Re-calculate secondary structure specific amino acid exchange matrices for the use of profile analysis

Marlies van der Wees

28 august 2009

ABSTRACT Profile analysis is a sequence comparison method to detect distantly related proteins. In 1991, Lüthy et al. extended profile analysis by creating secondary structure specific exchange matrices. In the meantime, several thousands of structures were added yearly to the PDB structure database. Hence it is desirable to re-calculate a set of secondary structure based matrices, as they lead to more up to date results when applied to profile analysis. The re-calculation is done with the BLOSUM algorithm, since BLOSUM exchange matrices proved superior to PAM matrices in alignments and homology searching. The first step in the creation of new exchange matrices is splitting the complete Blocks+ database into secondary structure specific sub databases for alpha helix, beta strand and other structures. Next, the original BLOSUM c-code is used to calculate the actual tables. The result is a new set of secondary structure specific exchange matrices which can be used for calculating profiles.

INTRODUCTION

In 1987 Gribskov et al. [1] introduced profile analysis, a sequence comparison method. In this method a position specific scoring table, the profile, is used to detect distantly related proteins. As a first step in profile analysis, a profile has to be created for a certain protein family or folding class. The starting point for the composition of the profile is a probe consisting of some representative sequences of the protein family or folding class. From this probe the profile is generated with the use of the Dayhoff mutational (PAM250) matrix. For each position in the probe the likelihood of appearance of each of the 20 amino acids on that position is calculated, as well as the likelihood for a gap to occur on that position. This results in a matrix with 21 columns (20 columns with the likelihood scores for the 20 amino acids and one column with the score for a gap) and as many rows as the length of the probe.

Once the profile is generated, databases can be searched for proteins that are related to (or members of) the protein family or folding class. Gribskov et al. showed that the profile method can distinguish all members of two tested protein families from all other sequences within a database.

However, when creating a profile for a protein family, Gribskov et al. did not distinguish between secondary structure types. For this reason, Lüthy et al. [2] extended the profile method for several classes of secondary structures. Whereas the original profile method uses one mutational matrix to calculate the profile for all positions in the probe, the extended method makes use of distinct mutational tables for positions that are in distinct secondary structures.

With these secondary structure specific matrices Lüthy et al. created profiles for three protein families (the globin family, the immunoglobulin family and the insecticyanin folding class) and examined if the new profiles performed better in finding distantly related sequences than profiles created with the original Dayhoff matrix. The results turned out that both profiles were equally effective in recognizing closely related protein sequences, but that the secondary structure based profiles were more effective in detecting distantly related sequences than the original profiles.

Since the publication of Lüthy's paper in 1991, several thousands of new structures have been added yearly to the PDB [3]. For this reason, it is desirable to repeat the study done by Lüthy, as the innumerable new structures might lead to different results. The first step in the repeat of the extended profile method is the creation of new secondary structure specific amino acid exchange matrices. These matrices can be used widely to generate profiles for several protein families and folding classes. Therefore the aim of this study is to create three new exchange matrices for amino acids in alpha helices, beta strands or other secondary structures respectively.

However, in the meantime also new algorithms to calculate mutational matrices have been introduced. Both the original profile analysis and the extended method used the PAM250 amino acid exchange matrix for generating the profiles. The PAM (point accepted mutation) matrix is based on single amino acid substitutions in closely related sequences, with at

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4																			
R	-1	5																		
N	-2	0	6																	
D	-2	-2	1	6																
C	0	-3	-3	-3	9															
Q	-1	1	0	0	-3	5														
E	-1	0	0	2	-4	2	5													
G	0	-2	0	-1	-3	-2	-2	6												
H	-2	0	1	-1	-3	0	0	-2	8											
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
T	0	-1	0	-1	-1	-1	-1	2	-2	-1	-1	-1	-1	-2	-1	1	5			
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Fig. 1: Original BLOSUM matrix created by Henikoff and Henikoff.

least 85% similarity. To calculate the PAM1 mutation probability matrix, observed amino acid substitutions are counted in sequences that differ only one out of 100 amino acids. In this way, a substitution rate of one mutation per 100 positions is estimated. The PAM250 matrix is obtained by multiplying the PAM1 matrix 250 times by itself. Consequently, the PAM250 matrix presumes a substitution rate of 250 single amino acid mutations per 100 positions. Next, the observed substitutions in the PAM250 mutation probability matrix are divided by the normalized frequencies for the substituted amino acid. Taking logarithms of these values results in a log-odd matrix, which shows the likelihood for each amino acid to be substituted by any other amino acid. [4]

In 1992, Henikoff and Henikoff introduced another series of mutational tables, the BLOSUM series [5]. The BLOSUM matrices are derived from protein blocks in the Blocks database. Protein blocks are short multiple sequence alignments with no gaps, representing highly conserved protein segments. A BLOSUM matrix consists of log-odd scores for the occurrence of amino acid pairs on a position in a block. These log-odd scores are calculated as follows:

$$s_{ij} = \lambda \cdot \log\left(\frac{q_{ij}}{e_{ij}}\right)$$

In this scoring formula λ is a scaling factor, q_{ij} is the observed frequency of an i, j amino acid pair and e_{ij} is the expected frequency of the same pair, which is simply the product of the relative frequencies of both amino acids i and j. s_{ij} takes the logarithm of the fraction q_{ij}/e_{ij}, thus s_{ij} will be zero when the observed frequencies equal the expected frequencies, negative when the observed frequencies are lower than expected and positive when the observed frequencies are higher than expected.

Unlike the PAM matrices, the BLOSUM series focus on mutations between more distantly related proteins. Therefore a clustering percentage is used to group sequences which are identical for at least the defined percentage. As a consequence, sequences which are highly identical are clustered and weighted as a single sequence. In this way, the contribution of closely related sequences is reduced. The most commonly used BLOSUM matrix is the BLOSUM62, which uses a clustering percentage of 62%. Figure 1 shows the original BLOSUM62 matrix generated by Henikoff and Henikoff.

In several studies the BLOSUM matrices proved to perform significantly better in alignments and homology searching of distantly related sequences than the PAM matrices. [5, 6, 7] Therefore the newly created matrices in this study are calculated by the BLOSUM algorithm instead of the PAM procedure.

```
HAKYWCGTLFMNIVQTRDEP  blocks sequence
EEEE  SSS EEEE  TT H  secondary structure extracted from DSSP file
--------------------P  sequence copied to alpha helix (H) sub database
----WCGTLF----QTRDE-  sequence copied to beta strand (E) sub database
HAKY------MNIV------  sequence copied to other sec struct sub database
```

Fig. 2: Example of a blocks sequence with corresponding secondary structure. The lower three lines show how the sequence is split and copied to three sub databases.

METHODS

The composition of secondary structure based mutational matrices consisted of two steps. First, the original Blocks+ database[1] [8] was split into three sub databases, containing blocks of protein segments which are in alpha helices, beta strands or other secondary structures respectively. Next, the BLOSUM construction c-code[2] made by Henikoff and Henikoff was applied to the three sub databases. In this way three different BLOSUM matrices were constructed for three different secondary structure types.

Construction of secondary structure specific blocks databases

Temporary structure specific blocks databases

In order to split the original Blocks+ database into three secondary structure based sub databases, information about the secondary structure of the amino acids in the aligned sequences had to be known. This information was extracted from the DSSP database [9], in which secondary structures of proteins in the PDB are documented. However, proteins in DSSP files are stored by their PDB name, while sequences in the Blocks+ database are listed with a UniProt identifier. Therefore, PDB and UniProt names had to be coupled. This was done by composing two dictionaries with mappings of different types of protein identifiers. The first dictionary was made from the *PDB master* database[1], and coupled each block accession number to all the PDB names of sequences in the concerning block.

In order to create the second dictionary, a database with PDB sequences[3] was BLASTed against uniref100 sequences from UniProt[4]. This BLAST was processed non-redundant, so that identical PDB sequences appeared only once in the BLAST result. Identical chains and duplicate PDB names were all represented by the first identifier that was found to be 100% identical to a UniProt sequence. With the use of the two dictionaries, all PDB identifiers corresponding to the aligned sequences in a certain block were known.

As a subsequent step in the splitting of the Blocks+ database into sub databases, the complete Blocks+ database was processed block by block. For each block the block accession number was extracted. If this number was listed in the first dictionary, the block was processed line by line, or sequence by sequence. If the number was not listed in the first dictionary, the complete block was neglected.

Next, for each sequence the second dictionary was searched for the UniProt identifier of the sequence. All coupled PDB identifiers were listed, provided that they were also coupled to the block number in the first dictionary. Then, the DSSP file for the first PDB name in the list was opened and, on condition that the amino acid sequence in the DSSP file was sufficiently equal (see *Effect of different percentages of unknown residues*) to the Blocks sequence, the secondary structure of the sequence was extracted. Using this structure, three sub sequences were composed for the three types of secondary structure. In these sequences, hyphens (–) were inserted on positions that are not part of the concerning structure. Figure 2 shows an example of how a sequence is split into three sub sequences.

[1] http://blocks.fhcrc.org/blocks/uploads/blocks
[2] http://blocks.fhcrc.org/blocks/uploads/blosum
[3] ftp://ftp.wwpdb.org/pub/pdb/derived_data/pdb_seqres.txt.gz
[4] ftp://ftp.uniprot.org/pub/databases/uniprot/uniref/uniref100/uniref100.fasta.gz

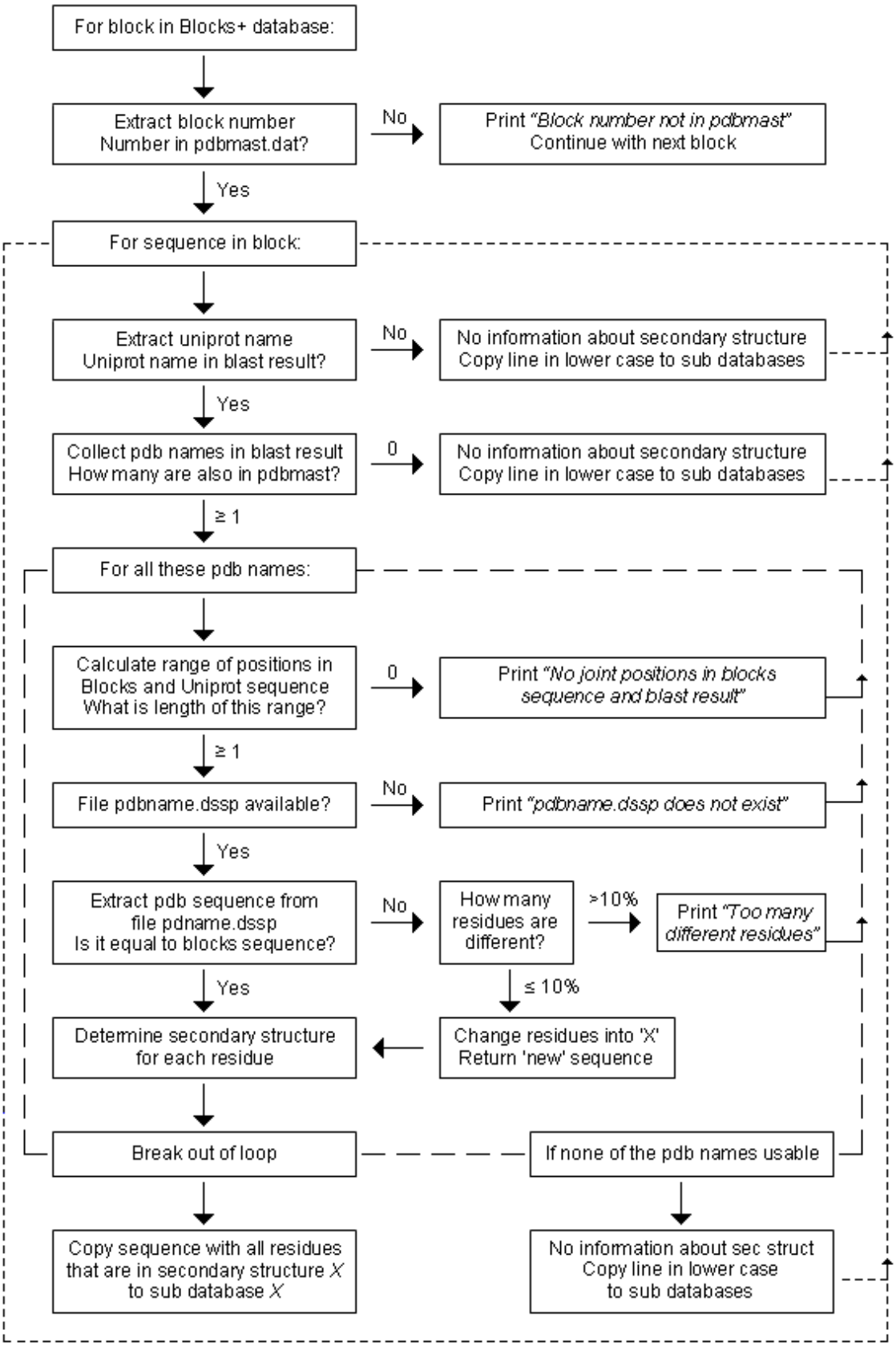

Fig. 3: Flow chart of blocks process function. The diagram is executed for each block in the original Blocks+ database.

Obviously, a lot of exceptions needed to be specified. For instance, in many cases the DSSP file was not available. For other PDB names the sequence segment that appeared in the BLAST result was not the same as the segment aligned in the block. Besides, there were also sequences with simply too many unknown residues. In all these situations, the same procedure as described above was attempted to finish for the next PDB identifier in the list.

In case that the procedure was not applicable for any of the PDB names corresponding to the sequence, the complete sequence was changed into lower case letters en copied to all three sub databases. In this way the sequence was still able to contribute to the later calculation of the BLOSUM matrix, but no secondary structure information was derived from it.

A flow chart of the processing of each block is shown in figure 3.

Effect of different percentages of unknown residues

Preceding the extraction of the secondary structure from a DSSP file, both the DSSP sequence and the Blocks sequence were compared. In this way, wrongly coupled sequences were detected and their secondary structure was not copied to the sub databases. However, many DSSP sequences were not completely equal to their corresponding Blocks sequence, due to incidentally unknown residues in the DSSP file. Preferably, these sequences would not be excluded. Therefore, a maximum percentage of residues allowed to be unequal had to be set. As long as the number of unknown amino acids in a sequence did not exceed this percentage, the unknown amino acids were replaced by a character 'X'.

It is known that the BLOSUM c-code is able to deal with several X-es in sequences, as the original Blocks+ database also contains characters X instead of ordinary residues. However, the effect of larger amounts of X-es is not yet known. For this reason, the procedure described above was executed with three different preset maximum percentages of unknown residues, namely 5%, 10% and 20%. For all of these percentages the number of remaining blocks in the sub databases was counted. In addition, the fraction of sequences with known secondary structure divided by sequences with unknown secondary structure within each block was calculated. In this way, possible effects of the preset percentage could be shown.

Final structure specific blocks databases

The three temporary sub databases that were created, consisted of complete sequences in lower case letters and split sequences made complete by hyphens. Of these, only the latter had known secondary structures. Therefore, only these sequences were used to determine whether a position in the block was considered to be part of alpha helix, beta strand or other structures. When doing this, the databases were processed block by block. Within each block, sequences were analysed per position. If all positions contained just lower case letters, the complete block was removed, as there was no secondary structure information available. If a position contained at least one hyphen, the position was not consequently given one secondary structure, and hence removed from all sequences in the block. As a result, three final sub database were left with blocks of which all residues in the sequences were presumed to be part of the same secondary structure.

Construction of BLOSUM62 matrices for each secondary structure

As mentioned before, the original c-code for the construction of BLOSUM matrices was used to calculate the secondary structure specific matrices. The clustering percentage was set at 62%, as the BLOSUM62 matrix is the most commonly used matrix of the BLOSUM series. First, a new BLOSUM62 matrix was calculated over the complete Blocks+ database. This was done to compare the overall substitution rates from the actual Blocks+ database (containing 29068 blocks) and the original one (consisting of 2106 blocks). In this way, insight was gained in differences that are not caused by secondary structures. Next, the blosum script was executed over the three sub databases. As a result, BLOSUM62 matrices were created for the three structure types alpha helix, beta strand and other.

RESULTS

Comparison of actual and original complete BLOSUM62 matrix

The firstly calculated matrix was the BLOSUM62 matrix of the complete actual Blocks+ database, which is shown in figure 4. The upper half of the figure shows the difference of this table with the original BLOSUM62 matrix. As can be seen, a lot of scores differ slightly. However, scores that were negative in the original BLOSUM62 matrix remain negative, and scores that were positive are still positive. One amino acid that has to be highlighted, is tryptophan (W). All substitution scores of tryptophan except one have become less negative. This means that tryptophan seems more mutable in the sequences in the actual Blocks+ database than it was in the original Blocks database.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	
	0	0	-1	-1	-1	0	0	0	-1	0	0	0	0	0	0	0	0	-1	0	0	A
		0	0	-2	-1	0	-1	-1	-1	-1	0	-1	-1	-1	-1	0	0	-2	-1	-1	R
A	4		0	-1	-2	0	0	0	0	-1	-1	0	-1	-1	-1	0	0	-2	-1	-1	N
R	-1	5		0	-1	-1	0	0	-1	0	-1	-1	0	0	-1	0	0	-2	-1	0	D
N	-1	0	6		-1	-1	-1	-2	-2	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-2	C
D	-1	0	2	6		0	0	-1	-1	-1	0	0	-1	-1	0	0	-1	-1	0	0	Q
C	1	-2	-1	-2	10		-1	0	0	0	-1	0	0	0	0	-1	0	-1	0	0	E
Q	-1	1	0	1	-2	5		0	-1	-1	-1	-1	-1	-1	-1	0	-1	0	-1	0	G
E	-1	1	0	2	-3	2	6		0	-1	-1	-1	-1	0	-1	-1	-1	-2	-1	-1	H
G	0	-1	0	-1	-1	-1	-2	6		0	0	0	0	-1	-1	0	0	-2	0	0	I
H	-1	1	1	0	-1	1	0	-1	8		0	0	0	-2	-1	0	0	-2	-1	0	L
I	-1	-2	-2	-3	0	-2	-3	-3	-2	4		-1	-1	0	0	-1	0	-1	0	0	K
L	-1	-2	-2	-3	-1	-2	-2	-3	-2	2	4		0	-1	0	0	-1	-1	-1	0	M
K	-1	3	0	0	-2	1	1	-1	0	-3	-2	6		0	-2	0	-1	-2	0	-1	F
M	-1	-2	-1	-3	0	-1	-2	-2	-1	1	2	-2	5		-1	-1	0	-3	-1	0	P
F	-2	-2	-2	-3	-1	-2	-3	-2	-1	1	2	-3	1	6		0	-1	-1	0	0	S
P	-1	-1	-1	0	-2	-1	-1	-1	-1	-2	-2	-1	-2	-2	8		-1	-1	0	0	T
S	1	-1	1	0	0	0	-1	0	0	-2	-2	-1	-1	-2	0	4		-1	-1	-2	W
T	0	-1	0	-1	0	0	-1	-1	-1	-1	-1	-1	0	-1	-1	2	4		-1	0	Y
W	-2	-1	-2	-2	-1	-1	-2	-2	0	-1	0	-2	0	3	-1	-2	-1	12		0	V
Y	-2	-1	-1	-2	-1	-1	-2	-2	1	-1	0	-2	0	3	-2	-2	-2	3	8		
V	0	-2	-2	-3	1	-2	-2	-3	-2	3	1	-2	1	0	-2	-2	0	-1	-1	4	
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	

Fig. 4: The lower half of the figure shows the BLOSUM62 matrix composed from the actual Blocks+ database. The upper half shows the difference with the original BLOSUM62 matrix (see figure 1), obtained by substracting the original matrix.

Effect of different percentages of unknown residues

In total three different maximum percentage of permitted unknown residues within a sequence were tested, namely 5%, 10% and 20%. The number of blocks remaining with the different thresholds are shown in table 1. It is clear that a lower percentage leads to less blocks remaining, because several sequences that slightly exceed the threshold, are not contributing with their secondary structure. However, the BLOSUM62 matrices composed for the databases with the three different percentages, do not show significant differences. Some scores increase or decrease by one point, but the majority of the scores in the matrices are the same for all three thresholds.

The average fractions of sequences with known structures and sequences with unknown structures are 0.0193 for the 5% threshold, 0,0199 for the 10% threshold and 0,0203 for the 20% threshold. For all threshold percentages holds that on average only one in fifty sequence has known secondary structure. From these results it can be concluded that no significant effect is caused by different percentages of tolerated unknown residues, at least for percentages below 20%. All matrices shown in figures 5-10 are composed with the 10% threshold.

Total number of blocks in Blocks+ database	29068
Blocks with accession number in pdbmast.dat	16876
Maximum number of unknown residues set at 20%	
Number of blocks in alpha helix specific database	2971
Number of blocks in beta strand specific database	3285
Number of blocks in other structures specific database	4595
Maximum number of unknown residues set at 10%	
Number of blocks in alpha helix specific database	2929
Number of blocks in beta strand specific database	3208
Number of blocks in other structures specific database	4481
Maximum number of unknown residues set at 5%	
Number of block in alpha helix specific database	2833
Number of block in beta strand specific database	3062
Number of block in other structures specific database	4272

Table 1: Number of blocks in different databases.

Comparison of actual and original secondary structure specific matrices

Figures 5-7 show the secondary structure specific BLOSUM62 matrices. The upper halfs of the figures show the difference with the secondary structure based matrices generated by Lüthy et al. In the new secondary structure based matrices the same tendency as in the complete BLOSUM62 matrix can be seen for tryptophan. Again it seems more mutable and hence less conserved. Though, this difference was also shown by Henikoff and Henikoff [5], and may be caused partly by the different algorithms that were used for the construction of the matrices.

Comparison of secondary structure specific and complete matrices

Figures 8-10 again show the secondary structure specific BLOSUM62 matrices. This time the upper halfs show the difference with the actual complete BLOSUM62 matrix. When comparing the secondary structure based matrices with the complete BLOSUM62 matrix, a couple of amino acids attract attention. First, cysteine (C) has a clearly lower conserving score in the alpha helix specific matrix than in the other matrices. This was also the most important difference among the matrices in Lüthy's paper [2]. Therefore it can be concluded that cysteine seems less conserved in alpha helices than in other secondary structures.

Second, both tryptophan (W) and tyrosine (Y) show the opposite, they seem to be less conserved in beta strands and other secondary stuctures. In alpha helices tryptophan and tyrosine have conserving scores that are equal to the scores in the complete matrix.

Finally, the majority of the diagonal scores is lower in all of the secondary structure specific matrices than in the complete matrix. This is probably caused by the fact that the secondary structure specific matrices are calculated over blocks databases with substantial lower numbers of blocks.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	
	1	-1	-1	-1	1	-1	-1	1	1	0	1	0	-1	1	0	1	0	2	0	1	A
		2	-1	0	-2	0	1	-1	1	-1	-1	1	-1	0	-1	-1	-1	2	-1	-1	R
A	3		4	0	-1	0	-1	-1	0	-1	-1	1	0	0	0	-1	0	3	0	0	N
R	-1	4		2	-3	0	2	-1	0	-1	0	0	-1	0	0	-1	0	3	-1	-1	D
N	-1	0	5		1	-2	-2	1	0	-1	1	-1	0	0	-2	0	-1	0	0	1	C
D	-1	0	1	5		3	1	-1	1	-1	0	1	-1	0	-1	0	-1	1	0	-1	Q
C	1	-2	-1	-3	6		2	-1	0	-1	0	1	-1	0	-1	0	-1	3	0	-1	E
Q	-1	1	1	1	-2	4		1	-1	-1	0	-1	0	1	0	0	0	4	0	0	G
E	-1	1	0	3	-3	2	4		1	0	0	0	0	1	-1	0	0	2	1	0	H
G	1	-1	0	-1	1	-1	-1	5		0	1	-1	0	0	0	-1	0	2	0	1	I
H	0	1	1	0	-1	1	0	-1	6		-2	0	1	2	0	0	0	2	1	1	L
I	-1	-2	-2	-2	0	-2	-2	-2	-1	4		2	-1	1	0	-1	-1	1	0	-1	K
L	-1	-2	-2	-2	0	-1	-2	-2	-1	2	3		0	0	-1	-1	0	1	1	0	M
K	-1	2	1	0	-2	1	1	-1	0	-2	-2	5		-3	1	0	1	1	0	1	F
M	-1	-1	-1	-2	0	-1	-2	-1	-1	1	2	-2	4		2	-1	0	4	-1	0	P
F	-1	-2	-2	-2	0	-2	-3	-2	0	0	2	-2	1	6		1	1	3	-1	-1	S
P	0	-1	0	1	-2	0	0	0	-1	-1	-2	0	-2	-2	8		2	1	0	0	T
S	1	-1	0	0	1	0	0	1	0	-2	-2	-1	-1	-2	0	3		-5	-1	1	W
T	0	-1	0	0	0	-1	-1	0	0	0	-1	-1	0	-1	0	2	3		-2	1	Y
W	-2	-1	-1	-1	-1	-1	-1	-1	0	-1	0	-3	0	3	-1	-1	-2	11		-1	V
Y	-2	-2	-1	-2	-1	-1	-2	-1	1	-1	0	-2	0	3	-2	-2	-1	3	8		
V	0	-2	-1	-2	1	-2	-2	-1	-1	2	1	-2	1	0	-1	-1	0	-1	-1	3	
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	

Fig. 5: The lower half of the figure shows the alpha helix specific BLOSUM62 matrix. The upper half shows the difference with the alpha helix specific PAM250 matrix as calculated by Lüthy et al, obtained by substracting the latter.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	
	1	-1	-1	-1	2	-1	-1	0	-1	0	0	-1	0	0	-1	-1	0	1	0	1	A
		0	0	0	0	0	1	-1	1	-1	0	2	0	-1	1	0	-1	0	0	-1	R
A	3		2	1	0	-1	-1	0	0	0	0	0	-1	0	-1	0	0	1	-1	-1	N
R	-1	4		1	1	-1	0	0	-2	0	0	-1	-1	0	1	-1	-1	2	-1	0	D
N	-1	0	4		0	0	0	1	-1	1	3	0	3	2	0	1	1	4	1	3	C
D	-1	0	2	6		1	1	-1	0	0	0	0	0	-1	0	1	0	2	-1	0	Q
C	1	-2	-1	-1	10		1	-2	-1	-1	-1	0	-1	-1	1	0	0	2	-2	0	E
Q	-1	1	0	0	-1	4		1	0	-1	1	-1	-1	0	1	-1	-1	2	0	0	G
E	-1	1	0	2	-2	2	4		0	0	0	0	-1	0	1	0	0	0	-1	1	H
G	1	-1	0	0	-1	-1	-1	6		0	0	-1	0	0	-1	0	0	0	1	0	I
H	-1	1	1	0	-2	1	0	-1	6		0	-1	0	1	-1	0	0	2	0	0	L
I	-1	-2	-1	-2	-2	-1	-2	-2	-2	3		1	-1	-1	1	1	0	1	-1	-1	K
L	-1	-1	-1	-2	0	-1	-2	-1	-1	1	3		0	1	0	-1	-1	3	0	0	M
K	-1	3	1	0	-2	1	1	-1	0	-2	-2	4		-1	-1	0	0	0	1	1	F
M	0	0	-1	-2	0	0	-1	-1	-1	1	1	-1	3		-2	0	1	4	-1	-1	P
F	-1	-2	-1	-2	-1	-2	-2	-2	0	0	1	-2	1	5		0	0	2	-1	0	S
P	0	0	-1	0	-1	0	0	0	0	-1	-1	0	0	-1	6		1	1	0	0	T
S	0	0	1	0	0	1	1	0	0	-1	-2	1	-1	-1	0	3		-4	1	1	W
T	0	0	0	-1	-1	0	0	-1	0	-1	-1	0	-1	-1	1	1	3		1	1	Y
W	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	0	-1	1	2	1	0	-1	9		1	V
Y	-1	-1	-1	-2	-1	-1	-2	-1	1	-1	-1	-2	0	3	-1	-1	-1	2	6		
V	0	-2	-2	-2	0	-1	-1	-1	-1	1	1	-2	0	0	-1	-1	0	-1	-1	3	
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	

Fig. 6: The lower half of the figure shows the beta strand specific BLOSUM62 matrix. The upper half shows the difference with the beta strand specific PAM250 matrix as calculated by Lüthy et al, obtained by substracting the latter.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	
	1	0	-1	-1	2	0	0	-1	-1	0	0	0	0	2	0	0	0	0	1	0	A
		1	0	0	0	0	0	0	0	0	0	1	-1	1	0	-1	0	-1	-1	-1	R
A	2		3	0	0	0	0	0	0	-1	0	0	-1	0	0	0	0	0	-1	-1	N
R	0	4		2	0	0	0	0	0	-1	-1	0	-1	0	0	0	-1	0	0	0	D
N	-1	0	4		-1	-1	0	1	1	1	1	0	2	2	2	1	0	-1	0	1	C
D	-1	0	1	5		3	0	0	0	-1	0	0	0	1	1	0	0	0	0	-1	Q
C	1	-1	-1	-2	10		1	-1	0	0	0	-1	0	1	0	-1	0	1	0	-1	E
Q	0	1	0	0	-2	4		0	0	0	0	0	1	1	1	0	0	1	0	-1	G
E	0	0	0	1	-2	1	3		1	0	0	0	0	1	0	0	0	1	1	0	H
G	-1	-1	0	-1	-1	-1	-1	4		-1	0	0	-1	0	1	0	0	1	0	0	I
H	-1	0	0	0	-1	0	0	-1	7		-2	1	0	0	1	0	1	1	0	0	L
I	0	-1	-1	-2	0	-1	-1	-2	-1	4		-1	0	2	1	0	0	1	0	0	K
L	0	-1	-1	-2	0	-1	-1	-2	-1	2	3		-1	0	1	0	0	1	1	-1	M
K	0	2	0	0	-2	1	0	-1	-1	-1	-1	3		-3	2	1	1	2	1	1	F
M	0	-1	-1	-2	1	0	-1	-1	0	1	2	-1	4		-1	0	0	1	1	0	P
F	0	-1	-1	-2	0	-1	-1	-2	0	1	1	-1	1	5		0	0	0	0	0	S
P	0	-1	-1	-1	-1	0	0	-1	-1	-1	-1	0	-1	-1	5		0	1	0	0	T
S	0	-1	0	0	0	0	-1	0	0	-1	-1	0	-1	-1	0	2		-4	-2	0	W
T	0	0	0	-1	-1	0	0	-1	-1	0	0	0	0	-1	-1	1	3		-1	0	Y
W	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	-1	1	3	-1	-1	-1	9		-1	V
Y	0	-1	-1	-1	0	-1	-1	-2	1	0	0	-1	1	3	-1	-1	-1	2	5		
V	0	-1	-1	-1	0	-1	-1	-2	-1	2	1	-1	1	1	-1	-1	0	0	0	3	
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	

Fig. 7: The lower half of the figure shows the BLOSUM62 matrix based on other secondary structures. The upper half shows the difference with the other structures specific PAM250 matrix as calculated by Lüthy et al, obtained by substracting the latter.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	
	-1	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	A
		-1	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	-1	0	R
A	3		-1	-1	0	1	0	0	0	0	0	1	0	0	1	-1	0	1	0	1	N
R	-1	4		-1	-1	0	1	0	0	1	1	0	1	1	1	0	1	1	0	1	D
N	-1	0	5		-4	0	0	2	0	0	1	0	0	1	0	1	0	0	0	0	C
D	-1	0	1	5		-1	0	0	0	0	1	0	0	0	1	0	-1	0	0	0	Q
C	1	-2	-1	-3	6		-2	1	0	1	0	0	0	0	1	1	0	1	0	0	E
Q	-1	1	1	1	-2	4		-1	0	1	1	0	1	0	1	1	1	1	1	2	G
E	-1	1	0	3	-3	2	4		-2	1	1	0	0	1	0	0	1	0	0	1	H
G	1	-1	0	-1	1	-1	-1	5		0	0	1	0	-1	1	0	1	0	0	-1	I
H	0	1	1	0	-1	1	0	-1	6		-1	0	0	0	0	0	0	0	0	0	L
I	-1	-2	-2	-2	0	-2	-2	-2	-1	4		-1	0	1	1	0	0	-1	0	0	K
L	-1	-2	-2	-2	0	-1	-2	-2	-1	2	3		-1	0	0	0	0	0	0	0	M
K	-1	2	1	0	-2	1	1	-1	0	-2	-2	5		0	0	0	0	0	0	0	F
M	-1	-1	-1	-2	0	-1	-2	-1	-1	1	2	-2	4		0	0	1	0	0	1	P
F	-1	-2	-2	-2	0	-2	-3	-2	0	0	2	-2	1	6		-1	0	1	0	1	S
P	0	-1	0	1	-2	0	0	0	-1	-1	-2	0	-2	-2	8		-1	-1	1	0	T
S	1	-1	0	0	1	0	0	1	0	-2	-2	-1	-1	-2	0	3		-1	0	0	W
T	0	-1	0	0	0	-1	-1	0	0	0	-1	-1	0	-1	0	2	3		0	0	Y
W	-2	-1	-1	-1	-1	-1	-1	-1	0	-1	0	-3	0	3	-1	-1	-2	11		-1	V
Y	-2	-2	-1	-2	-1	-1	-2	-1	1	-1	0	-2	0	3	-2	-2	-1	3	8		
V	0	-2	-1	-2	1	-2	-2	-1	-1	2	1	-2	1	0	-1	-1	0	-1	-1	3	
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	

Fig. 8: The lower half of the figure shows the alpha helix specific BLOSUM62 matrix. The upper half shows the difference with the actual complete BLOSUM62 matrix, obtained by substracting the complete BLOSUM62 matrix.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	
	-1	0	0	0	0	0	0	1	0	0	0	0	1	1	1	-1	0	1	1	0	A
		-1	0	0	0	0	0	0	0	0	1	0	2	0	1	1	1	0	0	0	R
A	3		-2	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	0	N
R	-1	4		0	1	-1	0	1	0	1	1	0	1	1	0	0	0	1	0	1	D
N	-1	0	4		0	1	1	0	-1	-2	1	0	0	0	1	0	-1	0	0	-1	C
D	-1	0	2	6		-1	0	0	0	1	1	0	1	0	1	1	0	0	0	1	Q
C	1	-2	-1	-1	10		-2	1	0	1	0	0	1	1	1	2	1	1	0	1	E
Q	-1	1	0	0	-1	4		0	0	1	2	0	1	0	1	0	0	1	1	2	G
E	-1	1	0	2	-2	2	4		-2	0	1	0	0	1	1	0	1	1	0	1	H
G	1	-1	0	0	-1	-1	-1	6		-1	-1	1	0	-1	1	1	0	0	0	-2	I
H	-1	1	1	0	-2	1	0	-1	6		-1	0	-1	-1	1	0	0	0	-1	0	L
I	-1	-2	-1	-2	-2	-1	-2	-2	-2	3		-2	1	1	1	2	1	1	0	0	K
L	-1	-1	-1	-2	0	-1	-2	-1	-1	1	3		-2	0	2	0	-1	1	0	-1	M
K	-1	3	1	0	-2	1	1	-1	0	-2	-2	4		-1	1	1	0	-1	0	0	F
M	0	0	-1	-2	0	0	-1	-1	-1	1	1	-1	3		-2	0	2	2	1	1	P
F	-1	-2	-1	-2	-1	-2	-2	-2	0	0	1	-2	1	5		-1	-1	2	1	1	S
P	0	0	-1	0	-1	0	0	0	0	-1	-1	0	0	-1	6		-1	0	1	0	T
S	0	0	1	0	0	1	1	0	0	-1	-2	1	-1	-1	0	3		-3	-1	0	W
T	0	0	0	-1	-1	0	0	-1	0	-1	-1	0	-1	-1	1	1	3		-2	0	Y
W	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	0	-1	1	2	1	0	-1	9		-1	V
Y	-1	-1	-1	-2	-1	-1	-2	-1	1	-1	-1	-2	0	3	-1	-1	-1	2	6		
V	0	-2	-2	-2	0	-1	-1	-1	-1	1	1	-2	0	0	-1	-1	0	-1	-1	3	
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	

Fig. 9: The lower half of the figure shows the beta strand specific BLOSUM62 matrix. The upper half shows the difference with the actual complete BLOSUM62 matrix, obtained by substracting the complete BLOSUM62 matrix.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	
	-2	1	0	0	0	1	1	-1	0	1	1	1	1	2	1	-1	0	1	2	0	A
		-1	0	0	1	0	-1	0	-1	1	1	-1	1	1	0	0	1	0	0	1	R
A	2		-2	-1	0	0	0	0	-1	1	1	0	0	1	0	-1	0	1	0	1	N
R	0	4		-1	0	-1	-1	0	0	1	1	0	1	1	-1	0	0	1	1	2	D
N	-1	0	4		0	0	1	0	0	0	1	0	1	1	1	0	-1	0	1	-1	C
D	-1	0	1	5		-1	-1	0	-1	1	1	0	1	1	1	0	0	0	0	1	Q
C	1	-1	-1	-2	10		-3	1	0	2	1	-1	1	2	1	0	1	1	1	1	E
Q	0	1	0	0	-2	4		-2	0	1	1	0	1	0	0	0	0	1	0	1	G
E	0	0	0	1	-2	1	3		-1	1	1	-1	1	1	0	0	0	0	0	1	H
G	-1	-1	0	-1	-1	-1	-1	4		0	0	2	0	0	1	1	1	1	1	-1	I
H	-1	0	0	0	-1	0	0	-1	7		-1	1	0	-1	1	1	1	0	0	0	L
I	0	-1	-1	-2	0	-1	-1	-2	-1	4		-3	1	2	1	1	1	1	1	1	K
L	0	-1	-1	-2	0	-1	-1	-2	-1	2	3		-1	0	1	0	0	1	1	0	M
K	0	2	0	0	-2	1	0	-1	-1	-1	-1	3		-1	1	1	0	0	0	1	F
M	0	-1	-1	-2	1	0	-1	-1	0	1	2	-1	4		-3	0	0	0	1	1	P
F	0	-1	-1	-2	0	-1	-1	-2	0	1	1	-1	1	5		-2	-1	1	1	1	S
P	0	-1	-1	-1	-1	0	0	-1	-1	-1	-1	0	-1	-1	5		-1	0	1	0	T
S	0	-1	0	0	0	0	-1	0	0	-1	-1	0	-1	-1	0	2		-3	-1	1	W
T	0	0	0	-1	-1	0	0	-1	-1	0	0	0	0	-1	-1	1	3		-3	1	Y
W	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	-1	1	3	-1	-1	-1	9		-1	V
Y	0	-1	-1	-1	0	-1	-1	-2	1	0	0	-1	1	3	-1	-1	-1	2	5		
V	0	-1	-1	-1	0	-1	-1	-2	-1	2	1	-1	1	1	-1	-1	0	0	0	3	
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	

Fig. 10: The lower half of the figure shows the other secondary structure specific BLOSUM62 matrix. The upper half shows the difference with the actual complete BLOSUM62 matrix, obtained by substracting the complete BLOSUM62 matrix.

DISCUSSION

The newly created secondary structure specific exchange matrices provide a good starting point for the calculation of profiles for the profile method. However, several steps in the production of the matrices still have to be improved. First, the DSSP files used are not the most recent entries. Most of them were last updated in 2004 or 2005. Moreover, for a lot of PDB sequences no DSSP file was available. More recent downloads probably lead to DSSP files of more PDB entries, and they might have less unknown amino acids. Besides, several PDB sequences have different numbering of the amino acids in the DSSP file and the BLAST result. This is also expected to be caused by the fact that old DSSP files were used.

Second, a non-redundant BLAST was performed. This means that all 100% identical PDB sequences with different identifiers were only coupled once to a UniProt name. As a result, only one DSSP file was opened for each Blocks sequence. While different PDB sequences or protein chains may have identical amino acids, their secondary structures might be different. Consequently, it is possible that the secondary structure extracted from the DSSP file is not the most representative. Currently, the script (see Appendix) uses the first DSSP entry with a valid sequence and extracts the secondary structure from the corresponding file and chain. A better solution may be the use of an extra mapping with all identical PDB sequences. Then, for each of these PDB identifiers the secondary structure has to be extracted. Finally, a consensus or majority structure has to be calculated, which then can be used for splitting the sequence into secondary structure parts.

Another point that is susceptible to improvement, is the opening of the DSSP files. Several blocks contain sequence segments of the same protein. In addition, different but similar proteins often have identical sequence segments. In both cases the same DSSP file is opened several times. The scipt can run faster if every DSSP file is only opened once, and the necessary data is provisionally stored.

Besides the possible improvements, some other issues are worth considering. For instance, as mentioned before, the big majority (∼98%) of all sequences in most blocks are copied straight to the sub databases, as there is no information available about the secondary structure. Nevertheless, these structures do join in the final sub databases. There might be situations in which most of the amino acids on a position have another secondary structure than assigned. Moreover, there are a lot of blocks in which only one sequence in the entire block has a known secondary structure. This sequence will establish the secondary stucture for the other sequences in the block, though might not be representative. On the

other hand, one sequence per block is not sufficient to calculate a BLOSUM matrix, so either the sequences with unknown structure have to be taken into account or the complete block should be neglected.

In addition to this paper, the secondary structure specific matrices can be further extended. In their paper, Lüthy et al. did not only distinguish between alpha helix, beta strand and other secondary structures, but also between interior and exterior residues. Interior and exterior residues were estimated by calculating the accessible surface area from the DSSP files. This resulted in several classes of mutational matrices, which could be compared when using the profile method. The classes used by Lüthy were a class with three matrices (alpha helix, beta strand and other), a class with two matrices (inside resiues and outside residues) and a combined class with six matrices (alpha inside, alpha outside, beta inside, beta outside, other inside, other outside). In a next study, the matrices in this paper can be extended to the three classes of matrices.

References

[1] M. Gribskov A.D. McLachlan and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA*, 84:4355–4358, 1987.

[2] R. Lüthy A.D. McLachlan and D. Eisenberg. Secondary structure-based profiles: Use of structure-conserving scoring tables in searching protein sequence databases for structural similarities. *PROTEINS: Structure, Function and Genetics*, 10:229–239, 1991.

[3] http://www.wwpdb.org/stats.html.

[4] Paul Higgs and Teresa Attwood. *Bioinformatics and Molecular Evolution*, pages 65–75. Blackwell Publishing, 2005.

[5] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89:10915–10919, 1992.

[6] W. Pearson. Comparison of methods for searching protein sequence databases. *Protein Science*, 4:1145–1160, 1995.

[7] S. Henikoff and J.G. Henikoff. Performance evaluation of amino acid substitution matrices. *PROTEINS: Structure, Function and Genetics*, 17:49–61, 1993.

[8] J.G. Henikoff S. Henikoff and S. Pietrokovski. Blocks+: a non-redundant database of protein alignment blocks derived from multiple complations. *Bioinformatics*, 15:471–479, 1999.

[9] W. Kabsch and C. Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22:2577–2637, 1983.

APPENDIX: PYTHON SCRIPTS

Main script

```
#This code runs through the original blocks database and splits sequences into parts with the same
#secondary structure (alpha helix, beta strand or other). From these parts of sequences three new
#blocks databases are composed, each one with only those residues that are part of the same
#secondary structure.

#The definitions of executed functions can be found in the file functions.py


#IMPORT FUNCTIONS-------------------------------------------------------------------------------
import functions


#COUNT NUMBER OF BLOCKS IN ORIGINAL BLOCKS DATABASE---------------------------------------------
Blocks = open('/home/feenstra/xd/blocks/blocks/blocks.dat','r')  #original blocks database
NumOfBlocks = functions.CountNumberOfBlocks(Blocks)
print NumOfBlocks, 'blocks in original blocks database\n'
Blocks.close()


#OPEN FILES-------------------------------------------------------------------------------------
Blocks = open('/home/feenstra/xd/blocks/blocks/blocks.dat','r')
Pdbmast = open('/home/feenstra/xd/blocks/blocks/pdbmast.dat','r').readlines()
Uniref = open('/home/ibivu/data/marlies/pdbNR_vs_uniref100_tab','r').readlines()

HelTemp = open('blocks_hel_temp.dat','w')       #temporary helix blocks database
ExtTemp = open('blocks_ext_temp.dat','w')       #temporary strand blocks database
CoiTemp = open('blocks_coi_temp.dat','w')       #temporary coil blocks database

HelFinal = open('blocks_hel.dat','w')           #final helix blocks database
ExtFinal = open('blocks_ext.dat','w')           #final strand blocks database
CoiFinal = open('blocks_coi.dat','w')           #final coil blocks database


#COUPLE BLOCK NUMBERS TO PDB NAMES AND UNIPROT NAMES TO PDB NAMES -------------------------------
NumAndPdb = functions.CoupleBlockNumToPdbNames(Pdbmast)
UniAndPdb = functions.CoupleUniNamesToPdbNames(Uniref)


#COPY FIRST 7 LINES OF BLOCKS DATABASE TO FINAL SUB DATABASES-----------------------------------
BlocksHeader = functions.ReadOneBlock(Blocks)
for j in range(len(BlocksHeader)):
   functions.CopyToSubDatabases(HelFinal,ExtFinal,CoiFinal,BlocksHeader[j])


#PROCESS BLOCK SEQUENCE BY SEQUENCE-------------------------------------------------------------
for i in range(1,NumOfBlocks+1):
   OneBlock = functions.ReadOneBlock(Blocks)    #list with lines of exactly one block
   functions.ProcessBlock(OneBlock,NumAndPdb,UniAndPdb,HelTemp,ExtTemp,CoiTemp)
```

```
#CLOSE AND REOPEN SUB DATABASES-----------------------------------------------------------------
HelTemp.close()
ExtTemp.close()
CoiTemp.close()

HelTemp = open('blocks_hel_temp.dat','r')
ExtTemp = open('blocks_ext_temp.dat','r')
CoiTemp = open('blocks_coi_temp.dat','r')


#COUNT NUMBER OF BLOCKS IN SUB BLOCKS DATABASES-------------------------------------------------
NumOfBlocks = functions.CountNumberOfBlocks(HelTemp)
print '\n', NumOfBlocks, 'blocks in sub blocks databases'

HelTemp.close()
HelTemp = open('blocks_hel_temp.dat','r')


#COPY PARTS OF BLOCKS FROM TEMPORARY TO FINAL SUB DATABASES-------------------------------------
functions.TempToFinal(NumOfBlocks,HelTemp,HelFinal)
functions.TempToFinal(NumOfBlocks,ExtTemp,ExtFinal)
functions.TempToFinal(NumOfBlocks,CoiTemp,CoiFinal)
```

Additional functions

```
#This script contains all the functions that are used in the script pars.py


#COUNT NUMBER OF BLOCKS IN BLOCKS DATABASE------------------------------------------------------
def CountNumberOfBlocks(File):

    List = []
    for Line in File:
        if Line.startswith('ID '):
            List.append(1)

    return List.count(1)


#COUPLE BLOCK NUMBERS TO PDB NAMES IN DICTIONARY------------------------------------------------
def CoupleBlockNumToPdbNames(File):

    BlockNums = []
    for Line in File:
        BlockNums.append(Line.split()[0])          #make list of block numbers

    Dictionary = {}
    for BlockNum in list(set(BlockNums)):          #remove duplicates block numbers
        PdbNames = []
        for Line in File:
            if Line.startswith(BlockNum):
                PdbNames.append(Line.split()[1])
        Dictionary[BlockNum] = PdbNames            #couple list of pdb names to block number

    return Dictionary
```

```
#PARSE BLAST RESULT-----------------------------------------------------------------------------
def ParseBlastResult(Line):

    PdbName = Line.split()[0]
    PdbStart = int(Line.split()[6])              #start position of pdb sequence in blast result
    PdbEnd = int(Line.split()[7])
    UniStart = int(Line.split()[8])              #start position of uniprot sequence
    UniEnd = int(Line.split()[9])

    return (PdbName,PdbStart,UniStart,UniEnd)


#COUPLE UNIPROT NAMES TO PDB NAMES IN DICTIONARY-------------------------------------------------
def CoupleUniNamesToPdbNames(File):

    UniNames = []
    for Line in File:
        UniNames.append(Line.split()[1][10:])     #make list of pdb names

    Dictionary = {}
    for UniName in list(set(UniNames)):           #remove duplicates
        PdbNames = []
        for Line in File:
            if Line.find(UniName) != -1:
                (PdbName,PdbStart,UniStart,UniEnd) = ParseBlastResult(Line)
                PdbNames.append((PdbName,PdbStart,UniStart,UniEnd))
        Dictionary[UniName] = PdbNames            #couple list of uniprot names to pdb name

    return Dictionary


#READ ONE BLOCK----------------------------------------------------------------------------------
def ReadOneBlock(File):

    Lines=[]
    for Line in File:
        Lines.append(Line)
        if Line.startswith('    ===') or Line.startswith('//'):
            break

    return Lines                                  #list with lines of exactly one block


#COPY LINES FROM BLOCKS DATABASE TO SUB DATABASES------------------------------------------------
def CopyToSubDatabases(HelTemp,ExtTemp,CoiTemp,Line):

    HelTemp.write(Line)
    ExtTemp.write(Line)
    CoiTemp.write(Line)
```

```
#PARSE BLOCKS DATABASE-------------------------------------------------------------------------------
def ParseBlocksDatabase(Line):

    BlocksSeq = Line.split(') ')[1].split()[0]   #blocks sequence
    BlocksLength = len(BlocksSeq)
    BlocksStart = int(Line.split(') ')[0][-4:])  #start position of blocks sequence
    BlocksEnd = BlocksStart + BlocksLength

    return (BlocksSeq,BlocksLength,BlocksStart,BlocksEnd)


#CALCULATE POSITIONS OF AMINO ACIDS-------------------------------------------------------------------
def CalculatePositions(Line,PdbStart,UniStart,UniEnd):

    (BlocksSeq,BlocksLength,BlocksStart,BlocksEnd) = ParseBlocksDatabase(Line)

    Range = range(max(UniStart,BlocksStart),min(UniEnd,BlocksEnd)) #range of sequence positions

    DiffEnum = UniStart-PdbStart                 #difference in enumeration in uniprot and pdb

    DsspPos = []
    for Position in Range:
        DsspPos.append(Position-DiffEnum)        #range of sequence positions in dssp file

    BlocksPos = {}
    for i in range(BlocksLength):
        BlocksPos[BlocksStart+i] = BlocksSeq[i]  #couple positions to residues in dictionary

    return (Range,DsspPos,BlocksPos)


#PARSE BLOCKS SEQUENCE--------------------------------------------------------------------------------
def ParseBlocksSeq(Range,Positions,Line):

    OriginalBlocksSeq = ParseBlocksDatabase(Line)[0]

    CompleteBlocksSeq = ""
    ReducedBlocksSeq = ""

    for Position in sorted(Positions.keys()):
        if Position in Range:                    #residues that are in dssp file
            CompleteBlocksSeq += Positions[Position]
            ReducedBlocksSeq += Positions[Position]
        elif Position not in Range:
            CompleteBlocksSeq += '-'             #residues that are not in dssp file

    NewLine = Line.replace(OriginalBlocksSeq,CompleteBlocksSeq)

    return (NewLine,ReducedBlocksSeq)
```

```
#PARSE DSSP FILE-------------------------------------------------------------------------------
def ParseDsspFile(DsspPos,PdbName):

    try:
        Dssp = open('/home/ibivu/data/dssp/'+PdbName[:4]+'.dssp','r')

        Start = False
        Sequence = ""
        SecStruct = ""

        for DsspLine in Dssp:
            if DsspLine.split()[0] == '#':          #indicate start of sequence
                Start = True
                continue

            if not Start:                           #skip introduction
                continue

            Chain = DsspLine[11]
            if Chain == PdbName[-1]:
                Position = int(DsspLine[0:5])

                if Position in DsspPos:
                    Sequence += DsspLine[13]
                    SecStruct += DsspLine[16]

        return (Sequence,SecStruct)

    except IOError:
        print 'File', PdbName[:4]+'.dssp does not exist'


#GET SECONDARY STRUCTURE OF SEQUENCE-----------------------------------------------------------
def GetSecStruct(DsspPos,PdbName,NewLine,BlocksSeq,HelTemp,ExtTemp,CoiTemp):

    HelSequence = ""
    ExtSequence = ""
    CoiSequence = ""

    SecStruct = ParseDsspFile(DsspPos,PdbName)[1]

    for i in range(len(SecStruct)):
        if SecStruct[i] == 'H':
            HelSequence += BlocksSeq[i]             #append residues in alpha helix
            ExtSequence += '-'
            CoiSequence += '-'
        elif SecStruct[i] == 'E':
            HelSequence += '-'
            ExtSequence += BlocksSeq[i]             #append residues in beta strand
            CoiSequence += '-'
        else:
            HelSequence += '-'
            ExtSequence += '-'
            CoiSequence += BlocksSeq[i]             #append all other residues

    HelTemp.write(NewLine.replace(BlocksSeq,HelSequence))
    ExtTemp.write(NewLine.replace(BlocksSeq,ExtSequence))
    CoiTemp.write(NewLine.replace(BlocksSeq,CoiSequence))
```

```
#COMPARE SEQUENCES IN BLOCKS DATABASE AND DSSP FILE-----------------------------------------------
def CompareSeqs(AllPdbNames,UniAndPdb,UniName,Line,HelTemp,ExtTemp,CoiTemp):

    for i in range(len(AllPdbNames)):
        (PdbName,PdbStart,UniStart,UniEnd) = UniAndPdb[UniName][i]
        (Range,DsspPos,BlocksPos) = CalculatePositions(Line,PdbStart,UniStart,UniEnd)

        if Range == []:
            print UniName, PdbName,': No joint positions in blocks seq and blast result'
            if i != len(AllPdbNames)-1:            #all pdb names except the last one
                continue
            elif i == len(AllPdbNames)-1:          #last pdb name in list of pdb names
                CopyToSubDatabases(HelTemp,ExtTemp,CoiTemp,Line.lower())
                break                              #break out of loop if no result for any pdb name

        else:
            try:
                DsspSeq = ParseDsspFile(DsspPos,PdbName)[0]
            except TypeError:                      #dssp file not available
                if i != len(AllPdbNames)-1:
                    continue
                elif i == len(AllPdbNames)-1:
                    CopyToSubDatabases(HelTemp,ExtTemp,CoiTemp,Line.lower())
                    break

            (NewLine,BlocksSeq) = ParseBlocksSeq(Range,BlocksPos,Line)

            if BlocksSeq == DsspSeq:
                GetSecStruct(DsspPos,PdbName,NewLine,BlocksSeq,HelTemp,ExtTemp,CoiTemp)
                break

            elif BlocksSeq != DsspSeq:
                DiffResidues = ""
                if len(BlocksSeq) == len(DsspSeq):
                    for j in range(len(BlocksSeq)):
                        if BlocksSeq[j] == DsspSeq[j]:
                            DiffResidues += BlocksSeq[j]
                        if BlocksSeq[j] != DsspSeq[j]:
                            DiffResidues += 'X'        #replace different resiues by 'X'

                    if DiffResidues.count('X') <= 0.1*len(BlocksSeq):  #max 10% of residues different
                        DiffResLine = Line.replace(BlocksSeq,DiffResidues)
                        GetSecStruct(DsspPos,PdbName,DiffResLine,DiffResidues,HelTemp,ExtTemp,CoiTemp)
                        break

                    else:
                        print UniName, PdbName,':', BlocksSeq, 'differs too much from', DsspSeq
                        if i != len(AllPdbNames)-1:
                            continue
                        elif i == len(AllPdbNames)-1:
                            CopyToSubDatabases(HelTemp,ExtTemp,CoiTemp,Line.lower())
                            break

                elif len(BlocksSeq) != len(DsspSeq):
                    print UniName,PdbName, ':', BlocksSeq, 'has not the same length as', DsspSeq
                    if i != len(AllPdbNames)-1:
                        continue
                    elif i == len(AllPdbNames)-1:
                        CopyToSubDatabases(HelTemp,ExtTemp,CoiTemp,Line.lower())
                        break
```

```
#PROCESS SEQUENCE-------------------------------------------------------------------------------
def ProcessSequence(Line,NumAndPdb,UniAndPdb,HelTemp,ExtTemp,CoiTemp,Num):

    UniName = Line[Line.find('|')+1:Line.find('|')+7]  #extract uniprot name

    if UniName not in UniAndPdb:
        CopyToSubDatabases(HelTemp,ExtTemp,CoiTemp,Line.lower())

    elif UniName in UniAndPdb:

        AllPdbNames = []
        for i in range(len(UniAndPdb[UniName])):
            PdbName = UniAndPdb[UniName][i][0]
            if PdbName in NumAndPdb[Num]:
                AllPdbNames.append(PdbName)          #pdb names in both pdbmast and blast result

        if len(AllPdbNames) == 0:
            CopyToSubDatabases(HelTemp,ExtTemp,CoiTemp,Line.lower())

        if len(AllPdbNames) >= 1:
            CompareSeqs(AllPdbNames,UniAndPdb,UniName,Line,HelTemp,ExtTemp,CoiTemp)


#PROCESS BLOCK----------------------------------------------------------------------------------
def ProcessBlock(Block,NumAndPdb,UniAndPdb,HelTemp,ExtTemp,CoiTemp):

    for Line in Block:

        if Line.startswith('AC '):
            Num = Line.split()[1][:-2]               #extract block number

            if Num in NumAndPdb:
                for i in range(4):                   #copy first four lines of block to subdatabases
                    CopyToSubDatabases(HelTemp,ExtTemp,CoiTemp,Block[i])

            elif Num not in NumAndPdb:               #no information, neglect complete block
                print 'Block number', Num, 'not in pdbmast'
                break

        elif Line.startswith('ID ') or Line.startswith('BL ') or Line.startswith('DE '):
            pass

        elif len(Line) <= 3:                         #copy empty lines and '//' to sub databases
            CopyToSubDatabases(HelTemp,ExtTemp,CoiTemp,Line)

        else:
            ProcessSequence(Line,NumAndPdb,UniAndPdb,HelTemp,ExtTemp,CoiTemp,Num)


#COUNT NUMBER OF UPPERCASE LETTERS ON POSITION IN BLOCKS----------------------------------------
def CountUpperCaseLetters(Position):

    import re
    UpperCase=re.findall("[A-Z]",Position)       #make list of uppercase letters on position

    return len(UpperCase)
```

```
#MAKE LIST OF SEQUENCES IN BLOCK-----------------------------------------------------------------
def MakeListOfLongSeqs(Block):

    ListOfLongSeqs = []
    for Line in Block:

        if Line.startswith('ID ') or Line.startswith('AC ') or Line.startswith('DE ')
            or Line.startswith('BL '):
            pass

        elif len(Line) <= 3:
            pass

        else:
            Seq = Line.split(') ')[1].split()[0]
            ListOfLongSeqs.append(Seq)              #append sequence to list of sequences

    return ListOfLongSeqs


#MAKE LIST OF POSITIONS IN BLOCK-----------------------------------------------------------------
def MakeListOfPos(ListOfLongSeqs):

    ListOfPos = []
    for i in range(len(ListOfLongSeqs[0])):
        Position = ""

        for j in range(len(ListOfLongSeqs)):
            Position += (ListOfLongSeqs[j][i])      #AA's on one particular position in all sequences

        Upper = CountUpperCaseLetters(Position)     #count number of upper case letters on position
        Hyphen = Position.count('-')                #count number of hyphens on position

        if Upper >= 1 and Hyphen == 0:              #at least one uppercase letter and no hyphens
            ListOfPos.append(Position)

    return ListOfPos


#MAKE LIST OF SEQUENCES WITHOUT HYPHENS----------------------------------------------------------
def MakeListOfShortSeqs(ListOfPos):

    ListOfShortSeqs = []
    for i in range(len(ListOfPos[0])):
        Sequence = ""

        for j in range(len(ListOfPos)):             #make sequence of all complete positions
            Sequence += ListOfPos[j][i]
        ListOfShortSeqs.append(Sequence)            #append sequence to list of sequences

    return ListOfShortSeqs


#COUPLE ORIGINAL SEQUENCES TO REDUCED SEQUENCES--------------------------------------------------
def CoupleLongSeqsToShortSeqs(ListOfLongSeqs,ListOfShortSeqs):

    Dictionary = {}
    for i in range(len(ListOfLongSeqs)):
        LongSeq = ListOfLongSeqs[i]
        ShortSeq = ListOfShortSeqs[i]
        Dictionary[LongSeq] = ShortSeq

    return Dictionary
```

```
#COPY BLOCKS FROM TEMPORARY SUB DATABASE TO FINAL SUB DATABASE-------------------------------------
def TempToFinal(NumOfBlocks,TempFile,FinalFile):

    FracOfSeqs = []
    for i in range(NumOfBlocks):
        OneBlock = ReadOneBlock(TempFile)

        ListOfLongSeqs = MakeListOfLongSeqs(OneBlock)
        ListOfPos = MakeListOfPos(ListOfLongSeqs)

        if len(ListOfPos) != 0:                    #neglect blocks with no positions left
            UpperCase = CountUpperCaseLetters(ListOfPos[0])
            TotalSeqs = float(len(ListOfPos[0]))
            FracOfSeqs.append(UpperCase/TotalSeqs) #fraction of sequences with known and unknown struct

            ListOfShortSeqs = MakeListOfShortSeqs(ListOfPos)

            LongAndShortSeqs = CoupleLongSeqsToShortSeqs(ListOfLongSeqs,ListOfShortSeqs)

            for Line in OneBlock:
                if Line.startswith('ID ') or Line.startswith('AC ') or Line.startswith('DE ')
                    or Line.startswith('BL '):
                    FinalFile.write(Line)

                elif len(Line) <= 3:
                    FinalFile.write(Line)

                else:
                    Seq = Line.split(') ')[1].split()[0]
                    if Seq in LongAndShortSeqs.keys():
                        NewLine = Line.replace(Seq,LongAndShortSeqs[Seq])
                        FinalFile.write(NewLine.upper())

    FracOfSeqs.sort()
    print FracOfSeqs                               #list of sorted fractions
```

Blocks count script

```
#This script is used to count the number of blocks in a sub blocks database

Blocks=open(raw_input('Enter file name: '),'r') #open blocks database

List = []
for Line in Blocks:
    if Line.startswith('ID '):
        List.append(1)

print List.count(1)
```