs.

We performed the repeat analysis with HHrepID on the sequences of all 23 representative OMBBs (Table 7.1). In 14 of these sequences, repeats are found with a P-value of better than 10^{-2} (Table 8.2 and Fig. 8.4). Almost all of the identified sequence repeat units coincide with the structural $\beta\beta$ hairpins. In six cases, all $\beta\beta$ hairpins present are correctly identified as repeats, in a further four cases more than half of the hairpins are detected as repeat units. To quantify if HHrepID is able to correctly identify the structural repeats on the sequence level, we used the program TM-SCORE (Zhang and Skolnick, 2004) to measure the RMSD between all aligned pairs of C_α atoms. The RMSD column shows the median of the median RMSDs of each repeat unit with all other repeat units. 11 out of the 14 OMBBs have median RMSDs below 2.5Å, demonstrating that the repeats found in sequence are not spurious and

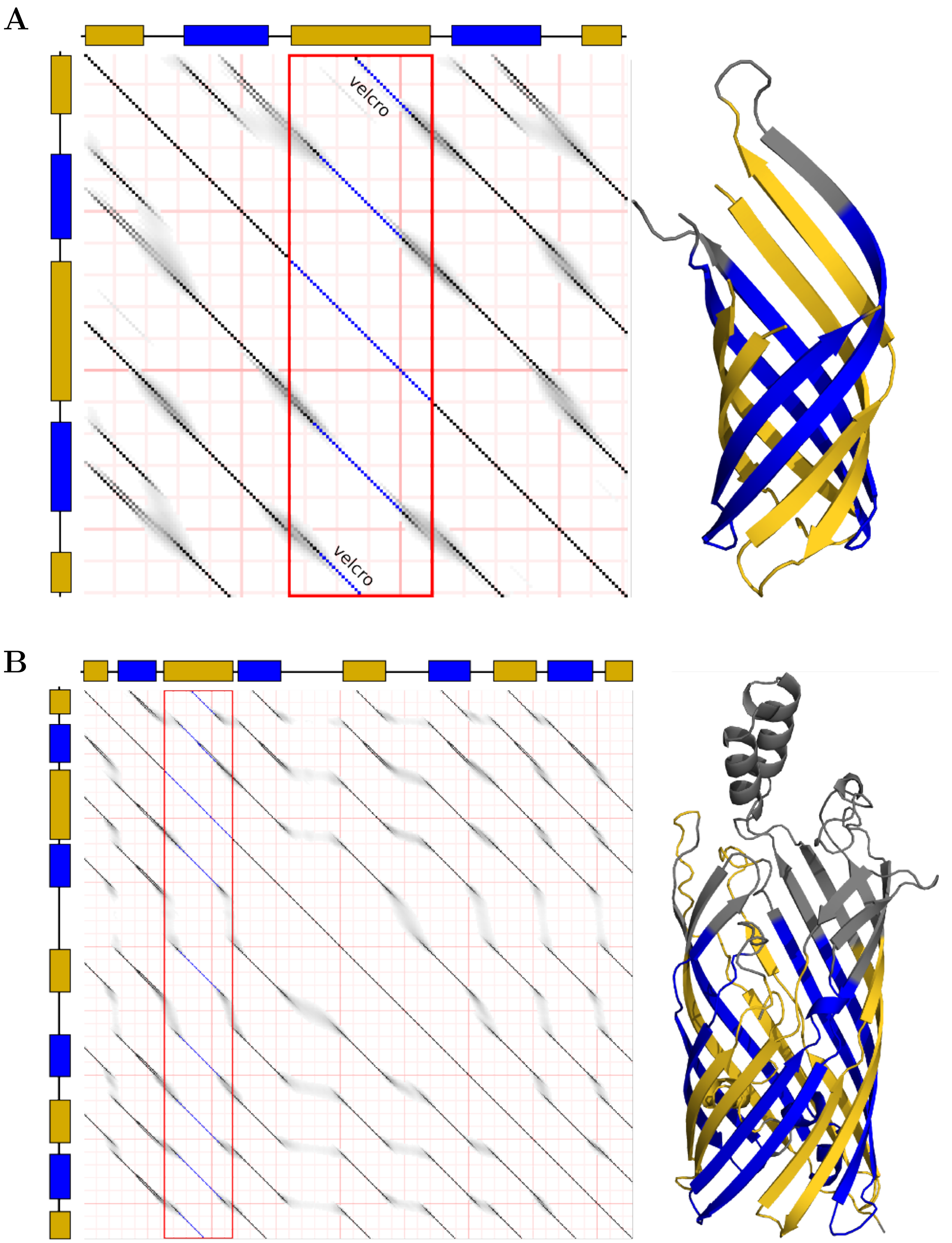


Figure 8.6.: Many OMBB sequences contain a hidden repeat pattern. (A) Self-comparison dot plot of the β-barrel domain of 8-stranded *OmpA* (Pautsch and Schulz, 2000) generated by HHrepID (Biegert and Söding, 2008). (B) Dot plot of 14-stranded long-chain fatty acid transporter *FadL* (Berg et al., 2004). The probability for each pair of residues to be homologous is coded in shades of gray, from 1.0 (black) via 0.5 (medium gray) to 0.0 (white). The blue and yellow colored boxes on the top and on the left of the dot plot represent the colored segments in the 3D-structures on the right.

coincide well with the structural repeats. The distinctness and regularity of repeat patterns detected by HHrepID are demonstrated in the dot plots of *OmpA* and *FadL* in Fig. 8.6. In the dot plot, the probability for each pair of residues to be homologous is coded in shades of gray. Clearly, the identified repeats (colored blue and yellow in the OMBB structures) coincide well with the structural $\beta\beta$ hairpins repeats. We also analyzed the 474 clusters of putative bacterial OMBBs from the pooled cluster map (Fig. 8.3) and confidently predicted repeats (P-value $< 10^{-2}$) in 281 clusters (59%) (Fig. 8.5).

We note that the detected internal sequence similarities are not unique to the OMBB metafold. We previously identified a 4-fold symmetry in many superfamilies of the $(\beta\alpha)_8$ barrel fold (Söding et al., 2006a), for example, suggesting that they, too, evolved by amplification of a shorter module. However, in lipocalins and streptavidin-like β-barrels (SCOP IDs b.60 and b.61.1), the two groups that are structurally most similar to the 8-stranded OMBBs, we found sequence repeats in only one of the 26 sequences from SCOP25_1.75, with marginal significance (P-value $= 0.002$).

8.3. Sequence similarity not due to structural convergence

It could be argued that the significant but weak sequence similarities among bacterial OMBBs and between their $\beta\beta$ hairpins are the result of structurally induced sequence convergence: The hairpin structures could have evolved convergently as a solution to the problem of forming stable membrane-embedded barrels and these structural and functional constraints exerted similar, detectable constraints on their sequences. In this case, we would expect to see a positive correlation between the structural and the sequence similarity of $\beta\beta$ hairpins.

To investigate this question, we manually divided all 23 single-chain bacterial OMBBs from the SCOP25 set (v1.73, Table 7.1) into overlapping double hairpins with periplasmic N- and C-termini (see 7.3). We used TM-ALIGN to perform structural searches with each double hairpin through the SCOP database from which all OMBB structures had been removed. For each matched pair, we also calculated the profile-profile similarity score using HHsearch but based on the fixed alignment from TM-ALIGN. Each blue dot in the scatter plot in Figure 8.7 marks the structural and sequence similarity scores for a matched pair. Strikingly, the sequence similarity is essentially independent of structural similarity for these analogous matches (Pearson correlation -0.02). In other words, structurally induced sequence convergence is not detectable.

How are matches between OMBB double hairpins distributed with respect to this reference distribution? We compared all double hairpins from canonical OMBBs with each other, using TM-ALIGN and HHsearch as before. The scores are shown as red dots in Figure 8.7. Intriguingly, the distribution looks just as expected if OMBBs diverged from a common ancestor. First, the structural similarity scores are positively correlated with sequence similarity scores (Pearson correlation 0.31), reflecting the varying degree of divergence from

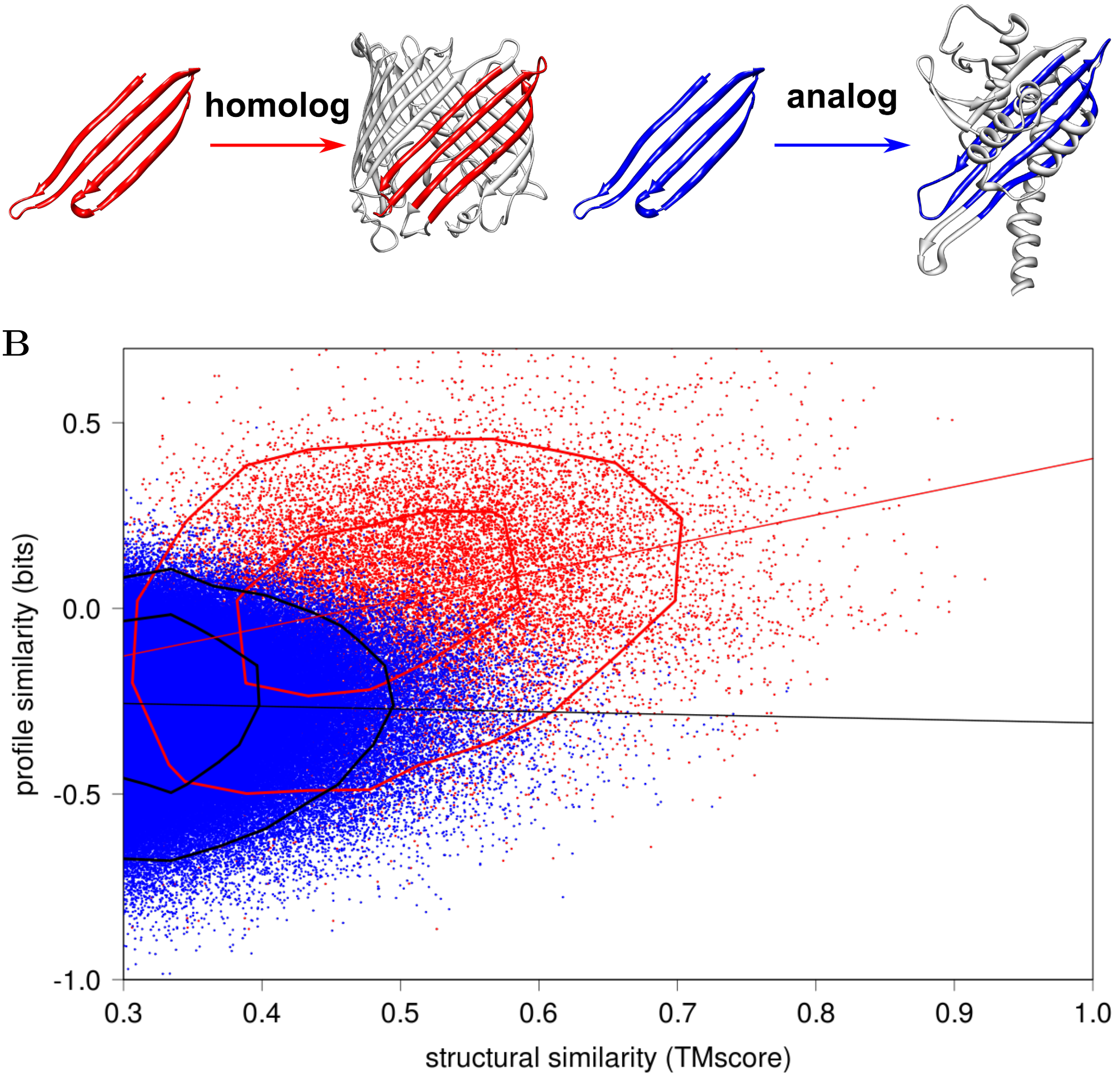


Figure 8.7.: The sequence similarity between OMBBs cannot be explained by structurally induced sequence convergence. (A) Perform a structural alignment between all double hairpins from single-chain bacterial OMBBs and all other double hairpins from OMBBs (red), and between double hairpins and all proteins in the PDB minus OMBBs (blue). (B) Profile-profile and structural similarity scores with the same coloring as in (A).

the common ancestor. Second, the red distribution is significantly shifted to higher sequence similarity scores with respect to the reference distribution over the whole range of structural similarity. This invalidates structure-induced sequence convergence as cause for the elevated sequence similarities among OMBB hairpins.

One might expect the sequence similarity to depend more strongly on functional properties than on structure. For example, the vast majority of OMBBs possess a C-terminal signal sequence in their last β-strand, which is needed for the insertion into the outer membrane (Robert et al., 2006). To investigate the influence of the C-terminal signal sequence on the sequence similarities, we highlighted all cases within the red distribution in which the compared double hairpins both contained the last β-strand (carrying the C-terminal signal sequence) (Fig. 8.8D). While a few of these comparisons result in sequence similarities that are among the highest observed in Figure 8.7, most are distributed just as the other red points. But even if other functional constraints would contribute significantly to the vertical scattering in Figure 8.7, which is strong in comparison to the correlation of structure and sequence similarity, the sheer number of dots in the red distribution allows us to clearly discern this correlation among the noise.

Could the differences in the red and blue distributions be explained through the similarities in global structural architecture among OMBB proteins? To investigate this, we selected an improved reference set of analogous proteins from the PDB. The folds most similar in structure to OMBBs are the lipocalin-like and streptavidin-like β-barrels (SCOP IDs b.60 and b.61.1). Figure 8.8C shows the results of the comparison of OMBBs with all proteins in SCOP25_1.75 from these two groups. The points lie well within the original blue distribution (with Pearson correlation 0.04), confirming the previous result.

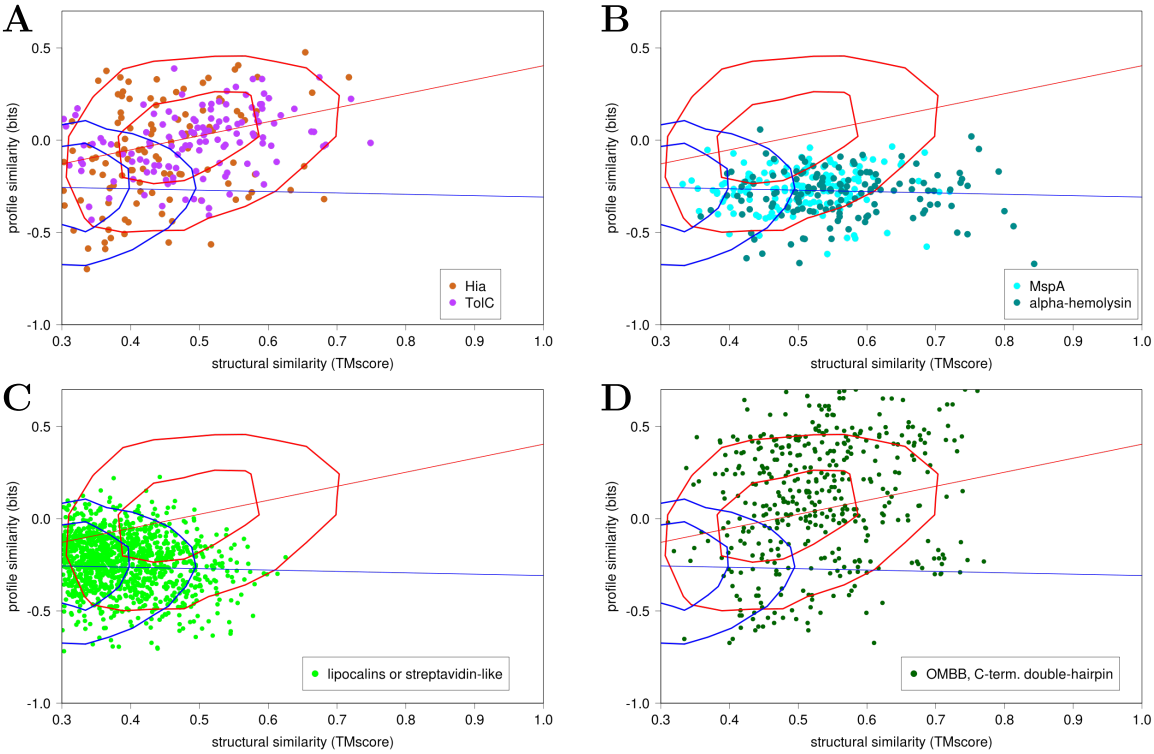


Figure 8.8.: Profile-profile and structural similarity scores between all double hairpins from single-chain bacterial OMBBs and all other double hairpins from OMBBs (red contour plot, homolog), and between double hairpins and all proteins in the PDB without OMBBs (blue contour plot, analog). (A-D) Highlighted are all hits between double hairpins from single-chain OMBBs and double hairpins from a special group of proteins: (A) multichain OMBBs (*Hia* and *TolC*), (B) non-homologous, atypical TMBBs with a OMBB-like structure (*MspA* and *α-hemolysin*), (C) lipocalins and streptavidin-like proteins (SCOP IDs b.60 and b.61.1, similar in structure to OMBBs), and (D) double hairpins from OMBBs containing the last C-terminal β-strand.

To clarify the relationship between the multi-chain OMBBs *Hia* and *TolC* and the single-chain OMBBs, we compared their double hairpins with the double hairpins from single-chain OMBBs. The resulting 2D score distributions in Figure 8.8A are in good agreement with the red distribution, identifying both proteins as members of the large superfamily of canonical OMBBs.

But could the sequence similarities between OMBBs be explained by similar constraints through being embedded in a membrane? To address this question, we derived a better reference score distribution using the atypical TMBBs α*-hemolysin* and *MspA*, which can be assumed to be unrelated to the canonical, single-chain, bacterial OMBBs. Since both these proteins possess only a single $\beta\beta$ hairpin in each chain, we concatenated two identical hairpins to generate double hairpins (see 7.3). We compared these two double hairpins with all double hairpins from canonical OMBBs in the same way as before. The resulting distribution of sequence and structure similarity scores is shown in Figure 8.8B. Clearly, the new reference distribution lies just about the horizontal regression line from the previous reference distribution, confirming the previous results.

The mechanisms of membrane insertion certainly differ between the canonical OMBBs and the atypical TMBBs. The functional requirement of membrane insertion can induce restraints that might lead to similarities in sequence. If this was the explanation for the observed similarities, we would expect the sequence similarity between OMBBs to be independent of structural similarity, just as we observe for the score distributions of analogous matches (Fig. 8.7 and Fig. 8.8B). What we find, however, is a clear correlation of sequence with structural similarity (Fig. 8.7). The common origin and subsequent divergence of bacterial OMBB hairpins therefore presents the most plausible explanation.

9. Discussion

In our first approach to investigate the evolutionary origin of OMBBs, we demonstrated that 22 out of 23 OMBBs of known structure can be multiply linked to a core group at low false positive rates, by direct comparison of their profile HMMs and by an exhaustive, transitive, sequence search method. If the observed similarities between OMBB profile HMMs were caused by structurally induced sequence convergence, we would expect our transitive searches to pick up sequences of other proteins that are similar in their hydrophobicity patterns and structures, such as lipocalins or cytosolic β-barrels. These again should be able to detect yet other folds similar to them, and so on. In contrast to structural similarity, homology is Boolean and transitive (Sadreyev et al., 2009) and as such is much better able to explain the low error rates and near-completeness of our cluster maps.

In the second approach, clear repeat patterns were detected within most sequences of *bona fide* OMBBs for the first time. The repeating sequence unit coincides well with the $\beta\beta$ structural repeat unit. We have previously found repeat patterns in many other structurally repetitive folds, such as $(\beta\alpha)_8$ barrels, which possess a $(\beta\alpha)_2$ repeat unit (Söding et al., 2006a). Our results support the notion that most domains which show residual sequence similarities between structural repeats originated through repeat amplification.

Third, we showed that the scenario of structure-induced sequence convergence cannot be invoked to explain the observed sequence similarities. At the same structural similarity, $\beta\beta$ hairpins from single-chain OMBBs are clearly more similar in sequence among themselves than they are to hairpins from the atypical TMBBs, which are structurally very similar to the canonical OMBBs and are subject to the same constraints from the membrane.

We found evidence for the homology of all major groups of OMBBs from Gram-negative bacteria, including *TolC*, the trimeric autotransporters, and cyanobacterial OMBBs, and we could link these with the mitochondrial OMBBs *VDAC1* and *Sam50*. Since mitochondria and chloroplasts descended from endosymbiotic α-proteobacteria and cyanobacteria, we expect most or all eukaryotic OMBBs to be related to the large OMBB metafold (Alva et al., 2008) delineated in this study. In addition, 8 proteins from the Gram-positive *Clostridiae* are contained in our pooled map. This observation would nicely fit the recent hypothesis that Gram-negative bacteria arose from an ancestral Clostridium symbiotically engulfing an actinobacterium (Lake, 2009).

From a methodological perspective, the combined sequence and structure analysis represents an advance for protein evolutionary studies: It allows to distinguish between two opposing scenarios, convergent and divergent evolution, and to demonstrate that structure-induced sequence convergence can be neglected. Furthermore, it strengthens the confidence in the common belief that profile-profile comparison methods excel in distinguishing remotely homologous from structurally analogous proteins.

From an evolutionary perspective, our results demonstrate the importance of repeat amplification for the origin of protein folds. The common evolutionary origin of OMBBs therefore also lends further credence to the hypothesis that, when proteins began to take over many of the functions from RNA, protein domains arose by amplification and recombination from smaller peptide modules, which originally evolved as cofactors in the RNA world (Söding and Lupas, 2003). The ancestral OMBB $\beta\beta$ hairpin may well have been among this pool of ancient peptide modules from which the first proteins were formed.

Part III.

Appendix

A. Characteristics & Definitions

A.1. Length distributions of various databases

The UniProt database from March 29, 2011 consists of 14 423 061 sequences with an average length of 322.502 residues. The shortest sequence has a length of 2, the longest of 36 805 residues. The non-redundant (NR) database from NCBI (August 12, 2011) has 14 605 097 sequences with an average length 342.131. The shortest sequence has a length of 6, the longest of 36 805 residues. The test set for the runtime benchmark contains 100 random proteins from the NCBI NR database with an average length of 354.3. The shortest sequence has a length of 30, the longest of 2102 residues. Figure A.1 and Table A.1 shows the length distribution of these sets.

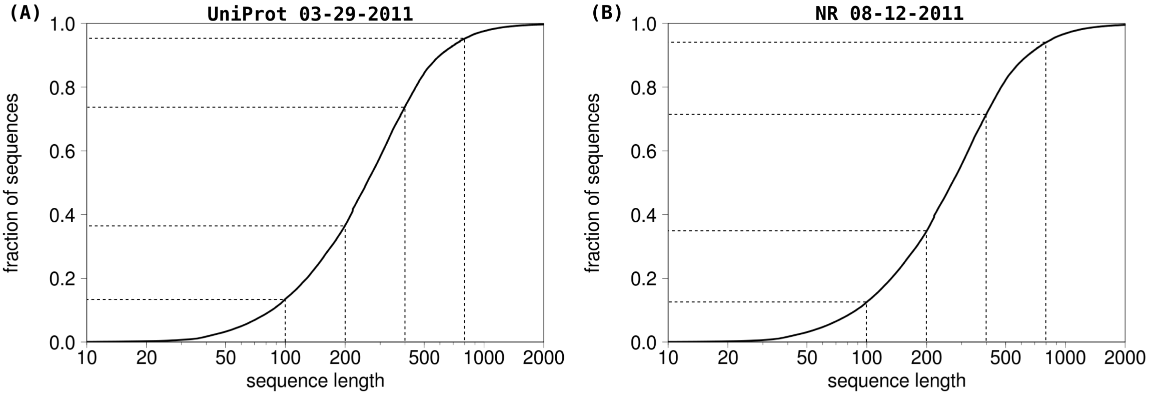


Figure A.1.: Length distribution of the UniProt database from March 29, 2011 and the NR database from August 12, 2011.

A.2. Effective number of sequences

The effective number of sequences (Neff) at column i of a multiple alignment is calculated on the subalignment M_i formed by all sequences with a residue in column i and by all columns with at most 10% terminal gaps in these sequences. A terminal gap is a gap that lies either to the left or to the right of the entire sequence. For each column j of M_i we calculate amino acid frequencies $p(j,x)$, using the Henikoff sequence weighting scheme. Then the number of

Table A.1.: Length distribution of the UniProt database from March 29, 2011, the NR database from August 12, 2011, and the random test set for the runtime benchmark

Length	number of sequences		
	UniProt	NR	random NR-seqs
< 50	448 610	430 158	3
$50 - 100$	1 465 504	1 371 763	11
$100 - 150$	1 697 832	1 655 875	14
$150 - 200$	1 632 375	1 593 513	6
$200 - 250$	1 702 938	1 663 011	14
$250 - 300$	1 407 702	1 388 825	6
$300 - 400$	2 274 856	2 326 880	17
$400 - 500$	1 534 458	1 573 495	12
$500 - 1000$	1 905 776	2 134 912	11
> 1000	353 010	466 665	6

effective sequences is

$$\mathrm{Neff}(i) = \exp\left(-\frac{1}{L_i}\sum_{j\in M_i}\sum_{x=1}^{20} p(j,x)\log p(j,x)\right) \tag{A.1}$$

Here, L_i is the number of columns in M_i.

A.3. False discovery rate

The false discovery rate (FDR) is the expected proportion of false positives (FP) from all seen hits. It is defined by:

$$FDR = \frac{FP}{FP+TP} \tag{A.2}$$

For example, if we have seen 1000 hits and 100 of them are false positives, then the false discovery rate is 10%. For homology detection the relevant FDR range is roughly between 0.1% and 10%, since an FDR of 10% corresponds to a *marginal FDR* of $\sim$ 50%. The marginal FDR measures what fraction of predictions with scores between a threshold S and some $S + \Delta S$ are correct. With other words, if a line in a benchmark plot cross the 10% FDR line, every second new match can be expected to be an false positiv.

B. Quality of HHblits confidence values

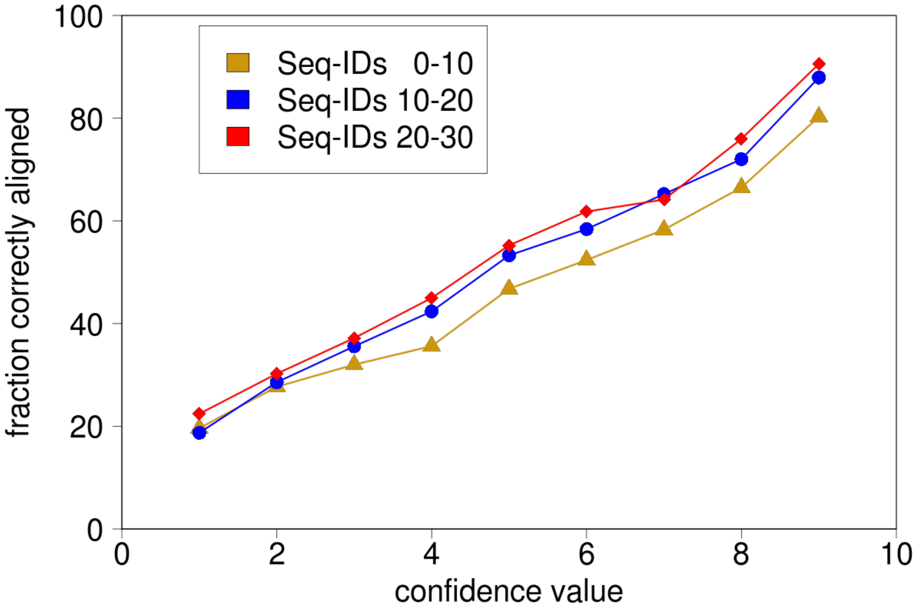


Figure B.1.: Relationship between HHblits confidence estimates from the maximum accuracy algorithm and the probability for a residue pair to be correctly aligned. The confidence values have an excellent correlation with the fraction of correctly aligned columns, and are nearl independent of the sequence identity between query and template sequences.

C. Pfam families with newly identified homologous structure

Table C.1.: List of 394 Pfam families for which no homologous template is known, HMMER3 has no match in the PDB below an E-value of $< 10^{-3}$ and for which HHblits has a match in the PDB database with E-value $< 10^{-3}$. In each row, the best HHblits match is given with its E-value and the coverage of the Pfam query. The last column specifies the HMMER3 E-value for the best match, an '-' indicates that HMMER3 has no matches up to the default reporting E-value threshold of 10.

Pfam-ID	**HHblits**			**HMMER**
	hit	**E-value**	**cov(%)**	**E-value**
PF03115	3hag	2e-101	53.24	-
PF11838	2xdt	1.1e-55	99.77	1
PF09562	2oa9	1.2e-54	98.85	-
PF10991	1je5	4e-54	96.02	-
PF04412	1c96	2.6e-53	83.81	0.006
PF09863	3iv3	1.2e-50	95.77	0.0048
PF12043	3c5n	1.9e-49	98.81	0.26
PF11047	3cxb	2e-49	73.10	0.049
PF07718	1r4x	3.8e-49	86.38	0.051
PF07632	2mas	2.1e-47	95.94	-
PF08010	2b3w	4.2e-45	96.58	0.14
PF11329	3eu8	9.5e-44	98.64	0.31
PF03813	1q79	1e-43	47.19	0.017
PF05316	3bbn	3.4e-41	90.71	-
PF09520	1wte	5.9e-40	97.33	-
PF12264	1qqp	7.5e-38	90.26	0.043
PF11161	2ra9	7.2e-37	77.46	-
PF09739	3f8t	2.8e-36	70.51	-
PF10963	3fgx	4.5e-36	100.00	0.0027
PF05864	2waq	5.8e-36	87.30	0.022
PF04486	1tuw	8.2e-36	77.39	-
PF05714	1w33	1e-35	86.26	0.078
PF05428	3kq4	1e-34	92.90	0.06
PF07588	1koe	2.1e-34	95.83	0.059
PF09536	2kii	1.1e-33	95.60	-
PF03662	1qw9	1.5e-33	99.38	-
PF11443	2nvo	1.6e-33	92.88	-
PF05538	2odj	1.7e-33	95.05	-
PF07520	1yuw	2.9e-33	58.91	-
PF06124	2r41	4.1e-33	100.00	-
PF05291	2ilr	4.7e-33	60.86	0.073
PF06045	1nkg	2.3e-31	92.65	-
PF06787	2a9s	4.6e-31	99.38	-
PF10023	1z5h	8.3e-31	93.51	0.061
PF05986	3ghm	1.4e-30	100.00	-
PF05482	1tr2	4e-30	85.30	-
PF11686	1se7	1.6e-29	100.00	-
PF07307	3nf2	1.8e-29	80.95	0.027
PF07528	1px5	2e-29	94.19	0.46
PF03254	2de0	2.4e-29	74.48	0.5
PF07395	1lrz	5.9e-29	98.86	-
PF10287	3iln	6.3e-29	91.81	-
PF01531	2hhc	7.1e-29	92.88	0.12
PF06074	3kdr	1.8e-28	56.66	0.14
PF06128	1yyh	2e-28	99.65	0.17
PF02088	1dec	1.2e-27	100.00	0.011
PF05136	3kdr	1.2e-27	90.17	-
PF07756	2qc0	1.5e-27	97.69	-
PF04681	1z3q	1.8e-27	86.45	-
PF09865	2w7q	1.8e-27	96.79	-
PF08189	2b5b	3e-27	94.87	0.031
PF10770	2plg	5.3e-27	82.88	0.014
PF11841	3dad	6e-27	99.36	-
PF06245	1vk1	6.1e-27	52.51	0.78
PF05060	1fo8	7.7e-27	75.42	0.062
PF12243	3d9j	1.1e-26	99.27	-
PF09960	2w3z	1.6e-26	43.59	1.2
PF10962	1v9m	2.1e-26	87.63	-
PF07014	2nw8	2.3e-26	95.20	-
PF10100	3c7a	2.7e-26	92.56	0.25
PF04841	1got	7.2e-26	70.80	-
PF07608	2yzy	1e-25	96.71	-
PF07379	3ci0	1.3e-25	79.43	-
PF03687	3bry	3.2e-25	87.35	0.11
PF07592	3hot	3.6e-25	93.57	-
PF05651	2a2l	7.2e-25	82.22	0.0057
PF12260	2acx	1.1e-24	89.76	-
PF06230	1zcz	1.5e-24	69.67	-
PF05213	1vgj	3.9e-24	66.67	0.0084
PF11768	1vyh	4.8e-24	61.72	0.097
PF04551	1tx2	4.9e-24	71.68	0.07
PF05550	2wur	5.5e-24	74.40	-
PF03290	1th0	7.4e-24	53.85	1.5
PF09927	2jxp	8.9e-24	99.15	6.2
PF09807	3bs4	9.5e-24	97.60	0.087
PF06199	2k4q	1.1e-23	100.00	0.013
PF08472	1tp6	1.2e-23	84.21	-
PF08553	1got	1.7e-23	42.52	0.15
PF11340	3cz8	1.8e-23	86.14	0.17
PF09892	3na6	2.3e-23	84.54	0.091

Table C.1 continue

Pfam-ID	**HHblits**			**HMMER**
	hit	**E-value**	**cov(%)**	**E-value**
PF09674	1kea	5.4e-23	81.06	0.012
PF05551	1a73	5.6e-23	52.24	0.91
PF08695	2ciu	7.4e-23	76.56	0.053
PF04405	2k5e	7.9e-23	98.21	0.0013
PF10179	1fnh	8.7e-23	95.62	0.66
PF02677	1wy5	9.2e-23	94.12	9.9
PF04708	3guv	1e-22	62.02	-
PF07006	2k3d	1.2e-22	66.40	-
PF10250	2hhc	1.8e-22	94.17	0.099
PF11813	3h2d	1.9e-22	71.67	-
PF08928	2fef	2.3e-22	100.00	0.029
PF09859	3itq	2.5e-22	83.73	-
PF10107	3fov	8.5e-22	48.75	0.89
PF09843	2zws	8.6e-22	73.60	-
PF08470	2vxr	1.2e-21	59.88	-
PF11017	2hcy	2.4e-21	98.75	0.061
PF04937	2bw3	2.6e-21	99.35	0.21
PF10222	1h54	2.7e-21	56.50	-
PF03336	2jig	2.9e-21	46.15	0.23
PF10677	2f1c	3.4e-21	97.85	0.48
PF06420	1h2i	3.9e-21	56.50	-
PF06674	3dtd	4.7e-21	46.13	3
PF09796	2fyu	7.8e-21	87.10	0.33
PF06951	1lwb	1e-20	50.56	-
PF06437	2fue	1.2e-20	64.90	-
PF10748	2ivw	2.2e-20	72.39	-
PF07894	1byr	2.7e-20	58.80	0.0045
PF00609	2bon	4e-20	100.00	-
PF09517	1yd6	5.6e-20	69.01	0.77
PF11039	2vzy	5.9e-20	98.01	0.11
PF03351	1d7b	9.8e-20	96.80	0.0047
PF12541	1ogo	1e-19	79.50	-
PF11680	3k44	1.2e-19	63.64	0.9
PF12439	1v7w	1.5e-19	99.55	-
PF05176	3gkn	2e-19	62.70	-
PF09810	3l0a	2.1e-19	54.39	0.011
PF10738	3lyd	2.3e-19	55.51	0.0022
PF05046	2ogh	2.7e-19	88.89	0.043
PF05342	3n6z	3.4e-19	48.98	-
PF10288	1ni5	3.4e-19	98.95	0.14
PF09778	3erv	4.4e-19	99.12	0.0024
PF04788	3bk5	5.1e-19	84.50	-
PF03214	1qg8	6.2e-19	34.00	-
PF06544	2iyg	6.3e-19	93.42	0.39
PF11854	2guf	7.5e-19	82.01	-
PF01973	2p2v	1e-18	85.88	0.0029
PF07845	1m2d	1e-18	78.95	0.012
PF08885	1ivn	1.1e-18	79.34	0.051
PF08642	1jbi	1.4e-18	85.27	3.5
PF07607	1z5h	1.4e-18	96.00	0.39
PF06241	1lnq	1.4e-18	78.16	-
PF07076	2qcp	1.4e-18	83.54	0.26
PF10365	3km5	1.5e-18	89.51	0.49
PF07959	1yp2	1.5e-18	65.83	0.38
PF10934	2ia7	1.7e-18	89.32	-
PF02411	2h3o	3.5e-18	54.78	-
PF10012	3h96	3.7e-18	80.00	0.028

Pfam-ID	**HHblits**			**HMMER**
	hit	**E-value**	**cov(%)**	**E-value**
PF07115	2ia7	4.1e-18	90.09	0.0018
PF07293	3i9v	6.3e-18	97.44	-
PF06477	2ag4	7.4e-18	97.32	-
PF06021	1sqh	1.1e-17	99.51	-
PF03018	2brj	1.2e-17	77.78	0.0043
PF04114	3k9t	1.3e-17	44.98	-
PF05565	2p2u	1.3e-17	74.84	0.016
PF07905	2ioj	1.3e-17	91.06	0.012
PF11751	3bry	1.4e-17	89.78	-
PF09565	2c1l	1.5e-17	61.20	-
PF08734	2zbc	1.8e-17	82.42	0.0045
PF11019	3mc1	3.2e-17	75.37	0.0029
PF06044	2jne	3.4e-17	18.43	-
PF04865	3h2t	5e-17	74.60	-
PF12362	2aya	5.2e-17	84.62	-
PF05991	1exn	6.8e-17	98.14	-
PF06622	1o9y	7.3e-17	24.59	-
PF08521	2kse	1.3e-16	97.95	0.05
PF08480	1ru4	1.4e-16	99.46	-
PF03302	1yy9	1.8e-16	77.92	-
PF07506	1zx4	2.1e-16	98.83	0.0047
PF01185	2fmc	5.1e-16	79.63	3.1
PF10087	2iw1	6.2e-16	95.65	0.057
PF11814	3erv	6.2e-16	90.00	0.15
PF03452	1xhb	7.3e-16	93.31	-
PF10703	2p8g	8.9e-16	30.65	0.12
PF02413	2kz6	1.2e-15	57.14	0.71
PF07327	1wqj	1.3e-15	55.14	0.35
PF11356	2ivw	1.5e-15	54.17	0.22
PF12055	1k1x	2.3e-15	58.25	-
PF09337	3nnq	5.1e-15	100.00	0.022
PF08424	3dss	5.1e-15	90.61	0.031
PF08170	3gir	8.5e-15	88.78	0.046
PF08379	3isr	1.1e-14	100.00	-
PF06805	3dwg	1.3e-14	42.70	-
PF09363	1umd	2.4e-14	72.41	-
PF10826	2fe3	2.7e-14	85.19	0.46
PF07611	3bma	3.4e-14	80.00	0.021
PF08757	3dnu	3.5e-14	58.54	-
PF07461	1tvg	4.2e-14	31.39	0.029
PF09941	1vet	7.7e-14	91.67	-
PF10141	2zxr	8.4e-14	75.13	0.11
PF09366	2pcs	9.6e-14	93.71	-
PF05272	2dhr	1e-13	64.00	1.2
PF07618	1y6u	1.2e-13	98.25	0.46
PF11824	2okx	1.3e-13	77.90	0.038
PF10483	3bs4	3.4e-13	79.18	-
PF10029	3c12	3.5e-13	81.51	0.095
PF11863	3hxl	4e-13	95.55	0.016
PF06147	3g27	4.2e-13	42.93	0.1
PF10743	1y6u	4.2e-13	70.93	0.047
PF10246	1k0r	4.3e-13	82.86	-
PF11288	3fak	5.7e-13	66.99	0.0043
PF05227	1vls	8.2e-13	97.10	0.17
PF11845	3b42	8.3e-13	59.88	2.2
PF10302	2bps	8.8e-13	34.29	0.019
PF11312	3mgg	9.4e-13	55.44	-

Table C.1 continue

Pfam-ID	**HHblits**			**HMMER**
	hit	**E-value**	**cov(%)**	**E-value**
PF11071	1s2d	1.8e-12	99.29	0.3
PF10037	1xi4	2.3e-12	75.06	-
PF04781	1elw	2.9e-12	96.72	0.27
PF09530	2i71	4.1e-12	74.59	0.023
PF07800	3knv	4.9e-12	80.62	0.1
PF08156	3id6	6.1e-12	100.00	2.8
PF06381	3kdr	8.8e-12	95.51	0.05
PF04244	2wq7	9.2e-12	69.78	-
PF10127	3c18	1.2e-11	79.20	-
PF08130	1w9n	1.2e-11	51.79	-
PF11959	3hft	1.4e-11	84.09	0.27
PF02066	1m0j	1.5e-11	51.85	3.2
PF04986	1omh	1.6e-11	33.68	-
PF04082	2veq	1.6e-11	29.30	-
PF10138	3ibz	1.6e-11	78.49	0.14
PF06075	2b29	2.1e-11	20.55	-
PF03490	2plc	2.5e-11	72.55	-
PF01927	3ga8	2.9e-11	37.58	0.035
PF09345	1h4x	2.9e-11	84.00	0.13
PF11761	3eeq	2.9e-11	100.00	0.1
PF06881	2e31	3.4e-11	70.09	0.011
PF04492	3e6c	3.5e-11	82.00	0.02
PF06890	2p5z	4.2e-11	46.03	-
PF08303	1yj5	4.7e-11	98.82	0.011
PF03281	1px5	6.3e-11	97.49	0.12
PF08497	2yxb	1.5e-10	47.75	-
PF10030	2jyx	1.5e-10	63.83	-
PF11997	3c48	1.7e-10	99.64	-
PF02697	3fmt	1.7e-10	86.67	0.0022
PF02474	1m4i	2.1e-10	73.10	-
PF12226	3iyo	2.5e-10	43.04	-
PF10567	1l3k	2.7e-10	69.36	-
PF09826	1fwx	4.3e-10	81.37	0.014
PF03158	2xeh	5.9e-10	75.65	-
PF08371	3hsi	6.6e-10	86.42	-
PF02666	2gpr	9.4e-10	90.23	-
PF06676	2waq	1.4e-09	37.86	0.59
PF06322	3e7l	1.5e-09	67.19	0.09
PF08685	1z3u	1.9e-09	24.00	-
PF07508	2r0q	1.9e-09	56.14	-
PF09317	2z1q	2e-09	48.75	-
PF07328	2ba3	2.2e-09	27.89	-
PF10908	1zat	2.9e-09	75.70	-
PF10474	2fji	3e-09	96.58	-
PF02521	3jty	3.5e-09	57.42	-
PF08499	3g4g	3.7e-09	88.52	0.025
PF06669	3d9x	3.8e-09	97.14	-
PF11954	1ei5	4e-09	63.87	0.028
PF07202	1h3i	4.5e-09	71.98	0.22
PF12010	1j1n	5.1e-09	94.20	0.12
PF04936	3hot	7.7e-09	78.49	8.1
PF08116	1c6w	7.8e-09	87.10	0.017
PF12000	3fro	8.5e-09	69.59	-
PF11766	1n67	8.6e-09	97.18	-
PF11711	2qv7	1.2e-08	32.09	-
PF06883	1twf	1.7e-08	100.00	2
PF08074	1ofc	1.7e-08	41.07	1.3

Pfam-ID	**HHblits**			**HMMER**
	hit	**E-value**	**cov(%)**	**E-value**
PF08465	1p6x	2.1e-08	96.97	-
PF05263	2o8x	2.5e-08	48.89	0.095
PF12128	1w1w	3.2e-08	6.19	0.0075
PF09889	1lv3	3.3e-08	46.55	0.059
PF04450	1z5h	4.3e-08	86.00	0.023
PF11308	2zxq	4.5e-08	49.71	-
PF09597	2e8n	4.7e-08	98.25	2.3
PF09824	2p4w	5.3e-08	78.75	0.0017
PF10780	1s3a	5.3e-08	100.00	-
PF11658	3lxq	6.1e-08	58.70	-
PF06011	1nep	7.2e-08	22.92	-
PF04407	3dcm	8.4e-08	95.43	0.0061
PF11853	2odj	8.5e-08	69.22	0.049
PF09582	1p90	9.6e-08	50.46	-
PF07610	2qsv	1e-07	100.00	0.015
PF11325	2vw9	1.2e-07	98.85	-
PF10686	2nx2	1.2e-07	88.73	0.017
PF01439	2kak	1.3e-07	93.67	0.34
PF08737	2fau	1.3e-07	77.88	-
PF05380	1rw3	1.7e-07	86.67	-
PF07699	2hey	1.9e-07	100.00	0.0045
PF06378	1h2i	2.5e-07	80.50	6.1
PF05610	2apn	3.6e-07	89.47	0.19
PF10758	3hxl	4.1e-07	99.45	-
PF04189	1yb2	4.4e-07	77.70	-
PF09855	2k4x	5.5e-07	82.81	0.22
PF06355	1gwy	6.1e-07	76.52	-
PF05895	2fl8	6.6e-07	28.31	-
PF04917	1oqw	6.8e-07	15.71	0.0036
PF04572	2vk9	7e-07	80.00	0.087
PF10144	3b42	7.5e-07	54.29	0.061
PF08405	3i86	8.4e-07	12.01	-
PF07087	1lwb	9.7e-07	77.17	-
PF09970	2fcl	9.7e-07	77.30	-
PF06977	1npe	1e-06	89.81	0.0024
PF07429	2gek	1.1e-06	80.06	-
PF10349	2hth	1.3e-06	32.41	-
PF10116	3e20	1.3e-06	98.57	-
PF08417	3gke	1.3e-06	59.43	-
PF10711	2vxz	1.9e-06	82.65	0.014
PF12303	2wg3	2.1e-06	61.70	0.0061
PF07813	3epv	2.2e-06	88.46	0.0022
PF10373	1ya0	3.5e-06	72.60	-
PF02681	2ipb	4.2e-06	90.54	0.27
PF11897	2gj4	4.2e-06	42.86	0.008
PF10987	3h35	4.5e-06	72.65	0.1
PF07617	3ia8	4.7e-06	98.18	-
PF03345	2gk3	4.9e-06	54.17	-
PF03850	3ibs	5.2e-06	80.92	0.011
PF07919	2icn	5.3e-06	62.70	0.069
PF10781	1whg	5.3e-06	98.39	-
PF07107	3ec9	5.5e-06	69.39	0.0012
PF05782	1kxp	5.5e-06	52.22	-
PF12578	1lw3	6.2e-06	66.29	0.29
PF04377	3gkr	6.4e-06	95.35	-
PF11903	1baz	6.6e-06	47.95	-
PF12073	1pjr	6.8e-06	94.23	0.26

Table C.1 continue

Pfam-ID	**HHblits**			**HMMER**
	hit	**E-value**	**cov(%)**	**E-value**
PF04312	1hjr	8.3e-06	76.98	0.025
PF07480	3epv	8.6e-06	94.74	0.58
PF12525	1v9n	8.7e-06	86.67	0.28
PF04413	1vgv	8.8e-06	86.34	-
PF05091	3fqi	9.4e-06	53.89	0.39
PF10367	1chc	9.7e-06	29.36	6.2
PF06353	2fph	1e-05	36.81	-
PF06239	1xi4	1e-05	65.67	-
PF08192	1hpg	1.1e-05	13.75	-
PF08579	1xi4	1.2e-05	82.50	-
PF05510	1u2c	1.3e-05	39.78	-
PF11833	1faf	1.3e-05	22.06	3.9
PF06823	2l1s	1.4e-05	80.33	-
PF09984	3b42	1.5e-05	93.29	-
PF09352	2qgp	1.6e-05	43.68	-
PF06375	2g30	1.9e-05	34.78	-
PF03406	1h6w	2.1e-05	90.70	-
PF12340	3ly5	2.1e-05	77.73	0.036
PF10952	2fbn	2.2e-05	88.03	-
PF10240	2qp2	2.2e-05	37.80	-
PF02754	3cf4	2.5e-05	80.95	0.91
PF10941	1uf3	2.5e-05	55.08	0.054
PF05689	1f00	2.5e-05	99.45	-
PF01941	2p02	2.6e-05	91.86	0.62
PF11839	1jcd	2.7e-05	39.76	-
PF06956	1xmx	2.7e-05	75.94	-
PF07585	2x55	2.8e-05	62.71	-
PF06448	1lsh	3.2e-05	39.33	-
PF10865	1ilo	3.2e-05	51.28	1
PF10505	3fqi	3.5e-05	72.90	-
PF11865	2qk1	3.9e-05	96.89	0.65
PF12222	1pgs	4e-05	69.09	0.21
PF00746	2ww8	4.1e-05	92.50	0.36
PF09759	1xqr	4.9e-05	72.63	1.2
PF11834	2dnf	5.1e-05	98.48	5.4
PF10126	3dfe	5.5e-05	92.86	0.068
PF07878	1nla	5.7e-05	92.00	-
PF07505	3c8f	7.2e-05	81.89	0.1
PF04155	1yo3	7.6e-05	72.97	-
PF10904	1j8b	7.6e-05	63.37	-
PF10706	2fcl	8.7e-05	88.51	-
PF01963	2g5g	8.9e-05	97.76	0.32
PF11336	2o4v	9.3e-05	77.48	0.014
PF03249	1p4t	9.4e-05	17.98	-
PF04599	1rxw	9.8e-05	65.00	-
PF01696	1pcl	0.0001	49.61	-
PF06702	1cja	0.00011	54.02	-
PF10122	2jr6	0.00013	80.39	10
PF11112	1z4h	0.00015	86.84	-
PF11849	3e0y	0.00016	94.77	-
PF06904	1lbu	0.00018	54.82	0.015
PF05918	1b3u	0.00019	60.90	0.0019
PF09854	2qgp	0.00019	23.82	0.021
PF10497	1wil	0.00025	61.76	0.71
PF07802	2k3j	0.00028	81.43	0.64
PF04305	3chh	0.00028	73.09	0.092
PF08736	2i1j	0.00028	51.06	-
PF06974	2jgp	0.0003	98.04	-
PF08288	3fro	0.0003	82.22	-
PF00242	1wz4	0.00031	7.33	-
PF09538	1vd4	0.00036	23.02	2.9
PF10673	3lub	0.00037	66.21	-
PF12215	2cqs	0.00038	63.13	-
PF11379	2cqy	0.00038	22.10	-
PF03258	2w7a	0.00045	46.67	0.094
PF10115	2kon	0.00046	76.34	-
PF08498	1wg8	0.00049	76.12	-
PF05444	3laq	0.00049	87.82	2.9
PF05869	3lkd	0.00055	81.40	0.06
PF11001	1wij	0.00058	65.61	0.0013
PF10309	3d45	0.00063	96.67	0.036
PF04049	3kae	0.00065	86.05	-
PF11006	2pxg	0.00067	87.21	1.6
PF07295	1lko	0.00068	25.85	0.096
PF10407	2ns5	0.00069	94.67	2.9
PF10165	1xm9	0.00073	77.28	-
PF08749	2plg	0.00074	92.41	-
PF04904	1rg6	0.00076	78.05	0.13
PF07409	2ia7	0.00078	63.25	0.049
PF12416	2dmh	0.00085	97.09	-
PF09576	1v54	0.00086	91.23	-
PF08004	2cob	0.00086	44.27	-
PF09415	1b67	0.00087	91.78	0.028
PF09894	1iru	0.0009	92.23	4.3
PF07855	2vfx	0.00091	95.73	0.16
PF10790	2al3	0.00095	92.11	0.019

D. HHblits Userguide

Quick guide to HHsearch/HHblits

Version 2.0.0 (June 2011)

©Michael Remmert & Johannes Söding

For bug reports, questions, or comments please contact johannes@soeding.com

HHsearch/HHblits is a software suite for detecting remote homologues of proteins and for generating high-quality alignments for homology modeling and function prediction. HHsearch is based on the pairwise comparison of profile hidden Markov models (HMMs). HHblits builds on top of HHsearch and achieves almost the same sensitivity as HHsearch at up to 1000 times the speed, by running a very fast profile-profile comparison prefilter before the full pairwise comparison of HMMs. HHblits can be used in an iterative search mode in combination with an HMM-encoded version of the UniProt or nr datbases. HHblits is faster than PSI-BLAST yet it achieves much higher sensitivity and generates alignments of much better quality.

In addition to the command line package described here, two web servers HHblits and HHpred are available at `http://toolkit.tuebingen.mpg.de` and `http://toolkit.lmb.uni-muenchen.de` that run the HHsearch and HHblits software and offer extended interactive functionality, such as options for checking query and template alignments, histogram views of alignments, building 3D models with MODELLER etc. In the latest CASP competition (2010), a fully automated version of HHpred based on HHsearch and HHblits was ranked best out of the 81 servers in template-based structure prediction, the category most relevant for biological applications, while having response times of minutes instead of days as most other servers (`http://predictioncenter.org/casp9/groups_analysis.cgi?type=server&tbm=on&submit=Filter`)

D.1. Obtaining HHblits and the databases

Binaries can be downloaded for Linux x86 (32bit), Linux AMD64, and Apple PPC OS X can be downloaded at

`ftp://toolkit.lmb.uni-muenchen.de/HHblits/`

In the subdirectory `databases/` different databases can be downloaded:

1	NR20	(c) Remmert & Soeding, based on NR, clustered to 20 % seq. id.
2	UniProt20	(c) Remmert & Soeding, based on UniProt, clustered to 20 % seq. id.
3	pdb70	(c) Remmert & Soeding, based on PDB, updated weekly
4	scop70	(c) Remmert & Soeding, based on SCOP, updated with SCOP

All databases consists of an HMM database, an A3M database and a CS-database for prefiltering.

D.2. Obtaining HHsearch and the databases

Binaries can be downloaded for Linux x86 (32bit), Linux AMD64, Windows x86 (now includes multi-threading support), Apple PPC OS X, and SUN Solaris can be downloaded at

```
ftp://toolkit.lmb.uni-muenchen.de/HHsearch
  or
ftp://ftp.tuebingen.mpg.de/pub/protevo/HHsearch/
```

In the subdirectory `databases/` many databases can be downloaded:

1*	pdb70	(c) J. Soeding, based on PDB, updated weekly
2*	scop70	(c) J. Soeding, based on SCOP, updated with SCOP
3*	PfamA	http://www.sanger.ac.uk/Software/Pfam/
4*	SMART	http://smart.embl-heidelberg.de/}, downloaded from NCBI site
5*	PfamB	based on ProDom, downloaded from Pfam site
6*	COG	http://www.ncbi.nlm.nih.gov/COG/new/
7*	KOG	http://www.ncbi.nlm.nih.gov/COG/new/
8*	CD/NCBI	http://www.ncbi.nlm.nih.gov/Structure/cdd/cdd.shtml
9	Panther	http://www.pantherdb.org/, from InterPro
10	TIGRFAMs	http://tigrblast.tigr.org/web-hmm/, from InterPro
11	PIRSF	http://pir.georgetown.edu/pirsf/, from InterPro
12	Superfamily	http://supfam.mrc-lmb.cam.ac.uk/SUPERFAMILY/, from InterPro
13	CATH/Gene3D	http://cathwww.biochem.ucl.ac.uk/latest/, from InterPro

The eight databases marked by an asterisc can be downloaded with HMMs in HHsearch format (*.hhm.tar files) and the *multiple sequence alignments (MSAs)* in A3M format (*.a3m.tar files). The *.hhm.tar and *.a3m.tar files untar into thousands of separate files, so before unnzipping and untarring, first create a directory for the database. Note that you can transform all A3M files to FASTA by using the `reformat.pl` script supplied with HHsearch:

```
> ./reformat.pl '*.a3m' .fas
```

For the other five databases, you can download the HMM models in HMMer format as *.hmm.tar files. The *.hmm.tar files untar into a single concatenated HMMer file. For theses databases, unfortunately no alignments are publicly available (info to the contrary is wellcome.)

D.3. Getting started

D.3.1. Overview of programs

`hhmake`	`Build an HMM from an input MSA in A2M, A3M, or FASTA format`
`hhblits`	`Iteratively search a database of HMMs with a query sequence or MSA`
`hhsearch`	`Search a database of HMMs with a query MSA or HMM`
`hhalign`	`Pairwise alignment of two HMMs/MSAs, dot plots etc.`
`hhfilter`	`Filter MSA by maximum sequence identity, coverage, etc.`
`reformat.pl`	`Reformat one or many MSAs`
`addss.pl`	`Add secondary structure information to a given MSA`
`create_db.pl`	`Build HHblits HMM- and/or A3M-databases`
`create_cs_db.pl`	`Build a HHblits prefilter database`
`buildali.pl`	`Build a PSI-BLAST MSA from a sequence or MSA, add sec. structure`
`alignhits.pl`	`Extract an MSA from a BLAST/PSI-BLAST output`
`hhmakemodel.pl`	`Generate MSAs or coarse 3D models from HHsearch results file`
`Align.pm`	`Perl package for local and global sequence-sequence alignment`

Call a program without arguments (or with -h) to get a more detailed description of its syntax.

D.3.2. Generate a MSA by an iterative HHblits search

The best way to generate a MSA (Multiple Sequence Alignment) is to use the HHblits software from this package. HHblits performs an iterative HMM-HMM comparison and has a runtime comparable to that of PSI-BLAST. This runtime is achived by a fast profile-profile prefilter based on SSE2 instructions, which reduces the number of comparisons in the time-consuming HMM-HMM comparison step.

HHblits needs for running the path to the two needed libraries (`context_data.lib` and `cs219.lib` in the `libs`-directory of this package) and the basename of the database (e.g. `databases/uniprot20`). The following database files must be present:

`BASENAME.cs219`	`database for the prefiltering step`

```
BASENAME_hhm_db              HMM databases
BASENAME_hhm_db.index        index table for HMM databases
```

When performing more than 1 search iteration or if an MSA should be generated, you also need the following databases:

```
BASENAME_a3m_db              A3M databases
BASENAME_a3m_db.index        index table for A3M databases
```

These default parameters for HHblits can be adapted in the configuration file `.hhdefaults` in the binary directory or they have to be specified by each calling of HHblits. All default options can be overridden on the command line.

The database scan with HHblits can be started by typing:

```
> ./hhblits -i query.seq -d databases/uniprot20
```

The complete search results, including the pairwise query-template alignments, are written to the default output file, `query.hhr`. If you are interested in the MSA, you have to add the `-oa3m` option:

```
> ./hhblits -i query.seq -d databases/uniprot20 -oa3m query.a3m
```

A special parameter `mact` can be used to choose the tradeoff between sensitivity and precision. With a low mact-value (e.g. `-mact 0.01`) you get very sensitive, but not so precise alignments, whereas a search with a high mact-value (e.g. `-mact 0.9`) results in shorter, but very precise alignments. The default value of mact in HHblits is 0.5.

If everything works, you will obtain an A3M-formatted MSA in file query.a3m. For obtaining other formats you can use the `reformat.pl` script, e.g. for reformatting the MSA to CLUSTAL format type:

```
> ./reformat.pl a3m clu query.a3m query.clu
```

The next step would be to add secondary structure information to this alignment. This can be done by using the script `addss.pl` (you have to fill in a few paths at the top of the script for needed binaries):

```
> ./addss.pl query.a3m
```

Now you can generate a hidden Markov model (HMM) from this MSA:

```
> ./hhmake -i query.a3m
```

D.3.3. Generate a MSA by buildali.pl

Another way uses the script `buildali.pl` from this package, but we recommend to use the more sensitive method HHblits. `buildali.pl` builds an A3M alignment with iterated PSI-BLAST. It checks after each round for each HSP (i.e. for each PSI-BLAST matched sequence fragment) if both its ends have a sufficiently high score per column (e.g. 1/6 bits/column) with the query sequence, or rather with a 'core profile' of sequences most similar to the query. It prunes the HSP ends that do not have sufficient similarity, thus largely suppressing the frequent problem of profile corruption coming from the ends of domains, coiled coil regions, or low complexity regions. In a representative all-against-all benchmark on SCOP20, `buildali.pl` was able to reduce the number of high-scoring false-positives by a factor of approximately five, while only slightly reducing sensitivity for very remote homologs. `buildali.pl` also automatically adds PSIPRED secondary structure prediction.

To get `buildali.pl` to run you have to fill in a few paths at the top of the script, for example to the BLAST directory, PSIPRED data and binaries directories etc. Some of the same paths also have to be inserted at the top of the script `alignhits.pl` which is called by `buildali.pl`. Finally, you need to have a non-redundant database like the nr from NCBI filtered to 90 and 70 percent (e.g. by using the program CD-HIT from Weizhong Lee, `http://cd-hit.org/`). You can call these databases nr90 and nr70, for example, and would set the $dbbase variable to "some_path/nr". The script then adds the "90" and "70" as needed. It will first search the nr90 until more than 50 sequences are found, whence it will switch to the nr70. To test `buildali.pl` you might as well set a link from the nr to nr90 and nr70, which means `buildali.pl` will search the entire nr (which is slower and a little less sensitive). Then start `buildali.pl` with your sequence:

```
> ./buildali.pl query.seq
```

If everything works, you will obtain an A3M-formatted MSA in file query.a3m as a result. (Make sure all paths are correct and your active shell is bash: `ln -s /bin/bash /bin/sh.`)

Now you can generate a hidden Markov model (HMM) from this MSA:

```
> ./hhmake -i query.a3m
```

D.3.4. Search with a profile HMM through a database

For searching with a profile HMM as query, you can either use HHblits or HHsearch. HHblits uses a prefilter and performs the HMM-HMM comparison only on a small subsets of HMMs that pass this prefilter. Therefore, HHblits has a dramatical shorter runtime by a small decrease in sensitivity (see Figure D.1).

Using HHblits follows the same syntax as described in the previous section with an A3M-alignment or an HMM-profile as input and a single iteration:

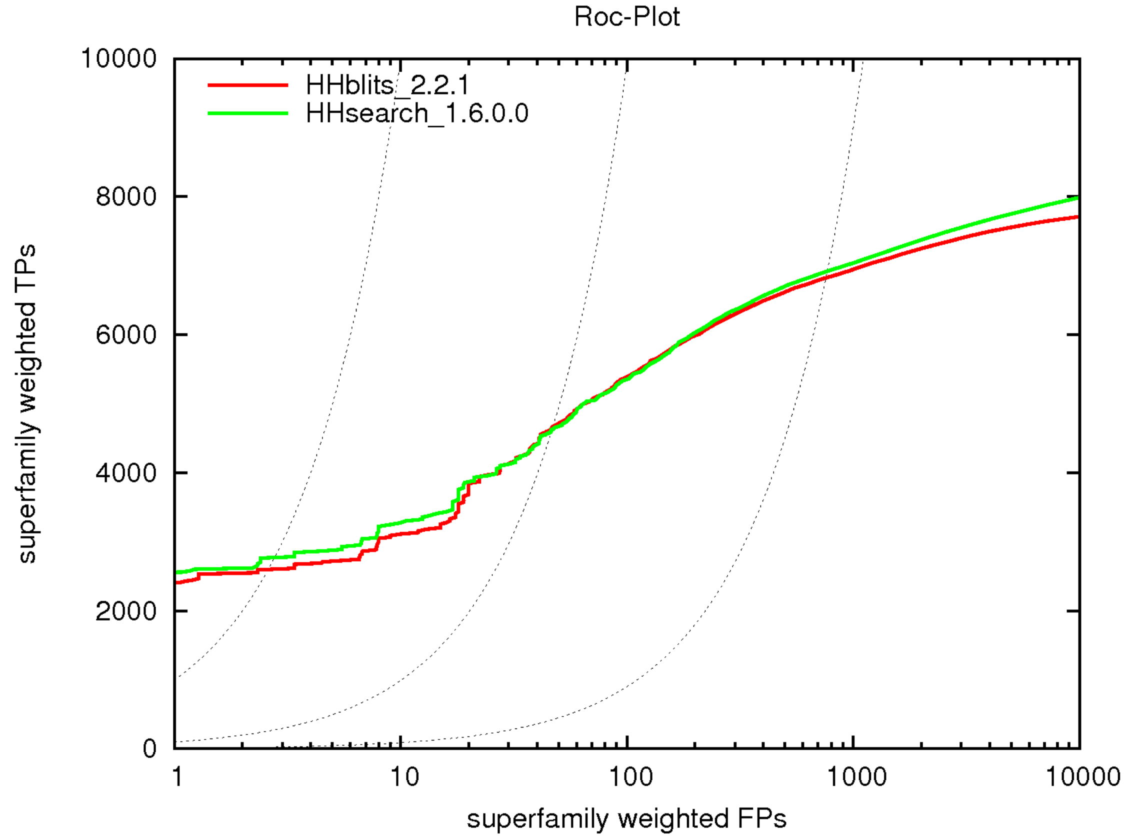


Figure D.1.: Benchmark of HHsearch and HHblits on a SCOP20 dataset.

```
> ./hhblits -i query_profile.a3m -d databases/uniprot20 -n 1
```

To start using HHsearch, untar a database first, e.g.

```
> cd scop70_1.72pre
> tar -xzvf scop70_1.72pre.hhm.tar.gz
```

The name `scop70_1.72pre` stands for 'SCOP domain database version 1.72 (pre-SCOP) filtered to 70% maximum sequence identity'.

Then, to generate a database file by concatenating all *.hhm files type (under LINUX)

```
> cat *.hhm > scop70_1.72pre.hhm
```

Test whether hhsearch works by typing

```
> ./hhsearch -i d1hxn__.hhm -d scop70_1.72pre.hhm
```

You should see a dot printed for every twenty HMMs processed until at the end hhsearch prints out a list with the best hits from the database. The complete search results, including the pairwise query-template alignments, are written to the default output file, `d1hxn__.hhr`. The format (HH results format, HHR) was designed to be easily parsable.

Instead of using the hmm file you can also take the a3m file for the scan. It is automatically converted to an HMM by HHsearch before starting the scan:

```
> ./hhsearch -i d1hxn__.a3m -d scop70_1.72pre.hhm
```

D.3.5. Building customized databases

HHblits databases

If you want to build your own database from a set of sequences, call

```
>   ./hhblits -i <seqfile> -o /dev/null -oa3m <MSA-file>
```

for every sequence in your database, as described in subsection 3.2. Default paramters are up to 2 search iterations at an E-value of 1E-3. These can be changed with the `'-n <int>'` and `'-e <float>'` options (Call `hhblits` without parameters for a complete list of options).

The next step is to add secondary structure prediction from PSIPRED (D. Jones, 1999) to the MSA. This can be done by the script `addss.pl` (When the sequence has a SCOP or PDB identifier as first word in its name, the script tries to add the DSSP states as well. You need to give the path to your local pdb or dssp directory for this to work.):

```
>   ./addss.pl <MSA-file>
```

Now you can use the two scripts `create_cs_db.pl` and `create_db.pl` to generate the needed databases for HHblits. In both scripts you have to adapt some path at the beginning.

The first script `create_cs_db.pl` generates the column state database needed for the prefilter step in HHblits. You can start it by typing:

```
> ./create_cs_db.pl -i <MSA-dir> -o databases/DB.cs219
```

The other script `create_db.pl` generates the HMM- and A3M-databases:

```
> ./create_db.pl -a3mdir <MSA-dir> -oa3m databases/DB_a3m_db
  -ohhm databases/DB_hhm_db
```

A list of additional options for these scripts can be retrieved by calling the scripts without parameters.

HHsearch databases

If you want to build your own database from a set of sequences, call

```
>   ./hhblits -i <seqfile> -o /dev/null -oa3m <MSA-file>
```

for every sequence in your database, as described in subsection 3.2. Default paramters are up to 2 search iterations at an E-value of 1E-3. These can be changed with the `'-n <int>'` and `'-e <float>'` options (Call `hhblits` without parameters for a complete list of options).

The next step is to add secondary structure prediction from PSIPRED (D. Jones, 1999) to the MSA. This can be done by the script `addss.pl` (When the sequence has a SCOP or PDB identifier as first word in its name, the script tries to add the DSSP states as well. You need to give the path to your local pdb or dssp directory for this to work.):

```
>   ./addss.pl <MSA-file>
```

Then generate an HHM file for each MSA by typing

```
> ls | grep "\.a3m\$" | xargs hhmake -i
```

if your MSAs have extension a3m. You can then concatenate your individual HMMs into your database:

```
> cat *.hhm > yourDB.hhm
```

(or, if maximum buffer size is exceeded,

```
> ls | grep "\.hhm\$" | xargs -i cat {} >> yourDB.hhm).
```

By default, the option -M first will be used. This means that exactly those columns of the MSAs which contain a residue in the query sequence will be assigned to Match / Delete states, the others will be assigned to Insert states. (The query sequence is the first sequence not containing secondary structure information.) Alternatively, you may want to apply the 50%-gap rule by typing `-M 50`, which assigns only those columns to Insert states which contain more than 50% gaps. The `-M first` option makes sense if your alignment can best be viewed as a seed sequence plus aligned homologs to reinforce it with evolutionary information. This is the case in the SCOP and PDB versions of our HMM databases, since here MSAs are built around a single seed sequence (the one with known structure). On the contrary, when your alignment represents an entire family of homologs and no sequence in particular, it is best to use the 50% gap rule. This is the case for Pfam or SMART MSAs, for instance. Despite its simplicity, the 50% gap rule has been shown to perform well in practice.

When calling hhmake, you may also apply several filters, such as maximum pairwise sequence identity (`-id <int>`), minium sequence identity with query sequence (`-qid <int>`), or miniumum coverage with query (`-cov <int>`). But beware of making your MSAs too restrictive, as this will lower the sensitivity for remote homologs.

D.3.6. Maximum Accuracy alignment algorithm

HHblits and HHsearch use a better alignment algorithm than the quick and standard Viterbi method to generate the final HMM-HMM alignments. Both realign all diplayed alignments in a second stage using the more accurate Maximum Accurracy (MAC) algorithm (Durbin, Eddy, Krough, Mitchison: Biological sequence analysis, page 95; extension to HMM-HMM: Biegert, Lupas, and Söding (2008) *Bioinformatics.* **24**, 807–814.). The Viterbi algorithm is employed for searching and ranking the matches. The realignment step is parallelized (`-cpu <int>`) and typically takes a few seconds only.

Please note: Using different alignment algorithms for scoring and aligning has the disadvantage that the pairwise alignments that are displayed are not always very similar to those that are used to calculate the scores! This can lead to the confusing results where alignments of only one or a few residues length may have probabilities of 50% or more. In such cases, run the search again with the `-norealign` option, which will skip the MAC-realignment step. This will allow you to check if the Viterbi alignments are valid at all, which they will probably not be. The length of the MAC alignments can therefore give you additional information to decide if a match is valid. In order to avoid confusion for users of our HHpred server, the `-norealign` option is the default there, whereas for you pros who dare to use the command line package, realigning is done by default.

The posterior probability threshold is controlled with the -mact [0,1[option. This parameter controls the alignment algorithm's greediness. More precisely, the MAC algorithm finds the alignment that maximizes the sum of posterior probabilites minus mact for each aligned pair. Global alignments are generated with -mact 0, whereas -mact 0.5 will produce quite conservative local alignments.

The -global and -local options now refer to both the Viterbi search stage as well as the MAC realignment stage. With -global (-local), the posterior probability matrix will be calculated for global (local) alignment. When -global is used in conjunction with -realign, the mact parameter is automatically set to 0 in order to produce global alignments. In other words, both following two commands will give global alignments:

```
> ./hhsearch -i <query> -d <db.hhm> -realign -mact 0
> ./hhsearch -i <query> -d <db.hhm> -realign -global
```

The first version uses *local* Viterbi to search and then uses MAC to realign the proteins globally (since mact is 0) on a *local* posterior probability matrix. The second version uses *global* Viterbi to search and then realigns globally (since mact is automatically set to 0) on a *global* posterior matrix. To detect and align remote homologs, for which sometimes only parts of the sequence are conserved, the first version is clearly better. It is also more robust. If you expect to find globally alignable sequence homologs, the second option might be preferable. In that case, it is recommended to run both versions and compare the results.

D.3.7. How can I verify if a database match is homologous?

Here is a list of things to check if a database match really is at least locally homologous.

- Check probability and E-value: HHsearch can detect homologous relationships far beyond the twilight zone, i.e. below 20% sequence identity. Sequence identity is therefore not an appropriate measure of relatedness anymore. The estimated probability of the template to be (at least partly) homologous to your query sequence is the most important criterion to decide whether a template HMM is actually homologous or just a high-scoring chance hit. When it is larger than 95%, say, the homology is nearly certain. Roughly speaking, one should give a hit serious consideration (i.e. check the other points in this list) whenever (1) the hit has > 50% probability, or (2) it has > 30% probability and is among the top three hits. The E-value is an alternative measure of statistical significance. It tells you how many chance hits with a score better than this would be expected if the database contained only hits unrelated to the query. At E-values below one, matches start to get marginially significant. Contrary to the probability, when calculating the E-value HHpred does not take into account the secondary structure similarity. Therefore, the probability is a more sensitive measure than the E-value.

- Check if homology is biologically suggestive or at least reasonable: Does the database hit have a function you would expect also for your query? Does it come from an organism that is likely to contain a homolog of your query protein?

- Check secondary structure similarity: If the secondary structure of query and template is very different or you can't see how they could fit together in 3D, then this is a reason to distrust the hit. (Note however that if the query alignment contains only a single sequence, the secondary structure prediction is quite unreliable and confidence values are overestimated.)

- Check relationship among top hits: If several of the top hits are homologous to each other, (e.g. when they are members of the same SCOP superfamily), then this will considerably reduce the chances of all of them being chance hits, especially if these related hits are themselves not very similar to each other. Searching the SCOP database is very usefull precisely for this reason, since the SCOP family identifier (e.g. a.118.8.2) allows to tell immediately if two templates are likely homologs.

- Check for possible conserved motifs: Most homologous pairs of alignments will have at least one (semi-)conserved motif in common. You can identify such putative (semi-)conserved motifs by the agglomeration of three or more well-matching columns (marked with a '|' sign between the aligned HMMs) occurring within a few residues, as well as by matching consensus sequences. Some false positive hits have decent scores due to a similar amino acid composition of the template. In these cases, the alignments tend to be long and to lack conserved motifs.

- Check residues and role of conserved motifs: If you can identify possible conserved motifs: are the corresponding conserved template residues involved in binding or enzymatic function?
- Check query and template alignments!: A corrupted query or template alignment is the main source of high-scoring false positives. The two most common sources of corruption in an alignment are (1) non-homologous sequences, especially repetitive or low-complexity sequences in the alignment, and (2) non-homologous fragments at the ends of the aligned database sequences that are due to PSI-BLAST's greedyness. Check the query and template MSAs in an alignment viewer such as JalView or ALNEDIT.
- Realign with other parameters: change the alignment parameters. Choose global instead of local mode, for instance, if you expect your query to be globally homologous to the putative homolog. Try to improve the probability by changing the values for minimum coverage or minimum sequence identity. You can also run the query HMM against other databases.
- Try to use manualPSI-BLAST iterations (use FASTA in first round if you can) to try to find more distant homologs for your query alignment and jump-start HHsearch with the manually enriched alignment.
- Try out other structure prediction servers!: A list of servers can be found by Battey, J.N. *et al.* (2007) Automated server predictions in CASP7. *Proteins* **69**:68-82.
- Verify predictions experimentally: The ultimate confirmation of a homologous relationship or structural model is, of course, the experimental verification of some of its key predictions, such as testing the binding to certain ligands by binding assays, measuring biochemical activity, or comparing the knock-out phenotype with the one obtained when the putative functional residues are mutated.

D.4. HHsearch/HHblits output: hit list and pairwise alignments

D.4.1. Summary hit list

Do a search with the N-terminal domain of DNA polymerase beta against the SCOP domains:

```
> ./hhsearch -i d1tv9a1.hhm -d scop.hhm -cpu 2
Search results will be written to d1tv9a1.hhr
Query file is in HHM format
```

```
Read in HMM d1tv9a1 with 87 match states and effective number of sequences = 5.6
.................................................. 1000 HMMs searched
.................................................. 2000 HMMs searched
.................................................. 3000 HMMs searched
.................................................. 4000 HMMs searched
.................................................. 5000 HMMs searched
.................................................. 6000 HMMs searched
.................................................. 7000 HMMs searched
.................................................. 8000 HMMs searched
.................................................. 9000 HMMs searched
.................................................. 10000 HMMs searched
.................................................. 11000 HMMs searched
.................................................. 12000 HMMs searched
......
Fitting scores with EVD (first round) ...
Fitting scores with EVD (second round) ...
Realigning 50 query-template alignments with maximum accuracy (MAC) algorithm ...
..
Query         d1tv9a1 a.60.6.1 (A:5-91) DNA polymerase beta, N-terminal (8 kD)-domain {Human}
Match_columns 87
No_of_seqs    103 out of 203
Neff          5.6
Searched_HMMs 12156
Date          Sat Nov  3 08:40:24 2007
Command       hhsearch -i /data/hhpred/scop70_1.72pre/d1tv9a1.hhm -d scop.hhm

 No Hit                             Prob E-value P-value  Score    SS Cols Query HMM  Template HMM
  1 d1tv9a1 a.60.6.1 (A:5-91) DNA  100.0 4.6E-34 3.8E-38  202.8   9.4   87    1-87      1-87  (87)
  2 d1jmsa1 a.60.6.1 (A:148-242) T 100.0 7.3E-29   6E-33  174.7   9.7   85    1-86     11-95  (95)
  3 e2bcqa1 a.60.6.1 (A:252-327) D  99.9 1.9E-25 1.6E-29  156.2   6.9   76    7-83      1-76  (76)
  4 d1mun__ a.96.1.2 (-) Catalytic  94.7   0.062 5.1E-06   28.8   7.3   56   17-73     72-129 (225)
  5 d1rrqa1 a.96.1.2 (A:9-229) Cat  94.4   0.059 4.8E-06   28.9   6.7   57   16-73     69-127 (221)
  6 d1keaa_ a.96.1.2 (A:) Thymine-  93.8   0.057 4.7E-06   29.0   5.6   57   17-73     75-133 (217)
  7 e2bgwa1 a.60.2.98 (A:160-229)   93.3    0.06   5E-06   28.9   5.1   26   50-75     42-67  (70)
  8 d2abk__ a.96.1.1 (-) Endonucle  93.0    0.13 1.1E-05   27.1   6.3   44   31-74     85-130 (211)
  9 d1orna_ a.96.1.1 (A:) Endonucl  92.7    0.11   9E-06   27.5   5.6   44   31-74     86-131 (214)
 10 e1x2ia1 a.60.2.98 (A:2-69) ATP  92.5   0.033 2.7E-06   30.3   2.8   27   50-76     39-65  (68)
 11 d1kfta_ a.60.2.3 (A:) Excinucl  92.4   0.037   3E-06   30.0   2.9   26   50-75     31-56  (56)
 12 d1vdda_ e.49.1.1 (A:) Recombin  92.1   0.036   3E-06   30.1   2.5   40   45-84      3-43  (199)
 13 d1m3qa1 a.96.1.3 (A:136-325) 8  90.6    0.29 2.3E-05   25.2   5.9   51   22-73     61-123 (190)
 14 d1cuk_2 a.60.2.1 (65-142) DNA   90.3   0.092 7.6E-06   27.9   3.2   38   34-71     18-62  (78)
 15 e2a1jb1 a.60.2.98 (B:219-296)   90.2    0.21 1.7E-05   25.9   5.0   58   16-75     15-73  (78)
 16 d1ixra1 a.60.2.1 (A:63-135) DN  90.1   0.096 7.9E-06   27.8   3.1   37   35-71     20-63  (73)
 17 d1bvsa2 a.60.2.1 (A:64-134) DN  88.8    0.14 1.2E-05   26.9   3.2   37   34-70     18-61  (71)
 18 d1mpga1 a.96.1.3 (A:100-282) 3  85.8       1 8.6E-05   22.2   6.2   50   23-72     71-127 (183)
 19 d1dgsa1 a.60.2.2 (A:401-581) N  83.3    0.46 3.8E-05   24.1   3.5   27   51-77    137-163 (181)
 20 e2a1ja1 a.60.2.98 (A:837-898)   82.8    0.55 4.5E-05   23.7   3.7   50   25-77      9-58  (62)
 21 d1pu6a_ a.96.1.5 (A:) 3-Methyl  82.0    0.63 5.2E-05   23.4   3.8   23   51-73    118-140 (217)
 22 d1t4ga1 a.60.4.1 (A:5-64) DNA   76.6    0.28 2.3E-05   25.3   0.6   25   50-74     29-53  (60)
 23 d1b22a_ a.60.4.1 (A:) DNA repa  73.8    0.71 5.9E-05   23.1   2.1   24   50-73     40-63  (70)
 24 e1wuda1 a.60.8.1 (A:530-606) H  73.4     4.3 0.00036   18.8   6.0   59    8-69      3-62  (77)
 25 d1szpa1 a.60.4.1 (A:81-144) DN  68.2     1.5 0.00012   21.3   2.7   24   50-73     33-56  (64)
 26 d1jiha2 e.8.1.7 (A:1-389) DNA   67.2     1.1 9.3E-05   22.0   1.9   23   55-77    302-324 (389)
 27 d1pzna1 a.60.4.1 (A:35-95) DNA  67.0     1.2  0.0001   21.8   2.1   24   50-73     31-54  (61)
 28 d1ngna_ a.96.1.2 (A:) Mismatch  62.3     4.9 0.00041   18.5   4.4   41   31-77     76-116 (144)
 29 d1d8ba_ a.60.8.1 (A:) HRDC dom  61.9     5.3 0.00043   18.4   4.5   59   12-73      6-68  (81)
```

```
30 d1a77_1 a.60.7.1 (209-316) Fla  53.5     3.2 0.00027   19.5   2.3   24   56-81     20-43  (108)
31 d1lb2b_ a.60.3.1 (B:) C-termin  52.9      11 0.00087   16.7   4.8   25   51-75     36-60  (72)
32 d1doqa_ a.60.3.1 (A:) C-termin  50.8      12   0.001   16.4   4.9   37   37-75     18-62  (69)
33 e2bcqa2 a.60.12.1 (A:329-385)   48.0     3.7  0.0003   19.2   1.9   34   49-86      4-37  (57)
34 d1s1hm_ i.1.1.1 (M:) 70S ribos  44.6       3 0.00025   19.7   1.0   23   53-75     16-38  (131)
35 d1rxwa1 a.60.7.1 (A:220-324) F  44.3     6.5 0.00054   17.9   2.7   25   55-81     18-42  (105)
36 d1tv9a2 a.60.12.1 (A:92-148) D  42.5     5.3 0.00044   18.3   2.0   31   51-85      5-35  (57)
37 d1t94a2 e.8.1.7 (A:75-407) DNA  41.9     5.6 0.00046   18.2   2.0   18   55-72    277-294 (333)
38 d1im4a_ e.8.1.7 (A:) DinB homo  41.3     5.8 0.00048   18.1   2.0   18   55-72    185-202 (209)
39 e1ul1x1 a.60.7.1 (X:218-357) F  40.7     6.4 0.00053   17.9   2.2   13   57-69     2
...
```

The summary hit list that is written to the screen shows the best hits from the database, ordered by the probability of being a true positive (column 4: 'Prob'). The meaning of the columns is the following:

Column 1 'No':	Index of hit
Column 2 'Hit':	First 30 characters of domain description (from nameline of query sequence)
Column 3 'Prob':	Probability of target to be a true positive For the probability of being a true positive, the secondary structure score in column 7 is taken into account, together with the raw score in column 6 ('Score'). True positives are defined to be either globally homologous or they are at least homologous in parts, and thereby locally similar in structure. More precisely, the latter criterion demands that the MAXSUB score between query and hit is at least 0.1. In almost all cases the structural similarity will we be due to a global OR LOCAL homology between query and target.
Column 4 'E-value':	E-value and P-value are calculated without taking the secondary structure into account! The E-value gives the average number of false positives ('wrong hits') with a score better than the one for the target when scanning the datbase. It is a measure of reliability: E-values near to 0 signify a very reliable hit, an E-value of 10 means about 10 wrong hits are expected to be found in the database with a score at least this good.
Column 5 'P-value':	The P-value is just the E-value divided by the number of sequences in the database. It is the probability that in a PAIRWISE comparison a wrong hit will score at least this good.
Column 6 'Score':	Raw score, does not include the secondary structure score

Column 7 ’SS’: Secondary structure score This score tells how well the PSIPRED-predicted (3-state) or actual DSSP-determined (8-state) secondary structure sequences agree with each other. PSIPRED confidence values are used in the scoring, low confidences getting less statistical weight.

Column 8 ’Cols’: The number of aligned Match columns in the HMM-HMM alignment.

Columns 9,10: Range of aligned match states from query HMM

Columns 11,12: Range of aligned match states from target HMM

Column 14: Number of match states in target HMM

D.4.2. HMM-HMM pairwise alignments

The output file d1bpya1.hhr contains the same hit list plus the pairwise HMM alignments. One example is give here:

```
No 4
>d1mun__ a.96.1.2 (-) Catalytic domain of MutY {Escherichia coli} SCOP: d1muya_ d1kg5a_ d1kg2a_ d1kg6a_
Probab=94.69   E-value=0.062  Score=28.83  Aligned_columns=56  Identities=25%

Q ss_dssp             HHHHHHHHTTCCHHHHHHHHHHHHHHHH-HCCSCCC-CHHHHHTSTTCCHHHHHHHHHHH
Q ss_pred             HHHHHHHhcCCCCchHHHHHHHHHHHHH-hCCCCcc-ChHHHhhCCCcChHHHHHHHHHH
Q d1tv9a1          17 ELANFEKNVSQAIHKYNAYRKAASVIA-KYPHKIK-SGAEAKKLPGVGTKIAEKIDEFL   73 (87)
Q Consensus       17 eia~~~e~~~en~~rv~AYr~Aa~~l~--l~~~i~-----l~~lpgIG~~ia~~I~Ei~   73 (87)
                      ++..+..-.|=. .|.+...+++..|. .+...+. +.++|.+|||||+.+|..|.-+.
T Consensus       72 ~l~~~i~~~G~~-ka~~l~------i~-----g~ip~~~~eL~~LpGVG~kTA~~VL~~a  129 (225)
T d1mun__         72 EVLHLWTGLGYY-ARARNLHKAAQQVATLHGGKFPETFEEVAALPGVGRSTAGAILSLS  129 (225)
T ss_dssp             HHHHHHTTSCCT-HHHHHHHHHHHHHHHHHSTTSCCCSHHHHHTSTTCCHHHHHHHHHHH
T ss_pred             HHHHHHHhhhhh-HHHHHHHHHHHHHHHHHHcCCccccChHHHHhcCCCcHHHHHHHHHHh
Confidence             344433333332 25555666676654 4555554 46889999999999999998664
```

This is a typical example of local homology, detectable at both the sequence and the structural level, which is embedded in globally non-homologous structures with different overall folds. This sequence- and structure-similar motif, called ’helix-hairpin-helix’ (HhH), makes unspecific contacts with DNA and is described in [Doherty, Serpell, Ponting, NAR 1996]. See [Söding J and Lupas AN, Bioessays 2003] for a hypothesis relating to the pervasiveness of recurring homologous peptide fragments.

The pairwise alignment consists of one or more blocks with the following lines:

```
Q ss_dssp:      the query secondary structure as determined by DSSP (when available)
Q ss_pred:      the query secondary structure as predicted by PSIPRED (when available)
Q scop-id:      the query sequence
Q Consensus:    the query alignment consensus sequence
```

The predicted secondary structure states are shown in capital letters if the PSIPRED confidence value is between 0.7 and 1.0, for lower confidence values they are given in lowercase letters. With the option `'-ssconf'`, `'ss_conf'` lines can be added to the alignments which report the PSIPRED confidence values by numbers between 0 and 9 (as in versions up to 1.5).

The consensus sequence uses capital letters for well conserved columns and lower case for partially conserved columns. Unconserved columns are marked by a tilde ' '. Roughly speaking, amino acids that occur with >= 60% probability (before adding pseudocounts) are written as capital letters and amino acids that have >= 40% probability are written as lower case letters, where gaps are included in the fraction counts. More precisely, when the gap-corrected amino acid fraction

$$p_i(a) * N_eff(i)/(N_eff + 1)$$

is above 0.6 (0.4) an upper (lower) case letter is used for amino acid a. Here, $p_i(a)$ is the emission probability for a in column i, N_eff is the effective number of sequences in the entire multiple alignment (between 1 and 20) and $N_eff(i)$ is the effective number of sequences in the subalignment consisting of those sequences that do not have a gap in column i. These percentages increase approximately inversely proportionally with the fraction of gaps in the column, hence a column with only cysteins and 50% gaps gets a lower case letter.

The line in the middle shows the column score between the query and target amino acid distributions. It gives a valuable indication for the alignment quality.

```
= : column score below -1.5
- : column score between -1.5 and -0.5
. : column score between -0.5 and +0.5
+ : column score between +0.5 and +1.5
| : column score above   +1.5
```

(A unit of column score corresponds approximately to 0.6 bits.) From the column score line the excellent alignment around the highly conserved `'LPGIG'` motif in the turn between two helices is evident. The alignment around the first helix by contrast scores only slightly better than zero per residue and is therefore not very reliable.

After the template block, which consists of the following lines,

```
T Consensus:    the target alignment consensus sequence
T scop-id:      the target domain sequence
T ss_dssp:      the target secondary structure as determined by DSSP (when available)
T ss_pred:      the target secondary structure as predicted by PSIPRED (when available)
```

The last line in the block (`Confidence`) reports the reliability of the pairwise query-template alignment. The confidence values are obtained from the posterior probabilities calculated in the Forward-Backward algorithm. A value of 8 indicates a probability that this pair of HMM columns is correctly aligned between 0.8 and 0.8999. The `Confidence` line is only displayed when the -realign option is acitive.

D.5. File formats

D.5.1. Input alignment formats

HMMs can be read by HHsearch/HHblits in its own .hhm format, as well as in HMMer format (.hmm). Performance is not as good for HMMER-format as for hhm format, so please use our hhm format if possible. HMMER's hmm format can be converted to hhm format simply with hhmake:

```
> ./hhmake -i test.hmm -o test.hhm
```

This works only for a single HMM per file, not for concatenated HMMs. A safer way to effect the conversion is to call hhmake with the original alignment file. Note: you may add predicted secondary structure to the hmm file with `addss.pl` before the conversion to hhm format.

Multiple alignments can be read in A2M, A3M, or aligned FASTA format. (Check the -M option for using an input format different from the default A3M). You can transform MSAs from Clustal or Stockholm format to A3M or aligned FASTA with the `reformat.pl` utility supplied in this package.

To reformat from Clustal fromat to A3M:

```
> ./reformat.pl test.aln test.a3m
```

or explicitely, if the formats can not be recognized from the extensions:

```
> ./reformat.pl clu a3m test.clustal test.a3m
```

To reformat from Stockholm to aligned FASTA:

```
> ./reformat.pl test.sto test.fas
```

Example for aligned FASTA format:

```
>d1a1x__ b.63.1.1 (-) p13-MTCP1 {Human (Homo sapiens)}
PPDHLWVHQEGIYRDEYQRTWVAVVEE--E--T--SF---------LR----------ARVQQIQVPLG-------DAARPSHLLTS-----QL
>gi|6678257|ref|NP_033363.1|:(7-103) T-cell lymphoma breakpoint 1 [Mus musculus]
HPNRLWIWEKHVYLDEFRRSWLPVVIK--S--N--EK---------FQ----------VILRQEDVTLG-------EAMSPSQLVPY-----EL
>gi|7305557|ref|NP_038800.1|:(8-103) T-cell leukemia/lymphoma 1B, 3 [Mus musculus]
PPRFLVCTRDDIYEDENGRQWVVAKVE--T--S--RSpygsrietcIT----------VHLQHMTTIPQ-------EPTPQQPINNN-----SL
>gi|11415028|ref|NP_068801.1|:(2-106) T-cell lymphoma-1; T-cell lymphoma-1A [Homo sapiens]
HPDRLWAWEKFVYLDEKQHAWLPLTIEikD--R--LQ---------LR----------VLLRREDVVLG-------RPMTPTQIGPS-----LL
>gi|7305561|ref|NP_038804.1|:(7-103) T-cell leukemia/lymphoma 1B, 5 [Mus musculus]
----------GIYEDEHHRVWIAVNVE--T--S--HS---------SHgnrietcvt-VHLQHMTTLPQ-------EPTPQQPINNN-----SL
>gi|7305553|ref|NP_038801.1|:(5-103) T-cell leukemia/lymphoma 1B, 1 [Mus musculus]
LPVYLVSVRLGIYEDEHHRVWIVANVE--TshS--SH---------GN----------RRRTHVTVHLW-------KLIPQQVIPFNplnydFL
```

```
>gi|27668591|ref|XP_234504.1|:(7-103) similar to Chain A, Crystal Structure Of Murine Tcl1
-PDRLWLWEKHVYLDEFRRSWLPIVIK--S--N--GK---------FQ----------VIMRQKDVILG-------DSMTPSQLVPY-----EL
>gi|27668589|ref|XP_234503.1|:(9-91) similar to T-cell leukemia/lymphoma 1B, 5;
-PHILTLRTHGIYEDEHHRLWVVLDLQ--A--ShlSF---------SN----------RLLIYLTVYLQqgvafplESTPPSPMNLN-----GL
>gi|7305559|ref|NP_038802.1|:(8-102) T-cell leukemia/lymphoma 1B, 4 [Mus musculus]
PPCFLVCTRDDIYEDEHGRQWVAAKVE--T--S--SH---------SPycskietcvtVHLWQMTTLFQ-------EPSPDSLKTFN-----FL
>gi|7305555|ref|NP_038803.1|:(9-102) T-cell leukemia/lymphoma 1B, 2 [Mus musculus]
---------PGFYEDEHHRLWMVAKLE--T--C--SH---------SPycnkietcvtVHLWQMTRYPQ-------EPAPYNPMNYN-----FL
```

The sequence name and its description must be contained in a single name line beginning with the > symbol and followed directly by the sequence name. The residue data is contained in one or more lines of arbitrary length following the name line. No empty lines should be used. In aligned FASTA the gaps are written with ’-’ and the n’th letter of each sequence (except newlines) is understood to build the n’th column of the multiple alignment.

The same alignment in A2M format looks like this:

```
>d1a1x__ b.63.1.1 (-) p13-MTCP1 {Human (Homo sapiens)}
PPDHLWVHQEGIYRDEYQRTWVAVVEE..E..T..SF.........LR..........ARVQQIQVPLG.......DAARPSHLLTS.....QL
>gi|6678257|ref|NP_033363.1|:(7-103) T-cell lymphoma breakpoint 1 [Mus musculus]
HPNRLWIWEKHVYLDEFRRSWLPVVIK..S..N..EK.........FQ..........VILRQEDVTLG.......EAMSPSQLVPY.....EL
>gi|7305557|ref|NP_038800.1|:(8-103) T-cell leukemia/lymphoma 1B, 3 [Mus musculus]
PPRFLVCTRDDIYEDENGRQWVVAKVE..T..S..RSpygsrietcIT..........VHLQHMTTIPQ.......EPTPQQPINNN.....SL
>gi|11415028|ref|NP_068801.1|:(2-106) T-cell lymphoma-1; T-cell lymphoma-1A [Homo sapiens]
HPDRLWAWEKFVYLDEKQHAWLPLTIEikD..R..LQ.........LR..........VLLRREDVVLG.......RPMTPTQIGPS.....LL
>gi|7305561|ref|NP_038804.1|:(7-103) T-cell leukemia/lymphoma 1B, 5 [Mus musculus]
----------GIYEDEHHRVWIAVNVE..T..S..HS.........SHgnrietcvt.VHLQHMTTLPQ.......EPTPQQPINNN.....SL
>gi|7305553|ref|NP_038801.1|:(5-103) T-cell leukemia/lymphoma 1B, 1 [Mus musculus]
LPVYLVSVRLGIYEDEHHRVWIVANVE..TshS..SH.........GN..........RRRTHVTVHLW.......KLIPQQVIPFNplnydFL
>gi|27668591|ref|XP_234504.1|:(7-103) similar to Chain A, Crystal Structure Of Murine Tcl1
-PDRLWLWEKHVYLDEFRRSWLPIVIK..S..N..GK.........FQ..........VIMRQKDVILG.......DSMTPSQLVPY.....EL
>gi|27668589|ref|XP_234503.1|:(9-91) similar to T-cell leukemia/lymphoma 1B, 5;
-PHILTLRTHGIYEDEHHRLWVVLDLQ..A..ShlSF.........SN..........RLLIYLTVYLQqgvafplESTPPSPMNLN.....GL
>gi|7305559|ref|NP_038802.1|:(8-102) T-cell leukemia/lymphoma 1B, 4 [Mus musculus]
PPCFLVCTRDDIYEDEHGRQWVAAKVE..T..S..SH.........SPycskietcvtVHLWQMTTLFQ.......EPSPDSLKTFN.....FL
>gi|7305555|ref|NP_038803.1|:(9-102) T-cell leukemia/lymphoma 1B, 2 [Mus musculus]
---------PGFYEDEHHRLWMVAKLE..T..C..SH.........SPycnkietcvtVHLWQMTRYPQ.......EPAPYNPMNYN.....FL
```

A2M format is derived from aligned FASTA format. It looks very similar, but it distinguishes between match/delete columns and insert columns. This information is important to uniquely specify how an alignment is transformed into an HMM. The match/delete columns use upper case letters for residues and the ’-’ symbol for deletions (gaps). The insert columns use lower case letters for the inserted residues. Gaps aligned to inserted residues are written as ’.’ Lines beginning with a hash # symbol will be treated as commentary lines in HHsearch/HHblits (see below).

The same alignment in A3M:

```
>d1a1x__ b.63.1.1 (-) p13-MTCP1 {Human (Homo sapiens)}
```

```
PPDHLWVHQEGIYRDEYQRTWVAVVEEETSFLRARVQQIQVPLGDAARPSHLLTSQL
>gi|6678257|ref|NP_033363.1|:(7-103) T-cell lymphoma breakpoint 1 [Mus musculus]
HPNRLWIWEKHVYLDEFRRSWLPVVIKSNEKFQVILRQEDVTLGEAMSPSQLVPYEL
>gi|7305557|ref|NP_038800.1|:(8-103) T-cell leukemia/lymphoma 1B, 3 [Mus musculus]
PPRFLVCTRDDIYEDENGRQWVVAKVETSRSpygsrietcITVHLQHMTTIPQEPTPQQPINNNSL
>gi|11415028|ref|NP_068801.1|:(2-106) T-cell lymphoma-1; T-cell lymphoma-1A [Homo sapiens]
HPDRLWAWEKFVYLDEKQHAWLPLTIEikDRLQLRVLLRREDVVLGRPMTPTQIGPSLL
>gi|7305561|ref|NP_038804.1|:(7-103) T-cell leukemia/lymphoma 1B, 5 [Mus musculus]
----------GIYEDEHHRVWIAVNVETSHSSHgnrietcvtVHLQHMTTLPQEPTPQQPINNNSL
>gi|7305553|ref|NP_038801.1|:(5-103) T-cell leukemia/lymphoma 1B, 1 [Mus musculus]
LPVYLVSVRLGIYEDEHHRVWIVANVETshSSHGNRRRTHVTVHLWKLIPQQVIPFNplnydFL
>gi|27668591|ref|XP_234504.1|:(7-103) similar to Chain A, Crystal Structure Of Murine Tcl1
-PDRLWLWEKHVYLDEFRRSWLPIVIKSNGKFQVIMRQKDVILGDSMTPSQLVPYEL
>gi|27668589|ref|XP_234503.1|:(9-91) similar to T-cell leukemia/lymphoma 1B, 5;
-PHILTLRTHGIYEDEHHRLWVVLDLQAShlSFSNRLLIYLTVYLQqgvafplESTPPSPMNLNGL
>gi|7305559|ref|NP_038802.1|:(8-102) T-cell leukemia/lymphoma 1B, 4 [Mus musculus]
PPCFLVCTRDDIYEDEHGRQWVAAKVETSSHSPycskietcvtVHLWQMTTLFQEPSPDSLKTFNFL
>gi|7305555|ref|NP_038803.1|:(9-102) T-cell leukemia/lymphoma 1B, 2 [Mus musculus]
---------PGFYEDEHHRLWMVAKLETCSHSPycnkietcvtVHLWQMTRYPQEPAPYNPMNYNFL
```

The A3M format is a condensed version of A2M format. It is obtained by omitting all ’.’ symbols from A2M format. Hence residues emitted by Match states of the HMM are in upper case, residues emitted by Insert states are in lower case and deletions are written ’-’. A3M-formatted alignments can be reformatted to other formats like FASTA or A2M with the `reformat.pl` utility:

```
./reformat.pl test.a3m test.a2m
```

Lines beginning with a hash # symbol will be treated as commentary lines in HHsearch/HHblits (see below). Please note that A3M, though very practical and space-efficient, is not a standard format, and the name A3M is our personal invention.

Secondary structure information in A3M/A2M or FASTA MSAs for HHsearch/HHblits

The alignments read in by HHblits, HHsearch or HHmake can also contain secondary structure information. This information can be included in sequences with special names, like in this A3M file:

```
>ss_dssp
CCSEEEEEETTEEEETTSCEEEEEEEECSSCEEEEEECCCCCCCSCCCHHHHTTCSSCSEEEEETTTEEEETTSC
>aa_dssp
PPDHLWVHQEGIYRDEYQRTWVAVVEEETSFLRARVQQIQVPLGDAARPSHLLTSQLPLMWQLYPEERYMDNNSR
>aa_pred
PPDHLWVHQEGIYRDEYQRTWVAVVEEETSFLRARVQQIQVPLGDAARPSHLLTSQLPLMWQLYPEERYMDNNSR
>ss_pred
CCCEEEEECCCEECCCCCEEEEEEEEEECCCCCCEEEEEEEECCCCCCCCCCCCCCCCCEEEEECCCCCEECCCCC
>ss_conf
987689961870104587078999970578640132153103788788777774424614787217702035631
>d1a1x__ b.63.1.1 (-) p13-MTCP1 {Human (Homo sapiens)}
PPDHLWVHQEGIYRDEYQRTWVAVVEEETSFLRARVQQIQVPLGDAARPSHLLTSQLPLMWQLYPEERYMDNNSR
>gi|6678257|ref|NP_033363.1|:(7-103) T-cell lymphoma breakpoint 1 [Mus musculus]
```

```
HPNRLWIWEKHVYLDEFRRSWLPVVIKSNEKFQVILRQEDVTLGEAMSPSQLVPYELPLMWQLYPKDRYRSCDSM
>gi|7305557|ref|NP_038800.1|:(8-103) T-cell leukemia/lymphoma 1B, 3 [Mus musculus]
PPRFLVCTRDDIYEDENGRQWVVAKVETSRSpygsrietcITVHLQHMTTIPQEPTPQQPINNNSLPTMWRLESMNTYTGTDGT
>gi|11415028|ref|NP_068801.1|:(2-106) T-cell lymphoma-1; T-cell lymphoma-1A [Homo sapiens]
HPDRLWAWEKFVYLDEKQHAWLPLTIEikDRLQLRVLLRREDVVLGRPMTPTQIGPSLLPIMWQLYPDGRYRSSDSS
```

The sequence with name `>ss_dssp` contains the 8-state DSSP-determined secondary structure. `>aa_dssp` and `>aa_pred` contain the same residues as the query sequence (`>d1a1x__` in this case). They are optional and used merely to check whether the secondary structure states have correctly been assigned to the alignment. `>ss_pred` contains the 3-state secondary structure predicted by PSIPRED, and `>ss_conf` conains the corresponding confidence values. The query sequence is the first sequence that does not start with a special name. It is not marked explicitely.

Name lines in alignments

If you would like to create HMMs from alignments with a specified name which differ from the name of the first sequence, you can do so by adding name lines to your FASTA, A2M, or A3M alignment:

```
#PF02043 Bac_chlorC:  Bacteriochlorophyll C binding protein
>ss_pred
CCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHCC
>ss_conf
9863234658887677777750019999999998878886403445886666775655576678777660667633039
>CSMA_CHLAU/1-79
ATRGWFSESSAQVAQIGDIMFQGHWQWVSNALQATAAAVDNINRNAYPGVSRSGSGEGAFSSSPSNGFRPKRIRSRFNR
>CSMA_CHLPH/2-51
NGGGVFTDILAASGRIFEVMVEGHWATVGYLFDSLGKGVSRINQNAYGNM-----------------------------
...
```

When creating an HMM from an A3M file with hhmake, the first word of the name line is used as the name and file name of the HMM (PF02043 in this case). The following is an optional description. The descriptions will appear in the hit list and alignment section of the search results. The name lines can be arbitrarily long and there can be any number of name/description lines included, marked by a '#' as the first character in the line. Note that name lines are read by HHmake but are not a part of the standard definition of the FASTA or A2M format.

D.5.2. HHsearch/HHblits model format (hhm-format)

HHsearch/HHblits uses a format that is similar to HMMER format. This is the example of an hhm model file produced by HHmake:

```
HHsearch 1.5
NAME  d1mvfd_ b.129.1.1 (D:) MazE {Escherichia coli}
FAM   b.129.1.1
```

```
FILE  d1mvfd_
COM   hhmake1 -i d1mvfd_.a3m -o test.hhm
DATE  Wed May 14 10:41:06 2008
LENG  44 match states, 44 columns in multiple alignment
FILT  32 out of 35 sequences passed filter (-id 90 -cov 0 -qid 0 -qsc -20.00 -diff 100)
NEFF  4.0
SEQ
>ss_dssp
CBCEEETTEEEEECCHHHHHHTTCCTTCBEEEEEETTEEEEEEC
>ss_pred
CCCCCCCCCCCCCCHHHHHHHHCCCCCCEEEEEEECCEEEEEEC
>ss_conf
93233467666600578899808998986889874993798739
>Consensus
sxIxKWGNSxAvRlPaxlxxxlxlxxgdxixxxxxxxxivlxPv
>d1mvfd_ b.129.1.1 (D:) MazE {Escherichia coli}
SSVKRWGNSPAVRIPATLMQALNLNIDDEVKIDLVDGKLIIEPV
>gi|10176344|dbj|BAB07439.1|:(1-43) suppressor of ppGpp-regulated growth inhibitor [Bacillus halodurans]
TTIQKWGNSLAVRIPNHYAKHINVTQGSEIELSLgSDQTIILKP-
>gi|50120611|ref|YP_049778.1|:(3-43) suppressor of growth inhibitory protein ChpA [Erwinia carotovora]
-TVKKWGNSPAIRLSSSVMQAFDMTFNDSFDMEIRETEIALIP-
>gi|44064461|gb|EAG93225.1|:(2-42) unknown [environmental sequence]
-SVVKWGSYLAVRLPAELVLELGLKEGDEIDLVKDDGPVRVR--
>gi|31442758|gb|AAP55635.1|:(1-44) PemI-like protein [Pediococcus acidilactici]
TRLAKWGNSKAARIPSQIIKQLKLDDNQDMTITIENGSIVLTPI
>gi|44419085|gb|EAJ13619.1|:(3-43) unknown [environmental sequence]
SAIQKWGNSAAVRLPAVLLEQIDASVGSSLNADVRPDGVLLSP-
>gi|24376549|gb|AAN57947.1|:(3-44) putative cell growth regulatory protein [Streptococcus mutans UA159]
SAINKWGNSSAIRLPKQLVQELQLQTNDVLDYKVSGNKIILEKV
>gi|11344928|gb|AAG34554.1|:(1-44) MazE [Photobacterium profundum]
TQIRKIGNSLGSIIPATFIRQLELAEGAEIDVKTVDGKIVIEPI
#
NULL  3706 5728 4211 4064 4839 3729 4763 4308 4069  3323  5509 4640 4464 4937 4285 4423 3815 3783 6325 4665
HMM   A    C    D    E    F    G    H    I    K     L     M    N    P    Q    R    S    T    V    W    Y
      M->M M->I M->D I->M I->I D->M D->D Neff NeffI NeffD
      0    *    *    0    *    0    *    *    *     *
S 1   *    *    *    *    *    *    *    *    *     *     *    *    *    *    *    1012 988  *    *    *    1
      0    *    *    *    *    *    *    2817 0     0

S 2   2307 *    *    *    *    *    *    *    *     *     *    *    *    3178 3009 2179 1546 *    *    *    2
      0    *    *    *    *    *    *    3447 0     0

V 3   *    *    *    *    *    *    *    917  *     3009  *    *    *    *    *    *    *    1530 *    *    3
      0    *    *    *    *    *    *    3447 0     0
  .
  .
  .
V 44  *    *    *    *    *    *    *    1309 *     *     *    *    *    *    *    *    *    745  *    *    44
      0    *    *    0    *    *    *    2533 0     0

//
```

The first line (`HHsearch 1.5`) gives the format version, which corresponds to the HHsearch version for which this format was first introduced. Newer versions of HHsearch/HHblits may use previous format versions. The `NAME` line gives the name of the HMM and an optional description. The first 30 characters of this field are used in the summary hit list of the search results in hhr format, the full name line is given above the query-template alignments of the search results. The `FAM` line contains the family if the sequence is from SCOP or Pfam. `COM` is the command that was used to generate the file. `NEFF` is the diversity of the alignment, calculated as exp of the negative entropy averaged over all columns of the alignment.

The `SEQ` section contains a number of aligned, representative (pseudo) sequences in A3M format and is terminated with a line containing only a `#`. The first sequence represents the DSSP secondary structure (if available, i.e. if contained in the A3M or FASTA alignment from which the HMM model was built), the second and third sequences contain the predicted secondary structure and the corresponding confidence values in the range 0–9 (if available). The fourth sequence is the consensus annotation sequence that is shown in the pairwise query-template alignments in the hhsearch output. The first *real* sequence after the pseudo sequences is the *seed* or *master* sequence from which the alignment was built (`>d1mvfd_`, in our example). If the alignment does not represent a single master sequence but an entire family, as in the case of Pfam alignments for example, the first real sequence may be a consensus sequence calculated for the entire alignment. This master sequence is shown in the pairwise query-template alignments in the hhsearch output.

The next line specifies the null model frequencies, which are extracted from the selected substitution matrix used to add pseudocounts. Each of the positive integers is equal to 1000 times the negative logarithm of the amino acid frequency (which is between 0 and 1):

$$-1000 \times \log_2(frequency) \tag{D.1}$$

After the two annotation lines that specify the order of columns for the emission and transition probabilities that follow, there is a line which is not currently read by HHsearch and that lists the transition frequencies from the begin state to the first Match state, Insert state and Delete state.

The last block contains two lines for each column of the HMM. The first line starts with the amino acid in the master sequence at that column in the HMM and the column number. Following are 20 positive integers representing the match state amino acid emission frequencies (see eq. D.1). Asterisks `*` stand for a frequency of 0 (which would otherwise be represented by 99999). Please note that, unlike in HMMer format, *the emission frequencies do not contain pseudo-counts* in the HHsearch model format. The second line contains the seven transition frequencies (eq. D.1) and three subalignment diversities. `Neff` specifies to the local diversity, i.e. the diversity of the subalignment of all sequences containing a residue at this particular column of the alignment. Similarly, `Neff_I` refers to the alignment of all sequences containing an insert at this position, and `Neff_D` refers to the aligment of all sequences having a Delete at this position. The end of the model is indicated by a line containing only \\.

D.6. Summary of command-line parameters

This is just a brief summary of command line parameters for the various binaries and perl scripts as they are displayed by the programs when calling them without command line parameters. On the help pages of our HHpred/HHblits web servers

`http://toolkit.tuebingen.mpg.de` or `http://toolkit.lmb.uni-muenchen.de`

you can find more detailed explanations about some of the input parameters ('Paramters' section) and about how to interpret the output ('Results' section). The FAQ section contains valuable practical hints on topics such as how to validate marginally significant database matches or how to avoid high-scoring false positives.

D.6.1. `hhmake` – build an HMM from an input MSA

Build an HMM from an input alignment in A2M, A3M, or FASTA format. or convert between HMMER format (.hmm) and HHsearch format (.hhm). A database file is generated by simply concatenating these HMM files.

```
HHmake version 1.6.1.0 (April 2011)
Build an HMM from an input alignment in A2M, A3M, or FASTA format.
or convert between HMMER format (.hmm) and HHsearch format (.hhm).
A database file is generated by simply concatenating these HMM files.
Soding, J. Protein homology detection by HMM-HMM comparison. Bioinf. 2005, 21, 951-960.
(C) Johannes Soeding (see LICENSE file)

Usage: hhmake -i file [options]
 -i <file>     query alignment (A2M, A3M, or FASTA), or query HMM

Output options:
 -o <file>     HMM file to be written to  (default=<infile.hhm>)
 -a <file>     HMM file to be appended to
 -v <int>      verbose mode: 0:no screen output  1:only warings  2: verbose
 -seq <int>    max. number of query/template sequences displayed (def=10)
               Beware of overflows! All these sequences are stored in memory.
 -cons         insert consensus as main representative sequence of HMM
 -name <name>  use this name for HMM (default: use name of first sequence)

Filter input alignment (options can be combined):
 -id   [0,100] maximum pairwise sequence identity (%) (def=90)
 -diff [0,inf[ filter most diverse set of sequences, keeping at least this
               many sequences in each block of >50 columns (def=100)
 -cov  [0,100] minimum coverage with query (%) (def=0)
 -qid  [0,100] minimum sequence identity with query (%) (def=0)
 -neff [1,inf] target diversity of alignment (default=off)
 -qsc  [0,100] minimum score per column with query  (def=-20.0)
```

```
Input alignment format:
 -M a2m        use A2M/A3M (default): upper case = Match; lower case = Insert;
               '-' = Delete; '.' = gaps aligned to inserts (may be omitted)
 -M first      use FASTA: columns with residue in 1st sequence are match states
 -M [0,100]    use FASTA: columns with fewer than X% gaps are match states

Other options:
 -def          read default options from ./.hhdefaults or <home>/.hhdefault.

Example: hhmake -i test.a3m
```

D.6.2. `hhblits` – iteratively search a database of HMMs with a query sequence or MSA

```
HHblits version 2.2.13 (May 2011)
Fast homology detection method HHblits to iteratively search a HMM database
by HMM-HMM comparison.
to be published.
(C) Michael Remmert and Johannes Soeding

Usage: hhblits -i query [options]

 -i <file>      input query (single FASTA-sequence, A3M- or FASTA-MSA, HMM-file)

Options:
 -d    <base>   database basename (default=/cluster/databases/hhblits/uniprot20)
 -n     [1,8]   number of iterations (default=2)
 -e     [0,1]   E-value cutoff for inclusion in result alignment (def=0.001)

Needed libraries
 -context_data  <file> context_data library (default=context_data.lib)
 -cs_lib        <file> cs-library (default=cs219.lib)

Input alignment format:
 -M a2m        use A2M/A3M (default): upper case = Match; lower case = Insert;
               '-' = Delete; '.' = gaps aligned to inserts (may be omitted)
 -M first      use FASTA: columns with residue in 1st sequence are match states
 -M [0,100]    use FASTA: columns with fewer than X% gaps are match states

Output options:
 -o <file>      write results in standard format to file (default=<infile.hhr>)
 -oa3m <file>   write pairwise alignments in A3M format (default=none)
 -opsi <file>   write pairwise alignments in PSI format (default=none)
 -ohhm <file>   write HHM file of the pairwise alignments (default=none)
 -oalis <base>  write pairwise alignments in A3M format after each round (default=none)

HMM-HMM alignment options:
 -norealign     do NOT realign displayed hits with MAC algorithm (def=realign)
 -mact [0,1[    posterior probability threshold for MAC re-alignment (def=0.500)
                Parameter controls alignment greediness: 0:global >0.1:local
 -glob/-loc     use global/local Viterbi alignment for searching/ranking (def=local)

Other options:
 -v <int>       verbose mode: 0:no screen output  1:only warings  2: verbose (def=2)
 -cpu <int>     number of CPUs to use (for shared memory SMPs) (default=1)

An extended list of options can be obtained by using ’--help all’ as parameter


Example: hhblits -i query.fas -oa3m query.a3m -n 2
```

D.6.3. `hhsearch` – search a database of HMMs with a query MSA or HMM

```
HHsearch version 1.6.1.0 (April 2011)
Search a database of HMMs with a query alignment or query HMM
Soding, J. Protein homology detection by HMM-HMM comparison. Bioinf. 2005, 21, 951-960.
(C) Johannes Soeding (see LICENSE file)

Usage: hhsearch -i query -d database [options]
 -i <file>     input query alignment (A2M, A3M, FASTA) or HMM
 -d <file>     HMM database of concatenated HMMs in hhm, HMMER, or A3M format,
               OR, if file has extension pal, list of HMM file names, one per
               line. Multiple dbs, HMMs, or pal files with -d ’<db1> <db2>...’

Output options:
 -cal          calibrate query HMM (write mu and lamda into hhm file)
 -o <file>     write results in standard format to file (default=<infile.hhr>)
 -ofas <file>  write pairwise alignments in FASTA (-oa2m: A2M, -oa3m: A3M) format
 -v <int>      verbose mode: 0:no screen output  1:only warings  2: verbose
 -seq <int>    max. number of query/template sequences displayed (def=1)
 -nocons       don’t show consensus sequence in alignments (default=show)
 -nopred       don’t show predicted 2ndary structure in alignments (default=show)
 -nodssp       don’t show DSSP 2ndary structure in alignments (default=show)
 -ssconf       show confidences for predicted 2ndary structure in alignments
 -aliw <int>   number of columns per line in alignment list (def=80)
 -p <float>    minimum probability in summary and alignment list (def=20)
 -E <float>    maximum E-value in summary and alignment list (def=1E+06)
 -Z <int>      maximum number of lines in summary hit list (def=500)
 -z <int>      minimum number of lines in summary hit list (def=10)
 -B <int>      maximum number of alignments in alignment list (def=500)
 -b <int>      minimum number of alignments in alignment list (def=10)
               Remark: you may use ’stdin’ and ’stdout’ instead of file names

Filter input alignment (options can be combined):
 -id   [0,100] maximum pairwise sequence identity (%) (def=90)
 -diff [0,inf[ filter most diverse set of sequences, keeping at least this
               many sequences in each block of >50 columns (def=100)
 -cov  [0,100] minimum coverage with query (%) (def=0)
 -qid  [0,100] minimum sequence identity with query (%) (def=0)
 -qsc  [0,100] minimum score per column with query  (def=-20.0)

Input alignment format:
 -M a2m        use A2M/A3M (default): upper case = Match; lower case = Insert;
               ’-’ = Delete; ’.’ = gaps aligned to inserts (may be omitted)
 -M first      use FASTA: columns with residue in 1st sequence are match states
 -M [0,100]    use FASTA: columns with fewer than X% gaps are match states

HMM-HMM alignment options:
 -realign      realign displayed hits with max. accuracy (MAC) algorithm
 -norealign    do NOT realign displayed hits with MAC algorithm (def=realign)
 -mact [0,1[   posterior probability threshold for MAC re-alignment (def=0.300)
               Parameter controls alignment greediness: 0:global >0.1:local
 -glob/-loc    use global/local alignment mode for searching/ranking (def=local)
 -alt <int>    show up to this many significant alternative alignments(def=2)
 -excl <range> exclude query positions from the alignment, e.g. ’1-33,97-168’
 -shift [-1,1] score offset (def=-0.01)
 -corr [0,1]   weight of term for pair correlations (def=0.10)
 -ssm  0-4     0:   no ss scoring
               1,2: ss scoring after or during alignment  [default=2]
               3,4: ss scoring after or during alignment, predicted vs. predicted
 -ssw [0,1]    weight of ss score  (def=0.11)

Other options:
 -def          read default options from ./.hhdefaults or <home>/.hhdefault.
               Write ’hhsearch’, ’hhmake’ and/or ’hhfilter’ etc. in one line,
               followed by its list of options, one per line.
 -cpu <int>    number of CPUs to use (for shared memory SMPs) (default=1)

An extended list of options can be obtained by using ’--help all’ as parameter


Example: hhsearch -i a.1.1.1.a3m -d scop70_1.71.hhm
```

D.6.4. `hhalign` – Align a query MSA/HMM to a template MSA/HMMt

Align a query alignment/HMM to a template alignment/HMM by HMM-HMM alignment. If only one alignment/HMM is given it is compared to itself and the best off-diagonal alignment plus all further non-overlapping alignments above significance threshold are shown. The command also allows to sample alignments randomly, to generate png-files with dot plots showing alignments or to print out a list of indices of aligned residue pairs.

```
HHalign version 1.6.1.0 (April 2011)
Align a query alignment/HMM to a template alignment/HMM by HMM-HMM alignment
If only one alignment/HMM is given it is compared to itself and the best
off-diagonal alignment plus all further non-overlapping alignments above
significance threshold are shown.
Soding, J. Protein homology detection by HMM-HMM comparison. Bioinf. 2005, 21, 951-960.
(C) Johannes Soeding (see LICENSE file)

Usage: hhalign -i query [-t template] [options]
 -i <file>     input query alignment  (fasta/a2m/a3m) or HMM file (.hhm)
 -t <file>     input template alignment (fasta/a2m/a3m) or HMM file (.hhm)
 -png <file>   write dotplot into PNG-file (default=none)

Output options:
 -o <file>     write output alignment to file
 -ofas <file>  write alignments in FASTA, A2M (-oa2m) or A3M (-oa3m) format
 -Oa3m <file>  write query alignment in a3m format to file (default=none)
 -Aa3m <file>  append query alignment in a3m format to file (default=none)
 -atab <file>  write alignment as a table (with posteriors) to file (default=none)
 -index <file> use given alignment to calculate Viterbi score (default=none)
 -v <int>      verbose mode: 0:no screen output  1:only warings  2: verbose
 -seq  [1,inf[ max. number of query/template sequences displayed  (def=1)
 -nocons       don’t show consensus sequence in alignments (default=show)
 -nopred       don’t show predicted 2ndary structure in alignments (default=show)
 -nodssp       don’t show DSSP 2ndary structure in alignments (default=show)
 -ssconf       show confidences for predicted 2ndary structure in alignments
 -aliw int     number of columns per line in alignment list (def=80)
 -P <float>    for self-comparison: max p-value of alignments (def=0.001
 -p <float>    minimum probability in summary and alignment list (def=0)
 -E <float>    maximum E-value in summary and alignment list (def=1E+06)
 -Z <int>      maximum number of lines in summary hit list (def=100)
 -z <int>      minimum number of lines in summary hit list (def=1)
 -B <int>      maximum number of alignments in alignment list (def=100)
 -b <int>      minimum number of alignments in alignment list (def=1)
 -rank int     specify rank of alignment to write with -Oa3m or -Aa3m option (def=1)

Dotplot options:
 -dthr <float> probability/score threshold for dotplot (default=0.50)
 -dsca <int>   if value <= 20: size of dot plot unit box in pixels
               if value > 20: maximum dot plot size in pixels (default=600)
 -dwin <int>   average score over window [i-W..i+W] (for -norealign) (def=10)
 -dali <list>  show alignments with indices in <list> in dot plot
               <list> = <index1> ... <indexN>  or  <list> = all

Filter input alignment (options can be combined):
 -id   [0,100] maximum pairwise sequence identity (%) (def=90)
 -diff [0,inf[ filter most diverse set of sequences, keeping at least this
               many sequences in each block of >50 columns (def=100)
 -cov  [0,100] minimum coverage with query (%) (def=0)
 -qid  [0,100] minimum sequence identity with query (%) (def=0)
 -qsc  [0,100] minimum score per column with query  (def=-20.0)

Input alignment format:
 -M a2m        use A2M/A3M (default): upper case = Match; lower case = Insert;
               '-' = Delete; '.' = gaps aligned to inserts (may be omitted)
 -M first      use FASTA: columns with residue in 1st sequence are match states
 -M [0,100]    use FASTA: columns with fewer than X% gaps are match states

HMM-HMM alignment options:
 -glob/-loc    global or local alignment mode (def=local)
 -alt <int>    show up to this number of alternative alignments (def=1)
 -realign      realign displayed hits with max. accuracy (MAC) algorithm
 -norealign    do NOT realign displayed hits with MAC algorithm (def=realign)
 -mact [0,1[   posterior probability threshold for MAC alignment (def=0.300)
               A threshold value of 0.0 yields global alignments.
 -sto <int>    use global stochastic sampling algorithm to sample this many alignments
 -excl <range> exclude query positions from the alignment, e.g. '1-33,97-168'
 -shift [-1,1] score offset (def=-0.010)
 -corr [0,1]   weight of term for pair correlations (def=0.10)
 -ssm  0-4     0:no ss scoring [default=2]
               1:ss scoring after alignment
               2:ss scoring during alignment
 -ssw  [0,1]   weight of ss score  (def=0.11)

 -def          read default options from ./.hhdefaults or <home>/.hhdefault.

Example: hhalign -i T0187.a3m -t d1hz4a_.hhm -png T0187pdb.png
```

D.6.5. `hhfilter` – filter an MSA

Filter an alignment by maximum pairwise sequence identity, minimum coverage, minimum sequence identity or score per column to the first (seed) sequence etc.

```
HHfilter version 1.6.1.0 (April 2011)
Filter an alignment by maximum sequence identity of match states and minimum coverage
Soding, J. Protein homology detection by HMM-HMM comparison. Bioinf. 2005, 21, 951-960.
(C) Johannes Soeding (see LICENSE file)

Usage: hhfilter -i infile -o outfile [options]
 -i <file>     read input file in A3M/A2M or FASTA format
 -o <file>     write to output file in A3M format
 -a <file>     append to output file in A3M format

Options:
 -v <int>      verbose mode: 0:no screen output  1:only warings  2: verbose
 -id   [0,100] maximum pairwise sequence identity (%) (def=90)
 -diff [0,inf[ filter most diverse set of sequences, keeping at least this
               many sequences in each block of >50 columns (def=0)
 -cov  [0,100] minimum coverage with query (%) (def=0)
 -qid  [0,100] minimum sequence identity with query (%) (def=0)
 -qsc  [0,100] minimum score per column with query  (def=-20.0)
 -neff [1,inf] target diversity of alignment (default=off)
 -def          read default options from ./.hhdefaults or <home>/.hhdefault.

Input alignment format:
 -M a2m        use A2M/A3M (default): upper case = Match; lower case = Insert;
               '-' = Delete; '.' = gaps aligned to inserts (may be omitted)
 -M first      use FASTA: columns with residue in 1st sequence are match states
 -M [0,100]    use FASTA: columns with fewer than X% gaps are match states

Example: hhfilter -id 50 -i d1mvfd_.a2m -o d1mvfd_.fil.a2m
```

D.6.6. `reformat.pl` – reformat one or many alignments

Read one or many multiple alignments in one format and write them in another format

```
Usage: reformat.pl [informat] [outformat] infile outfile [options]
  or   reformat.pl [informat] [outformat] 'fileglob' .ext [options]

Available input formats:
   fas:     aligned fasta; lower and upper case equivalent, '.' and '-' equivalent
   a2m:     aligned fasta; inserts: lower case, matches: upper case, deletes: '-',
            gaps aligned to inserts: '.'
   a3m:     like a2m, but gaps aligned to inserts MAY be omitted
   sto:     Stockholm format; sequences in several blocks with sequence name at
            beginning of line (hmmer output)
   psi:     format as read by PSI-BLAST using the -B option (like sto with -M first -r)
   clu:     Clustal format; sequences in several blocks with sequence name at beginning
```

```
            of line
Available output formats:
   fas:     aligned fasta; all gaps '-'
   a2m:     aligned fasta; inserts: lower case, matches: upper case, deletes: '-',
            gaps aligned to inserts: '.'
   a3m:     like a2m, but gaps aligned to inserts are omitted
   sto:     Stockholm format; sequences in just one block, one line per sequence
   psi:     format as read by PSI-BLAST using the -B option
   clu:     CLUSTAL format
If no input or output format is given the file extension is interpreted as format
specification ('aln' as 'clu')

Options:
  -v int    verbose mode (0:off, 1:on)
  -num      add number prefix to sequence names: 'name', '1:name' '2:name' etc
  -noss     remove secondary structure sequences (beginning with >ss_)
  -sa       do not remove solvent accessibility sequences (beginning with >sa_)
  -M first  make all columns with residue in first seuqence match columns
            (default for output format a2m or a3m)
  -M int    make all columns with less than X% gaps match columns
            (for output format a2m or a3m)
  -r        remove all lower case residues (insert states)
            (AFTER -M option has been processed)
  -r int    remove all lower case columns with more than X% gaps
  -g ''     suppress all gaps
  -g '-'    write all gaps as '-'
  -uc       write all residues in upper case (AFTER other options have been processed)
  -lc       write all residues in lower case (AFTER other options have been processed)
  -l        number of residues per line (for CLUSTAL, FASTA, A2M, A3M formats)
            (default=100)
  -d        maximum number of characers in nameline (default=1000)

Examples: reformat.pl 1hjra.a3m 1hjra.a2m
          (same as reformat.pl a3m a2m 1hjra.a3m 1hjra.a2m)
          reformat.pl test.a3m test.fas -num -r 90
          reformat.pl fas sto '*.fasta' .stockholm
```

D.6.7. `buildali.pl` – build a PSI-BLAST alignment from a sequence or MSA

Build alignment for query sequence (FASTA) or query alignment (A2M, A3M, or aligned FASTA):

- Build profile with several interations of PSI-BLAST

- If query alignment contains < 20 sequences and query has < 50 residues and query can be extended by more than 10% of residues then do search with extended query and merge alignments
- Include dssp states if available
- Include psipred secondary structure prediction

```
Usage: buildali.pl infile [outdir] [options]

General options:
 -v   <int>    verbose mode (def=$v)
 -u            update: do not overwrite *.a3m files already existing (def=off)
 -old [dir]    if a file with same name is found in old database, jumpstart
               PSI-BLAST with this file
 -cpu <int>    number of CPUs to use when calling blastpgp (default=$cpu)
 -cn           create alignment file <basename>.<n>.a3m after each PSI-BLAST round

Options for building alignments:
 -extend       force query extension: psiblast with original AND extended sequence
               and merge alignments (def=off)
 -n    <int>   maximum number of psiblast iterations  (def=$maxiter)
 -e    <float> E-value for inclusion in PSI-BLAST profile (def=$Eult)
 -id   <int>   maximum pairwise sequence identity in % (def=$id)
 -qid  <int>   minimum sequence identity with query in % (def=0)
 -diff <int>   maximum number of maximally different sequences in output alignment
               (default=off)
 -cov  <int>   minimum coverage in % (Coverage = length of HSP / length of query)
               (def=$cov)
 -len  <int>   minimum number of residues in HSP (def=$min_hitlen)
 -b    <float> minimum per-residue bit score with query at ends of HSPs
 -bl   <float> lenient -b value used for ends of HSP where query sequence overlaps
               less than 50 residues (default=0)
 -bs   <float> strict -b value used for ends of HSP where query sequence overlaps
               more than 50 residues (default=0.167)
 -p    <float> only for extended search: maximum p-value of HSP IN MATCH COLUMNS
               for inclusion into alignment (def=$pmax)
 -core         when input is multiple alignment: build a core alignment with BLAST
               (don't use the supplied input alignment
               as a core alignment )
 -lc           filter out low complexity regions in query sequence
               (only for PSI-BLAST search)
 -ihs <int>    run quick intermediate HMM search (HHsenser) if less than <int>
               sequences found (def=off)
 -noss         omit secondary structure (predicted or DSSP)
 -db <basename> basename of sequence database, e.g. /cluster/databases/nr_euk
              ( =>  nr_euk90f, nr_euk70f)

Input formats:
 -fas         aligned FASTA input format; the first sequence (=query sequence) will
              define match columns
 -a3m         A3M input format (default); the first sequence (=query sequence) will
              (re)define match columns
 -clu         CLUSTAL format
 -sto:        Stockholm format; sequences in just one block, one line per sequence

Example: buildali.p test.a3m > ./builddb.log &
Usage: buildali.pl infile [outdir] [options]
```

D.6.8. `addss.pl` – add predicted secondary structure to an MSA or HMM

Add DSSP states (if available) and PSIPRED secondary structure prediction to a multiple sequence alignment. Input is a multiple sequence alignment or a HMMER (multi-)model file. Allowed input formats are A2M/FASTA (default), A3M (-a3m), CLUSTAL (-clu), STOCKHOLM (-sto), HMMER (-hmm). If the input file is an alignment, the output file is in A3M with default name <basename>.a3m. If the input file is in HMMER format, the output is the same as the input, except that records SSPRD and SSCON are added to each model which contain predicted secondary structure and confidence values. In this case the output file name is obligatory and must be different from the input file name. (Remark: A3M looks misaligned but it is not. To reconvert to FASTA, type `reformat.pl file.a3m file.fas`. For an explanation of the A3M format, see the HHsearch README file.

```
Usage: perl addss.pl <ali file> [<outfile>] [-fas|-a3m|-clu|-sto|-hmm]
```

D.6.9. `create_db.pl` – creates HHblits databases from HMMER-files, HHM-files or A3M-files

Creates HHblits databases from HMMER-files, HHM-files or A3M-files. The recommended way to use this script is to start with a directory of A3M-files (-a3mdir <DIR>) and let this script generates an A3M- database (-oa3m <FILE>) and an HHM-database (-ohhm <FILE>). If you already have HHM-models for your A3M-files, you can use them as additional input (-hhmdir <DIR>). If you don't need the A3M-database, you can also start this script with an directory of HHM-files (-hhmdir <DIR>) and as output only the HHM-database (-ohhm <FILE>).

```
Usage: perl create_db.pl -i <dir> [options]

Options:
  -a3mdir <dir>  Input directory (directories) with A3M-files
  -hhmdir <dir>  Input directory (directories) with HHM- or HMMER-files
                 (WARNING! Using HMMER dbs could result in a decreased sensitivity!)

  -oa3m  <FILE>  Output filename for the A3M database
                 (if not given, no A3M database will be build)
  -ohhm  <FILE>  Output filename for the HHM database
                 (if not given, no HHM database will be build)

  -a3mext        Extension of A3M-files (default: a3m)
  -hhmext        Extension of HHM- or HMMER-files (default: hhm)

  -append        If the output file exists, append new files (default: overwrite)

  -v [0-5]       verbose mode (default: 2)

Examples:

   perl create_db.pl -a3mdir /databases/scop_a3ms -oa3m /databases/scop_a3m_db
   -ohhm /databases/scop_hhm_db

   perl create_db.pl -a3mdir /databases/scop_a3ms -hhmdir /databases/scop_hhms
   -oa3m /databases/scop_a3m_db -ohhm /databases/scop_hhm_db

   perl create_db.pl -hhmdir /databases/scop_hhms -ohhm /databases/scop_hhm_db
```

D.6.10. `create_cs_db.pl` – creates HHblits prefilter database

Create a HHblits prefilter database from HMMER-files, HMM-files or A3M-files

```
Usage: perl create_cs_db.pl -i <dir> [options]

Options:
  -i <dir>    Input directory with HMMER-, HMM- or A3M-files
  -o <file>   Output file for the CS-database (default: <indir>.cs219)

  -ext <ext>  File extension, which identifies file-typ (default: a3m)
              - A3M-type  : a3m
              - HHM-type  : hhm
              - HMMER-type: hmmer or hmm

  -append     append to CS-database (default: overwrite file, if existing)

  -v [0-5]    verbose mode (default: 2)
```

D.6.11. `alignhits.pl` – extract an alignment from a BLAST/PSI-BLAST output

Extract a multiple alignment of hits from Blast or PsiBlast output (as text file, not html)

```
Usage:   alignhits.pl blast-file alignment-file [options]
Options for thresholds
  -e   e-value   : maximum e-value (default=0.0001)
  -qid percent   : minimum sequence identity to query in % (default=0)
                   (seq-id = # identities in match columns / # hit residues in
                    match columns)
  -cov coverage : minimum coverage in % (default=0)
  -emin e-value : minimum e-value (default=-1)
Options for output format:
  -psi           : PsiBlast-readable format; inserts relative to query (=first)
                   sequence omitted, capitalization of residues same as
                   query sequence (default)
  -a2m           : like FASTA, but capitalization of residues same as query sequence,
                   deletes '-', gaps aligned to lower case columns '.'
  -a3m           : like -a2m, but gaps aligned to inserts omitted
  -ufas          : unaligned fasta format (without gaps)
  -fas           : aligned fasta; all residues upper case, all gaps '-'
Other options:
  -v             : verbose mode (default=off)
  -append        : append output to file (default=overwrite)
  -best          : extract only the best HSP per sequence (default=off)
  -q   file      : insert a2m-formatted  query sequence into output alignment;
                   upper/lower case determines match/insert columns
  -Q   file      : like -q, but all query residues will be match states (upper case)
  -p   p-value   : maximum p-value of HSP IN MATCH COLUMNS
                   (with query or -P alignment) (default=1)
  -qsc value     : minimum score per column in bits (with query or -P alignment)
                   (default=-10)
  -b   float     : HSP pruning: min per-residue score in bits
                   (with query or -B alignment) at ends of HSPs
  -bl  float     : lenient HSP pruning: min per-residue score in bits
                   (with query or -B alignment) at ends of HSPs. Used when number of
                   endgaps at the one end < bg (see -bg) (default=-10)
  -bs  float     : strict HSP pruning: like -b, but used when number of endgaps >= bg
                   (default=-10)
  -bg  int       : below this number of end gaps the lenient HSP pruning score is used,
                :  above the strict score is employed (default=30)
  -P   file      : read alignment file (in psiblast-readable format) and calculate PSSM
                   to be used with option -p (only in conjunction with -q or -Q options)
  -B   file      : read alignment file (in psiblast-readable format) and calculate PSSM
```

```
                  to be used with option -b (only in conjunction with -q or -Q options)

Examples:
alignhits.pl 1enh.out 1enh.psi
alignhits.pl 1enh.out 1enh.a3m -e 1E-4 -cov 50 -s/c 1 -a2m
```

D.6.12. `hhmakemodel.pl` – generate MSAs or coarse 3D models from HHsearch results file

From the top hits in an hhsearch output file (hhr), you can

- generate a MSA (multiple sequence alignment) containing all representative template sequences from all selected alignments (options -fas, -a2m, -a3m, -pir)
- generate several concatenated pairwise alignments in AL format (option -al)
- generate several concatenated coarse 3D models in PDB format (option -ts)

In PIR, PDB and AL format, the pdb files are required in order to read the pdb residue numbers and ATOM records. The PIR formatted file can be used directly as input to the MODELLER homology modelling package.

```
 Usage: hhmakemodel.pl [-i] file.hhr [options]

 Options:
  -i   <file.hhr>        results file from hhsearch with hit list and alignments
  -fas <file.fas>        write a FASTA-formatted multiple alignment to file.fas
  -a2m <file.a2m>        write an A2M-formatted multiple alignment to file.a2m
  -a3m <file.a3m>        write an A3M-formatted multiple alignment to file.a3m
  -m   <int> [<int> ...] pick hits with specified indices  (default='-m 1')
  -p   <probability>     minimum probability threshold
  -e   <E-value>         maximum E-value threshold
  -q   <query_ali>       use the full-length query sequence in the alignment
                         (not only the aligned part);
                         the query alignment file must be in HHM, FASTA, A2M,
                         or A3M format.
  -N                     use query name from hhr filename (default: use same
                         name as in hhr file)
  -first                 include only first Q or T sequence of each hit in MSA
  -v                     verbose mode

 Options when database matches in hhr file are PDB or SCOP sequences
  -pir <file.pir>        write a PIR-formatted multiple alignment to file.pir
  -ts  <file.pdb>        write the PDB-formatted models based on *pairwise*
                         alignments into file.pdb
  -al  <file.al>         write the AL-formatted *pairwise* alignments into file.al
  -d   <pdbdirs>        directories containing the pdb files (for PDB, SCOP, or DALI
                         sequences)
  -s   <int>            shift the residue indices up/down by an integer
  -CASP                 formatting for CASP (for -ts, -al options)
                         (default: LIVEBENCH formatting)

 Options when query is compared to itself (for repeat detection)
  -conj                 include also conjugate alignments in MSA (with query and
                         template exchanged)
  -conjs                include conjugate alignments and sort by ascending diagonal
                         value (i.e. i0-j0)
```

D.7. Changes from previous versions

(For a full history, see accompanying file CHANGES.)

D.7.1. 2.0.0 (June 2011)

- Include iterative HMM-HMM comparison method HHblits.
- Increase speed by using SSE3 instructions in some functions.
- Adding new option "-atab" for writing alignment as a table (with posteriors) to file.
- HHsearch is now able to read HMMER3 profiles (but should not be used due to a loss of sensitivity).

D.7.2. 1.6.0 (2010)

- A new procedure for estimation of P- and E-values has been implemented that circumvents the need to calibrate HMMs. Calibration can still be done if desired. By default, however, HHsearch now estimates the lamda and mu parameters of the extreme value distribution (EVD) for each pair of query and database HMMs from the lengths of both HMMs and the diversities of their underlying alignments. Apart from saving the time for calibration, this procedure is more reliable and noise-resistant. This change only applies to the default local search mode. For global searches, nothing has changed. Note that E-values in global search mode are unreliable and that sensitivity is reduced.
  Old calibrations can still be used:
  -calm 0 : use empirical query HMM calibration (old default)
  -calm 1 : use empirical db HMM calibration
  -calm 2 : use both query and db HMM calibration
  -calm 3 : use neural network calibration (new default)

- Previous versions of HHsearch sometimes showed non-homologous hits with high probabilities by matching long stretches of secondary structure states, in particular long helices, in the absence of any similarity in the amino acid profiles. Capping the SS score by a linear function of the profile score now effectively suppresses these spurious high-scoring false positives.
- The output format for the query-template alignments has slightly changed. A 'Confidence' line at the bottom of each alignment block now reports the posterior probabilities for each alignment column when the -realign option is active (which it is by default). These probabilities are calculated in the Forward-Backward algorithm that is needed as input for the Maximium ACcuracy alignment algorithm. Also, the lines 'ss_conf' with the confidence values for the secondary structure prediction are omitted by default. (They can be displayed with option '-showssconf'). To compensate, secondary structure predictions with confidence values between 7 and 9 are given in capital letters, while for the predictions with values between 0 and 6 lower-case letters are used.
- In the hhsearch output file in the header lines before each query-database alignment, the substitution matrix score (without gap penalties) of the query with the database sequence is now reported in bits per column. Also, the sum of probabilities for each pair of aligned residues from the MAC algorithm is reported here (0 if no MAC alignment is performed).
- The buildali.pl script now uses context-specific iterative BLAST (CSI-BLAST) instead of PSI-BLAST. This considerably increases the sensitivity of buildali.pl/HHsearch.
- Removed a bug which produced a segfault for input alignments with more than 15000 match columns. Now, the HHsearch binaries will issue a warning and will transform only the first 15000 match columns into an HMM.
- Removed a bug in the multi-threading code that could lead to occasional hang-ups (race condition) in situations where slow file access was impeding program execution and inter-thread signaling was unreliable.
- Removed a memory leak and optimized memory management.
- Removed a bug in hhalign that could lead to unreasonably significant E-values and probabilities due to calibration problems.
- HHsearch now performs realign with MAC-alignment only around Viterbi-hit.

D.7.3. 1.5.0 (August 2007)

- By default, HHsearch realigns all displayed alignments in a second stage using the more accurate Maximum Accuracy (MAC) alignment algorithm (Durbin, Eddy, Krough, Mitchison: Biological sequence analysis, page 95; HMM-HMM version: J. Söding, unpublished). As before, the Viterbi algorithm is employed for searching and ranking the matches. The realignment step is parallelized (`-cpu <int>`) and typically takes a few seconds only. You can switch off the MAC realignment with the -norealign option. The posterior probability threshold is controlled with the -mact [0,1[option. This parameter controls the alignment algorithm's greediness. More precisely, the MAC algorithm finds the alignment that maximizes the sum of posterior probabilites minus mact for each aligned pair. Global alignments are generated with -mact 0, whereas -mact 0.5 will produce quite conservative local alignments. Default value is -mact 0.35, which produces alignments of roughly the same length as the Viterbi algorithm. The -global and -local (default) option now refer to both the Viterbi search stage as well as the MAC realignment stage. With -global (-local), the posterior probability matrix will be calculated for global (local) alignment. Note that '-local -mact 0' will produce global alignments from a local posterior probability matrix (which is not at all unreasonable).

- An amino acid compositional bias correction is now performed by default. This increases the sensitivity by 25% at 0.01 errors per query and by 5% at 0.1 errors per query. By recalibrating the Probabilities, the increased selectivity of this new version allows to give higher probabilities for the same P-values. Also, the score offset could be increased from -0.1 bits to 0 as a consequence.

- The algorithm that filters the set of the most diverse sequences (option -diff) has been improved. Before, it determined the set of the N most diverse sequences. In the case of multi-domain alignments, this could lead to severely underrepresented regions. E.g. when the first domain is only covered by a few fairly similar sequences and the second by hundreds of very diverse ones, most or all of the similar ones were removed. The '-diff N' option now filters the most diverse set of sequences, keeping at least N sequences in each block of >50 columns. This generally leads to a total number of sequences that is larger than N. Speed is similar. The default is '-diff 100' for hhmake and hhsearch. Speed is similar. Use -diff 0 to switch this filter off.

- The sensitivity for the -global alignment option has been significantly increased by a more robust statistical treatment. The sensitivity in -global mode is now only 0-10% lower than for the default -local option on a SCOP benchmark, i.e. when the query or the templates represent single structural domains. The E-values are now more realistic, although still not as reliable as for -local. The Probabilities were recalibrated.

- A new binary hhalign has been added. It is similar to hhsearch, but performs only pairwise comparisons. It can produce dot plots, tables of aligned residues, and it can sample alternative alignments stochastically. It uses the MAC algorithm by default.

- HHsearch and hhalign can generate query-template multiple alignments in FASTA, A2M, or A3M format with the -ofas, -oa2m, -oa3m options
- Returned error values were changed to comply with convention that 0 means no errors:
  1. Finished successfully
  2. Format error in input files
  3. File access error
  4. Out of memory
  5. Syntax error on command line
  6. Internal logic error (please report)
  7. Internal numeric error (please report)
  8. Other
- Added script `buildali.pl <file>` to automatically build PSI-BLAST mutliple sequence alignments, including predicted and DSSP secondary structure. `buildali.pl` is much more robust to alignment corruption by non-homologous fragment by pruning sequences individually from both ends as necessary (J. Söding, unpublished).
- Added script `hhmakemodel.pl <file.hhr>` that parses hhsearch results files and can generate FASTA or PIR multiple alignments or build rough 3D models.
- Moved memory allocation from stack to heap to avoid segmentation faults under some Windows systems.
- Removed a bug due to which pseudocounts where added to HMMer HMMs (which already have their own pseudocounts added). This bug reduced sensitivity for HMMs read in HMMer format.
- Removed a bug due to which the query-template alignments where not displayed on some platforms when output was directed to stdout
- Removed a bug that caused occasional segfaults under SunOS when reading HMMer files
- Added multi-threading (`-cpu <int>`) for Windows x86 platform
- Cleaned up output formatting of summary list for Windows x86
- Stopped support for the Alpha/DEC platform

Is anyone still interested in Mac OSX/PPC or SunOS support?

D.8. License

The HHsearch/HHblits software package is distributed under the terms of the *Attribution-NonCommercial-3.0 license* from Creative Commons (`http://creativecommons.org/licenses/by-nc/3.0/`). Here is a human-readable summary.

Summary of license

You are free

- to copy, distribute, display and perform the work
- to make derivative works

under the following conditions:

- Attribution: You must give the original author credit.
- Noncommercial: You may not use this work for commercial purposes.

For any reuse or distribution, you must make clear to others the license terms of this work. Any of these conditions, in particular the second condition restricting the use for commercial purposes, can be waived if you get permission from the copyright holder. Your fair use and other rights are in no way affected by the above.

For the legally binding, full text of the license, please refer to `http://creativecommons.org/licenses/by-nc/3.0/`

If you would like to use this software for commercial purposes, please contact the copyright holder at johannes@soeding.com

Bibliography

S. Abeln, C. Teubner, and C. M. Deane. Using phylogeny to improve genome-wide distant homology recognition. *PLoS Comput Biol* **2007**;*3*(1).

S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol* **1990**;*215*:403–410.

S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res* **1997**;*25*:3389–3402.

V. Alva, M. Ammelburg, J. Söding, and A. N. Lupas. On the origin of the histone fold. *BMC Struct Biol* **2007**;*7*:17–17.

V. Alva, K. K. Koretke, M. Coles, and A. N. Lupas. Cradle-loop barrels and the concept of metafolds in protein classification by natural descent. *Curr Opin Struct Biol* **2008**;*18*(3):358–365.

T. Arnold, M. Poynor, S. Nussberger, A. N. Lupas, and D. Linke. Gene duplication of the eight-stranded beta-barrel OmpX produces a functional pore: a scenario for the evolution of transmembrane beta-barrels. *J Mol Biol* **2007**;*366*:1174–1184.

Z. Aydin, A. Singh, J. Bilmes, and W. S. Noble. Learning sparse models for a dynamic bayesian network classifier of protein secondary structure. *BMC Bioinformatics* **2011**;*12*(1):154–154.

P. G. Bagos, T. D. Liakopoulos, I. C. Spyropoulos, and S. J. Hamodrakas. PRED-TMBB: a web server for predicting the topology of β-barrel outer membrane proteins. *Nucleic Acids Res* **2004**; *32*:400–404.

B. v. d. Berg, P. N. Black, W. M. Clemons, and T. A. Rapoport. Crystal structure of the long-chain fatty acid transporter FadL. *Science* **2004**;*304*(5676):1506–1509.

F. S. Berven, K. Flikka, H. B. Jensen, and I. Eidhammer. BOMP: a program to predict integral β-barrel outer membrane proteins encoded within genomes of Gram-negative bacteria. *Nucleic Acids Res* **2004**;*32*:394–399.

A. Biegert, C. Mayer, M. Remmert, J. Söding, and A. Lupas. The MPI Bioinformatics Toolkit for protein sequence analysis. *Nucleic Acids Res* **2006**;*34*:W335–W339.

A. Biegert and J. Söding. De novo identification of highly diverged protein repeats by probabilistic consistency. *Bioinformatics* **2008**;*24*(6):807–814.

A. Biegert and J. Söding. Sequence context-specific profiles for homology searching. *Proc Natl Acad Sci U S A* **2009**;*106*(10):3770–3775.

H. R. Bigelow, D. S. Petrey, J. Liu, D. Przybylski, and B. Rost. Predicting transmembrane β-barrels in proteomes. *Nucleic Acids Res* **2004**;*32*:2566–2577.

P. J. Brennan and H. Nikaido. The envelope of mycobacteria. *Annu Rev Biochem* **1995**;*64*:29–63.

J. L. Cayol, B. Ollivier, B. K. Patel, G. Prensier, J. Guezennec, and J. L. Garcia. Isolation and characterization of Halothermothrix orenii gen. nov., sp. nov., a halophilic, thermophilic, fermentative, strictly anaerobic bacterium. *Int J Syst Bacteriol* **1994**;*44*(3):534–540.

V. Chaudhary, F. Liu, V. Matta, and L. Yang. Parallel implementations of local sequence alignment: hardware and software, chapter 10. Wiley Online Library, **2006**;.

L. Chen, A. L. DeVries, and C. H. Cheng. Convergent evolution of antifreeze glycoproteins in Antarctic notothenioid fish and Arctic cod. *Proc Natl Acad Sci U S A* **1997**;*94*(8):3817–3822.

H. Cheng, B. H. Kim, and N. V. Grishin. Discrimination between Distant Homologs and Structural Analogs: Lessons from Manually Constructed, Reliable Data Sets. *J Mol Biol* **2008**;*377*:1265–1278.

G. Chukkapalli, C. Guda, and S. Subramaniam. Sledgehmmer: a web server for batch searching the pfam database. *Nucleic acids research* **2004**;*32*(suppl 2):W542.

B. Clantin, A. S. Delattre, P. Rucktooa, N. Saint, A. C. Méli, C. Locht, F. Jacob-Dubuisson, and V. Villeret. Structure of the membrane protein FhaC: a member of the Omp85-TpsB transporter superfamily. *Science* **2007**;*317*(5840):957–961.

M. Coles, M. Hulko, S. Djuranovic, V. Truffault, K. Koretke, J. Martin, and A. N. Lupas. Common evolutionary origin of swapped-hairpin and double-psi beta barrels. *Structure* **2006**;*14*(10):1489–1498.

R. R. Copley, R. B. Russell, and C. P. Ponting. Sialidase-like Asp-boxes: sequence-similar structures within different protein folds. *Protein Sci* **2001**;*10*(2):285–292.

J. L. Corden. Tails of rna polymerase ii. *Trends Biochem Sci* **1990**;*15*(10):383–387.

M. O. Dayhoff, R. M. Schwartz, and B. C. Orcut. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure* **1978**;*5*:345–352.

S. Derrien and P. Quinton. Parallelizing hmmer for hardware acceleration on fpgas. In Application-specific Systems, Architectures and Processors, 2007. ASAP. IEEE International Conf. on. IEEE, **2007**; pages 10–17.

S. Derrien and P. Quinton. Hardware acceleration of hmmer on fpgas. *Journal of Signal Processing Systems* **2010**;*58*(1):53–67.

S. Dietmann and L. Holm. Identification of homology in protein structure classification. *Nat Struct Biol* **2001**;*8*(11):953–957.

C. Do, S. Gross, and S. Batzoglou. Contralign: discriminative training for protein sequence alignment. In Research in Computational Molecular Biology. Springer, **2006**; pages 160–174.

C. Dong, K. Beis, J. Nesper, A. L. Brunkan-LaMontagne, B. R. Clarke, C. Whitfield, and J. H. Naismith. Wza the translocon for E. coli capsular polysaccharides defines a new class of membrane proteins. *Nature* **2006**;*444*:226–229.

R. F. Doolittle. Convergent evolution: the need to be explicit. *Trends Biochem Sci* **1994**;*19*(1):15–18.

R. F. Doolittle, D. F. Feng, S. Tsang, G. Cho, and E. Little. Determining divergence times of the major kingdoms of living organisms with a protein clock. *Science* **1996**;*271*(5248):470–477.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. Biological Sequence Analysis: Probabilistic Models of proteins and nucleic acids. Cambridge University Press, **1998**.

D. Duy, J. Soll, and K. Philippar. Solute channels of the outer membrane: from bacteria to chloroplasts. *Biol Chem* **2007**;*388*(9):879–889.

S. R. Eddy. A new generation of homology search tools based on probabilistic inference. *Genome Inform* **2009**;*23*(1):205–211.

R. Edgar. Search and clustering orders of magnitude faster than blast. *Bioinformatics* **2010**; *26*(19):2460.

M. Eigen and P. Schuster. The hypercycle. A principle of natural self-organization. Part A: Emergence of the hypercycle. *Naturwissenschaften* **1977**;*64*(11):541–565.

M. Faller, M. Niederweis, and G. E. Schulz. The structure of a mycobacterial outer-membrane channel. *Science* **2004**;*303*(5661):1189–1192.

M. Farrar. Striped smith-waterman speeds database searches six times over other simd implementations. *Bioinformatics* **2007**;*23*(2):156–161.

J. S. Fetrow and A. Godzik. Function driven protein evolution. A possible proto-protein for the RNA-binding proteins. *Pac Symp Biocomput* **1998**;pages 485–496.

A. V. Finkelstein and O. B. Ptitsyn. Why do globular proteins fit the limited set of folding patterns? *Prog Biophys Mol Biol* **1987**;*50*(3):171–190.

R. D. Finn, J. Mistry, B. Schuster-Böckler, S. Griffiths-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. *Nucleic Acids Res* **2006**;*34*(Database issue):247–251.

R. D. Finn, J. Mistry, J. Tate, P. Coggill, A. Heger, J. E. Pollington, O. L. Gavin, P. Gunasekaran, G. Ceric, K. Forslund, L. Holm, E. L. Sonnhammer, S. R. Eddy, and A. Bateman. The pfam protein families database. *Nucleic Acids Res* **2010**;*38*(Database issue):211–222.

T. Frickey and A. N. Lupas. CLANS: a Java application for visualizing protein families based on pairwise similarity. *Bioinformatics* **2004**;*20*:3702–3704.

I. Friedberg and A. Godzik. Connecting the protein structure universe by using sparse recurring fragments. *Structure* **2005**;*13*(8):1213–1224.

J. E. Gewehr, V. Hintermair, and R. Zimmer. AutoSCOP: automated prediction of SCOP classifications using unique pattern-class mappings. *Bioinformatics* **2007**;*23*(10):1203–1210.

M. W. Gonzalez and W. R. Pearson. Homologous over-extension: a challenge for iterative similarity searches. *Nucleic Acids Res* **2010**;*38*(7):2177–2189.

O. Gotoh. An improved algorithm for matching biological sequences. *J Mol Biol* **1982**;*162*(3):705–708.

S. Griep and U. Hobohm. Pdbselect 1992-2009 and pdbfilter-select. *Nucleic Acids Res* **2010**; *38*(Database issue):318–319.

N. V. Grishin. Two tricks in one bundle: helix-turn-helix gains enzymatic activity. *Nucleic Acids Res* **2000**;*28*(11):2229–2233.

N. V. Grishin. KH domain: one motif, two folds. *Nucleic Acids Res* **2001**;*29*(3):638–643.

M. Hauser and J. Söding. kclust: fast and sensitive clustering of large protein sequence databases. *in preparation* **2011**;.

G. Hautbergue and V. Goguel. Activation of the cyclin-dependent kinase ctdk-i requires the heterodimerization of two unstable subunits. *J Biol Chem* **2001**;*276*(11):8005–8013.

J. G. Henikoff and S. Henikoff. Using substitution probabilities to improve position-specific scoring matrices. *Comput Appl Biosci* **1996**;*12*(2):135–143.

S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A* **1992**;*89*(22):10915–10919.

D. G. Higgins and P. M. Sharp. Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene* **1988**;*73*(1):237–244.

A. Hildebrand, M. Remmert, A. Biegert, and J. Söding. Fast and accurate automatic structure prediction with hhpred. *Proteins* **2009**;*77 Suppl 9*:128–132.

L. Holm and C. Sander. Decision support system for the evolutionary classification of protein structures. *Proc Int Conf Intell Syst Mol Biol* **1997**;*5*:140–146.

I. Holmes and R. Durbin. Dynamic programming alignment accuracy. *J Comput Biol* **1998**;*5*(3):493–504.

D. Horn, M. Houston, and P. Hanrahan. Clawhmmer: A streaming hmmer-search implementation. In ACM/IEEE Conference on Supercomputing. IEEE Computer Society, **2005**; page 11.

D. C. Jeffares, A. M. Poole, and D. Penny. Relics from the RNA world. *J Mol Evol* **1998**;*46*(1):18–36.

D. T. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *J Mol Biol* **1999**;*292*:195–202.

J. C. Jones, H. P. Phatnani, T. A. Haystead, J. A. MacDonald, S. M. Alam, and A. L. Greenleaf. C-terminal repeat domain kinase i phosphorylates ser2 and ser5 of rna polymerase ii c-terminal domain repeats. *J Biol Chem* **2004**;*279*(24):24957–24964.

H. Kim, B. Erickson, W. Luo, D. Seward, J. H. Graber, D. D. Pollock, P. C. Megee, and D. L. Bentley. Gene-specific rna polymerase ii phosphorylation and the ctd code. *Nat Struct Mol Biol* **2010**;*17*(10):1279–1286.

R. Koebnik, K. P. Locher, and P. V. Gelder. Structure and function of bacterial outer membrane proteins: barrels in a nutshell. *Mol Microbiol* **2000**;*37*:239–253.

S. S. Krishna and N. V. Grishin. Structurally analogous proteins do exist! *Structure* **2004**;*12*(7):1125–1127.

S. S. Krishna, R. I. Sadreyev, and N. V. Grishin. A tale of two ferredoxins: sequence similarity and structural differences. *BMC Struct Biol* **2006**;*6*:8–8.

J. A. Lake. Evidence for an early prokaryotic endosymbiosis. *Nature* **2009**;*460*(7258):967–971.

T. Lassmann and E. L. Sonnhammer. Kalign - an accurate and fast multiple sequence alignment algorithm. *BMC Bioinformatics* **2005**;*6*:298–298.

D. Lavenier and V. Nguyen. Seed-based parallel protein sequencecomparison combining multithreading, gpu, and fpga technologies. *Bioinformatics* **2010**;*1*:15.

Y. Li, Y. Takagi, Y. Jiang, M. Tokunaga, H. Erdjument-Bromage, P. Tempst, and R. D. Kornberg. A multiprotein complex that interacts with rna polymerase ii elongator. *J Biol Chem* **2001**;*276*(32):29628–29631.

H. Lican and H. Ya. Gpu based parallel computing on blast program. In Networking and Distributed Computing (ICNDC), 2010 First International Conference on. IEEE, **2010**; pages 379–380.

L. Ligowski and W. Rudnicki. An efficient implementation of smith waterman algorithm on gpu using cuda, for massively parallel scanning of sequence databases. In Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE, **2009**; pages 1–8.

C. Ling and K. Benkrid. Design and implementation of a cuda-compatible gpu-based core for gapped blast algorithm. *Procedia Computer Science* **2010**;*1*(1):495–504.

Y. Liu, D. L. Maskell, and B. Schmidt. Cudasw++: optimizing smith-waterman sequence database searches for cuda-enabled graphics processing units. *BMC Res Notes* **2009**;*2*:73–73.

D. B. Lukatsky, B. E. Shakhnovich, J. Mintseris, and E. I. Shakhnovich. Structural similarity enhances interaction propensity of proteins. *J Mol Biol* **2007**;*365*(5):1596–1606.

B. M. Lunde, S. L. Reichow, M. Kim, H. Suh, T. C. Leeper, F. Yang, H. Mutschler, S. Buratowski, A. Meinhart, and G. Varani. Cooperative interaction of transcription termination factors with the rna polymerase ii c-terminal domain. *Nat Struct Mol Biol* **2010**;*17*(10):1195–1201.

A. N. Lupas, C. P. Ponting, and R. B. Russell. On the evolution of protein folds: are similar motifs in different protein folds the result of convergence, insertion, or relics of an ancient peptide world? *J Struct Biol* **2001**;*134*(2-3):191–203.

T. Madej, A. R. Panchenko, J. Chen, and S. H. Bryant. Protein homologous cores and loops: important clues to evolutionary relationships between structurally similar proteins. *BMC Struct Biol* **2007**;*7*:23–23.

S. Manavski and G. Valle. Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment. *BMC bioinformatics* **2008**;*9*(Suppl 2):S10.

A. Mayer, M. Lidschreiber, M. Siebert, K. Leike, J. Söding, and P. Cramer. Uniform transitions of the general rna polymerase ii transcription complex. *Nat Struct Mol Biol* **2010**;*17*(10):1272–1278.

A. D. McLachlan. Repeating sequences and gene duplication in proteins. *J Mol Biol* **1972**;*64*(2):417–437.

A. D. McLachlan. Gene duplication and the origin of repetitive protein structures. *Cold Spring Harb Symp Quant Biol* **1987**;*52*:411–420.

A. Meinhart and P. Cramer. Recognition of rna polymerase ii carboxy-terminal domain by 3'-rna-processing factors. *Nature* **2004**;*430*(6996):223–226.

I. Melvin, J. Weston, C. Leslie, and W. S. Noble. Rankprop: a web server for protein remote homology detection. *Bioinformatics* **2009**;*25*(1):121–122.

S. Moslavac, O. Mirus, R. Bredemeier, J. Soll, A. von Haeseler, and E. Schleiff. Conserved pore-forming regions in polypeptide-transporting proteins. *FEBS J* **2005**;*272*(6):1367–1378.

A. G. Murzin. Sweet-tasting protein monellin is related to the cystatin family of thiol proteinase inhibitors. *J Mol Biol* **1993**;*230*(2):689–694.

A. G. Murzin. How far divergent evolution goes in proteins. *Curr Opin Struct Biol* **1998**;*8*(3):380–387.

A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J Mol Biol* **1995**;*247*:536–540.

N. Nagano, E. G. Hutchinson, and J. M. Thornton. Barrel structures in proteins: automatic identification and classification including a sequence analysis of TIM barrels. *Protein Sci* **1999**; *8*(10):2072–2084.

N. Nagano, C. A. Orengo, and J. M. Thornton. One fold with many functions: the evolutionary relationships between TIM barrel families based on their sequences, structures and functions. *J Mol Biol* **2002**;*321*(5):741–765.

S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol* **1970**;*48*(3):443–453.

A. F. Neuwald, J. S. Liu, and C. E. Lawrence. Gibbs motif sampling: detection of bacterial outer membrane protein repeats. *Protein Science* **1995**;*4*:1618–1632.

T. X. Nguyen, E. R. Alegre, and S. T. Kelley. Phylogenetic analysis of general bacterial porins: a phylogenomic case study. *J Mol Microbiol Biotechnol* **2006**;*11*(6):291–301.

V. Nguyen and D. Lavenier. Plast: parallel local alignment search tool for database comparison. *BMC bioinformatics* **2009**;*10*(1):329.

W. S. Noble, R. Kuang, C. Leslie, and J. Weston. Identifying remote protein homologs by network propagation. *FEBS J* **2005**;*272*(20):5119–5128.

T. Oliver, L. Yeow, and B. Schmidt. High performance database searching with hmmer on fpgas. In 2007 IEEE International Parallel and Distributed Processing Symposium. IEEE, **2007**; page 258.

L. E. Orgel. Prebiotic chemistry and the origin of the RNA world. *Crit Rev Biochem Mol Biol* **2004**; *39*(2):99–123.

S. A. Paschen, T. Waizenegger, T. Stan, M. Preuss, M. Cyrklaff, K. Hell, D. Rapaport, and W. Neupert. Evolutionary conservation of biogenesis of beta-barrel membrane proteins. *Nature* **2003**; *426*:862–866.

A. Pautsch and G. E. Schulz. High-resolution structure of the OmpA membrane domain. *J Mol Biol* **2000**;*298*(2):273–282.

W. R. Pearson. Effective protein sequence comparison. *Methods Enzymol* **1996**;*266*:227–258.

J. Pei and N. V. Grishin. AL2CO: calculation of positional conservation in a protein sequence alignment. *Bioinformatics* **2001**;*17*(8):700–712.

J. Peng and J. Xu. Boosting protein threading accuracy. *Lect Notes Comput Sci* **2009**;*5541*:31–31.

S. Pietrokovski. Searching databases of conserved sequence regions by aligning protein multiple-alignments. *Nucleic Acids Res* **1996**;*24*(19):3836–3845.

B. Rekapalli, C. Halloy, and I. Zhulin. Hsp-hmmer: a tool for protein domain identification on a large scale. In Proceedings of the 2009 ACM symposium on Applied Computing. ACM, **2009**; pages 766–770.

H. Remaut, C. Tang, N. S. Henderson, J. S. Pinkner, T. Wang, S. J. Hultgren, D. G. Thanassi, G. Waksman, and H. Li. Fiber formation across the bacterial outer membrane by the chaperone/usher pathway. *Cell* **2008**;*133*:640–652.

H. Remaut and G. Waksman. Structural biology of bacterial pathogenesis. *Curr Opin Struct Biol* **2004**;*14*(2):161–170.

M. Remmert, A. Biegert, A. Hauser, and J. Söding. HHblits: Lightning-fast iterative protein sequence searching by HMM-HMM alignment. *Nat Methods* **2011**;*accepted.*

M. Remmert, A. Biegert, D. Linke, A. N. Lupas, and J. Söding. Evolution of outer membrane beta-barrels from an ancestral beta beta hairpin. *Mol Biol Evol* **2010**;*27*(6):1348–1358.

M. Remmert, D. Linke, A. N. Lupas, and J. Söding. HHomp–prediction and classification of outer membrane proteins. *Nucleic Acids Res* **2009**;*37*(Web Server issue):446–451.

V. Robert, E. B. Volokhina, F. Senf, M. P. Bos, P. Van Gelder, and J. Tommassen. Assembly factor Omp85 recognizes its outer membrane protein substrates by a species-specific C-terminal motif. *PLoS Biol* **2006**;*4*(11).

T. Rognes. Faster smith-waterman database searches with inter-sequence simd parallelisation. *BMC bioinformatics* **2011**;*12*(1):221.

T. Rognes and E. Seeberg. Six-fold speed-up of smith-waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics* **2000**;*16*(8):699–706.

B. Rost, C. Sander, and R. Schneider. Redefining the goals of protein secondary structure prediction. *J Mol Biol* **1994**;*235*(1):13–26.

N. Ruiz, D. Kahne, and T. J. Silhavy. Advances in understanding bacterial outer-membrane biogenesis. *Nat Rev Microbiol* **2006**;*4*(1):57–66.

R. B. Russell, M. A. Saqi, R. A. Sayle, P. A. Bates, and M. J. Sternberg. Recognition of analogous and homologous protein folds: analysis of sequence and structure conservation. *J Mol Biol* **1997**; *269*(3):423–439.

R. Sadreyev and N. Grishin. COMPASS: a tool for comparison of multiple protein alignments with assessment of statistical significance. *J Mol Biol* **2003**;*326*(1):317–336.

R. I. Sadreyev and N. V. Grishin. Accurate statistical model of comparison between multiple sequence alignments. *Nucleic Acids Res* **2008**;.

R. I. Sadreyev, B. H. Kim, and N. V. Grishin. Discrete-continuous duality of protein structure space. *Curr Opin Struct Biol* **2009**;*19*(3):321–328.

A. Sali and T. L. Blundell. Comparative protein modelling by satisfaction of spatial restraints. *J Mol Biol* **1993**;*234*:779–815.

A. A. Samir, A. Ropolo, D. Grasso, R. Tomasini, J. C. Dagorn, N. Dusetti, J. L. Iovanna, and M. I. Vaccaro. Cloning and expression of the mouse pip49 (pancreatitis induced protein 49) mrna which encodes a new putative transmembrane protein activated in the pancreas with acute pancreatitis. *Mol Cell Biol Res Commun* **2000**;*4*(3):188–193.

X. Shao and N. V. Grishin. Common fold in helix-hairpin-helix proteins. *Nucleic Acids Res* **2000**; *28*(14):2643–2650.

F. Sievers, A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. Söding, J. D. Thompson, and D. G. Higgins. Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega. *Mol Syst Biol* **2011**;*7*:539–539.

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J Mol Biol* **1981**;*147*(1):195–197.

J. Söding. Protein homology detection by HMM-HMM comparison. *Bioinformatics* **2005**;*21*:951–960.

J. Söding, A. Biegert, and A. N. Lupas. The HHpred interactive server for protein homology detection and structure prediction. *Nucleic Acids Res* **2005**;*33*(Web Server issue):244–248.

J. Söding and A. N. Lupas. More than the sum of their parts: on the evolution of proteins from peptides. *Bioessays* **2003**;*25*(9):837–846.

J. Söding and M. Remmert. Protein sequence comparison and fold recognition: progress and good-practice benchmarking. *Curr Opin Struct Biol* **2011**;*21*(3):404–411.

J. Söding, M. Remmert, and A. Biegert. HHrep: de novo protein repeat detection and the origin of TIM barrels. *Nucleic Acids Res* **2006**a;*34*(Web Server issue):137–142.

J. Söding, M. Remmert, A. Biegert, and A. N. Lupas. HHsenser: exhaustive transitive profile search using HMM-HMM comparison. *Nucleic Acids Res* **2006**b;*34*:374–378.

T. G. Sokolova, J. M. González, N. A. Kostrikina, N. A. Chernyh, T. V. Slepova, E. A. Bonch-Osmolovskaya, and F. T. Robb. Thermosinus carboxydivorans gen. nov., sp. nov., a new anaerobic, thermophilic, carbon-monoxide-oxidizing, hydrogenogenic bacterium from a hot pool of Yellowstone National Park. *Int J Syst Evol Microbiol* **2004**;*54*(Pt 6):2353–2359.

L. Song, M. R. Hobaugh, C. Shustak, S. Cheley, H. Bayley, and J. E. Gouaux. Structure of staphylococcal alpha-hemolysin, a heptameric transmembrane pore. *Science* **1996**;*274*(5294):1859–1866.

E. L. L. Sonnhammer, S. R. Eddy, and R. Durbin. Pfam: A Comprehensive Database of Protein Families Based on Seed Alignments. *Proteins* **1997**;*28*:405–420.

E. Sotiriades and A. Dollas. A general reconfigurable architecture for the blast algorithm. *The Journal of VLSI Signal Processing* **2007**;*48*(3):189–208.

D. E. Sterner, J. M. Lee, S. E. Hardin, and A. L. Greenleaf. The yeast carboxyl-terminal repeat domain kinase ctdk-i is a divergent cyclin-cyclin-dependent kinase complex. *Mol Cell Biol* **1995**; *15*(10):5716–5724.

Y. Sun and J. Buhler. Designing patterns for profile hmm search. *Bioinformatics* **2007**;*23*(2):36–43.

A. Szalkowski, C. Ledergerber, P. Krähenbühl, and C. Dessimoz. Swps3 - fast multi-threaded vectorized smith-waterman for ibm cell/b.e. and x86/sse2. *BMC Res Notes* **2008**;*1*:107–107.

T. Takagi and T. Maruyama. Accelerating hmmer search using fpga. In Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on. IEEE, **2009**; pages 332–337.

Y. Takeuchi and H. Nikaido. Persistence of segregated phospholipid domains in phospholipid-lipopolysaccharide mixed bilayers: studies with spin-labeled phospholipids. *Biochemistry* **1981**; *20*(3):523–529.

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 Natl Acad Sci U S A* **1994**; *91*(25):12091–12095.

D. L. Theobald and D. S. Wuttke. Divergent evolution within protein superfolds inferred from profile-based phylogenetics. *J Mol Biol* **2005**;*354*(3):722–737.

R. Ujwal, D. Cascio, J. P. Colletier, S. Faham, J. Zhang, L. Toro, P. Ping, and J. Abramson. The crystal structure of mouse VDAC1 at 2.3 A resolution reveals mechanistic insights into metabolite gating. *Proc Natl Acad Sci U S A* **2008**;*105*(46):17742–17747.

L. Vasiljeva, M. Kim, H. Mutschler, S. Buratowski, and A. Meinhart. The nrd1-nab3-sen1 termination complex interacts with the ser5-phosphorylated rna polymerase ii c-terminal domain. *Nat Struct Mol Biol* **2008**;*15*(8):795–804.

R. Voulhoux and J. Tommassen. Omp85, an evolutionarily conserved bacterial protein involved in outer-membrane-protein assembly. *Res Microbiol* **2004**;*155*(3):129–135.

P. Vouzis and N. Sahinidis. Gpu-blast: using graphics processors to accelerate protein sequence alignment. *Bioinformatics* **2011**;*27*(2):182.

J. Walters, V. Balu, S. Kompalli, and V. Chaudhary. Evaluating the use of gpus in liver image segmentation and hmmer database searches. In Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE, **2009**; pages 1–12.

J. Walters, B. Qudah, and V. Chaudhary. Accelerating the hmmer sequence analysis suite using conventional processors. In Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on, volume 1. IEEE, **2006**; pages 6–pp.

H. Wang, B. Ooi, K. Tan, T. Ong, and L. Zhou. Blast++: Blasting queries in batches. *Bioinformatics* **2003**;*19*(17):2323.

Z. Wang, J. Liu, A. Sudom, M. Ayres, S. Li, H. Wesche, J. Powers, and N. Walker. Crystal structures of IRAK-4 kinase in complex with inhibitors: a serine/threonine kinase with tyrosine as a gatekeeper. *Structure* **2006**;*14*(12):1835–1844.

M. S. Waterman and M. Eggert. A new algorithm for best subsequence alignments with application to trna-rrna comparisons. *J Mol Biol* **1987**;*197*(4):723–728.

J. Weston, R. Kuang, C. Leslie, and W. S. Noble. Protein ranking by semi-supervised network propagation. *BMC Bioinformatics* **2006**;*7 Suppl 1*.

B. O. Wittschieben, G. Otero, T. de Bizemont, J. Fellows, H. Erdjument-Bromage, R. Ohba, Y. Li, C. D. Allis, P. Tempst, and J. Q. Svejstrup. A novel histone acetyltransferase is an integral subunit of elongating rna polymerase ii holoenzyme. *Mol Cell* **1999**;*4*(1):123–128.

A. Wozniak. Using video-oriented instructions to speed up sequence comparison. *Comput Appl Biosci* **1997**;*13*(2):145–150.

P. Yao, H. An, M. Xu, G. Liu, X. Li, Y. Wang, and W. Han. Cuhmmer: A load-balanced cpu-gpu cooperative bioinformatics application. In High Performance Computing and Simulation (HPCS), 2010 International Conference on. IEEE, **2010**; pages 24–30.

G. Yona and M. Levitt. Within the twilight zone: a sensitive profile-profile comparison tool based on information theory. *J Mol Biol* **2002**;*315*(5):1257–1275.

Y. Zhang and J. Skolnick. Scoring function for automated assessment of protein structure template quality. *Proteins* **2004**;*57*(4):702–710.

Y. Zhang and J. Skolnick. TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic Acids Res* **2005**;*33*(7):2302–2309.

Michael Remmert

Diploma in Bioinformatics

Personal Details

DATE & PLACE OF BIRTH	February 18, 1981, Cologne
NATIONALITY	German

Publications

2011	**HHblits: Lightning-fast iterative protein sequence searching by HMM-HMM alignment**, **M. Remmert**, A. Biegert, A. Hauser, J. Söding, *Nature Methods*, accepted.
2011	**Fast, scalable generation of high quality protein multiple sequence alignments using Clustal Ω**, F. Sievers, A. Wilm, D. Dineen, H. MacWilliam, **M. Remmert**, T. Gibson, K. Karplus, R. Lopez, J. Söding, J. Thompson, D. Higgins *Mol. Syst. Biol.7*, 539.
2011	**Protein sequence comparison and fold recognition: progress and good-practice benchmarking**, J. Söding, **M. Remmert** *Curr. Opin. Struct. Biol. 21*, 404 – 411.
2010	**Evolution of outer membrane β-barrels from an ancestral ββ-hairpin**, **M. Remmert**, A. Biegert, D. Linke, A. N. Lupas, J. Söding, *Mol. Biol. Evo. 27*(6), 1348 – 1358.
2010	**A galaxy of folds**, V. Alva, **M. Remmert**, A. Biegert, A. N. Lupas, J. Söding, *Protein Sci. 19*(1), 124 – 130.
2009	**Fast and accurate automatic structure prediction with HHpred**, A. Hildebrand, **M. Remmert**, A. Biegert, J. Söding, *Proteins 77*, 128 – 132.
2009	**HHomp–prediction and classification of outer membrane proteins**, **M. Remmert**, D. Linke, A. N. Lupas, J. Söding, *Nucleic Acids Res. 37*, w446 – 451.
2008	**PDBalert: automatic, recurrent remote homology tracking and protein structure prediction**, V. Agarwal, **M. Remmert**, A. Biegert, J. Söding, *BMC Struc. Biol. 8*, 51.
2006	**HHrep: *de novo* protein repeat detection and the origin of TIM barrels**, J. Söding, **M. Remmert**, A. Biegert, *Nucleic Acids Res. 34*, w137 – 142.
2006	**HHsenser: exhaustive transitive profile search using HMM-HMM comparison**, J. Söding, **M. Remmert**, A. Biegert, A. N. Lupas, *Nucleic Acids Res. 34*, w374 – 378.
2006	**The MPI Bioinformatics Toolkit for protein sequence analysis**, A. Biegert, C. Mayer, **M. Remmert**, J. Söding, A. N. Lupas, *Nucleic Acids Res. 34*, w335 – 339.
2006	**Identification of plant microRNA homologs**, T. Dezulian, **M. Remmert**, J. F. Palatnik, D. Weigel, D. H. Huson, *Bioinformatics 22* (3), 359 – 360.

Conference talks & poster presentations

JUL 2011	**ISMB & ECCB 2011**, *Vienna, Austria*, Software Demo: „HHblits: Lightning-fast iterative sequence searching by HMM-HMM comaprison“.
DEC 2010	**CASP9 Meeting 2010**, *Pacific Grove, California*, Poster: „HHblits: Lightning-fast iterative sequence searching by HMM-HMM comaprison“.
JUN 2009	**ISMB & ECCB 2009**, *Stockholm, Schweden*, Talk: „Protein homology searches and structure prediction with HHpred and the MPI/LMU Bioinformatics Toolkit“.
APR 2009	**Interact PhD Symposium**, *München, Deutschland*, Poster: „HHblits - Iterative HMM-HMM search detects twice as many homologous proteins as PSI-BLAST at the same speed “.
SEP 2008	**GCB 2008**, *Dresden, Deutschland*, Poster: „Origin of bacterial outer membrane β-barrels by multiple duplication of a $\beta\beta$ hairpin“.
APR 2008	**Gene Center Retreat**, *Wildbad Kreuth, Deutschland*, Poster: „Origin of bacterial outer membrane β-barrels by multiple duplication of a $\beta\beta$ hairpin“.
SEP 2006	**GCB 2006**, *Tübingen, Deutschland*, Poster: „The MPI Bioinformatics Toolkit for protein sequence analysis“.