Evaluating protein secondary structure predictions

by Kristian Kræmmer Nielsen, 26-08-1980, jkkn04
The University of Southern Denmark, Odense
http://jkkn.dk/sdu/proteins/

Supervisor: Lene Monrad Favrholdt
Assistant Supervisor: Fiona Nielsen
Individual student activity - Summer 2008

Contents

1 Overview and motivation

When studying proteins structures we use three different structures, the primary structure, which is the amino acid sequence of the protein, the secondary structure, which is the local conformation of the protein backbone, and the tertiary structure, which is the whole three-dimensional structure of the protein[1]. Since it is differcult to determine protein structures experimentally, lots of research have went into trying to predict the protein structure directly from the amino acid sequence instead. Over time multiple methods for predicting protein structures have been constructed and immediately a need for a method for evaluating how each predictor performs individually or in combination with others is essential. As a result af the evaluation it would be valuable to know on which factors each predictor is performing good and on which factors it is performing poorly. This knowledge might then be used to improve the original predictor. A lot of existing methods of evaluating predictors are already available, in this project we will look at Q_3, SOV_3 and MCC. These three evaluator all returns a score, which can be seen from the matter of predicting α-helix (H), β-strand (E) and coil (C) in the secondary structure and in case of Q_3 and SOV a total score can also be calculated. Common for them are that there are no other factors evaluated of the predictor, this is not much use in identifying the strong and weak points of the prediction, if trying to improve it. The three evaluators will be described in section 6.1.

The focus in this project have been to implement a new evaluation method where we evaluate the predictors in five different categories. Each time a predicted structure-element is not predicted as it otherwise is observed we calculate the element as a error by category. The five categories are, *bridges*, a predicted segment is extended across a gap in the observed protein segment; *expansion*,

a predicted segment is longer than the observed protein segment; *gap*, there is missing a segment from the predicted protein; *reduction*, a predicted segment is shorter than the observed segment; and *other* for each structure-element where the observed and predictor disagrees in any other way. This method is in this rapport implemented as a state machine and therefore referred to as of now as the State Machine Confusion Matrix (*SMCM*) method.

This article will in chapter 3, describe and define the evaluator and in chapter 4, describe an algorithm and chapter 5 the implementation released as a result of this project. Chapter 6 describes the application and the features of the entire program. Last a comparison have been made with other existing evaluators on two protein families, the chymotrypsin-protein family and the cytokines-protein family.

This project have been made as an individual student activity supervised by Lene Monrad Favrholdt, Southern University of Denmark and Fiona Nielsen, Centre for Molecular and Biomolecular Informatics, Radboud University Nijmegen, The Netherlands.

2 Terminology

This section briefly describes the terminology used in this rapport and project.

Predictor We will use this term for any algorithm or method, which is able to construct a secondary protein structure sequence. In the analysis chapter of this rapport we will be looking at SVM, SVM w/VITERBI, BRNN, BRNN w/VITERBI, HMM21 multisequence, HMM21 unisequence, HMM75 multisequence and HMM75 unisequence.

Observation We will use this term for any observed secondary protein sequence, which we will be evaluating our predictors against. In some literature this is also known as the assignment structure. In the analysis chapter of this rapport we will be looking at DSSP and STRIDE, using two translation types and KAKSI.

Element We will refer to an element as a single part of a sequence, as either one α-helix (H), β-strand (E) and coil (C) element. Also known as residues.

Element-state We will refer to an element as either in α-helix (H), β-strand (E) or coil (C) state.

Evaluator A method of evaluating a predictor. Already known methods are Q_3, SOV_3 and MCC. Chapter 3 to 4 looks into defining the new method.

Category We will use this term for each of the three categories in which we will count a mismatch between an predictor and an observation. In our method: *bridge*, *expansi*on, *reduction*, *gap* or *other*.

3 Analysis and definition of State Machine Confusion Matrix

First of all we need to define exactly what we are to be measuring and what results we are expecting to get from our new evaluator. In this chapter we will describe the details of each category and by example illustrate the different situation we want to be able to evaluate.

The primary idea of the evaluator is to get as much information as possible when comparing a predictor to an observation of a protein. We will therefore introduce five categories in which we will count each mismatch between a predicted element to a observed element and we will also keep a confusion matrix for each category using columns as the element-state the predictor had at the point of a mismatch and rows as the element-state the observation had at the point of a mismatch.

In the following sections we will refer to the bundled test suite data, which have been used to make basic tests of each category. The test marked TEST_A to TEST_O make up examples of the different categories and mixtures of these. The tests marked EXAM_A to EXAM_D are taken from [1]. The EXAM_L test has been taken from the task assignment paper.

3.1 Bridges

First category we will be defining is "bridges". We will define a bridge as a mismatching part of the predicted sequence that repeats the same element-state and where there is a correct element on each side of the mismatching part. Hence we say that the predictor "bridges" over some part of the actual observed sequence.

In Table 1 we show three examples, which demonstrate different scenarios in which bridges occur. In the first example it is shown how a bridge might be the only mismatch in a sequence. In the second example we see a chain of α-helix elements which make up multiple bridge sub-sequences. The last examples shows a mixture of a reduction/expansion mismatch and a bridge mismatch.

	Sequence	**Bridge length**	**Comment**
Predicted:	HHHHHHCCCCCEEE	2	
Observed:	HHCEHHHCCCCCEEE		*In this sequence, only mismatch is the bridge.*
Predicted:	HHHHHHHHHHHHHH	9	
Observed:	HCEHCEHCEHCEHCH		*In this sequence, only mismatch is the bridges.*
Predicted:	HHHHHCCEEEEEEEEE	2	
Observed:	HHHHCCEEEEECCEEE		*There are also 2 reductions and 2 expansions in this sequence.*

Table 1: TEST_B, TEST_C, EXAM_L. The underlined parts of the sequences represents the bridges. The bridge length is countered as how many elements which are in a bridge.

3.2 Gaps

Second category we will be redefining is "gaps". We will define a gap as any part of the predicted sequence which incorrectly changes element-state to something other than seen in the observed sequence, but where it in both ends still agrees with the observed. We note that we will actually define a "gap" exactly the same as a "bridge" but seen from the observed sequence. So we can say if the observed-sequence "bridges" the predicted-sequence has a "gap" and vice-versa.

In Table 2 we show four examples. The first example shows how a gap might occur as the only mismatch in a sequence. We notice that we do not care at all which element-states the predictor is crossing as the predictor and observed sequence agrees on the element-state at the ends and that the observed sequence does not change element-state doing the gap.

3.3 Expansions

Third category we will define is "expansions". We define this as any mismatch in the predicted sequence which is a continues repeation of the previous correct element-state or the coming correct element-state. So we say the element-state was expanded.

In the Table 3 we see eight examples upon expansions. First, fourth and seventh example shows that these can occur by themselves as only mismatch in a sequence and in both ends of

	Sequence	Gap length	Comment
Predicted:	HHCCEHHCCCCEEEE	3	
Observed:	HHHHHHHCCCCEEEE		*In this sequence, only mismatch is the gap.*
Predicted:	CHCHCHCHCHCHCHCH	6	
Observed:	CCCHHHHHCCCCCCCC		*Besides gaps, the last H is a reduction.*
Predicted:	CCHHCCHHHHCCCCCC	2	
Observed:	CCCHHHHHCCCCCCCC		*Besides the gap, there are 3 reductions and 3 expansions.*
Predicted:	CCCHHCHHCCCCCCCC	1	
Observed:	CCCHHHHHCCCCCCCC		*In this sequence, only mismatch is the gap.*

Table 2: TEST_A, EXAM_B, EXAM_C, EXAM_D. The underlined parts of the sequences represents the gaps. The gap length is countered as how many elements which are in a gap.

a correct part of the sequence. Second example shows a combination with an "other"-mismatch, defined below. Third, fifth and sixth example shows the most common case where a expansion occurs at the same time of a reduction of some other element, see the definition of a reduction below. The last example shows how a expansion can occur together with a reduction, but only partly overlapping each other.

3.4 Reductions

Fourth category we will define is "reductions". We define this as any mismatching element of the predicted sequence which is placed where the observed sequence either starts earlier in a element-state or continues to stay in a element-state but the predicted state does not, and only agrees in a shorter part than the observed sequence. Again we notice that this is as with "gaps" and "bridges"; a "reduction" can been seen as an "expansion" seen from the observed sequence. Notice that we, by this definition, often will count mismatches as both expansions and reductions.

And again this is best shown by examples, as we do in Table 4. The first and fifth example shows how a reduction can stand by itself in either or both ends of a sub-sequence. The second, third and fourth shows the typically case where it appears in conjunction with a expansion mismatch. The last example, as with expansions, shows how the two categories can also partly overlap each other.

3.5 Other

Last category we will define is "other". This we simply define as any other mismatch between the predicted and observed sequence. We define this by any mismatch, which have not already counted in any of the other categories.

The Table 5 shows two examples of where we have use for the "other" category. The first example where we have just seen an expansion but afterwards nothing matches. The last example is more obvious, the case where the predicted and observed sequence never agrees on anything.

4 Design and algorithm

In this chapter we will design and explain the algorithm used for obtaining the results for the State Machine Confusion Matrix.

	Sequence	Expansion length	Comment
Predicted:	HHHHHHHHHHHHHHH	8	
Observed:	CCCCCCHHHHHHHCC		*The expansions are the only mismatches.*
Predicted:	HHHHHHCHHHHHHHH	3	
Observed:	HHHCCEEHHHHHHHH		*Also one mismatch is registered as other.*
Predicted:	HHCCCCCCCCCCCCC	2	
Observed:	HHHHCCCCCCCCCCC		*Same elements are also registered as 2 reductions.*
Predicted:	HHHHHHHHEEEEEEE	3	
Observed:	HHHHHCCCEEEEEEE		*The expansion is the only mismatch.*
Predicted:	CCHHHHHHHHHHHHH	3	
Observed:	CCCCCHHHHHHHHHH		*Same elements are also registered as 3 reductions.*
Predicted:	CCEEEEHHHHHHHHH	3	
Observed:	CCCCEEEHHHHHHHH		*Same elements are also registered as 3 reductions.*
Predicted:	CCCCCHHHHHHHHHH	3	
Observed:	CCCEEEHHHHHHHHH		*The expansion is the only mismatch.*
Predicted:	CCCHHHHCCEEEEEE	2	
Observed:	CCCHHEEEEEEEEEE		*There are also a reduction length 4, 2 of which are the same elements.*

Table 3: TEST_D, TEST_F, TEST_G, TEST_H, TEST_I, TEST_J, TEST_K, TEST_M. The underlined parts of the sequences represents the expansions. The expansion length is countered as how many elements which are part of a expansion.

	Sequence	Reduction length	Comment
Predicted:	HHHHHHEEEEEECCC	6	
Observed:	HHHHHHHHHHHCCC		*The reduction is the only mismatch.*
Predicted:	HHCCCCCCCCCCCC	2	
Observed:	HHHHCCCCCCCCCC		*Same elements are also counted as an expansion, length 2.*
Predicted:	CCHHHHHHHHHHHH	3	
Observed:	CCCCCHHHHHHHHH		*Same elements are also counted as an expansion, length 3.*
Predicted:	CCEEEEHHHHHHHH	3	
Observed:	CCCCEEEHHHHHHH		*Same elements are also counted as an expansion, length 3.*
Predicted:	CCCCEEEEHHHHHH	4	
Observed:	CCCCCCHHHHHHHH		*The reductions are the only mismatches.*
Predicted:	CCCHHHHCCEEEEE	4	
Observed:	CCCHHEEEEEEEEE		*There is also an expansion of length 2.*

Table 4: TEST_E, TEST_G, TEST_I, TEST_J, TEST_L, TEST_M. The underlined parts of the sequences represents the reductions. The reduction length is countered as how many total elements are part of a reduction.

	Sequence	"Other" length	Comment
Predicted:	HHHHHHCHHHHHHHH	1	
Observed:	HHHCCEEHHHHHHHH		*There is also an expansion of length 3.*
Predicted:	EECHCHCHCHCHCHC	15	
Observed:	CCHEHEHEHEHEHEH		*All mismatches are registered as other.*

Table 5: TEST_F, TEST_O. The underlined parts of the sequences represents the "other" mismatches. The "other" length is countered as how many total elements are registered as an "other" mismatch.

First we notice that we in the previous chapter observed that a "gap" and a "bridge" are actually the same, if we swap the predicted and observed sequences, this also goes for an "expansion" which is the same as an "reduction" - again we just have to swap the two sequences. The last category "other" is more simple, as we can always just count this as the number of mismatching elements between the two sequences where we have not yet categories the mismatch otherwise.

4.1 Design

We decide that our main algorithm just have to implement a way to count a "bridge" and a to count a "expansion". When we have this implemented we will be able to swap the sequences and automatically we will also have an implementation for the "gap" and "reduction" categories.

The "other"-category we will count afterwards on any mismatch which have not yet been categorized.

We will use a state-machine to implement this, which works on the two sequences. The sequences will have to be of equal length and potential unknowns[1] have to be at the same locations. This preparation will not be part of our algorithm, but assumed done before processing.

The state-machine will have two main states, *CORRECT* or *MISMATCH* - determined by the current element we are looking at, are currently the agreeing state between the predicted and observed sequence or if they are currently disagreeing. We will work from the beginning of the sequence and forward, always advancing one element for each iteration.

4.2 Algorithm

An algorithm for calculating the bridge and expansion counts for a sequence is displayed as Algorithm 1.

First we initialize some variables (lines 2 to 6) we will later use for determining state, these are:

hasRepeated we will use this to determine if the predictor can have repeated a previous element. This is only set to `false` at end of the sequence. In the attached implementation also reset back to `false` after each unknown sub-sequence so that we do not miscount mismatches in such.

countRepeated we will keep track of how many times the current element has been repeated.

elementRepeated the previous element-state which the predictor had predicted.

linksToCorrect this will determine if we are repeating the same element-state as the two sequences last agreed upon.

mismatches the number of current mismatches we have to account for.

[1] elements left out of either the predicted or observed sequence, e.g. due to error

Algorithm 1 Pseudo code for main loop of algorithm for counting "bridges" and "expansions". This corresponds mostly to the function `processSeenFromSequence` in the attached implementation. The `register` method is a method that counts the last `countRepeated` element as mismatches of the specified category.

```
1  function (predicted, observed) {
2      var hasRepeated = false;
3      var countRepeated = 0;
4      var elementRepeated;
5      var linksToCorrect = false;
6      var mismatches = 0;
7
8      foreach (pElement, oElement in predicted, observed) {
9          if (pElement == oElement) {
10             // CORRECT state
11             if (hasRepeated && countRepeated > 0) {
12                 if (linksToCorrect && elementRepeated == pElement) {
13                     // We have a BRIDGE
14                     register(BRIDGE, countRepeated);
15                 } else if (linksToCorrect || elementRepeated == pElement) {
16                     // We have an EXPANSION
17                     register(EXPANSION, countRepeated);
18                 }
19                 countRepeated = 0;
20             }
21             if (mismatches > 0) {
22                 // register OTHER
23                 register(OTHER, mismatches);
24                 mistmatches = 0;
25             }
26
27             elementRepeated = pElement;
28             linksToCorrect = true;
29             hasRepeated = true;
30         } else {
31             // MISMATCH state
32             mismatches ++;
33
34             if (hasRepeated && elementRepeated == pElement) {
35                 countRepeated++;
36             } else {
37                  if (linksToCorrect) {
38                      // We have an EXPANSION (notice that countRepeat may
                            be zero)
39                      register(EXPANSION, countRepeated);
40                  }
41                  linksToCorrect = false;
42                  countRepeated = 1;
43             }
44             elementRepeated = pElement;
45             hasRepeated = true;
46         }
47     }
48     if (mismatches > 0) {
49         // register OTHER
50         register(OTHER, mismatches);
51         mistmatches = 0;
52     }
53 }
```

The main algorithm consists of a loop (lines 8-47) where we compare each element of the predicted and observed sequence with each other.

In case of mismatch (lines 31-45) we start by increasing our mismatch counter (line 32). If we have had a previously predicted element-state, we check to see if we are repeating the same state again (line 34), in that case we keep track of how many times we have repeated it. Otherwise we are no longer repeating the same element-state, but still we have a mismatch. We will now check if we have kept repeating a element-state, until just now, that we once agreed upon, if that is the case - we have seen an forward directed expansion and we will register that (line 39) before we reset our state and repeat counter (lines 41-42). Last but not least we update our repeat element no matter if it is the same or new one and we update `hasRepeated` to notice that we have a possible repeat-element (lines 44-45).

In case of a correct match between the two sequences (lines 10-28) we first need to check if we previously had a mismatch. We first check to see if we have had a possible repetition (line 11) and if so we need to determine what category. In the case where the element-state we have repeated is the same all the way through and including the element-state we now see as a match (line 12) we have seen a bridge and will register it as such, counting `countRepeated` elements backwards. Otherwise it cannot have been a bridge, but if either the repeated element-state were linked back to some element we agreed upon (forward expansion) or if the repeated element-state now agrees with the current element (backwards expansion) we have detected a expansion and will register it as such (lines 15-18). In all cases as soon as we see a match we will register any mismatches in between as the category "other" (line 23). Duly noted that our `register`-method should only actually count the mismatch as "other" if the provided mismatch location has not already been registered in another category. We will reset our counter to zero (line 24) and also update our variables containing the possible element for repetition (line 27), note that we agreed on this element-state (line 28) and that we now have a possible new candidate for repetition (line 29).

This algorithm will complete since we are always going one element forward in each sequence. The running time will be $O(n)$ where n is the length the sequences. The implementation of the `register` will optionally backtrack through the sequences but again maximum n number of elements, and since it is called twice for each mismatch, the running time can be limited at $\Theta(3 \cdot n)$.

To complete the calculation for the categories gaps and reductions, the same algorithm is used, but inputs are reversed. This way we will be counting gaps instead of bridges and reductions instead of expansions. Total running time can be reduced easily to $\Theta(4 \cdot n)$ if we make sure only to register other mismatches once and optimize so we only have one main loop. This is the case in the implementation released together with this article.

5 Implementation

Together with this article a practical implementation of the algorithm above have been created. The full source code for the application is shown in Appendix B. The *SMCM*-implementation is in the class `StateMachine`, as shown in Appendix B.1. The implementation of *SMCM* handles unknowns by seeing the sequence as multiple sub-sequences splitted in the places where unknowns occur. This way bridges, gaps, etc.. are not counted over the unknown space.

In the application, there has also been added support reading and parsing `.fasta`-files and `.pred`-files. These file formats are straight forward and described in the next section. In the following chapter we will introduce how to use the application.

The implementation introduces a full abstracted API for accessing the difference evaluation methods, it is implemented in Java[6] and the Java-doc is available from the homepage of this

article.

The evaluation method *SMCM*, can be accessed through abstraction (`AbstractEvaluation`) or directly through the `StateMachine` class.

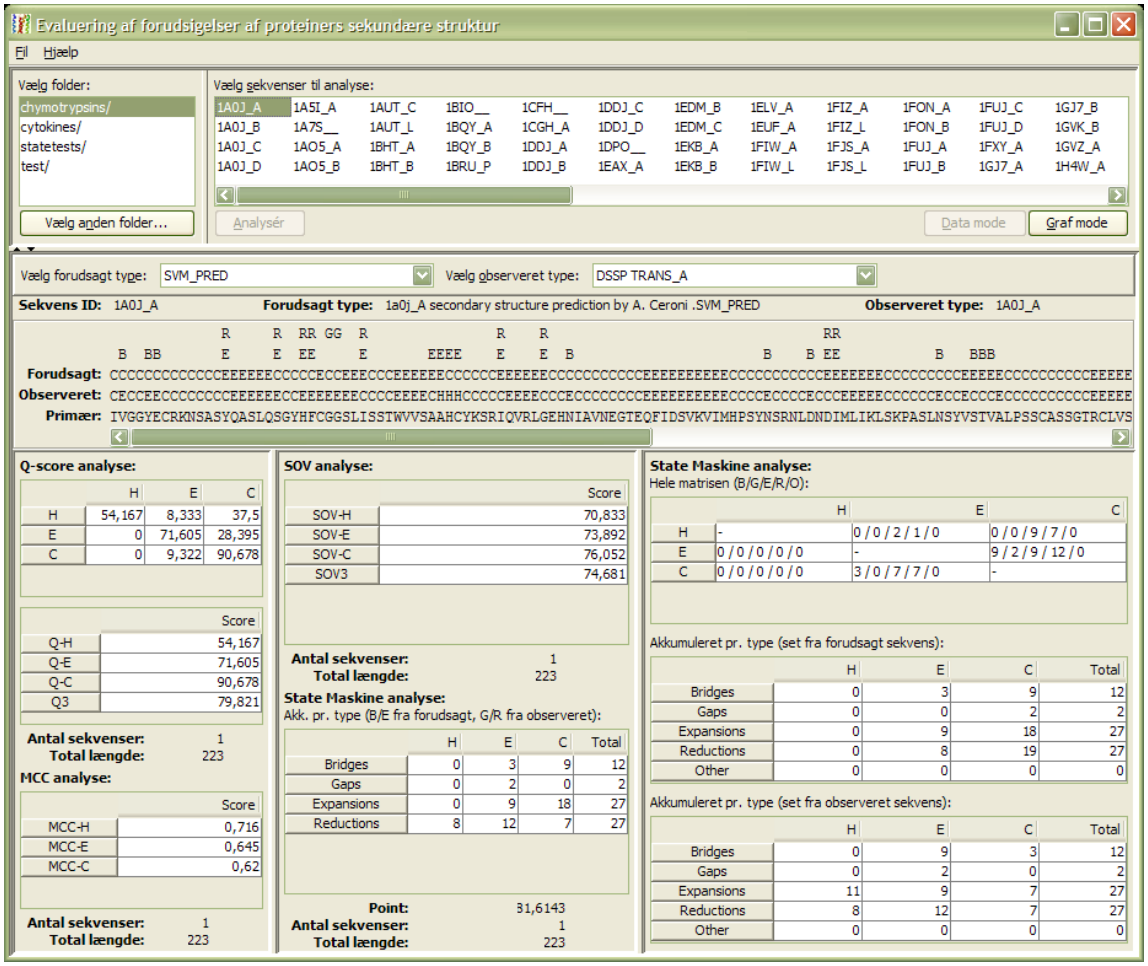

Figure 1: Screenshot of main window of application

5.1 File format

The application will read all files in a given directory containing files with the extension `.fasta` or `.pred`. It will determine the prediction/observation type by looking at the filename. It will recognize the following types if part of the filename: `DSSP`, `STRIDE`, `KAKSI`, `SVM_PRED`, `SVM_VITERBI`, `BRNN_PRED`, `BRNN_VITERBI`, `HMM21_MULTISEQ`, `HMM21_UNISEQ`, `HMM75_MULTISEQ`, `HMM75_UNISEQ` and `TEST_OBSERVATION` and `TEST_PREDICTION` (defined in `SequenceType`, see B.12) defaulting to seeing it as a primary sequence.

The file format is as follows.

A line beginning with ">" is seen as a title for the next sequence. The first five characters are translated into an identifier, rest of the line is left as a description.

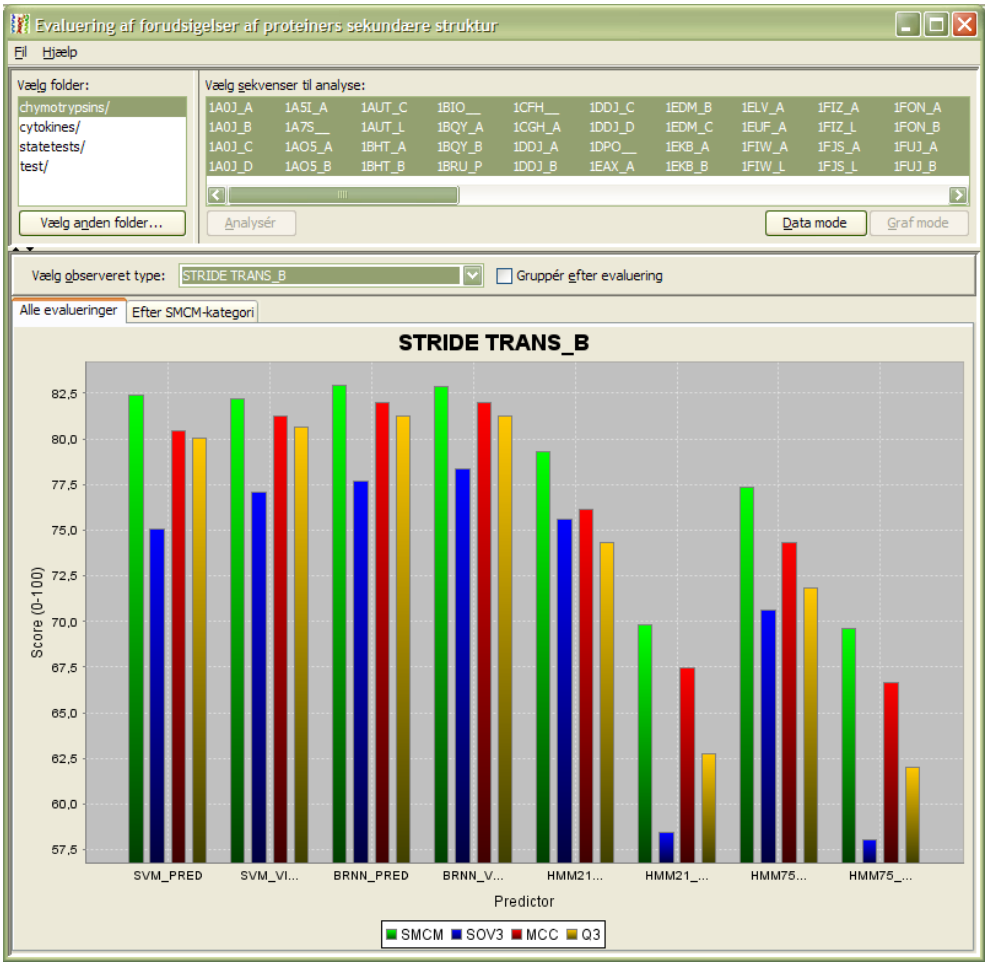

Figure 2: Graph mode

The following lines are joined to make up the sequence string.
For `DSSP` and `STRIDE`, translations are applied.
This behaviour is repeated, for each protein sequence, until the end of file is reached.

5.2 Third party dependencies

The application bundles the following third party libraries, which are required:

commons-lang[4] Apache's Jakarta's common module. This is used for easing the string manipulation done doing sequences translations.

swixml/jdom[5] This has been used to make up the Swing GUI used in the GUI application.

jcommon/jfreechart[3] This provides the nice graphs shown in the GUI application.

6 Introduction of the application

The application provides a full GUI for evaluating proteins and comparing them with four evaluators, presenting results as graphs and or raw data. It can export data as LaTeX, as also used for including the results for the chymotrypsin and cytokines families in Appendix A of this article. The application is written in Java and is run as a Java Web Start[7] application. It is supported and tested on Windows XP, Mac OSX, Linux Xandros and Linux Ubuntu. It requires no more than Java 1.5 installed on the machine in advance.

In the main screen you can select which folder to read proteins from, select which sequences you which to include and which combination of a predictor and observation you which to investigate.

Above the data, both sequences are shown and above these sequence, the *SMCM*-analysis results are shown so you can see where the algorithm have found gaps, expansions, bridges, reductions and other. These data are only shown if you select a single sequence, see Figure 1.

A selection of sequences can also be compared as a graph, this is shown in the graph mode, see Figure 2. Graphs can be exported as images by right clicked them.

6.1 Implementation of SOV_3, MCC and $Q - Score$

Besides implementation of the new *SMCM* algorithm, three other evaluation methods have also been implemented:

- SOV_3 - Segment Overlap Measure[2], this is a complex method which takes segment overlaps into account. The implementation is a simplified port to Java of the original program, written in C, written by Adam Zemla.
- MCC - Matthews Correlation Coefficient. This algorithm counts the number of True Positives (TP), True Negatives (TN), False Postives (FP) and False Negatives (FN), then calculates:

$$MCC_i = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FN)\,(TN + FP)\,(TP + FP)\,(TN + FN)}}$$

  This can be done for each element-state (i) or for all states.
- Q_3 The most classic method, where we simply count how many elements are observed correctly as a percentage of how many elements totally in the sequence. Again this can be seen from perspective of each possible element-state, Q_3 denotes for all three element-states.

For more information of these evaluators, we will refer you to [1].

7 Analysis of results

In this section we will compare our new evaluation results to other evaluations of the same predictors. For the analysis we have looked into two families of proteins, the chymotrypsin family and the cytokines family. In both case we have looked at the following predictors:

- HMM (Hidden Markov Models)
  - Using the 21-state model, uni-sequence
  - Using the 21-state model using multisequence information.
  - Using the 75-state model, uni-sequence
  - Using the 75-state model using multisequence information.
- SVM (Support Vector Machine)
  - By itself
  - Using a Viterbi decoder postprocessing
- BRNN (Bidirectional Recurrent Neural Network)
  - By itself
  - Using a Viterbi decoder post processing

For more information about these predictors we will, once again, refer you to [1].

For each of these predictors we will be comparing to observations of the same protein sequences obtained with each of the following methods:

- DSSP (Dictionary of Secondary Structure of Proteins)
  Since the output of DSSP can be interpreted differently we will be looking at two translation of DSSP:
  - Translation A, here we map:
    * `G` (3_{10}-helix), `I` (pi-helix), `H` (α-helix) to α-helix (H)
    * `E` (extended beta sheet) and `B` (beta bridge) to β-strand (E)
    * `T` (helix-turn), `S` (bend), "`.`" (other) and "`-`" (loop) to coil (C).
  - Translation B, here we only map `H` to α-helix (H) and `E` to β-strand (E), while everything else is mapped to coil (C).
- STRIDE (the secondary STRuctural IDEtification method)
  - Translation A, as defined above.
  - Translation B, as defined above.
- KAKSI (KAKSI means "two" in Finnish) and is based on $C\alpha$ distances and $\binom{\Phi}{\Psi}$angles.

Again we refer to [1] for more information about these methods.

7.1 Comparing Predictors and Observations

The author of this article is not a bioinformatician so it is left to the reader to decide which of these results are more or less useful. For full detail including all results of the new evaluator (referred to as $SMCM$) including similar results using the existing evaluators Q_3, SOV_3 and MCC is to be found in Appendix A.

In the following sections we will note some possible hypothesis while studying the new evaluation upon the various predictors.

To do preliminary comparison an optimistic score for $SMCM$ of all our five categories have been defined. This is defined as:

$$1 - \frac{bridges + gaps + \frac{1}{2}(expansions + reductions) + others}{length}$$

The reason for initially weighting both expansions and reduction as $\frac{1}{2}$ is that they typically overlap. Argumentatively you could weight the "other" category higher, since it occurs seldom and usually means that something is very wrong with the predictor.

7.2 The Chymotrypsin family

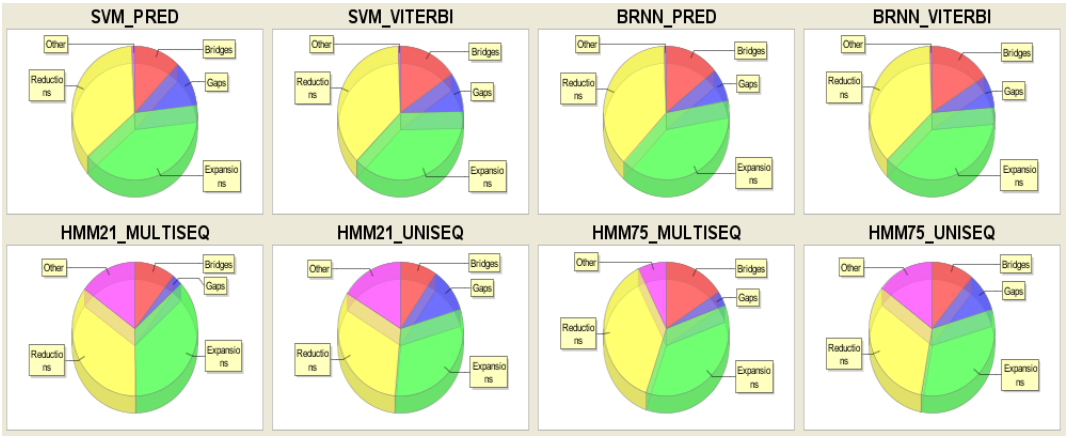

Figure 3: Chymotrypsins comparison of $SMCM$ categories against KAKSI observation sequence

In Figure 3, the mismatches of the predictors can be seen by our new $SMCM$-categories. The figure shows only the results against the KAKSI observation. We will refer to the application for other scenarios. In the following subsections we will look into noting some observations for the predictors, when predicting the chymotrypsin family.

All the data used in these observations can be found in the tables in Appendix A.1.

7.2.1 The DSSP (translation B) compared to predictors

We start by noticing that if comparing the SVM predictor to SVM using the Viterbi decoder, we get more 27% more bridges than normally. And only a small decrease in other categories. The SOV_3 computes a better score for the Viterbi decoder even the increase in bridges, while the Q_3 score is noting a small decrease and our "score" is nearly unchanged. We see the same type of behavior with the BRNN predictor compared to BRNN with Viterbi - an increase in bridges,

and only a slight reduction of gaps, expansions and reductions while our "other" category remains unchanged.

Looking at the HMM family of predictors we notices that if we use multisequences we get a drastically improvement in the categories gaps and other. Comparing HMM75 unisequence, which have 1,168 elements in the "other" category, we only have 321 in this category when using multisequences. We also see a slight decrease in bridges for HMM21 when using multisequences, but this is the opposite case for HMM75, where we see an increase of bridges.

We notice that the HMM family have much more elements in the the "other" category than both SVM and BRNN predictors. In the best case of HMM75 multisequence, 321 elements visa BRNN's only 13 elements.

7.2.2 The STRIDE (translation B) compared to predictors

Looking again at the SVM family we notice again that the Viterbi decoder increases our number of bridges, indicating that when we decode we might just be building bridges in the sequences. We only see a slight decrease of gaps and expansions and more surprisingly we also see a increase of reductions and a small increase in the category "other".

The same situation is seen for BRNN predictor, we see an increase (19%) of bridges when using the Viterbi decoder, while we again only see a small decrease of gaps, expansions and reductions.

In the HMM family of predictors we see much the same picture as when we compared with DSSP. We get a reduction in all categories if using multiple sequence, drastically in all categories, bridges (28%), gaps (82%), expansions (11%), reductions (15), other (36%) using the HMM21 predictor. But again, for its big-brother, we strangely see a increase of bridges (26%) if using multiple sequence.

7.2.3 The KAKSI observation compared to predictors

We see the same pattern with SVM. A slight increase of all categories, but increase of bridges (27%) if using the Viterbi decoder. This also is the case for the BRNN prediction. Surprisingly for BRNN we also see a small increase in the category "other" (29 to 31) if using the Viterbi decoder.

The HMM family still have much more elements in the "other" category and has 546 in the best case of HMM75 multisequence, vs. BRNN (no Viterbi) only has 29. Again we see the pattern that if we use multisequences we get a decrease of elements in every category. In the case of HMM75 multisequence we get a small increase of bridges (15%) as well.

7.3 The Cytokines family

In Figure 4, the mismatches of the predictors can be seen by our new $SMCM$-categories. The figure shows only the results against the KAKSI observation. We will refer to the application for other scenarios. In the following subsections we will look into noting some observations for the predictors, when predicting the cytokines family.

All the data used in these observations can be found in the tables in Appendix A.2.

7.3.1 The DSSP (translation B) compared to predictors

First looking at the SVM predictors we get some surprising results in that if using the Viterbi decoder, we get an increase in all categories; bridges (51%), expansions (8%), reductions (22%), other (30%) - except gaps where we see an decrease of 28%. In our SMCM-score this gives status quo for the Viterbi decoder, but in SOV_3 we notice a increase of score if using the Viterbi, while Q_3 remains nearly the same.

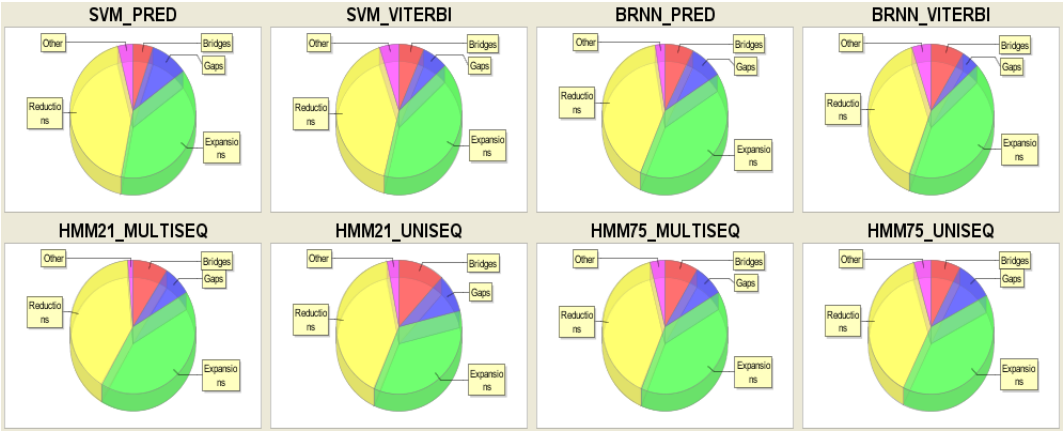

Figure 4: Cytokines comparison of $SMCM$ categories against KAKSI observation sequence

We see the same behavior with the BRNN predictor. If using Viterbi, mismatches in all categories except gaps increases. Worst off is the category "other", which increase from 29 to 54 elements (86%).

Looking at the HMM family we see a decrease of bridges and gaps if using multisequences, but now we have an increase of expansions and reductions. As with the chymotrypsin family we see a drastically decrease in the category "other".

The scores between Q_3, SOV_3 and $SMCM$ agrees and increases by a comparable factor if using multisequences.

7.3.2 The STRIDE (translation B) compared to predictors

Again if we look at the SVM predictor, we see a drastic increase of the SOV_3 score (from 69.7 to 79.1) if using the Viterbi decoder, while Q_3 only indicates a very small increase, MCC actually indicates a decrease and our $SMCM$-score indicates only a small increase of performance. We again see that we get more mismatches in the categories of bridges, expansions, reductions and "other" and only a small decrease in the gaps-category.

For the BRNN predictor, all scores except SOV_3 indicates a decrease of performance when using Viterbi. SOV_3 indicates a large performance increment (from 76.4 to 81.1). We see again more bridges, less gaps, but more expansions, reductions and the number of "other" elements doubles if using Viterbi.

Looking at HMM21 multisequence we see the first predictor which have zero mismatches in the "other" category and all factors decrease compared to the unisequence variant. The HMM79 version, on the other hand, increase in both expansion and reduction if using multisequence, but not in the same factor as it decreases in bridges, gaps and other.

7.3.3 The KAKSI observation compared to predictors

Looking at SVM, we see that, when comparing to KAKSI, we still have the increase of bridges (26%), expansions (7%) and other (19%) as we did when comparing to DSSP and STRIDE. But this time we see a decrease of reductions (4%).

The BRNN predictor shows the same pattern, having increased number of elements in all categories except gaps and reductions, if using Viterbi.

Studying the HMM predictors we see an increase of expansions (8%) if using multisequence, all other categories is decreased. The HMM21 multisequence predictor still performs best compared to HMM75 multisequence if looking at predictors with fewest mismatches in the category "other" (17 visa 51), while the other categories are close to the same. If comparing the Q_3 scores of HMM75 and HMM21 we see a small increase in favor of HMM75 while `MCC` scores increase slightly for α-helix (H) and coil (C), while β-strand (E) decreases. The SOV_3 score increase from (68.8 to 72.1) in favor of HMM75 multisequence.

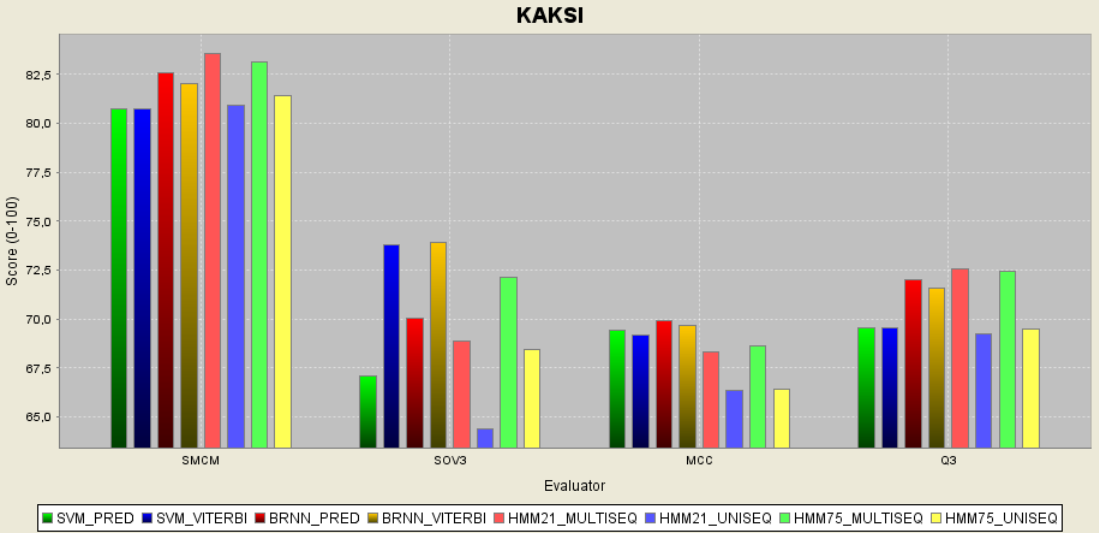

Figure 5: Predictors compared to KAKSI by total score, by the different evaluators on the cytokines protein family. They are not agreeing upon which predictor is best. The score for MCC is a sum of the individual scores. All scores have been transformed to percentages.

8 Conclusion

The algorithm gives some new details of each prediction, so we now know a little more about each predictor. The implementation is open so it can be used in other applications and/or used for further investigation. The application can read other proteins families/be used with other observation and predictors with very little, to nothing, needed to be changed in the code. The data output of this article might be used for further investigation.

References

[1] "Algorithms for Protein Secondary Structure Prediction", a comparison of prediction algorithms for two families of globular proteins, Bachelor's Project by Fiona Nielsen, May 31, 2005.

[2] "Secondary structure prediction accuracy evaluation", "SOV (Segment OVerlap) measure", Adam Zemla (11/16/1996), http://as2ts.llnl.gov/AS2TS/people/Zemla/

[3] "JFreeChart", v1.0.10, July 9th 2008, http://www.jfree.org/jfreechart/

[4] "commons lang", Apache Commons, v2.4, July 2008, http://commons.apache.org/lang/

[5] "SwiXML", v1.5 (build 149), Worf Paulus, http://swixml.org/

[6] "JDK 6 Documentation", http://java.sun.com/javase/6/docs/

[7] "Java Web Start Technology", http://java.sun.com/javase/technologies/desktop/javawebstart/

A Results

This appendix contains results calculated using our new evaluation method. All combinations of predictors and observed proteins are listed for each protein family. In the table, the confusion matrix for the $SMCM$ method are accumulated in the following manner:

- The categories "bridges" and "expansions" are accumulated by the element state the predictor had chosen in place of the correct element.
- The categories "gaps" and "reductions" are accumulated by the element state the observed protein had in the place where the predictor was wrong.
- The "other" category is only shown in total.

In the caption of the table, the $SMCM$ score is also shown.

For a complete interactive listing of all results and possible analyzing other proteins and proteins families, we refer to the application released together with this article.

A.1 Chymotrypsins

The following pages lists results for the chymotrypsin protein family.

DSSP TRANS_A compared to SVM_PRED

	H	E	C	Total/Score
Bridges	14	102	1,160	1,276
Gaps	4	162	133	299
Expansions	75	1,025	1,542	2,642
Reductions	347	1,033	868	2,248
Other	-	-	-	57
MCC	0.556	0.620	0.578	-
SOV	53.849	72.123	72.320	70.542
Q Score	49.908	72.061	88.610	77.708

Table 6: $Q_3 = 77.708$, $SOV_3 = 70.542$, $SMCM = 80.585\%$.

DSSP TRANS_A compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	14	104	1,334	1,452
Gaps	0	134	120	254
Expansions	67	1,024	1,462	2,553
Reductions	282	1,034	841	2,157
Other	-	-	-	48
MCC	0.545	0.622	0.578	-
SOV	53.463	73.134	72.114	70.888
Q Score	45.899	72.074	89.206	77.618

Table 7: $Q_3 = 77.618$, $SOV_3 = 70.888$, $SMCM = 80.432\%$.

DSSP TRANS_A compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	12	120	1,215	1,347
Gaps	0	70	117	187
Expansions	75	1,257	1,179	2,511
Reductions	307	809	868	1,984
Other	-	-	-	39
MCC	0.545	0.634	0.605	-
SOV	51.714	75.629	72.901	72.080
Q Score	45.023	76.745	88.067	78.547

Table 8: $Q_3 = 78.547$, $SOV_3 = 72.080$, $SMCM = 81.806\%$.

DSSP TRANS_A compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	12	123	1,348	1,483
Gaps	0	69	110	179
Expansions	77	1,278	1,070	2,425
Reductions	257	797	857	1,911
Other	-	-	-	39
MCC	0.527	0.634	0.603	-
SOV	50.338	75.752	72.570	71.805
Q Score	42.535	76.785	88.130	78.337

Table 9: $Q_3 = 78.337$, $SOV_3 = 71.805$, $SMCM = 81.575\%$.

DSSP TRANS_A compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	15	8	1,120	1,143
Gaps	4	36	61	101
Expansions	180	247	2,148	2,575
Reductions	247	1,675	557	2,479
Other	-	-	-	1,093
MCC	0.378	0.547	0.579	-
SOV	60.480	56.834	71.391	65.714
Q Score	59.493	47.958	92.045	72.132

Table 10: $Q_3 = 72.132$, $SOV_3 = 65.714$, $SMCM = 76.837\%$.

DSSP TRANS_A compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	35	45	1,114	1,194
Gaps	5	86	390	481
Expansions	786	737	1,828	3,351
Reductions	232	1,361	1,496	3,089
Other	-	-	-	1,590
MCC	0.231	0.386	0.430	-
SOV	48.040	47.384	62.488	56.386
Q Score	53.502	43.370	77.238	61.989

Table 11: $Q_3 = 61.989$, $SOV_3 = 56.386$, $SMCM = 69.118\%$.

DSSP TRANS_A compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	15	11	1,806	1,832
Gaps	1	19	82	102
Expansions	422	318	1,841	2,581
Reductions	238	1,972	599	2,809
Other	-	-	-	434
MCC	0.474	0.501	0.533	-
SOV	66.884	53.044	70.020	64.025
Q Score	68.203	44.094	91.271	71.270

Table 12: $Q_3 = 71.270$, $SOV_3 = 64.025$, $SMCM = 75.889\%$.

DSSP TRANS_A compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	21	42	1,512	1,575
Gaps	1	38	307	346
Expansions	854	817	1,862	3,533
Reductions	263	1,265	1,575	3,103
Other	-	-	-	1,204
MCC	0.288	0.365	0.395	-
SOV	51.766	47.090	61.811	55.937
Q Score	56.820	41.527	77.309	61.727

Table 13: $Q_3 = 61.727$, $SOV_3 = 55.937$, $SMCM = 69.318\%$.

DSSP TRANS_B compared to SVM_PRED

	H	E	C	Total/Score
Bridges	1	30	530	561
Gaps	4	160	499	663
Expansions	269	891	1,318	2,478
Reductions	263	1,010	909	2,182
Other	-	-	-	15
MCC	0.555	0.654	0.604	-
SOV	72.754	77.634	74.961	74.912
Q Score	61.169	77.302	86.136	80.518

Table 14: $Q_3 = 80.518$, $SOV_3 = 74.912$, $SMCM = 83.004\%$.

DSSP TRANS_B compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	1	30	684	715
Gaps	0	125	451	576
Expansions	202	888	1,206	2,296
Reductions	170	1,023	1,094	2,287
Other	-	-	-	19
MCC	0.584	0.657	0.616	-
SOV	73.303	79.055	77.302	77.025
Q Score	59.777	77.362	87.229	81.099

Table 15: $Q_3 = 81.099$, $SOV_3 = 77.025$, $SMCM = 82.849\%$.

DSSP TRANS_B compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	1	37	603	641
Gaps	0	70	417	487
Expansions	197	1,212	1,001	2,410
Reductions	184	801	1,276	2,261
Other	-	-	-	13
MCC	0.596	0.668	0.630	-
SOV	70.677	81.358	77.012	77.347
Q Score	59.638	82.228	85.414	81.547

Table 16: $Q_3 = 81.547$, $SOV_3 = 77.347$, $SMCM = 83.444\%$.

DSSP TRANS_B compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	1	37	723	761
Gaps	0	69	406	475
Expansions	174	1,228	903	2,305
Reductions	114	793	1,317	2,224
Other	-	-	-	13
MCC	0.591	0.668	0.631	-
SOV	69.624	81.534	78.163	78.118
Q Score	57.550	82.288	85.589	81.528

Table 17: $Q_3 = 81.528$, $SOV_3 = 78.118$, $SMCM = 83.268\%$.

DSSP TRANS_B compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	1	4	507	512
Gaps	4	36	160	200
Expansions	346	247	1,815	2,408
Reductions	170	1,673	804	2,647
Other	-	-	-	934
MCC	0.398	0.569	0.601	-
SOV	79.775	61.023	83.511	76.016
Q Score	73.417	51.385	89.497	75.132

Table 18: $Q_3 = 75.132$, $SOV_3 = 76.016$, $SMCM = 80.125\%$.

DSSP TRANS_B compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	1	16	705	722
Gaps	1	86	968	1,055
Expansions	617	603	1,595	2,815
Reductions	163	1,335	1,786	3,284
Other	-	-	-	1,448
MCC	0.227	0.394	0.423	-
SOV	62.174	50.338	61.788	58.278
Q Score	60.056	45.471	74.205	63.151

Table 19: $Q_3 = 63.151$, $SOV_3 = 58.278$, $SMCM = 70.120\%$.

DSSP TRANS_B compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	1	3	1,242	1,246
Gaps	1	19	336	356
Expansions	525	209	1,582	2,316
Reductions	150	1,954	821	2,925
Other	-	-	-	321
MCC	0.445	0.512	0.526	-
SOV	80.212	57.150	77.101	70.746
Q Score	78.288	46.654	87.927	73.018

Table 20: $Q_3 = 73.018$, $SOV_3 = 70.746$, $SMCM = 78.363\%$.

DSSP TRANS_B compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	0	5	1,057	1,062
Gaps	0	35	981	1,016
Expansions	603	623	1,646	2,872
Reductions	189	1,255	1,728	3,172
Other	-	-	-	1,168
MCC	0.254	0.369	0.387	-
SOV	63.891	50.291	61.459	57.871
Q Score	62.004	43.255	74.594	62.813

Table 21: $Q_3 = 62.813$, $SOV_3 = 57.871$, $SMCM = 70.151\%$.

STRIDE TRANS_A compared to SVM_PRED

	H	E	C	Total/Score
Bridges	4	119	1,212	1,335
Gaps	8	170	117	295
Expansions	55	920	1,767	2,742
Reductions	365	1,229	748	2,342
Other	-	-	-	56
MCC	0.568	0.617	0.578	-
SOV	54.730	71.406	71.102	69.639
Q Score	50.135	70.044	89.507	77.066

Table 22: $Q_3 = 77.066$, $SOV_3 = 69.639$, $SMCM = 79.866\%$.

STRIDE TRANS_A compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	4	121	1,410	1,535
Gaps	0	132	110	242
Expansions	50	919	1,670	2,639
Reductions	305	1,239	714	2,258
Other	-	-	-	48
MCC	0.555	0.618	0.577	-
SOV	54.348	72.675	70.758	70.057
Q Score	45.957	70.018	90.034	76.885

Table 23: $Q_3 = 76.885$, $SOV_3 = 70.057$, $SMCM = 79.649\%$.

STRIDE TRANS_A compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	4	124	1,305	1,433
Gaps	0	69	115	184
Expansions	52	1,156	1,369	2,577
Reductions	343	1,004	747	2,094
Other	-	-	-	35
MCC	0.559	0.629	0.603	-
SOV	52.756	75.410	71.202	71.151
Q Score	45.418	74.467	88.897	77.870

Table 24: $Q_3 = 77.870$, $SOV_3 = 71.151$, $SMCM = 81.011\%$.

STRIDE TRANS_A compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	4	127	1,439	1,570
Gaps	0	68	107	175
Expansions	54	1,177	1,261	2,492
Reductions	281	990	739	2,010
Other	-	-	-	35
MCC	0.539	0.628	0.601	-
SOV	51.320	75.521	71.015	70.971
Q Score	42.857	74.506	88.943	77.637

Table 25: $Q_3 = 77.637$, $SOV_3 = 70.971$, $SMCM = 80.804\%$.

STRIDE TRANS_A compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	5	4	1,200	1,209
Gaps	4	34	56	94
Expansions	150	200	2,386	2,736
Reductions	266	1,915	491	2,672
Other	-	-	-	1,084
MCC	0.385	0.542	0.574	-
SOV	61.304	56.556	69.813	64.562
Q Score	59.748	46.384	92.724	71.080

Table 26: $Q_3 = 71.080$, $SOV_3 = 64.562$, $SMCM = 75.756\%$.

STRIDE TRANS_A compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	21	69	1,190	1,280
Gaps	5	81	352	438
Expansions	771	636	1,977	3,384
Reductions	236	1,519	1,350	3,105
Other	-	-	-	1,604
MCC	0.240	0.388	0.430	-
SOV	49.069	47.662	62.204	56.186
Q Score	54.447	42.651	78.025	61.579

Table 27: $Q_3 = 61.579$, $SOV_3 = 56.186$, $SMCM = 68.729\%$.

STRIDE TRANS_A compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	5	5	1,962	1,972
Gaps	1	19	87	107
Expansions	392	298	2,014	2,704
Reductions	241	2,173	545	2,959
Other	-	-	-	408
MCC	0.484	0.490	0.526	-
SOV	67.382	52.397	68.329	62.656
Q Score	68.509	42.378	91.781	70.056

Table 28: $Q_3 = 70.056$, $SOV_3 = 62.656$, $SMCM = 74.673\%$.

STRIDE TRANS_A compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	14	68	1,541	1,623
Gaps	1	42	243	286
Expansions	811	749	2,111	3,671
Reductions	272	1,401	1,500	3,173
Other	-	-	-	1,213
MCC	0.293	0.358	0.387	-
SOV	52.813	46.538	61.401	55.370
Q Score	56.873	40.336	77.600	60.770

Table 29: $Q_3 = 60.770$, $SOV_3 = 55.370$, $SMCM = 68.837\%$.

STRIDE TRANS_B compared to SVM_PRED

	H	E	C	Total/Score
Bridges	1	25	560	586
Gaps	8	170	460	638
Expansions	235	827	1,491	2,553
Reductions	274	1,212	868	2,354
Other	-	-	-	15
MCC	0.576	0.650	0.600	-
SOV	74.997	76.593	75.477	75.031
Q Score	62.697	75.093	86.668	80.032

Table 30: $Q_3 = 80.032$, $SOV_3 = 75.031$, $SMCM = 82.416\%$.

STRIDE TRANS_B compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	1	25	737	763
Gaps	0	131	430	561
Expansions	162	827	1,393	2,382
Reductions	173	1,225	1,013	2,411
Other	-	-	-	19
MCC	0.609	0.652	0.613	-
SOV	75.274	78.093	77.649	77.076
Q Score	61.533	75.164	87.842	80.661

Table 31: $Q_3 = 80.661$, $SOV_3 = 77.076$, $SMCM = 82.192\%$.

STRIDE TRANS_B compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	1	29	647	677
Gaps	0	69	399	468
Expansions	151	1,124	1,201	2,476
Reductions	190	989	1,191	2,370
Other	-	-	-	13
MCC	0.627	0.666	0.627	-
SOV	73.018	81.095	77.416	77.702
Q Score	61.944	80.074	85.975	81.237

Table 32: $Q_3 = 81.237$, $SOV_3 = 77.702$, $SMCM = 82.947\%$.

STRIDE TRANS_B compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	1	29	772	802
Gaps	0	68	388	456
Expansions	123	1,139	1,098	2,360
Reductions	104	978	1,229	2,311
Other	-	-	-	13
MCC	0.624	0.667	0.628	-
SOV	71.156	81.290	78.555	78.383
Q Score	60.096	80.146	86.171	81.247

Table 33: $Q_3 = 81.247$, $SOV_3 = 78.383$, $SMCM = 82.825\%$.

STRIDE TRANS_B compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	1	4	551	556
Gaps	4	34	157	195
Expansions	305	194	2,036	2,535
Reductions	177	1,902	748	2,827
Other	-	-	-	911
MCC	0.410	0.564	0.594	-
SOV	81.545	60.647	83.614	75.620
Q Score	74.538	49.786	89.930	74.318

Table 34: $Q_3 = 74.318$, $SOV_3 = 75.620$, $SMCM = 79.318\%$.

STRIDE TRANS_B compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	1	15	752	768
Gaps	3	81	960	1,044
Expansions	565	536	1,758	2,859
Reductions	154	1,509	1,679	3,342
Other	-	-	-	1,428
MCC	0.234	0.393	0.420	-
SOV	64.393	50.558	61.951	58.440
Q Score	61.533	44.562	74.641	62.741

Table 35: $Q_3 = 62.741$, $SOV_3 = 58.440$, $SMCM = 69.806\%$.

STRIDE TRANS_B compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	1	3	1,340	1,344
Gaps	1	19	313	333
Expansions	466	199	1,776	2,441
Reductions	153	2,152	814	3,119
Other	-	-	-	304
MCC	0.443	0.503	0.515	-
SOV	81.903	56.302	77.555	70.602
Q Score	77.344	44.990	88.063	71.822

Table 36: $Q_3 = 71.822$, $SOV_3 = 70.602$, $SMCM = 77.327\%$.

STRIDE TRANS_B compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	0	5	1,064	1,069
Gaps	0	42	944	986
Expansions	483	589	1,878	2,950
Reductions	166	1,389	1,678	3,233
Other	-	-	-	1,235
MCC	0.260	0.361	0.378	-
SOV	66.977	49.502	61.952	58.038
Q Score	62.902	41.964	74.706	62.008

Table 37: $Q_3 = 62.008$, $SOV_3 = 58.038$, $SMCM = 69.610\%$.

KAKSI compared to SVM_PRED

	H	E	C	Total/Score
Bridges	3	114	655	772
Gaps	8	176	514	698
Expansions	251	756	1,575	2,582
Reductions	386	1,155	729	2,270
Other	-	-	-	45
MCC	0.536	0.628	0.568	-
SOV	68.878	74.062	72.611	71.987
Q Score	55.890	74.192	85.887	78.585

Table 38: $Q_3 = 78.585$, $SOV_3 = 71.987$, $SMCM = 81.232\%$.

KAKSI compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	3	114	860	977
Gaps	0	127	479	606
Expansions	182	756	1,484	2,422
Reductions	233	1,177	953	2,363
Other	-	-	-	42
MCC	0.554	0.630	0.578	-
SOV	68.422	75.677	74.563	73.661
Q Score	53.804	74.221	86.868	78.999

Table 39: $Q_3 = 78.999$, $SOV_3 = 73.661$, $SMCM = 80.868\%$.

KAKSI compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	4	117	776	897
Gaps	0	63	446	509
Expansions	173	1,086	1,281	2,540
Reductions	273	980	1,158	2,411
Other	-	-	-	29
MCC	0.569	0.639	0.586	-
SOV	66.451	79.080	74.734	74.573
Q Score	54.049	78.745	84.775	79.309

Table 40: $Q_3 = 79.309$, $SOV_3 = 74.573$, $SMCM = 81.378\%$.

KAKSI compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	4	117	885	1,006
Gaps	0	62	438	500
Expansions	146	1,100	1,221	2,467
Reductions	167	964	1,207	2,338
Other	-	-	-	31
MCC	0.565	0.638	0.586	-
SOV	65.181	79.131	75.368	74.930
Q Score	52.270	78.788	84.948	79.285

Table 41: $Q_3 = 79.285$, $SOV_3 = 74.930$, $SMCM = 81.240\%$.

KAKSI compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	4	25	749	778
Gaps	4	15	186	205
Expansions	281	352	2,093	2,726
Reductions	235	1,712	685	2,632
Other	-	-	-	1,109
MCC	0.385	0.517	0.567	-
SOV	74.977	57.514	81.177	72.626
Q Score	67.853	47.307	89.257	72.546

Table 42: $Q_3 = 72.546$, $SOV_3 = 72.626$, $SMCM = 77.280\%$.

KAKSI compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	3	27	879	909
Gaps	3	40	979	1,022
Expansions	622	588	1,775	2,985
Reductions	200	1,341	1,557	3,098
Other	-	-	-	1,545
MCC	0.226	0.361	0.407	-
SOV	60.549	49.055	60.140	56.705
Q Score	58.221	42.884	74.518	61.812

Table 43: $Q_3 = 61.812$, $SOV_3 = 56.705$, $SMCM = 68.963\%$.

KAKSI compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	3	12	1,167	1,182
Gaps	1	19	253	273
Expansions	472	164	2,060	2,696
Reductions	223	1,907	702	2,832
Other	-	-	-	546
MCC	0.421	0.497	0.523	-
SOV	75.075	55.775	75.542	68.452
Q Score	70.798	44.952	88.828	71.746

Table 44: $Q_3 = 71.746$, $SOV_3 = 68.452$, $SMCM = 77.308\%$.

KAKSI compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	2	15	1,010	1,027
Gaps	0	30	835	865
Expansions	546	522	1,975	3,043
Reductions	216	1,232	1,586	3,034
Other	-	-	-	1,368
MCC	0.259	0.362	0.388	-
SOV	63.813	48.598	61.451	57.274
Q Score	60.429	42.266	75.465	62.327

Table 45: $Q_3 = 62.327$, $SOV_3 = 57.274$, $SMCM = 70.006\%$.

A.2 Cytokines

The following pages lists results for the cytokines protein family.

DSSP TRANS_A compared to SVM_PRED

	H	E	C	Total/Score
Bridges	29	0	75	104
Gaps	119	0	31	150
Expansions	165	30	335	530
Reductions	366	18	184	568
Other	-	-	-	38
MCC	0.590	0.092	0.580	-
SOV	70.739	72.106	69.944	69.174
Q Score	78.543	46.250	77.529	71.068

Table 46: $Q_3 = 71.068$, $SOV_3 = 69.174$, $SMCM = 82.317\%$.

DSSP TRANS_A compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	37	0	106	143
Gaps	65	0	35	100
Expansions	173	30	350	553
Reductions	384	16	170	570
Other	-	-	-	43
MCC	0.583	0.087	0.566	-
SOV	79.556	70.562	68.350	74.937
Q Score	78.274	41.250	77.915	70.921

Table 47: $Q_3 = 70.921$, $SOV_3 = 74.937$, $SMCM = 82.180\%$.

DSSP TRANS_A compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	45	0	99	144
Gaps	95	0	24	119
Expansions	199	20	258	477
Reductions	261	12	188	461
Other	-	-	-	38
MCC	0.594	0.099	0.593	-
SOV	74.217	69.845	71.237	72.169
Q Score	82.841	38.750	76.680	73.402

Table 48: $Q_3 = 73.402$, $SOV_3 = 72.169$, $SMCM = 83.810\%$.

DSSP TRANS_A compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	45	0	121	166
Gaps	47	0	18	65
Expansions	215	19	294	528
Reductions	276	12	192	480
Other	-	-	-	60
MCC	0.581	0.094	0.584	-
SOV	79.253	69.722	70.486	75.623
Q Score	81.901	37.500	76.834	72.834

Table 49: $Q_3 = 72.834$, $SOV_3 = 75.623$, $SMCM = 83.284\%$.

DSSP TRANS_A compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	117	0	43	160
Gaps	59	0	27	86
Expansions	419	9	176	604
Reductions	197	12	314	523
Other	-	-	-	6
MCC	0.508	0.087	0.503	-
SOV	77.289	65.978	55.728	69.330
Q Score	89.120	23.750	56.139	71.489

Table 50: $Q_3 = 71.489$, $SOV_3 = 69.330$, $SMCM = 82.853\%$.

DSSP TRANS_A compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	156	0	47	203
Gaps	86	0	31	117
Expansions	393	16	189	598
Reductions	278	10	328	616
Other	-	-	-	17
MCC	0.446	0.079	0.455	-
SOV	71.068	67.363	52.789	65.211
Q Score	84.956	28.750	53.436	68.230

Table 51: $Q_3 = 68.230$, $SOV_3 = 65.211$, $SMCM = 80.151\%$.

DSSP TRANS_A compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	83	0	68	151
Gaps	71	0	29	100
Expansions	363	13	199	575
Reductions	229	14	288	531
Other	-	-	-	20
MCC	0.527	0.052	0.522	-
SOV	78.485	65.015	61.994	72.516
Q Score	86.333	21.250	62.625	71.468

Table 52: $Q_3 = 71.468$, $SOV_3 = 72.516$, $SMCM = 82.675\%$.

DSSP TRANS_A compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	82	0	59	141
Gaps	100	0	39	139
Expansions	398	18	227	643
Reductions	255	6	299	560
Other	-	-	-	44
MCC	0.456	0.028	0.470	-
SOV	74.491	63.876	57.779	68.322
Q Score	82.908	17.500	59.073	68.293

Table 53: $Q_3 = 68.293$, $SOV_3 = 68.322$, $SMCM = 80.540\%$.

DSSP TRANS_B compared to SVM_PRED

	H	E	C	Total/Score
Bridges	26	0	13	39
Gaps	115	0	64	179
Expansions	184	13	278	475
Reductions	310	18	228	556
Other	-	-	-	33
MCC	0.638	0.077	0.627	-
SOV	72.637	88.355	73.128	70.561
Q Score	81.973	50.820	76.594	72.960

Table 54: $Q_3 = 72.960$, $SOV_3 = 70.561$, $SMCM = 83.884\%$.

DSSP TRANS_B compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	33	0	26	59
Gaps	65	0	63	128
Expansions	200	12	299	511
Reductions	342	16	210	568
Other	-	-	-	43
MCC	0.634	0.073	0.620	-
SOV	82.069	87.272	73.925	77.431
Q Score	81.725	45.902	77.476	73.024

Table 55: $Q_3 = 73.024$, $SOV_3 = 77.431$, $SMCM = 83.820\%$.

DSSP TRANS_B compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	47	0	35	82
Gaps	84	0	66	150
Expansions	219	9	212	440
Reductions	224	12	211	447
Other	-	-	-	29
MCC	0.642	0.087	0.644	-
SOV	78.148	86.830	72.918	74.199
Q Score	86.125	44.262	75.848	75.105

Table 56: $Q_3 = 75.105$, $SOV_3 = 74.199$, $SMCM = 85.187\%$.

DSSP TRANS_B compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	47	0	44	91
Gaps	47	0	52	99
Expansions	239	9	242	490
Reductions	241	12	218	471
Other	-	-	-	54
MCC	0.633	0.087	0.640	-
SOV	81.664	86.830	75.776	78.223
Q Score	85.273	44.262	76.323	74.748

Table 57: $Q_3 = 74.748$, $SOV_3 = 78.223$, $SMCM = 84.767\%$.

DSSP TRANS_B compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	116	0	2	118
Gaps	59	0	69	128
Expansions	468	6	138	612
Reductions	173	12	404	589
Other	-	-	-	1
MCC	0.539	0.086	0.537	-
SOV	78.654	83.417	58.758	69.888
Q Score	91.341	27.869	55.020	71.531

Table 58: $Q_3 = 71.531$, $SOV_3 = 69.888$, $SMCM = 82.180\%$.

DSSP TRANS_B compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	156	0	0	156
Gaps	86	0	71	157
Expansions	443	9	155	607
Reductions	252	10	409	671
Other	-	-	-	15
MCC	0.476	0.079	0.487	-
SOV	72.686	85.040	55.575	66.010
Q Score	87.154	34.426	52.646	68.398

Table 59: $Q_3 = 68.398$, $SOV_3 = 66.010$, $SMCM = 79.668\%$.

DSSP TRANS_B compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	61	0	5	66
Gaps	69	0	57	126
Expansions	436	5	167	608
Reductions	200	14	376	590
Other	-	-	-	17
MCC	0.563	0.053	0.561	-
SOV	81.646	82.785	66.066	74.415
Q Score	88.964	24.590	61.737	72.161

Table 60: $Q_3 = 72.161$, $SOV_3 = 74.415$, $SMCM = 83.011\%$.

DSSP TRANS_B compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	85	0	13	98
Gaps	100	0	99	199
Expansions	409	3	176	588
Reductions	222	6	335	563
Other	-	-	-	41
MCC	0.496	0.027	0.512	-
SOV	76.156	81.521	60.583	69.055
Q Score	85.522	18.033	58.684	69.092

Table 61: $Q_3 = 69.092$, $SOV_3 = 69.055$, $SMCM = 80.793\%$.

STRIDE TRANS_A compared to SVM_PRED

	H	E	C	Total/Score
Bridges	11	0	72	83
Gaps	128	4	34	166
Expansions	121	28	361	510
Reductions	421	21	135	577
Other	-	-	-	35
MCC	0.595	0.080	0.578	-
SOV	69.339	77.141	69.792	67.936
Q Score	77.627	42.529	80.989	71.236

Table 62: $Q_3 = 71.236$, $SOV_3 = 67.936$, $SMCM = 82.601\%$.

STRIDE TRANS_A compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	13	0	96	109
Gaps	65	0	35	100
Expansions	131	26	381	538
Reductions	453	24	125	602
Other	-	-	-	40
MCC	0.593	0.076	0.566	-
SOV	81.803	76.349	68.590	76.813
Q Score	77.498	37.931	81.500	71.194

Table 63: $Q_3 = 71.194$, $SOV_3 = 76.813$, $SMCM = 82.780\%$.

STRIDE TRANS_A compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	23	0	95	118
Gaps	95	0	31	126
Expansions	158	16	286	460
Reductions	324	15	131	470
Other	-	-	-	32
MCC	0.598	0.091	0.592	-
SOV	77.674	75.709	70.912	74.485
Q Score	81.927	35.632	80.307	73.738

Table 64: $Q_3 = 73.738$, $SOV_3 = 74.485$, $SMCM = 84.420\%$.

STRIDE TRANS_A compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	23	0	110	133
Gaps	44	0	24	68
Expansions	166	16	327	509
Reductions	347	15	133	495
Other	-	-	-	59
MCC	0.589	0.084	0.587	-
SOV	83.922	75.249	69.984	78.738
Q Score	81.151	33.333	80.733	73.297

Table 65: $Q_3 = 73.297$, $SOV_3 = 78.738$, $SMCM = 83.978\%$.

STRIDE TRANS_A compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	78	0	41	119
Gaps	63	0	26	89
Expansions	351	9	180	540
Reductions	226	16	259	501
Other	-	-	-	8
MCC	0.530	0.086	0.517	-
SOV	81.860	72.404	58.124	73.985
Q Score	88.943	22.989	60.017	73.066

Table 66: $Q_3 = 73.066$, $SOV_3 = 73.985$, $SMCM = 84.514\%$.

STRIDE TRANS_A compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	120	0	41	161
Gaps	100	0	35	135
Expansions	331	16	195	542
Reductions	306	13	261	580
Other	-	-	-	21
MCC	0.459	0.074	0.469	-
SOV	74.965	73.412	55.144	69.008
Q Score	84.675	26.437	56.863	69.575

Table 67: $Q_3 = 69.575$, $SOV_3 = 69.008$, $SMCM = 81.539\%$.

STRIDE TRANS_A compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	55	0	61	116
Gaps	82	0	33	115
Expansions	291	12	206	509
Reductions	251	14	231	496
Other	-	-	-	21
MCC	0.552	0.053	0.541	-
SOV	80.260	71.098	65.006	74.975
Q Score	86.162	20.690	67.008	72.939

Table 68: $Q_3 = 72.939$, $SOV_3 = 74.975$, $SMCM = 84.136\%$.

STRIDE TRANS_A compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	56	0	59	115
Gaps	106	0	37	143
Expansions	322	17	225	564
Reductions	300	9	254	563
Other	-	-	-	57
MCC	0.465	0.027	0.476	-
SOV	75.667	70.387	59.952	69.967
Q Score	82.380	17.241	62.148	69.218

Table 69: $Q_3 = 69.218$, $SOV_3 = 69.967$, $SMCM = 81.529\%$.

STRIDE TRANS_B compared to SVM_PRED

	H	E	C	Total/Score
Bridges	8	0	14	22
Gaps	123	4	69	196
Expansions	138	24	307	469
Reductions	380	21	166	567
Other	-	-	-	22
MCC	0.638	0.085	0.625	-
SOV	71.477	86.060	73.913	69.717
Q Score	80.619	49.315	79.761	73.066

Table 70: $Q_3 = 73.066$, $SOV_3 = 69.717$, $SMCM = 84.062\%$.

STRIDE TRANS_B compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	10	0	29	39
Gaps	65	0	62	127
Expansions	150	22	328	500
Reductions	416	24	161	601
Other	-	-	-	28
MCC	0.636	0.080	0.616	-
SOV	83.776	84.944	75.242	79.125
Q Score	80.449	43.836	80.508	73.087

Table 71: $Q_3 = 73.087$, $SOV_3 = 79.125$, $SMCM = 84.346\%$.

STRIDE TRANS_B compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	25	0	25	50
Gaps	89	0	66	155
Expansions	171	12	255	438
Reductions	292	15	152	459
Other	-	-	-	24
MCC	0.642	0.092	0.640	-
SOV	80.632	84.304	74.627	76.429
Q Score	84.801	41.096	79.014	75.315

Table 72: $Q_3 = 75.315$, $SOV_3 = 76.429$, $SMCM = 85.755\%$.

STRIDE TRANS_B compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	25	0	40	65
Gaps	44	0	58	102
Expansions	177	12	281	470
Reductions	314	15	154	483
Other	-	-	-	50
MCC	0.635	0.086	0.635	-
SOV	86.092	83.845	76.064	81.130
Q Score	84.053	38.356	79.462	74.937

Table 73: $Q_3 = 74.937$, $SOV_3 = 81.130$, $SMCM = 85.418\%$.

STRIDE TRANS_B compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	75	0	2	77
Gaps	63	0	77	140
Expansions	405	8	155	568
Reductions	211	16	345	572
Other	-	-	-	0
MCC	0.553	0.090	0.544	-
SOV	83.022	80.721	60.484	73.651
Q Score	90.683	26.027	57.879	72.771

Table 74: $Q_3 = 72.771$, $SOV_3 = 73.651$, $SMCM = 83.452\%$.

STRIDE TRANS_B compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	108	0	0	108
Gaps	97	0	81	178
Expansions	396	14	168	578
Reductions	288	13	342	643
Other	-	-	-	14
MCC	0.482	0.078	0.494	-
SOV	76.909	81.822	57.312	69.353
Q Score	86.365	30.137	55.190	69.407

Table 75: $Q_3 = 69.407$, $SOV_3 = 69.353$, $SMCM = 80.856\%$.

STRIDE TRANS_B compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	34	0	5	39
Gaps	80	0	67	147
Expansions	359	10	177	546
Reductions	222	14	303	539
Other	-	-	-	14
MCC	0.585	0.054	0.578	-
SOV	83.100	79.392	67.992	75.815
Q Score	88.473	23.288	65.497	73.507

Table 76: $Q_3 = 73.507$, $SOV_3 = 75.815$, $SMCM = 84.388\%$.

STRIDE TRANS_B compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	58	0	10	68
Gaps	106	0	100	206
Expansions	348	14	182	544
Reductions	271	9	278	558
Other	-	-	-	33
MCC	0.501	0.030	0.520	-
SOV	77.190	78.526	63.028	70.563
Q Score	84.631	19.178	61.538	69.954

Table 77: $Q_3 = 69.954$, $SOV_3 = 70.563$, $SMCM = 81.960\%$.

KAKSI compared to SVM_PRED

	H	E	C	Total/Score
Bridges	50	0	30	80
Gaps	98	0	52	150
Expansions	125	8	454	587
Reductions	512	15	137	664
Other	-	-	-	60
MCC	0.551	0.071	0.542	-
SOV	68.620	93.225	67.147	67.109
Q Score	76.109	55.172	76.700	69.554

Table 78: $Q_3 = 69.554$, $SOV_3 = 67.109$, $SMCM = 80.751\%$.

KAKSI compared to SVM_VITERBI

	H	E	C	Total/Score
Bridges	51	0	50	101
Gaps	53	0	49	102
Expansions	140	8	482	630
Reductions	494	12	131	637
Other	-	-	-	79
MCC	0.552	0.067	0.532	-
SOV	77.277	91.895	67.279	73.793
Q Score	76.076	48.276	77.197	69.575

Table 79: $Q_3 = 69.575$, $SOV_3 = 73.793$, $SMCM = 80.751\%$.

KAKSI compared to BRNN_PRED

	H	E	C	Total/Score
Bridges	63	0	41	104
Gaps	79	0	39	118
Expansions	165	5	396	566
Reductions	409	9	154	572
Other	-	-	-	36
MCC	0.559	0.081	0.557	-
SOV	71.842	91.205	68.241	70.021
Q Score	80.414	46.552	75.788	72.014

Table 80: $Q_3 = 72.014$, $SOV_3 = 70.021$, $SMCM = 82.611\%$.

KAKSI compared to BRNN_VITERBI

	H	E	C	Total/Score
Bridges	63	0	61	124
Gaps	35	0	32	67
Expansions	175	5	435	615
Reductions	405	9	157	571
Other	-	-	-	71
MCC	0.550	0.082	0.549	-
SOV	76.917	91.205	68.553	73.950
Q Score	79.670	46.552	75.954	71.573

Table 81: $Q_3 = 71.573$, $SOV_3 = 73.950$, $SMCM = 82.023\%$.

KAKSI compared to HMM21_MULTISEQ

	H	E	C	Total/Score
Bridges	120	0	0	120
Gaps	57	0	39	96
Expansions	337	0	222	559
Reductions	280	10	246	536
Other	-	-	-	17
MCC	0.514	0.065	0.520	-
SOV	74.255	88.049	58.998	68.846
Q Score	88.249	25.862	58.789	72.540

Table 82: $Q_3 = 72.540$, $SOV_3 = 68.846$, $SMCM = 83.589\%$.

KAKSI compared to HMM21_UNISEQ

	H	E	C	Total/Score
Bridges	162	0	12	174
Gaps	87	0	48	135
Expansions	293	1	223	517
Reductions	323	7	260	590
Other	-	-	-	45
MCC	0.453	0.062	0.466	-
SOV	68.944	88.916	54.649	64.380
Q Score	84.364	31.034	55.556	69.260

Table 83: $Q_3 = 69.260$, $SOV_3 = 64.380$, $SMCM = 80.919\%$.

KAKSI compared to HMM75_MULTISEQ

	H	E	C	Total/Score
Bridges	86	0	28	114
Gaps	59	0	45	104
Expansions	282	0	250	532
Reductions	311	10	211	532
Other	-	-	-	51
MCC	0.532	0.043	0.544	-
SOV	76.581	87.663	64.591	72.118
Q Score	85.367	25.862	65.672	72.414

Table 84: $Q_3 = 72.414$, $SOV_3 = 72.118$, $SMCM = 83.158\%$.

KAKSI compared to HMM75_UNISEQ

	H	E	C	Total/Score
Bridges	94	0	18	112
Gaps	80	1	56	137
Expansions	294	6	282	582
Reductions	337	2	219	558
Other	-	-	-	64
MCC	0.468	0.029	0.489	-
SOV	73.962	85.895	59.228	68.464
Q Score	82.324	24.138	61.940	69.470

Table 85: $Q_3 = 69.470$, $SOV_3 = 68.464$, $SMCM = 81.434\%$.

B Source Code

This appendix contains a nearly complete listing of all source-code for the application. Some elements have been left out to shorten the listings, these include:

- SwiXML markup for the GUI windows.
- Localization for Danish and English, which is bundled with the application
- Utility classes for handling ressources.
- Build-files/scripts, webstart-descriptors, etc..

B.1 dk.sdu.imada.jkkn04.**Protein.tests.StateMachine**

```
1  package dk.sdu.imada.jkkn04.Protein.tests;
2
3  import java.text.CharacterIterator;
4  import java.text.StringCharacterIterator;
5  import java.util.ArrayList;
6  import java.util.EnumMap;
7  import java.util.EnumSet;
8  import java.util.List;
9  import java.util.Map;
10 import java.util.Set;
11
12 /**
13  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
14  */
15 public class StateMachine extends AbstractEvaluation {
16
17     /**
18      * Set of different confusion parameter which we measure
19      */
20     public enum ConfusionParameter {
21         /**
22          * In case that a predictor have made a state longer than it should be
23          */
24         EXPANSION,
25         /**
26          * In case that a predictor have made a state shorter than it should be
27          */
28         REDUCTION,
29         /**
30          * In case a predictor kept state and left out a state change
31          */
32         BRIDGE,
33         /**
34          * In case a predictor changes state in the middle but returned to the
                correct
35          * state afterwards
36          */
37         GAP,
38         /**
```

```
39          * In case of any other mismatch
40          */
41         OTHER;
42     }
43
44     /**
45      * This represents the confusion matrix for one confusion parameter.
46      */
47     protected class ConfusionCount {
48         /**
49          * Number of errors seen from one state to another.
50          */
51         public int matrix[][] = new int[State.COUNT][State.COUNT];
52     }
53
54     private enum MachineState {
55         CORRECT, MISMATCH, UNKNOWNS, END
56     }
57
58     /**
59      * Inner class used to represent location in a sequence (predicted or observed)
60      */
61     protected class SequenceCounter {
62
63
64         /**
65          * Create new Sequence counter for a given sequence string
66          * @param sequence sequence
67          */
68         public SequenceCounter(String sequence) {
69             this.iterator = new StringCharacterIterator(sequence);
70         }
71
72         /**
73          * Iterator used for iterating by each character of the sequence
74          */
75         CharacterIterator iterator;
76
77         /**
78          * Is set to true if a previous valid character was repeated in an invalid
79          * position
80          */
81         boolean hasRepeated = false;
82
83         /**
84          * Character which was repeated
85          */
86         char repeated;
87
88         /**
89          * Number of times character was repeated since last valid match
90          */
91         int countRepeated = 0;
```

```
92
93          /**
94           * Repeated character is linked all the may to last valid match
95           */
96          boolean linksToCorrect = false;
97      };
98
99      private int sequenceLength;
100     private Map<ConfusionParameter, ConfusionCount> confusionMatrix = null;
101
102     /* used while processing */
103     private SequenceCounter observ;
104     private SequenceCounter predict;
105
106     /* Mismatches registered on sequence */
107     private List<Set<ConfusionParameter>> mismatchesOnSeq;
108
109     /**
110      * Performs the calculation of the State-matrix
111      *
112      * @see
             dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#performEvaluation()
113      */
114     @Override
115     public void performEvaluation() {
116         final String seqObserved = observedProtein.getCleanSequence();
117         final String seqPredict = predictedProtein.getCleanSequence();
118
119         if (seqObserved.length() != seqPredict.length())
120             // FIXME: throw something
121             return;
122         sequenceLength = seqObserved.length();
123
124         /* Initialise matrix */
125         confusionMatrix = new EnumMap<ConfusionParameter, ConfusionCount>(
126                 ConfusionParameter.class);
127         for (final ConfusionParameter p : ConfusionParameter.values()) {
128             confusionMatrix.put(p, new ConfusionCount());
129         }
130
131         observ  = new SequenceCounter(seqObserved);
132         predict = new SequenceCounter(seqPredict);
133
134         /* No mismatches found yet, set all sets to empty */
135         mismatchesOnSeq = new ArrayList<Set<ConfusionParameter>>(sequenceLength);
136         for (int i = 0; i < sequenceLength; i++) {
137             mismatchesOnSeq.add(EnumSet.noneOf(ConfusionParameter.class));
138         }
139
140         int mismatches = 0;
141
142         MachineState currentState;
143         char chrObserved = observ.iterator.current();
```

```
144         char chrPredict = predict.iterator.current();
145 
146         do {
147             if (chrObserved == 'X' || chrPredict == 'X') {
148                 // skipping unknowns / processed as end of string
149                 currentState = MachineState.UNKNOWNS;
150             } else if (chrObserved == chrPredict) {
151                 if (chrObserved == CharacterIterator.DONE
152                         || chrPredict == CharacterIterator.DONE) {
153                     currentState = MachineState.END;
154                 } else {
155                     currentState = MachineState.CORRECT;
156                 }
157             } else {
158                 currentState = MachineState.MISMATCH;
159             }
160 
161             /*
162              * Count possible gaps / reductions seeing them from the observed
                    sequence
163              */
164             processSeenFromSequence(currentState, observ, ConfusionParameter.GAP,
165                     ConfusionParameter.REDUCTION);
166             /*
167              * Count possible bridges / expansion seeing them from the predicted
                    sequence
168              */
169             processSeenFromSequence(currentState, predict, ConfusionParameter.BRIDGE,
170                     ConfusionParameter.EXPANSION);
171 
172             switch (currentState) {
173             case END:
174             case CORRECT:
175             case UNKNOWNS:
176                 if (mismatches > 0) {
177                     registerOther(mismatches);
178                     mismatches = 0;
179                 }
180                 break;
181             case MISMATCH:
182                 mismatches++;
183                 break;
184             }
185 
186             if (currentState != MachineState.END) {
187                 chrObserved = observ.iterator.next();
188                 chrPredict = predict.iterator.next();
189             }
190 
191         } while (currentState != MachineState.END);
192 
193     }
194 
```

```
195     private void processSeenFromSequence(final MachineState currentState,
196             final SequenceCounter sequence, final ConfusionParameter bridge_or_gap,
197             final ConfusionParameter expansion_or_reduction) {
198         final char chr = sequence.iterator.current();
199         switch (currentState) {
200         case END:
201         case CORRECT:
202         case UNKNOWNS:
203             if (sequence.hasRepeated && sequence.countRepeated > 0) {
204                 if (sequence.linksToCorrect && sequence.repeated == chr) {
205                     /*
206                      * BRIDGE (linked all the way in prediction and still same state)
                            or
207                      * GAP (linked all the way in observation and still same state)
208                      */
209                     registerMismatch(bridge_or_gap, sequence.countRepeated);
210                 } else if (sequence.linksToCorrect || sequence.repeated == chr) {
211                     /*
212                      * EXPANSION (if repeated previous prediction or repeated current
213                      * prediction) or
214                      * REDUCTION (if repeated previous observation or repeated
215                      * current observation)
216                      */
217                     registerMismatch(expansion_or_reduction, sequence.countRepeated);
218                 }
219                 sequence.countRepeated = 0;
220             }
221             /*
222              * MATCH: store information about last matched character and mark as
                    linked to
223              * correct char
224              */
225             if (currentState == MachineState.CORRECT) {
226                 sequence.repeated = chr;
227                 sequence.linksToCorrect = true;
228                 sequence.hasRepeated = true;
229             } else {
230                 /* else we are at end of string/segment */
231                 sequence.linksToCorrect = sequence.hasRepeated = false;
232             }
233
234             break;
235         case MISMATCH:
236             if (sequence.hasRepeated && sequence.repeated == chr) {
237                 sequence.countRepeated++;
238             } else {
239                 if (sequence.linksToCorrect) {
240                     /* EXPANSION or REDUCTION (notice countRepeated may be zero) */
241                     registerMismatch(expansion_or_reduction, sequence.countRepeated);
242                 }
243                 sequence.linksToCorrect = false;
244                 sequence.countRepeated = 1;
245             }
```

```
246             sequence.repeated = chr;
247             sequence.hasRepeated = true;
248             break;
249         }
250     }
251
252     private void registerOther(final int mismatches) {
253         /*
254          * We look if we have mismatching entries which are not registered under
                any type
255          * if so, we will count the mismatches under OTHER
256          */
257
258         /* rewind iterators */
259         observ.iterator.setIndex(observ.iterator.getIndex() - mismatches);
260         predict.iterator.setIndex(predict.iterator.getIndex() - mismatches);
261
262         /* update other counts */
263         final ConfusionCount m = confusionMatrix.get(ConfusionParameter.OTHER);
264
265         /* count as "other" mismatch while no bridge/gap/reduction/expansion
                registered */
266         for (int x = 0, index = observ.iterator.getIndex(); x < mismatches; x++,
                index++) {
267             final Set<ConfusionParameter> set = mismatchesOnSeq.get(index);
268             if (set.isEmpty()) {
269                 /* mismatch not registered, register as other */
270                 final int i = mapCharToInt(observ.iterator.current());
271                 final int j = mapCharToInt(predict.iterator.current());
272                 set.add(ConfusionParameter.OTHER);
273                 m.matrix[i][j]++;
274             }
275             observ.iterator.next();
276             predict.iterator.next();
277         }
278     }
279
280     private void registerMismatch(final ConfusionParameter type, final int count) {
281
282         /* rewind iterators */
283         observ.iterator.setIndex(observ.iterator.getIndex() - count);
284         predict.iterator.setIndex(predict.iterator.getIndex() - count);
285
286         /* update counts */
287         final ConfusionCount m = confusionMatrix.get(type);
288
289         /* count sequences in confusion matrix */
290         for (int x = 0, index = observ.iterator.getIndex(); x < count; x++,
                index++) {
291             final int i = mapCharToInt(observ.iterator.current());
292             final int j = mapCharToInt(predict.iterator.current());
293
294             mismatchesOnSeq.get(index).add(type);
```

```
295
296             m.matrix[i][j]++;
297             observ.iterator.next();
298             predict.iterator.next();
299         }
300     }
301
302     /**
303      * Returns results for the State machine analysis.
304      *
305      * @see dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#getResults()
306      */
307     @Override
308     public Map<String, Object> getResults() {
309         final Map<String, Object> results = super.getResults();
310
311         /**
312          * We calculate a total sum, this is weighted as:<br/><br/>
313          *
314          * <pre>
315          * totalSum = (Bridges * 1 + Gaps *
316          * 1 + Expansions * 0.5 + Reductions * 0.5 + Other * 1).
317          * </pre>
318          *
319          * A percentage is calculated:
320          *
321          * <pre>
322          * 1 - (totalSum / sequenceLength) * 100
323          * </pre>
324          */
325         double totalSum = 0;
326
327         for (final Map.Entry<ConfusionParameter, ConfusionCount> e : confusionMatrix
328                 .entrySet()) {
329             final ConfusionParameter param = e.getKey();
330             final ConfusionCount count = e.getValue();
331             int sum = 0;
332             for (final State iState : State.values()) {
333                 final int i = iState.getInt();
334                 int sumOnObserved = 0;
335                 int sumOnPredicted = 0;
336                 for (final State jState : State.values()) {
337                     final int j = jState.getInt();
338                     results.put("State-" + iState.getChar() + jState.getChar() + "-"
339                             + param.name(), count.matrix[i][j]);
340                     sum += count.matrix[i][j];
341                     sumOnObserved += count.matrix[i][j];
342                     sumOnPredicted += count.matrix[j][i];
343                 }
344                 results.put(
345                         "State-Sum-" + iState.getChar() + "-Observed-" + param.name(),
346                         sumOnObserved);
347                 results.put("State-Sum-" + iState.getChar() + "-Predicted-"
```

```
348                     + param.name(), sumOnPredicted);
349
350             /*
351              * Summing by expansion/bridges on the predicted state and
                   reduction/gaps
352              * on the observed state
353              */
354             switch (param) {
355             case BRIDGE:
356             case EXPANSION:
357                 results.put("State-Sum-" + iState.getChar() + "-" + param.name(),
358                         sumOnPredicted);
359                 break;
360             case GAP:
361             case REDUCTION:
362                 results.put("State-Sum-" + iState.getChar() + "-" + param.name(),
363                         sumOnObserved);
364                 break;
365             case OTHER:
366                 break;
367             }
368         }
369         results.put("State-Sum-" + param.name(), sum);
370         totalSum += sum
371                 * ((param == ConfusionParameter.EXPANSION || param ==
                       ConfusionParameter.REDUCTION) ? 0.5
372                         : 1);
373     }
374
375     results.put("State-TotalSum", totalSum);
376     results.put("State", 100 * (1 - totalSum / sequenceLength));
377     results.put("length", sequenceLength);
378     return results;
379 }
380
381 /**
382  * Sum with another State Machine result
383  *
384  * @see
        dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#sumWith(dk.sdu.imada.jkkn04.Protei
385  */
386 @Override
387 public void sumWith(final AbstractEvaluation evaluation) {
388     super.sumWith(evaluation);
389     if (evaluation != null) {
390         if (!(evaluation instanceof StateMachine)) {
391             // FIXME: throw something
392         } else {
393             final StateMachine other = (StateMachine) evaluation;
394
395             /* Sum confusion matrixes */
396             for (final Map.Entry<ConfusionParameter, ConfusionCount> e :
                    other.confusionMatrix
```

```
397                     .entrySet()) {
398                 final ConfusionParameter param = e.getKey();
399                 final ConfusionCount otherCount = e.getValue();
400                 final ConfusionCount count = confusionMatrix.get(param);
401                 for (final State iState : State.values()) {
402                     final int i = iState.getInt();
403                     for (final State jState : State.values()) {
404                         final int j = jState.getInt();
405                         count.matrix[i][j] += otherCount.matrix[i][j];
406                     }
407                 }
408             }
409             sequenceLength += other.sequenceLength;
410         }
411     }
412 }
413
414 /**
415  * Returns two strings representing where we found the gaps, extensions,
        bridges,
416  * reductions,...
417  *
418  * @return two strings
419  */
420 public String[] getMismatchesBySequence() {
421     final StringBuffer strBridgeExpansionOther = new StringBuffer();
422     final StringBuffer strGapReduction = new StringBuffer();
423
424     if (sequences == 1) {
425         // only valid if representing a single sequence
426         for (int i = 0; i < sequenceLength; i++) {
427             final Set<ConfusionParameter> atPos = mismatchesOnSeq.get(i);
428             if (atPos.contains(ConfusionParameter.BRIDGE)) {
429                 strBridgeExpansionOther.append("B");
430             } else if (atPos.contains(ConfusionParameter.EXPANSION)) {
431                 strBridgeExpansionOther.append("E");
432             } else if (atPos.contains(ConfusionParameter.OTHER)) {
433                 strBridgeExpansionOther.append("O");
434             } else {
435                 strBridgeExpansionOther.append("_");
436             }
437             if (atPos.contains(ConfusionParameter.GAP)) {
438                 strGapReduction.append("G");
439             } else if (atPos.contains(ConfusionParameter.REDUCTION)) {
440                 strGapReduction.append("R");
441             } else {
442                 strGapReduction.append("_");
443             }
444         }
445     }
446     return new String[] { strBridgeExpansionOther.toString(),
447             strGapReduction.toString() };
448 }
```

```
449
450     /**
451      * Prints nice debug of where we found the mismatches.
452      */
453     public void printDebug() {
454         for (final ConfusionParameter p : ConfusionParameter.values()) {
455             final char type = p.name().toLowerCase().charAt(0);
456             for (int i = 0; i < sequenceLength; i++) {
457                 if (mismatchesOnSeq.get(i).contains(p)) {
458                     System.out.print(type);
459                 } else {
460                     System.out.print("_");
461                 }
462             }
463             System.out.println();
464         }
465         System.out.println(observedProtein.getCleanSequence());
466         System.out.println(predictedProtein.getCleanSequence());
467     }
468
469 }
```

B.2 dk.sdu.imada.jkkn04.**Protein.tests.QScore**

```
1  package dk.sdu.imada.jkkn04.Protein.tests;
2
3  import java.text.CharacterIterator;
4  import java.text.StringCharacterIterator;
5  import java.util.Map;
6
7  /**
8   * Implemenntation of the class QScore evalution of protein sequences.
9   *
10  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
11  */
12 public class QScore extends AbstractEvaluation {
13
14     private int resultMatrix[][][] = new int[State.COUNT][State.COUNT];
15     private int opsTotal[] = new int[State.COUNT];
16     private int totalSum;
17     private int sequenceLength;
18
19     /**
20      * Perform the QScore calcuation.
21      *
22      * @see
            dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#performEvaluation()
23      */
24     @Override
25     public void performEvaluation() {
26         final String seqObserved = observedProtein.getCleanSequence();
27         final String seqPredict = predictedProtein.getCleanSequence();
28
29         if (seqObserved.length() != seqPredict.length())
```

```
30            // FIXME: throw something
31            return;
32        sequenceLength = seqObserved.length();
33
34        final CharacterIterator iObserved = new
              StringCharacterIterator(seqObserved);
35        final CharacterIterator iPredict = new StringCharacterIterator(seqPredict);
36
37        resultMatrix = new int[State.COUNT][State.COUNT];
38        opsTotal = new int[State.COUNT];
39        totalSum = 0;
40        char chrObserved = iObserved.current();
41        char chrPredict = iPredict.current();
42        while (chrObserved != CharacterIterator.DONE
43              && chrPredict != CharacterIterator.DONE) {
44           final int intObserved = mapCharToInt(chrObserved);
45           final int intPredict = mapCharToInt(chrPredict);
46
47           if (intObserved != -1) {
48              if (intPredict != -1) {
49                 resultMatrix[intObserved][intPredict]++;
50                 if (intObserved == intPredict) {
51                    totalSum++;
52                 }
53              }
54              opsTotal[intObserved]++;
55           }
56
57           chrObserved = iObserved.next();
58           chrPredict = iPredict.next();
59        }
60
61     }
62
63     /**
64      * Returns the results of the QScore-evaluation. We return both a matrix:
            Q-HH, Q-HE,
65      * Q-HC, Q-EH,.... And a accumulated: Q-H, Q-E, Q-C And a total: Q3
66      *
67      * @see dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#getResults()
68      */
69     @Override
70     public Map<String, Object> getResults() {
71        final Map<String, Object> results = super.getResults();
72
73        for (final State iState : State.values()) {
74           final int i = iState.getInt();
75           for (final State jState : State.values()) {
76              final int j = jState.getInt();
77              results.put("Q-" + iState.name() + jState.name(), (opsTotal[i] == 0 ?
                    0
78                       : ((double) resultMatrix[i][j] / opsTotal[i] * 100)));
79           }
```

```
80            /* also adding diagonal */
81            results.put("Q-" + iState.name(), (opsTotal[i] == 0 ? 0
82                  : ((double) resultMatrix[i][i] / opsTotal[i] * 100)));
83         }
84         // Add total sum to results
85         results.put("Q3", (sequenceLength == 0 ? 0
86               : ((double) totalSum / sequenceLength * 100)));
87
88         results.put("length", sequenceLength);
89
90         return results;
91      }
92
93      /**
94       * Sums a QScore result with another QScore result
95       *
96       * @see
           dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#sumWith(dk.sdu.imada.jkkn04.Protei
97       */
98      @Override
99      public void sumWith(final AbstractEvaluation evaluation) {
100        super.sumWith(evaluation);
101        if (evaluation != null) {
102           if (!(evaluation instanceof QScore)) {
103              // FIXME: throw something
104           } else {
105              final QScore other = (QScore) evaluation;
106              for (final State state : State.values()) {
107                 final int i = state.getInt();
108
109                 // Sum result matrix
110                 for (final State stateJ : State.values()) {
111                    final int j = stateJ.getInt();
112                    resultMatrix[i][j] += other.resultMatrix[i][j];
113                 }
114
115                 opsTotal[i] += other.opsTotal[i];
116              }
117              totalSum += other.totalSum;
118              sequenceLength += other.sequenceLength;
119           }
120        }
121     }
122
123     /**
124      * Clones the QScore evaluation result.
125      *
126      * @see dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#clone()
127      */
128     @Override
129     public QScore clone() {
130        final QScore qscore = (QScore) super.clone();
131        qscore.opsTotal = qscore.opsTotal.clone();
```

```
132         qscore.resultMatrix = qscore.resultMatrix.clone();
133         return qscore;
134     }
135
136 }
```

B.3 dk.sdu.imada.jkkn04.**Protein.tests.MCC**

```
1  package dk.sdu.imada.jkkn04.Protein.tests;
2
3  import java.text.CharacterIterator;
4  import java.text.StringCharacterIterator;
5  import java.util.Map;
6
7  /**
8   * Implementation of the MCC evaluation
9   *
10  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
11  */
12 public class MCC extends AbstractEvaluation {
13
14     private double[] MCC = new double[State.COUNT];
15     private int sequenceLength;
16
17     /**
18      * Performs the MCC evaluations
19      *
20      * @see
            dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#performEvaluation()
21      */
22     @Override
23     public void performEvaluation() {
24         final String seqObserved = observedProtein.getSequenceWithoutUnknowns();
25         final String seqPredict = predictedProtein.getSequenceWithoutUnknowns();
26
27         if (seqObserved.length() != seqPredict.length())
28             // FIXME: throw something
29             return;
30         sequenceLength = seqObserved.length();
31
32         final CharacterIterator iObserved = new
               StringCharacterIterator(seqObserved);
33         final CharacterIterator iPredict = new StringCharacterIterator(seqPredict);
34
35         final int tp[] = new int[State.COUNT]; // True Positive
36         final int tn[] = new int[State.COUNT]; // True Negative
37         final int fp[] = new int[State.COUNT]; // False Positive
38         final int fn[] = new int[State.COUNT]; // False Negative
39
40         char chrObserved = iObserved.current();
41         char chrPredict = iPredict.current();
42         while (chrObserved != CharacterIterator.DONE
43                 && chrPredict != CharacterIterator.DONE) {
44             for (final State state : State.values()) {
```

```
45              final int c = state.getInt();
46              final char chr = state.getChar();
47              if (chrObserved == chr) {
48                 if (chrPredict == chr) {
49                    tp[c]++;
50                 } else {
51                    fn[c]++;
52                 }
53              } else {
54                 if (chrPredict == chr) {
55                    fp[c]++;
56                 } else {
57                    tn[c]++;
58                 }
59              }
60           }
61
62           chrObserved = iObserved.next();
63           chrPredict = iPredict.next();
64        }
65
66        // Summarize per state
67        for (final State state : State.values()) {
68           final int c = state.getInt();
69
70           final double divisor = Math.sqrt((tp[c] + fn[c]) * (tn[c] + fp[c])
71                 * (tp[c] + fp[c]) * (tn[c] + fn[c]));
72           double result = 0;
73           // If denominator is zero, we will return zero:
74           if (divisor != 0) {
75              result = (tp[c] * tn[c] - fp[c] * fn[c]) / divisor;
76           }
77
78           MCC[c] = result;
79
80        }
81
82     }
83
84     /**
85      * Return results from the evaluations, MCC-H, MCC-E, MCC-C, length
86      *
87      * @see dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#getResults()
88      */
89     @Override
90     public Map<String, Object> getResults() {
91        final Map<String, Object> results = super.getResults();
92
93        for (final State state : State.values()) {
94           final int c = state.getInt();
95           results.put("MCC-" + state.name(), MCC[c] / sequences);
96        }
97
```

```
98          results.put("length", sequenceLength);
99          return results;
100     }
101
102     /**
103      * Sums with another MCC evaluation
104      *
105      * @see
             dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#sumWith(dk.sdu.imada.jkkn04.Protei
106      */
107     @Override
108     public void sumWith(final AbstractEvaluation evaluation) {
109         super.sumWith(evaluation);
110         if (evaluation != null) {
111             if (!(evaluation instanceof MCC)) {
112                 // FIXME: throw something
113             } else {
114                 final MCC other = (MCC) evaluation;
115                 for (final State state : State.values()) {
116                     final int c = state.getInt();
117                     MCC[c] += other.MCC[c];
118                 }
119                 sequenceLength += other.sequenceLength;
120             }
121         }
122     }
123
124     /**
125      * Clones the MCC evaluator
126      *
127      * @see dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#clone()
128      */
129     @Override
130     public MCC clone() {
131         final MCC mcc = (MCC) super.clone();
132         mcc.MCC = mcc.MCC.clone();
133         return mcc;
134     }
135
136 }
```

B.4 dk.sdu.imada.jkkn04.**Protein.tests.SOV**

```
1  package dk.sdu.imada.jkkn04.Protein.tests;
2
3  import java.text.CharacterIterator;
4  import java.text.StringCharacterIterator;
5  import java.util.Map;
6
7  /**
8   * SOV (Segment OVerlap) measure Secondary structure prediction accuracy
9   * evaluation.<br/><br/>
10  *
11  * This is a port of the original program posted and written by Adam
```

```
12   * Zemla (11/16/1996)<br/>
13   *
14   * It was posted on http://as2ts.llnl.gov/AS2TS/download_area/<br/><br/>
15   * And originally written in C
16   *
17   * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
18   */
19  public class SOV extends AbstractEvaluation {
20
21      private static final boolean USE_SOV_1994_JMB_METHOD = false;
22      private static final double SOV_DELTA_S = 0.5;
23      private static final double SOV_DELTA = 1.0;
24
25      private double resultsSOV[] = new double[State.COUNT]; // results by H, E, C
26      private double totalSOV = 0;
27      private int sequenceLength = 0;
28
29      /**
30       * Performs the actual SOV calculation.
31       *
32       * @see
             dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#performEvaluation()
33       */
34      @Override
35      public void performEvaluation() {
36          final String seqObserved = observedProtein.getCleanSequence();
37          final String seqPredict = predictedProtein.getCleanSequence();
38
39          // Split by segments of unknown
40          final String[] segObserved = seqObserved.split("X+");
41          final String[] segPredict = seqPredict.split("X+");
42
43          if (segObserved.length != segPredict.length)
44              // FIXME: throw something
45              return;
46
47          resultsSOV = new double[State.COUNT];
48          totalSOV = 0;
49          int totalLength = 0;
50
51          for (int i = 0; i < segObserved.length; i++) {
52              if (segObserved[i].length() != segPredict[i].length())
53                  // FIXME: throw something
54                  return;
55              final int len = segPredict[i].length();
56
57              for (final State state : State.values()) {
58                  resultsSOV[state.getInt()] += calcSOVPart(segPredict[i],
                        segObserved[i],
59                          state.getChar())
60                          * len;
61              }
62
```

```
63            totalSOV += calcSOVPart(segPredict[i], segObserved[i]) * len;
64
65            totalLength += len;
66         }
67
68         if (totalLength > 0) {
69            for (final State state : State.values()) {
70               resultsSOV[state.getInt()] /= totalLength;
71            }
72            totalSOV /= totalLength;
73         }
74
75         sequenceLength = totalLength;
76
77      }
78
79      private double calcSOVPart(final String seqPredict, final String seqObserved) {
80         return calcSOVPart(seqPredict, seqObserved, null);
81      }
82
83      private double calcSOVPart(final String seqPredict, final String seqObserved,
84            final Character stateChar) {
85         final CharacterIterator iPredict = new StringCharacterIterator(seqPredict);
86         final CharacterIterator iObserved = new
               StringCharacterIterator(seqObserved);
87
88         double out = 0;
89         double s = 0;
90
91         char chrObserved = iObserved.first();
92         int totalResidues = 0;
93
94         while (chrObserved != CharacterIterator.DONE) {
95            if (stateChar == null || chrObserved == stateChar) {
96               totalResidues++;
97            }
98            chrObserved = iObserved.next();
99         }
100
101         chrObserved = iObserved.first();
102         while (chrObserved != CharacterIterator.DONE) {
103            final int beginSegment1 = iObserved.getIndex();
104            final char s1 = chrObserved;
105            do {
106               chrObserved = iObserved.next();
107            } while (chrObserved != CharacterIterator.DONE && s1 == chrObserved);
108            final int endSegment1 = iObserved.getIndex() - 1;
109            final int lengthSegment1 = endSegment1 - beginSegment1 + 1;
110
111            int multiple = 0;
112            char chrPredict = iPredict.first();
113
114            while (chrPredict != CharacterIterator.DONE) {
```

```
115             final int beginSegment2 = iPredict.getIndex();
116             final char s2 = chrPredict;
117             do {
118                chrPredict = iPredict.next();
119             } while (chrPredict != CharacterIterator.DONE && s2 == chrPredict);
120             final int endSegment2 = iPredict.getIndex() - 1;
121             final int lengthSegment2 = endSegment2 - beginSegment2 + 1;
122
123             if (stateChar == null || s1 == stateChar) {
124                if (s1 == s2 && endSegment2 >= beginSegment1
125                      && beginSegment2 <= endSegment1) {
126                   if (multiple > 0 && !USE_SOV_1994_JMB_METHOD) {
127                      totalResidues = totalResidues + lengthSegment1;
128                   }
129                   multiple++;
130                   int j1, j2, k1, k2;
131                   // Place first segment start offset in j1
132                   if (beginSegment1 > beginSegment2) {
133                      j1 = beginSegment1;
134                      j2 = beginSegment2;
135                   } else {
136                      j1 = beginSegment2;
137                      j2 = beginSegment1;
138                   }
139                   // Place last segment end offset in k2
140                   if (endSegment1 < endSegment2) {
141                      k1 = endSegment1;
142                      k2 = endSegment2;
143                   } else {
144                      k1 = endSegment2;
145                      k2 = endSegment1;
146                   }
147                   int d;
148                   final int minov = k1 - j1 + 1;
149                   final int maxov = k2 - j2 + 1;
150                   final int d1 = (int) Math.floor(lengthSegment1 * SOV_DELTA_S);
151                   final int d2 = (int) Math.floor(lengthSegment2 * SOV_DELTA_S);
152                   if (d1 <= d2 || USE_SOV_1994_JMB_METHOD) {
153                      d = d1;
154                   } else {
155                      d = d2;
156                   }
157                   if (d > minov) {
158                      d = minov;
159                   }
160                   if (d > maxov - minov) {
161                      d = maxov - minov;
162                   }
163                   double x = SOV_DELTA * d;
164                   x = (minov + x) * lengthSegment1;
165                   if (maxov > 0) {
166                      s += x / maxov;
167                   } else {
```

```
168                         System.err
169                                 .println(String
170                                         .format(
171                                                 "\n ERROR! minov = %-4d maxov = %-4d
                                                        length = %-4d d = %-4d   %4d %4d  %4d
                                                        %4d",
172                                                 minov, maxov, lengthSegment1, d,
173                                                 beginSegment1 + 1, endSegment1 + 1,
174                                                 beginSegment2 + 1, endSegment1 + 1));
175                         // FIXME: throw something
176                     }
177                 }
178             }
179         }
180     }
181     if (totalResidues > 0) {
182         out = s / totalResidues;
183     } else {
184         out = 1.0;
185     }
186     return out * 100;
187 }
188
189 /**
190  * Returns result of the SOV calculation We return SOV-H, SOV-E, SOV-C and a
        total
191  * SOV3
192  *
193  * @see dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#getResults()
194  */
195 @Override
196 public Map<String, Object> getResults() {
197     final Map<String, Object> results = super.getResults();
198
199     for (final State state : State.values()) {
200         final int c = state.getInt();
201         results.put("SOV-" + state.name(), resultsSOV[c] / sequences);
202     }
203     results.put("SOV3", totalSOV / sequences);
204     results.put("length", sequenceLength);
205
206     return results;
207 }
208
209 /**
210  * Summerize with another SOV result.
211  *
212  * @see
        dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#sumWith(dk.sdu.imada.jkkn04.Protei
213  */
214 @Override
215 public void sumWith(final AbstractEvaluation evaluation) {
216     super.sumWith(evaluation);
```

```
217         if (evaluation != null) {
218             if (!(evaluation instanceof SOV)) {
219                 // FIXME: throw something
220             } else {
221                 final SOV other = (SOV) evaluation;
222                 for (final State state : State.values()) {
223                     final int c = state.getInt();
224                     resultsSOV[c] += other.resultsSOV[c];
225                 }
226                 totalSOV += other.totalSOV;
227                 sequenceLength += other.sequenceLength;
228             }
229         }
230     }
231 
232     /**
233      * Clone the SOV result
234      *
235      * @see dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation#clone()
236      */
237     @Override
238     public AbstractEvaluation clone() {
239         final SOV sov = (SOV) super.clone();
240         sov.resultsSOV = sov.resultsSOV.clone();
241         return sov;
242     }
243 
244 }
```

B.5 dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation

```
1  package dk.sdu.imada.jkkn04.Protein.tests;
2  
3  import java.util.Map;
4  import java.util.TreeMap;
5  
6  import dk.sdu.imada.jkkn04.Protein.data.Protein;
7  
8  /**
9   * This abstract class can be implemented to offer a new evaluation type for
        comparing a
10  * predicted secondary protein sequence with a observed sequence.
11  *
12  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
13  */
14 public abstract class AbstractEvaluation implements Cloneable {
15 
16     /**
17      * Observed protein sequence to use for evaluation
18      */
19     protected Protein observedProtein;
20     /**
21      * Predicted protein sequence to use for evaluation
22      */
```

```
23      protected Protein predictedProtein;
24
25      /**
26       * Represents the different states in a protein sequence
27       */
28      public enum State {
29          /**
30           * Helix
31           */
32          H,
33          /**
34           * Strand
35           */
36          E,
37          /**
38           * Coil
39           */
40          C;
41
42          /**
43           * Total number of different states we handle.
44           */
45          final static int COUNT = State.values().length;
46
47          /**
48           * Character representing this state
49           *
50           * @return char (H/E/C)
51           */
52          public char getChar() {
53              return name().charAt(0);
54          }
55
56          /**
57           * Number representing this state
58           *
59           * @return number between -1 and 2 (-1 in case of X)
60           */
61          public int getInt() {
62              return mapCharToInt(getChar());
63          }
64      };
65
66      /**
67       * Number of sequences considered in this evaluation
68       */
69      protected int sequences = 1;
70
71      /**
72       * Returns results for this evaluation.<br/>
73       * Should be overridden to include the actual results
74       *
75       * @return Map of key/value of results
```

```
76      */
77     public Map<String, Object> getResults() {
78        final Map<String, Object> results = new TreeMap<String, Object>();
79        results.put("sequences", sequences);
80        return results;
81     }
82
83     /**
84      * Method for translating a character (H/E/C/X) into a number
85      *
86      * @param c H/E/C/X state
87      * @return number between -1 and 2 (-1 in case of X)
88      */
89     public static int mapCharToInt(final char c) {
90        switch (c) {
91        case 'H':
92           return 0;
93        case 'E':
94           return 1;
95        case 'C':
96           return 2;
97        case 'X':
98           return -1;
99        default:
100          System.err.println("Bad character: " + c);
101       }
102       return -1;
103    }
104
105    /**
106     * Method for translating number into a character (H/E/C/X)
107     *
108     * @param i Number between -1 and 2.
109     * @return H/E/C/X.
110     */
111    public static char mapIntToChar(final int i) {
112       switch (i) {
113       case 0:
114          return 'H';
115       case 1:
116          return 'E';
117       case 2:
118          return 'C';
119       case -1:
120          return 'X';
121       default:
122          System.err.println("Bad integer: " + i);
123       }
124       return '?';
125    }
126
127    /**
128     * Configures the evaluator with the observed protein sequence to use
```

```
129      *
130      * @param observedProtein observed sequence
131      */
132     public void setObservedProtein(final Protein observedProtein) {
133        this.observedProtein = observedProtein;
134     }
135
136     /**
137      * Configures the evaluator with the prediction protein sequence to use
138      *
139      * @param predictedProtein prediction sequence
140      */
141     public void setPredictedProtein(final Protein predictedProtein) {
142        this.predictedProtein = predictedProtein;
143     }
144
145     /**
146      * Main method, must be implemented to perform the evaluation
147      */
148     abstract public void performEvaluation();
149
150     /**
151      * Creates a clone of the result, useful when merging results
152      *
153      * @see java.lang.Object#clone()
154      */
155     @Override
156     public AbstractEvaluation clone() {
157        try {
158           final AbstractEvaluation clone = (AbstractEvaluation) super.clone();
159           clone.observedProtein = clone.predictedProtein = null;
160           return clone;
161        } catch (final CloneNotSupportedException e) {
162           e.printStackTrace();
163        }
164        return null;
165     }
166
167     /**
168      * Sums results from this evaluation with results for another sequence
169      *
170      * @param evaluation Another evaluation of same type
171      */
172     public void sumWith(final AbstractEvaluation evaluation) {
173        sequences += evaluation.sequences;
174     }
175
176  }
```

B.6 dk.sdu.imada.jkkn04.Protein.util.ProcessUtil

```
1  package dk.sdu.imada.jkkn04.Protein.util;
2
3  import java.util.EnumSet;
```

```
4  import java.util.HashMap;
5  import java.util.Map;
6  import java.util.Set;
7
8  import dk.sdu.imada.jkkn04.Protein.data.Protein;
9  import dk.sdu.imada.jkkn04.Protein.data.ProteinMap;
10 import dk.sdu.imada.jkkn04.Protein.data.SequenceTranslated;
11 import dk.sdu.imada.jkkn04.Protein.data.SequenceType;
12 import dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation;
13 import dk.sdu.imada.jkkn04.Protein.tests.MCC;
14 import dk.sdu.imada.jkkn04.Protein.tests.QScore;
15 import dk.sdu.imada.jkkn04.Protein.tests.SOV;
16 import dk.sdu.imada.jkkn04.Protein.tests.StateMachine;
17
18 /**
19  * Utility class to do processing of various kinds
20  *
21  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
22  */
23 public class ProcessUtil {
24
25     /**
26      * Class for representing multiple evaluations order by evaluation type
27      *
28      * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
29      */
30     public static class MultipleEvaluationsMap extends
31             HashMap<String, AbstractEvaluation> {
32         private static final long serialVersionUID = 1L;
33     };
34
35     /**
36      * Class for representing multiple evaluations ordered by observed and
          predicted
37      * sequence pair.
38      *
39      * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
40      */
41     public static class ObservationPredictionMap extends
42             HashMap<SequenceTranslated, Map<SequenceTranslated,
                    MultipleEvaluationsMap>> {
43         private static final long serialVersionUID = 1L;
44     };
45
46     /**
47      * Set of all known evaluators in this implementation
48      */
49     public static final AbstractEvaluation[] ALL_EVALUATIONS = { new QScore(), new
          SOV(),
50             new MCC(), new StateMachine() };
51
52     /**
53      * Method for performing all tests of a certain type
```

```
54      *
55      * @param pm Map of one protein, multiple sequence types
56      * @param observationTypes Observed translated types to use
57      * @param predictionTypes Predicted translated types to use
58      * @param evaluationClasses Evaluator classes to use (can be ALL_EVALUATIONS)
59      * @return map of all results for the given protein
60      */
61     public static ObservationPredictionMap performAllTests(final ProteinMap pm,
62           final Set<SequenceTranslated> observationTypes,
63           final Set<SequenceTranslated> predictionTypes,
64           final AbstractEvaluation[] evaluationClasses) {
65
66        final ObservationPredictionMap apm = new ObservationPredictionMap();
67
68        /*
69         * go through every combination of observations available to predictions
               available
70         */
71        for (final SequenceTranslated observationType : observationTypes) {
72           if (pm.containsKey(observationType)) {
73              final Protein observed = pm.get(observationType);
74              final Map<SequenceTranslated, MultipleEvaluationsMap> observedMap =
                    new HashMap<SequenceTranslated, MultipleEvaluationsMap>();
75
76              for (final SequenceTranslated predictionType : predictionTypes) {
77                 if (pm.containsKey(predictionType)) {
78                    final MultipleEvaluationsMap multiEvaluationsMap = new
                          MultipleEvaluationsMap();
79                    final Protein predicition = pm.get(predictionType);
80                    for (final AbstractEvaluation eval : evaluationClasses) {
81                       try {
82                          final AbstractEvaluation current = eval.clone();
83                          current.setObservedProtein(observed);
84                          current.setPredictedProtein(predicition);
85                          current.performEvaluation();
86
87                          multiEvaluationsMap.put(current.getClass().getName(),
88                                current);
89
90                       } catch (final IllegalArgumentException e) {
91                          e.printStackTrace();
92                       } catch (final SecurityException e) {
93                          e.printStackTrace();
94                       }
95                    }
96                    observedMap.put(predictionType, multiEvaluationsMap);
97                 }
98              }
99
100             if (!observedMap.isEmpty()) {
101                apm.put(observationType, observedMap);
102             }
103          }
```

```
104         }
105
106         return apm;
107     }
108
109     /**
110      * This summarises results for a protein into total sum.
111      *
112      * @param sum Map-object to use for summarisation (can be an empty map)
113      * @param map Object representing a result, which should be summed into
             <code>sum</code>.
114      */
115     public static void summerizeAll(final ObservationPredictionMap sum,
116             final ObservationPredictionMap map) {
117
118         for (final Map.Entry<SequenceTranslated, Map<SequenceTranslated,
                 MultipleEvaluationsMap>> observed : map
119                 .entrySet()) {
120             final SequenceTranslated observedType = observed.getKey();
121             Map<SequenceTranslated, MultipleEvaluationsMap> sumObserved = sum
122                     .get(observedType);
123             if (sumObserved == null) {
124                 sum
125                         .put(
126                                 observedType,
127                                 sumObserved = new HashMap<SequenceTranslated,
                                         MultipleEvaluationsMap>());
128             }
129             for (final Map.Entry<SequenceTranslated, MultipleEvaluationsMap>
                     prediction : observed
130                     .getValue().entrySet()) {
131                 final SequenceTranslated predictionType = prediction.getKey();
132                 MultipleEvaluationsMap sumPrediction =
                         sumObserved.get(predictionType);
133                 if (sumPrediction == null) {
134                     sumObserved.put(predictionType,
135                             sumPrediction = new MultipleEvaluationsMap());
136                 }
137                 for (final Map.Entry<String, AbstractEvaluation> evaluation :
                         prediction
138                         .getValue().entrySet()) {
139                     final AbstractEvaluation sumOfEvaluation = sumPrediction
140                             .get(evaluation.getKey());
141                     if (sumOfEvaluation == null) {
142                         sumPrediction.put(evaluation.getKey(), evaluation.getValue()
143                                 .clone());
144                     } else {
145                         sumOfEvaluation.sumWith(evaluation.getValue());
146                     }
147                 }
148             }
149         }
150
```

```
151     }
152
153     /**
154      * Shortcut to perform all evaluation by translated types
155      *
156      * @param pm Map of multiple sequences types of a protein
157      * @param observationTypes Observed translated types to use
158      * @param predictionTypes Prediction translated types to use
159      * @return result map ordered by translated types
160      */
161     public static ObservationPredictionMap performAllTests(final ProteinMap pm,
162             final Set<SequenceTranslated> observationTypes,
163             final Set<SequenceTranslated> predictionTypes) {
164         return performAllTests(pm, observationTypes, predictionTypes,
                ALL_EVALUATIONS);
165     }
166
167     /**
168      * Shortcut to perform all evaluation by type
169      *
170      * @param pm Map of multiple sequences types of a protein
171      * @param observationTypes Observed types to use
172      * @param predictionTypes Prediction types to use
173      * @return result map ordered by translated types
174      */
175     public static ObservationPredictionMap performAllTestsByType(final ProteinMap
            pm,
176             final Set<SequenceType> observationTypes,
177             final Set<SequenceType> predictionTypes) {
178         return performAllTests(pm, SequenceTranslated.buildAllOf(observationTypes),
179                 SequenceTranslated.buildAllOf(predictionTypes));
180     }
181
182     /**
183      * Shortcut to perform all evaluation by type
184      *
185      * @param pm Map of multiple sequences types of a protein
186      * @param observationTypes Observed types to use
187      * @param predictionTypes Prediction types to use
188      * @param evaluationClasses Evaluator classes to use (can be ALL_EVALUATIONS)
189      * @return result map ordered by translated types
190      */
191     public static ObservationPredictionMap performAllTestsByType(final ProteinMap
            pm,
192             final EnumSet<SequenceType> observationTypes,
193             final EnumSet<SequenceType> predictionTypes,
194             final AbstractEvaluation[] evaluationClasses) {
195         return performAllTests(pm, SequenceTranslated.buildAllOf(observationTypes),
196                 SequenceTranslated.buildAllOf(predictionTypes), evaluationClasses);
197     }
198
199 }
```

B.7 dk.sdu.imada.jkkn04.Protein.util.LaTeXFormatter

```
1  /**
2   *
3   */
4  package dk.sdu.imada.jkkn04.Protein.util;
5
6  import java.io.BufferedWriter;
7  import java.io.File;
8  import java.io.FileWriter;
9  import java.io.IOException;
10 import java.io.PrintWriter;
11 import java.util.Locale;
12 import java.util.Map;
13 import java.util.TreeSet;
14
15 import dk.sdu.imada.jkkn04.Protein.data.SequenceTranslated;
16 import dk.sdu.imada.jkkn04.Protein.tests.MCC;
17 import dk.sdu.imada.jkkn04.Protein.tests.QScore;
18 import dk.sdu.imada.jkkn04.Protein.tests.SOV;
19 import dk.sdu.imada.jkkn04.Protein.tests.StateMachine;
20 import dk.sdu.imada.jkkn04.Protein.util.ProcessUtil.MultipleEvaluationsMap;
21 import dk.sdu.imada.jkkn04.Protein.util.ProcessUtil.ObservationPredictionMap;
22
23 /**
24  * Small class to output a LaTeX file of tables of interesting results
25  *
26  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
27  */
28 public class LaTeXFormatter {
29
30     static {
31         Locale.setDefault(Locale.ENGLISH);
32     }
33
34     /**
35      * Writes a set of results to a LaTeX formatted file
36      *
37      * @param file File to write into
38      * @param results Results to write
39      */
40     public static void writeFile(File file, ObservationPredictionMap results) {
41
42         File out = file;
43
44         /* Add ".tex" extension if missing */
45         if (file.getName().indexOf('.') == -1) {
46             out = new File(file.getAbsoluteFile() + ".tex");
47         }
48
49         PrintWriter pw = null;
50         try {
51             pw = new PrintWriter(new BufferedWriter(new FileWriter(out)));
```

```
52
53          pw.println("%% Created by Protein Analyser
                (http://jkkn.dk/sdu/proteins/)");
54          pw.println();
55
56          /* For each observation compare to each prediction */
57          for (final SequenceTranslated observedType: new
                TreeSet<SequenceTranslated>(results.keySet())) {
58              final Map<SequenceTranslated, MultipleEvaluationsMap> mapByObserved =
                    results.get(observedType);
59              for (final SequenceTranslated predictionType: new
                    TreeSet<SequenceTranslated>(mapByObserved.keySet())) {
60                  final MultipleEvaluationsMap resultsByAnalyse =
                        mapByObserved.get(predictionType);
61                  final Map<String, Object> sm  =
                        resultsByAnalyse.get(StateMachine.class.getName()).getResults();
62                  final Map<String, Object> mcc =
                        resultsByAnalyse.get(MCC.class.getName()).getResults();
63                  final Map<String, Object> sov =
                        resultsByAnalyse.get(SOV.class.getName()).getResults();
64                  final Map<String, Object> q   =
                        resultsByAnalyse.get(QScore.class.getName()).getResults();
65
66                  pw.format("\\subsubsection*{%s compared to %s}",
67                          observedType.toString().replace("_", "\\_"),
68                          predictionType.toString().replace("_", "\\_"));
69                  pw.println();
70                  pw.println();
71                  pw.println("\\begin{table}[H]");
72                  pw.println("\\begin{centering}");
73                  pw.println("\\begin{tabular}{r|r|r|r|r|}");
74                  pw.println("& \\textbf{H} & \\textbf{E} & \\textbf{C} &
                        \\textbf{Total/Score}\\\\");
75                  pw.println("\\hline");
76
77                  for (String category: new String[] {"Bridges", "Gaps",
                        "Expansions", "Reductions"}) {
78
79                      String cateName = category.toUpperCase().substring(0,
                            category.length()-1); // cut off 's'
80                      pw.format("\\textbf{%s} & %d & %d & %d & %d\\\\",
81                              category,
82                              sm.get("State-Sum-H-" + cateName),
83                              sm.get("State-Sum-E-" + cateName),
84                              sm.get("State-Sum-C-" + cateName),
85                              sm.get("State-Sum-" + cateName)
86                              );
87                      pw.println();
88                  }
89                  pw.format("\\textbf{Other} & - & - & - & %d\\\\",
90                          sm.get("State-Sum-OTHER"));
91                  pw.println();
92
```

```
93
94                  pw.format("\\textbf{MCC} & %,.3f & %,.3f & %,.3f & -\\\\",
95                          mcc.get("MCC-H"),
96                          mcc.get("MCC-E"),
97                          mcc.get("MCC-C")
98                          );
99                  pw.println();
100
101                 pw.format("\\textbf{SOV} & %,.3f & %,.3f & %,.3f & %,.3f\\\\",
102                         sov.get("SOV-H"),
103                         sov.get("SOV-E"),
104                         sov.get("SOV-C"),
105                         sov.get("SOV3"));
106                 pw.println();
107
108                 pw.format("\\textbf{Q Score} & %,.3f & %,.3f & %,.3f & %,.3f\\\\",
109                         q.get("Q-H"),
110                         q.get("Q-E"),
111                         q.get("Q-C"),
112                         q.get("Q3"));
113                 pw.println();
114
115                 pw.println("\\hline");
116                 pw.println("\\end{tabular}");
117
118                 pw.println("\\par\\end{centering}");
119                 pw.format("\\caption{$Q_{3}=%,.3f$, $SOV_{3}=%,.3f$, $SMCM=%,.3f\\%%$.}",
120                         q.get("Q3"),
121                         sov.get("SOV3"),
122                         sm.get("State")
123                         );
124                 pw.println();
125                 pw.println("\\end{table}");
126                 pw.println();
127                 pw.println();
128
129             }
130         }
131
132     } catch (IOException e) {
133         // FIXME: Throw something
134         e.printStackTrace();
135     } finally {
136         if (pw != null) pw.close();
137     }
138
139 }
140
141 }
```

B.8 dk.sdu.imada.jkkn04.**Protein.data.Protein**

```
1 package dk.sdu.imada.jkkn04.Protein.data;
```

```
2
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.List;
6  import java.util.regex.Matcher;
7  import java.util.regex.Pattern;
8
9  import org.apache.commons.lang.StringUtils;
10
11 /**
12  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
13  */
14 public class Protein {
15
16     private static final Pattern IDENTIFIER_PATTERN = Pattern
17             .compile("^([A-Z0-9]{4})[:_]?([A-Z0-9]?).*");
18     private final String description;
19     private final String identifier;
20     private final String rawSequence;
21     private final String sequence;
22     private String cleanSequence;
23     private final SequenceType sequenceType;
24     private final TranslationType translationType;
25
26     /**
27      * Represents type of translation
28      */
29     public enum TranslationType {
30         /**
31          * Raw sequence, as it was read
32          */
33         RAW,
34         /**
35          * Normal translation ("b." mapped to "EC")
36          */
37         NORMAL,
38         /**
39          * STRIDE/DSSP ("GHIEBTS.-" mapped to "HHHEECCCC")
40          */
41         TRANS_A,
42         /**
43          * STRIDE/DSSP ("HEGITSB.-" mapped to "HECCCCCCC")
44          */
45         TRANS_B
46     }
47
48     /**
49      * Creates new protein sequence
50      *
51      * @param description Description for protein sequence
52      * @param rawSequence Raw sequence string
53      * @param predictionType Type of prediction/observation
54      * @param translationType Translation to use
```

```
55      */
56     public Protein(final String description, final String rawSequence,
57             final SequenceType predictionType, final TranslationType
                   translationType) {
58         super();
59         this.description = description.trim();
60         identifier = cleanIdentifer();
61 
62         this.rawSequence = rawSequence;
63         sequenceType = predictionType;
64         this.translationType = translationType;
65 
66         sequence = translatedProtein();
67         cleanSequence = null;
68     }
69 
70     private String cleanIdentifer() {
71         /* Clean identifier similar to how Fiona does it */
72         final Matcher XXXX_X =
                   IDENTIFIER_PATTERN.matcher(description.toUpperCase());
73         String clean = "";
74 
75         if (XXXX_X.matches()) {
76             clean = XXXX_X.group(1) + "_" + XXXX_X.group(2);
77             if (XXXX_X.group(2).length() == 0) {
78                 clean += "_"; // add anonymous chain identifier
79             }
80         } else {
81             clean = "BAD_NAME_FORMAT: " + description;
82             // throw something
83         }
84 
85         return clean;
86     }
87 
88     /**
89      * Returns description of protein sequence
90      *
91      * @return description
92      */
93     public String getDescription() {
94         return description;
95     }
96 
97     /**
98      * Returns identifier for protein sequence
99      *
100     * @return identifier
101     */
102    public String getIdentifier() {
103        return identifier;
104    }
105
```

```
106     /**
107      * Returns translated sequence
108      *
109      * @return translated sequence
110      */
111     public String getSequence() {
112        return sequence;
113     }
114
115     private String translatedProtein() {
116        String sequence = rawSequence;
117        final String upperCase = rawSequence.toUpperCase();
118
119        switch (translationType) {
120        case NORMAL:
121           sequence = StringUtils.replaceChars(rawSequence, "b.", "EC");
122           break;
123        case TRANS_A:
124           sequence = StringUtils.replaceChars(upperCase, "GHIEBTS.-", "HHHEECCCC");
125           break;
126        case TRANS_B:
127           sequence = StringUtils.replaceChars(upperCase, "HEGITSB.-", "HECCCCCCC");
128           break;
129        case RAW:
130        default:
131           break;
132        }
133
134        return sequence;
135     }
136
137     /**
138      * Creates a list of possible translation of the given protein
139      *
140      * @param p Protein to translate
141      * @return list of translations
142      */
143     public static List<Protein> getTranslatedProteins(final Protein p) {
144        final List<Protein> list = new ArrayList<Protein>(2);
145        if (p.sequenceType.hasExtendedChars()) {
146           list.add(new Protein(p.description, p.rawSequence, p.sequenceType,
147                 TranslationType.TRANS_A));
148           list.add(new Protein(p.description, p.rawSequence, p.sequenceType,
149                 TranslationType.TRANS_B));
150        } else if (!p.sequenceType.isPrimary()) {
151           list.add(new Protein(p.description, p.rawSequence, p.sequenceType,
152                 TranslationType.NORMAL));
153        } else {
154           list.add(p);
155        }
156        return list;
157     }
158
```

```
159     /**
160      * Returns a cleaned sequence (X in reference sequence is instead in current
             sequence)
161      *
162      * @return cleaned sequence
163      */
164     public String getCleanSequence() {
165        if (cleanSequence == null)
166           return sequence;
167        return cleanSequence;
168     }
169
170     /**
171      * Returns type of current sequence
172      *
173      * @return type of sequence
174      */
175     public SequenceType getSequenceType() {
176        return sequenceType;
177     }
178
179     /**
180      * Sets a reference sequence. This is used to filter out unknown. Unknowns (X)
             in
181      * reference sequence replaces anything in the same location in the cleaned
             version of
182      * the current sequence.
183      *
184      * @param reference Sequence to use as reference
185      */
186     public void setReferenceSequence(final Protein reference) {
187        cleanProtein(reference);
188     }
189
190     private void cleanProtein(final Protein reference) {
191        final String refSeq = reference.getSequence();
192
193        // Find positions of unknowns.
194        final List<Integer> positions = new ArrayList<Integer>();
195        int unknownPos = 0;
196        while ((unknownPos = refSeq.indexOf('X', unknownPos)) != -1) {
197           positions.add(unknownPos++);
198        }
199        if (!sequenceType.isObservation()) {
200           Collections.reverse(positions); // go backwards thru list
201        }
202
203        final StringBuilder cleanSeq = new StringBuilder(sequence);
204        for (final Integer pos : positions) {
205           if (!sequenceType.isObservation()) {
206              /*
207               * we remove unknowns inserting X's at their position appropriately as
208               * they appear in the reference
```

```
209                 */
210                cleanSeq.setCharAt(pos, 'X');
211            } else {
212                /*
213                 * we will insert X's appropriately as they appear in the reference
214                 */
215                cleanSeq.insert((int) pos, 'X');
216            }
217        }
218
219        cleanSequence = cleanSeq.toString();
220    }
221
222    /**
223     * Return translated cleaned sequence without unknowns.
224     *
225     * @return sequence where X's are removed.
226     */
227    public String getSequenceWithoutUnknowns() {
228        return getCleanSequence().replace("X", "");
229    }
230
231    /**
232     * Returns type of sequence
233     *
234     * @return sequence type
235     */
236    public TranslationType getTranslationType() {
237        return translationType;
238    }
239
240 }
```

B.9 dk.sdu.imada.jkkn04.**Protein.data.ProteinFile**

```
1  package dk.sdu.imada.jkkn04.Protein.data;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.net.URL;
7  import java.util.HashMap;
8  import java.util.Map;
9
10 import dk.sdu.imada.jkkn04.Protein.data.Protein.TranslationType;
11
12 /**
13  * Represents a file containing sequence data for a specific
        prediction/observation type
14  *
15  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
16  */
17 public class ProteinFile {
18
```

```
19      final private URL file;
20      final private Map<String, Protein> proteins;
21
22      /**
23       * Creates a new instance for the given file/resource
24       *
25       * @param file Resource to use
26       * @throws IOException In case of read-error
27       */
28      public ProteinFile(final URL file) throws IOException {
29          super();
30          this.file = file;
31          proteins = readFile(detectPredictionType());
32      }
33
34      private SequenceType detectPredictionType() {
35          final String filename = file.getFile().toUpperCase();
36          for (final SequenceType t : SequenceType.values()) {
37              if (filename.contains(t.name()))
38                  return t;
39          }
40          return SequenceType.PRIMARY;
41      }
42
43      private Map<String, Protein> readFile(final SequenceType predictionType)
44              throws IOException {
45          final BufferedReader bf = new BufferedReader(new InputStreamReader(file
46                  .openStream()));
47
48          final Map<String, Protein> proteins = new HashMap<String, Protein>();
49
50          String currentProteinId = null;
51          StringBuffer currentProtein = null;
52          String line;
53          while ((line = bf.readLine()) != null) {
54              if (line.startsWith(">")) {
55                  if (currentProteinId != null && currentProtein != null) {
56                      /*
57                       * if (currentProtein.length() == 0) { // format error }
58                       */
59                      final Protein p = new Protein(currentProteinId, currentProtein
60                              .toString(), predictionType, TranslationType.RAW);
61                      proteins.put(p.getIdentifier(), p);
62                  }
63                  currentProteinId = line.substring(1);
64                  currentProtein = new StringBuffer();
65              } else {
66                  if (currentProtein != null) {
67                      currentProtein.append(line);
68                  } else {
69                      // format error
70                  }
71              }
```

```
72          }
73
74          if (currentProteinId != null && currentProtein != null) {
75              final Protein p = new Protein(currentProteinId,
                    currentProtein.toString(),
76                      predictionType, TranslationType.RAW);
77              proteins.put(p.getIdentifier(), p);
78          }
79
80          bf.close();
81
82          return proteins;
83      }
84
85      /**
86       * Returns the read protein map.
87       *
88       * @return map of proteins in file.
89       */
90      public Map<String, Protein> getProteins() {
91          return proteins;
92      }
93
94  }
```

B.10 dk.sdu.imada.jkkn04.**Protein.data.ProteinMap**

```
1   package dk.sdu.imada.jkkn04.Protein.data;
2
3   import java.util.HashMap;
4
5   /**
6    * Represents a map of a protein for each known sequence- and translation type
         pair.
7    *
8    * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
9    */
10  public class ProteinMap extends HashMap<SequenceTranslated, Protein> {
11
12      /**
13       *
14       */
15      private static final long serialVersionUID = 1L;
16
17  }
```

B.11 dk.sdu.imada.jkkn04.**Protein.data.ProteinMapCollection**

```
1   package dk.sdu.imada.jkkn04.Protein.data;
2
3   import java.io.IOException;
4   import java.net.URL;
5   import java.util.ArrayList;
6   import java.util.Collections;
```

```
7  import java.util.HashMap;
8  import java.util.List;
9  import java.util.Map;
10 import java.util.Set;
11 import java.util.SortedSet;
12 import java.util.TreeSet;
13
14 import dk.jkkn.util.ResourceFilter;
15 import dk.jkkn.util.ResourceWrap;
16 import dk.jkkn.util.Resources;
17 import dk.sdu.imada.jkkn04.Protein.data.Protein.TranslationType;
18
19 /**
20  * Represents a complete collection of proteins
21  *
22  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
23  */
24 public class ProteinMapCollection {
25
26    /**
27     * The release bundle some example data, this is the default location of these
28     */
29    public final static URL DEFAULT_DATA_PATH = ProteinMapCollection.class
30          .getResource("/data/protein/");
31
32    final private HashMap<String, ProteinMap> collection = new HashMap<String,
          ProteinMap>();
33    final private SortedSet<String> proteinIdentifiers = new TreeSet<String>();
34
35    /**
36     * Creates a new collection and reads all files available in the given data
            directory.
37     * All files with the extensions ".pred" and ".fasta" is read.
38     *
39     * @param dataDir directory to read
40     * @throws IOException in case of a read error.
41     */
42    public ProteinMapCollection(final URL dataDir) throws IOException {
43       final List<ProteinFile> proteinFiles = new ArrayList<ProteinFile>();
44       for (final URL file : Resources.getDirectoryList(dataDir, new
             ResourceFilter() {
45
46          @Override
47          public boolean accept(final ResourceWrap file) {
48             return file.isFile()
49                   && file.canRead()
50                   && (file.getName().endsWith(".pred") || file.getName().endsWith(
51                         ".fasta"));
52          }
53
54       })) {
55          proteinFiles.add(new ProteinFile(file));
56       }
```

```
57
58          for (final ProteinFile pf : proteinFiles) {
59              /* Create a sorted set of all existing proteins */
60              proteinIdentifiers.addAll(pf.getProteins().keySet());
61              /* Create a map for every protein sequence type => protein */
62              for (final Protein p : pf.getProteins().values()) {
63                  ProteinMap pm;
64                  if (!collection.containsKey(p.getIdentifier())) {
65                      collection.put(p.getIdentifier(), pm = new ProteinMap());
66                  } else {
67                      pm = collection.get(p.getIdentifier());
68                  }
69                  for (final Protein tp : Protein.getTranslatedProteins(p)) {
70                      pm.put(new SequenceTranslated(tp.getSequenceType(), tp
71                              .getTranslationType()), tp);
72                  }
73              }
74          }
75
76          // Hardcoded reference type to be KAKSI:
77          setProteinReferenceSequenceType(new SequenceTranslated(SequenceType.KAKSI,
78                  TranslationType.NORMAL));
79
80      }
81
82      /**
83       * Creates a new collection for a given directory
84       *
85       * @param dataDir directory to read
86       * @throws IOException in case of a read error.
87       * @see #ProteinMapCollection(URL)
88       */
89      public ProteinMapCollection(final String dataDir) throws IOException {
90          this(new URL(DEFAULT_DATA_PATH, dataDir));
91      }
92
93      /**
94       * This determinds which type we should use as reference when reading the
             sequences.
95       * If unknowns are found in the reference, they will also be read as unknown
             in the
96       * other sequences.
97       *
98       * @param type Type to use as reference
99       */
100     public void setProteinReferenceSequenceType(final SequenceTranslated type) {
101         /*
102          * If we have a reference sequence (KAKSI), set it so we can clean the
                protein
103          * sequence appropriately
104          */
105         for (final ProteinMap pm : collection.values()) {
106             if (pm.containsKey(type)) {
```

```
107                 final Protein reference = pm.get(type);
108                 for (final Map.Entry<SequenceTranslated, Protein> entry :
                        pm.entrySet()) {
109                     if (!entry.getKey().equals(type)) {
110                         entry.getValue().setReferenceSequence(reference);
111                     }
112                 }
113             } else {
114                 // warning
115             }
116         }
117 
118     }
119 
120     /**
121      * Returns a map of all sequence identifiers
122      *
123      * @return map of ids
124      */
125     public Set<String> getProteinIdentifiers() {
126         return Collections.unmodifiableSet(proteinIdentifiers);
127     }
128 
129     /**
130      * Return a map for a given protein identifier
131      *
132      * @param identifier Protein identifier
133      * @return map
134      */
135     public ProteinMap getProteinMap(final String identifier) {
136         return collection.get(identifier);
137     }
138 
139 }
```

B.12 dk.sdu.imada.jkkn04.**Protein.data.SequenceType**

```
1  /**
2   *
3   */
4  package dk.sdu.imada.jkkn04.Protein.data;
5 
6  import java.util.EnumSet;
7 
8  /**
9   * Represents an observed or predicted sequence type
10  *
11  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
12  */
13 public enum SequenceType {
14     /**
15      * Primary sequence string
16      */
17     PRIMARY(),
```

```
18      /**
19       * DSSP type observation sequence
20       */
21      DSSP(EnumSet.of(SequenceAttributes.OBSERVATION,
            SequenceAttributes.EXTENDED_CHARS)),
22      /**
23       * STRIDE type observation sequence
24       */
25      STRIDE(EnumSet.of(SequenceAttributes.OBSERVATION,
            SequenceAttributes.EXTENDED_CHARS)),
26      /**
27       * KAKSI type observation sequence
28       */
29      KAKSI(EnumSet.of(SequenceAttributes.OBSERVATION)),
30      /**
31       * SVM type prediction sequence
32       */
33      SVM_PRED(EnumSet.of(SequenceAttributes.PREDICTION)),
34      /**
35       * SVM VITERBI type prediction sequence
36       */
37      SVM_VITERBI(EnumSet.of(SequenceAttributes.PREDICTION)),
38      /**
39       * BRNN type prediction sequence
40       */
41      BRNN_PRED(EnumSet.of(SequenceAttributes.PREDICTION)),
42      /**
43       * BRNN VITERBI type prediction sequence
44       */
45      BRNN_VITERBI(EnumSet.of(SequenceAttributes.PREDICTION)),
46      /**
47       * HMM21 Multiple sequence type prediction
48       */
49      HMM21_MULTISEQ(EnumSet.of(SequenceAttributes.PREDICTION)),
50      /**
51       * HMM21 Uni-sequence type prediction
52       */
53      HMM21_UNISEQ(EnumSet.of(SequenceAttributes.PREDICTION)),
54      /**
55       * HMM75 Multiple sequence type prediction
56       */
57      HMM75_MULTISEQ(EnumSet.of(SequenceAttributes.PREDICTION)),
58      /**
59       * HMM75 Uni-sequence type prediction
60       */
61      HMM75_UNISEQ(EnumSet.of(SequenceAttributes.PREDICTION)),
62
63      /**
64       * Observation type used for testing
65       */
66      TEST_OBSERVATION(EnumSet.of(SequenceAttributes.OBSERVATION)),
67
68      /**
```

```
69       * Prediction type used for testing
70       */
71      TEST_PREDICTION(EnumSet.of(SequenceAttributes.PREDICTION));
72
73      private enum SequenceAttributes {
74          PREDICTION, OBSERVATION, EXTENDED_CHARS
75      }
76
77      private final EnumSet<SequenceAttributes> attributes;
78
79      /**
80       * Set of all available prediction types
81       */
82      public static final EnumSet<SequenceType> PREDICTION_TYPES =
            getByAttribute(SequenceAttributes.PREDICTION);
83
84      /**
85       * Set of all avalable observation types
86       */
87      public static final EnumSet<SequenceType> OBSERVATION_TYPES =
            getByAttribute(SequenceAttributes.OBSERVATION);
88
89      private SequenceType(final EnumSet<SequenceAttributes> attributes) {
90          this.attributes = attributes;
91      }
92
93      private SequenceType() {
94          this(EnumSet.noneOf(SequenceAttributes.class));
95      }
96
97      /**
98       * Returns true if this is a prediction type
99       *
100      * @return true if prediction type
101      */
102     public boolean isPrediction() {
103         return attributes.contains(SequenceAttributes.PREDICTION);
104     }
105
106     /**
107      * Returns true if this is a observation type
108      *
109      * @return true if observation type
110      */
111     public boolean isObservation() {
112         return attributes.contains(SequenceAttributes.OBSERVATION);
113     }
114
115     /**
116      * Returns true if this uses 8 characters instead of 3.
117      *
118      * @return true if has extended characters
119      */
```

```
120     public boolean hasExtendedChars() {
121         return attributes.contains(SequenceAttributes.EXTENDED_CHARS);
122     }
123 
124     /**
125      * Return true if primary sequence
126      *
127      * @return true if primary
128      */
129     public boolean isPrimary() {
130         return this == SequenceType.PRIMARY;
131     }
132 
133     private static EnumSet<SequenceType> getByAttribute(final SequenceAttributes
          attribute) {
134         final EnumSet<SequenceType> set = EnumSet.noneOf(SequenceType.class);
135         for (final SequenceType type : SequenceType.values()) {
136             if (type.attributes.contains(attribute)) {
137                 set.add(type);
138             }
139         }
140         return set;
141     }
142 }
```

B.13 dk.sdu.imada.jkkn04.**Protein.data.SequenceTranslated**

```
1  package dk.sdu.imada.jkkn04.Protein.data;
2  
3  import java.util.HashSet;
4  import java.util.Set;
5  
6  import dk.sdu.imada.jkkn04.Protein.data.Protein.TranslationType;
7  
8  /**
9   * Represents a protein sequence type in a given translation form.
10  *
11  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
12  */
13 public class SequenceTranslated implements Comparable<SequenceTranslated> {
14 
15     /**
16      * Creates a new representative for a given protein type in a given translation
17      *
18      * @param sequenceType Protein sequence type
19      * @param translationType Translation type
20      */
21     public SequenceTranslated(final SequenceType sequenceType,
22             final TranslationType translationType) {
23         super();
24         this.sequenceType = sequenceType;
25         this.translationType = translationType;
26     }
27 
```

```
28      /**
29       * (non-Javadoc)
30       *
31       * @see java.lang.Object#hashCode()
32       */
33      @Override
34      public int hashCode() {
35          final int prime = 31;
36          int result = 1;
37          result = prime * result + ((sequenceType == null) ? 0 :
                sequenceType.hashCode());
38          result = prime * result
39                  + ((translationType == null) ? 0 : translationType.hashCode());
40          return result;
41      }
42
43      /**
44       * (non-Javadoc)
45       *
46       * @see java.lang.Object#equals(java.lang.Object)
47       */
48      @Override
49      public boolean equals(final Object obj) {
50          if (this == obj)
51              return true;
52          if (obj == null)
53              return false;
54          if (getClass() != obj.getClass())
55              return false;
56          final SequenceTranslated other = (SequenceTranslated) obj;
57          if (sequenceType == null) {
58              if (other.sequenceType != null)
59                  return false;
60          } else if (!sequenceType.equals(other.sequenceType))
61              return false;
62          if (translationType == null) {
63              if (other.translationType != null)
64                  return false;
65          } else if (!translationType.equals(other.translationType))
66              return false;
67          return true;
68      }
69
70      /**
71       * Returns name of sequence type appended with translation if not
72       * TranslationType.NORMAL
73       *
74       * @see java.lang.Object#toString()
75       */
76      @Override
77      public String toString() {
78          return sequenceType.name()
79                  + (translationType != TranslationType.NORMAL ? "_"
```

```
80                      + translationType.name() : "");
81      }
82
83      /**
84       * Return sequence type
85       *
86       * @return sequence type
87       */
88      public SequenceType getSequenceType() {
89         return sequenceType;
90      }
91
92      /**
93       * Return translation type
94       *
95       * @return type of translation
96       */
97      public TranslationType getTranslationType() {
98         return translationType;
99      }
100
101     private final SequenceType sequenceType;
102     private final TranslationType translationType;
103
104     /**
105      * Construct a name for this sequencetype/translation pair.
106      *
107      * @return type + ":" + translation
108      */
109     public String name() {
110        return sequenceType.name() + ":" + translationType;
111     }
112
113     /**
114      * Creates a set of all possible translations of a given set of sequence types
115      *
116      * @param types sequence types
117      * @return set of all translation of those sequence types
118      */
119     public static Set<SequenceTranslated> buildAllOf(final Set<SequenceType>
         types) {
120        final Set<SequenceTranslated> set = new HashSet<SequenceTranslated>();
121        for (final SequenceType type : types) {
122           for (final TranslationType transType : TranslationType.values()) {
123              set.add(new SequenceTranslated(type, transType));
124           }
125        }
126        return set;
127     }
128
129     /**
130      * (non-Javadoc)
131      *
```

```
132      * @see java.lang.Comparable#compareTo(java.lang.Object)
133      */
134     @Override
135     public int compareTo(final SequenceTranslated o) {
136         if (o.sequenceType != sequenceType)
137             return sequenceType.compareTo(o.sequenceType);
138         return translationType.compareTo(o.translationType);
139     }
140 
141 }
```

B.14 dk.sdu.imada.jkkn04.Protein.gui.ProteinApp

```
1  /**
2   *
3   */
4  package dk.sdu.imada.jkkn04.Protein.gui;
5  
6  import javax.swing.UIManager;
7  import javax.swing.UnsupportedLookAndFeelException;
8  
9  /**
10  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
11  */
12 public class ProteinApp {
13 
14     /**
15      * Main method - This makes up the main method of the GUI program for the
            evaluator
16      *
17      * @param args Not used
18      */
19     public static void main(final String[] args) {
20         try {
21             // UIManager.setLookAndFeel(
22             // "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
23             UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
24         } catch (final ClassNotFoundException e) {
25             e.printStackTrace();
26         } catch (final InstantiationException e) {
27             e.printStackTrace();
28         } catch (final IllegalAccessException e) {
29             e.printStackTrace();
30         } catch (final UnsupportedLookAndFeelException e) {
31             e.printStackTrace();
32         }
33 
34         try {
35             new ProteinWindowImpl();
36         } catch (final Exception e) {
37             e.printStackTrace();
38         }
39     }
40 
```

```
41 }
```

B.15 dk.sdu.imada.jkkn04.**Protein.gui.ProteinWindow**

```
1  package dk.sdu.imada.jkkn04.Protein.gui;
2
3  import java.awt.Component;
4  import java.awt.Container;
5  import java.awt.Desktop;
6  import java.awt.Dimension;
7  import java.awt.Font;
8  import java.awt.event.ActionEvent;
9  import java.net.URI;
10 import java.net.URL;
11 import java.util.Iterator;
12
13 import javax.swing.AbstractAction;
14 import javax.swing.Action;
15 import javax.swing.JComponent;
16 import javax.swing.JDialog;
17 import javax.swing.JFrame;
18 import javax.swing.JOptionPane;
19 import javax.swing.JPanel;
20 import javax.swing.JScrollBar;
21 import javax.swing.JScrollPane;
22 import javax.swing.ScrollPaneConstants;
23 import javax.swing.SwingUtilities;
24 import javax.swing.border.EmptyBorder;
25 import javax.swing.plaf.ScrollBarUI;
26
27 import org.swixml.ConverterLibrary;
28 import org.swixml.SwingEngine;
29 import org.swixml.SwingTagLibrary;
30
31 import dk.jkkn.util.swixml.ClassNameConverter;
32 import dk.jkkn.util.swixml.StringArrayConverter;
33
34 /**
35  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
36  */
37 public abstract class ProteinWindow {
38
39     static private final String HOMEPAGE = "http://jkkn.dk/sdu/proteins/";
40
41     private final JFrame mainFrame;
42
43     JPanel labelPaneForSeqs, gridForSeqs;
44     JScrollPane scrollPaneForSeqs;
45
46     /* Actions */
47
48     /**
49      * Action for exit menu item
50      */
```

```
51      public Action actionExit = new AbstractAction() {
52          private static final long serialVersionUID = 1L;
53
54          public void actionPerformed(final ActionEvent e) {
55              System.exit(0);
56          }
57      };
58
59      /**
60       * Action for help menu item (opens homepage)
61       */
62      public Action actionHelp = new AbstractAction() {
63          private static final long serialVersionUID = 1L;
64
65          public void actionPerformed(final ActionEvent e) {
66              boolean okay = false;
67              try {
68                  if (Desktop.isDesktopSupported()) {
69                      final Desktop desktop = Desktop.getDesktop();
70                      if (desktop.isSupported(Desktop.Action.BROWSE)) {
71                          desktop.browse(new URI(HOMEPAGE));
72                          okay = true;
73                      }
74                  }
75              } catch (final Throwable th) {
76                  th.printStackTrace();
77              }
78              if (!okay) {
79                  JOptionPane.showMessageDialog(getMainFrame(),
80                          "You can find the homepage at: \n" + HOMEPAGE, getMainFrame()
81                                  .getTitle(), JOptionPane.INFORMATION_MESSAGE);
82              }
83          }
84      };
85
86      /**
87       * Action for opening about dialog
88       */
89      public Action actionAbout = new AbstractAction() {
90          private static final long serialVersionUID = 1L;
91
92          public void actionPerformed(final ActionEvent e) {
93              JDialog dialog;
94              try {
95                  dialog = (JDialog) renderWindow(ProteinWindow.class
96                          .getResource("AboutWindow.xml"));
97                  dialog.setModal(true);
98                  dialog.pack();
99                  dialog.setLocationRelativeTo(getMainFrame());
100                 dialog.setVisible(true);
101             } catch (final Exception e1) {
102                 e1.printStackTrace();
103             }
```

```
104         }
105     };
106 
107     /**
108      * Action for closing about dialog
109      */
110     public Action actionCloseAbout = new AbstractAction() {
111         private static final long serialVersionUID = 1L;
112 
113         public void actionPerformed(final ActionEvent e) {
114             SwingUtilities.getWindowAncestor((Component) e.getSource()).dispose();
115         }
116     };
117 
118     /**
119      * Creates a new Protein GUI Window
120      *
121      * @throws Exception in case of parse error
122      */
123     public ProteinWindow() throws Exception {
124 
125         ConverterLibrary.getInstance().register(String[].class,
126                 new StringArrayConverter());
127         ConverterLibrary.getInstance().register(Class.class, new
                ClassNameConverter());
128         SwingTagLibrary.getInstance().registerTag("resulttable", ResultTable.class);
129 
130         final Container container = renderWindow(ProteinWindow.class
131                 .getResource("ProteinWindow.xml"));
132         mainFrame = (JFrame) container;
133         SwingEngine.setAppFrame(getMainFrame());
134 
135         // Manually we need to set the correct border needed for the horizontal
136         // scrollbar
137         final JScrollBar sc = scrollPaneForSeqs.getHorizontalScrollBar();
138         final ScrollBarUI scrollUI = sc.getUI();
139         final int scrollHeight = (int) scrollUI.getPreferredSize(sc).getHeight();
140         labelPaneForSeqs.setBorder(new EmptyBorder(0, 0, scrollHeight, 0));
141         scrollPaneForSeqs
142                 .setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
143         scrollPaneForSeqs
144                 .setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER);
145         // Synchronizes size of grid-fields (depends upon fonts used)
146         final int height = Math.max(
147                 labelPaneForSeqs.getComponent(3).getPreferredSize().height,
                    gridForSeqs
148                         .getComponent(0).getPreferredSize().height);
149 
150         labelPaneForSeqs.getComponent(0).setMinimumSize(new Dimension(1, height));
151         labelPaneForSeqs.getComponent(0).setMaximumSize(new Dimension(1, height));
152         labelPaneForSeqs.getComponent(0).setPreferredSize(new Dimension(1, height));
153 
154         mainFrame.setLocationRelativeTo(null);
```

```
155     }
156
157     /**
158      * Method for rendering a window, adds support for bold and text with increased
159      * fontsize
160      *
161      * @param swixml SWIXML resource
162      * @return Container for dialog/frame
163      * @throws Exception In case of any parsing errors
164      */
165     Container renderWindow(final URL swixml) throws Exception {
166
167         final SwingEngine swix = new SwingEngine(this);
168         final Container container = swix.render(swixml);
169
170         // Make headers bold
171         final Iterator<?> i = swix.getAllComponentItertor();
172         while (i.hasNext()) {
173             final Object o = i.next();
174             if (o instanceof JComponent) {
175                 final JComponent c = (JComponent) o;
176                 if (c.getClientProperty("makeBold") != null) {
177                     if (c.getClientProperty("makeBold").equals("true")) {
178                         c.setFont(c.getFont().deriveFont(Font.BOLD));
179                     }
180                     c.putClientProperty("makeBold", null);
181                 }
182                 if (c.getClientProperty("incrFont") != null) {
183                     try {
184                         c.setFont(c.getFont().deriveFont(
185                                 c.getFont().getSize()
186                                         + Float.parseFloat((String) c
187                                                 .getClientProperty("incrFont"))));
188                     } catch (final NumberFormatException nfe) {
189                         nfe.printStackTrace();
190                     } finally {
191                         c.putClientProperty("incrFont", null);
192                     }
193                 }
194             }
195         }
196         return container;
197     }
198
199     /**
200      * Returns frame for main window
201      *
202      * @return frame
203      */
204     public JFrame getMainFrame() {
205         return mainFrame;
206     }
207
```

```
208 }
```

B.16 dk.sdu.imada.jkkn04.**Protein.gui.ProteinWindowImpl**

```
1  package dk.sdu.imada.jkkn04.Protein.gui;
2
3  import java.awt.CardLayout;
4  import java.awt.Color;
5  import java.awt.Container;
6  import java.awt.GradientPaint;
7  import java.awt.GridLayout;
8  import java.awt.event.ActionEvent;
9  import java.awt.event.ActionListener;
10 import java.io.File;
11 import java.io.IOException;
12 import java.net.MalformedURLException;
13 import java.net.URL;
14 import java.text.DecimalFormat;
15 import java.text.NumberFormat;
16 import java.util.ArrayList;
17 import java.util.HashMap;
18 import java.util.List;
19 import java.util.Map;
20 import java.util.Set;
21 import java.util.TreeSet;
22
23 import javax.swing.AbstractAction;
24 import javax.swing.Action;
25 import javax.swing.JButton;
26 import javax.swing.JCheckBox;
27 import javax.swing.JComboBox;
28 import javax.swing.JFileChooser;
29 import javax.swing.JList;
30 import javax.swing.JMenuItem;
31 import javax.swing.JOptionPane;
32 import javax.swing.JPanel;
33 import javax.swing.JTabbedPane;
34 import javax.swing.JTextField;
35 import javax.swing.ListModel;
36 import javax.swing.event.ChangeEvent;
37 import javax.swing.event.ChangeListener;
38 import javax.swing.event.ListSelectionEvent;
39 import javax.swing.event.ListSelectionListener;
40 import javax.swing.filechooser.FileFilter;
41 import javax.swing.table.TableModel;
42 import javax.swing.text.JTextComponent;
43
44 import org.jfree.chart.ChartFactory;
45 import org.jfree.chart.ChartPanel;
46 import org.jfree.chart.JFreeChart;
47 import org.jfree.chart.axis.NumberAxis;
48 import org.jfree.chart.plot.CategoryPlot;
49 import org.jfree.chart.plot.PiePlot3D;
50 import org.jfree.chart.plot.PlotOrientation;
```

```
51  import org.jfree.chart.renderer.category.BarRenderer;
52  import org.jfree.data.category.DefaultCategoryDataset;
53  import org.jfree.data.general.DefaultPieDataset;
54
55  import dk.jkkn.util.ResourceFilter;
56  import dk.jkkn.util.ResourceWrap;
57  import dk.jkkn.util.Resources;
58  import dk.sdu.imada.jkkn04.Protein.data.Protein;
59  import dk.sdu.imada.jkkn04.Protein.data.ProteinMap;
60  import dk.sdu.imada.jkkn04.Protein.data.ProteinMapCollection;
61  import dk.sdu.imada.jkkn04.Protein.data.SequenceTranslated;
62  import dk.sdu.imada.jkkn04.Protein.data.SequenceType;
63  import dk.sdu.imada.jkkn04.Protein.data.Protein.TranslationType;
64  import dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation;
65  import dk.sdu.imada.jkkn04.Protein.tests.MCC;
66  import dk.sdu.imada.jkkn04.Protein.tests.QScore;
67  import dk.sdu.imada.jkkn04.Protein.tests.SOV;
68  import dk.sdu.imada.jkkn04.Protein.tests.StateMachine;
69  import dk.sdu.imada.jkkn04.Protein.tests.StateMachine.ConfusionParameter;
70  import dk.sdu.imada.jkkn04.Protein.util.LaTeXFormatter;
71  import dk.sdu.imada.jkkn04.Protein.util.ProcessUtil;
72  import dk.sdu.imada.jkkn04.Protein.util.ProcessUtil.MultipleEvaluationsMap;
73  import dk.sdu.imada.jkkn04.Protein.util.ProcessUtil.ObservationPredictionMap;
74
75  /**
76   * This class implements the logic for the Protein GUI Window.
77   *
78   * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
79   */
80  public class ProteinWindowImpl extends ProteinWindow {
81
82      private static final NumberFormat DECIMALFORMAT = new DecimalFormat("#.####");
83
84      private ProteinMapCollection proteinCollection = null;
85      ObservationPredictionMap analyseResultSum = null;
86      boolean disableAutoAnalysis = false;
87
88      private Object previousSelectedObs = null;
89      private Object previousSelectedPredict = null;
90      private Object previousSelectedObsGraph = null;
91
92      JMenuItem saveMenuItem;
93
94      JList folderList, sequenceList;
95      JButton selectFolder, analyseButton, dataModeButton, graphModeButton;
96
97      JPanel modePanel;
98
99      JComboBox selectPredType, selectObsType;
100
101     JTextField sequenceId, predictedType, observedType;
102     JTextField detection1, detection2, predicted, observed, primary;
103
```

```
104     ResultTable QScoreTable, QScoreAccumTable;
105     JTextField QSeqCount, QSeqLength;
106
107     ResultTable MCCTable;
108     JTextField MCCSeqCount, MCCSeqLength;
109
110     ResultTable SOVTable;
111     JTextField SOVSeqCount, SOVSeqLength;
112
113     ResultTable StateAccTable;
114     JTextField SMScore, SMSeqCount, SMSeqLength;
115     ResultTable StateTable, StateAccPredTable, StateAccObsTable;
116
117     JTabbedPane graphTabs;
118     JPanel graphPanel, piePanel;
119     JComboBox selectObsTypeGraph;
120     JCheckBox chartByEval;
121
122     private static JFreeChart chartResults;
123
124     // Folder button action
125     private final Action actionSelectFolder = new AbstractAction() {
126         private static final long serialVersionUID = 1L;
127         private final JFileChooser fileChooser = new JFileChooser();
128         {
129             fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
130         }
131
132         @Override
133         public void actionPerformed(final ActionEvent e) {
134             if (fileChooser.showOpenDialog(getMainFrame()) ==
                    JFileChooser.APPROVE_OPTION) {
135                 try {
136                     updateFolderList(fileChooser.getSelectedFile().toURI().toURL());
137                 } catch (final MalformedURLException e1) {
138                     e1.printStackTrace();
139                 }
140             }
141         }
142     };
143
144     // Analyse button action
145     private final Action actionAnalyse = new AbstractAction() {
146         private static final long serialVersionUID = 1L;
147
148         @Override
149         public void actionPerformed(final ActionEvent e) {
150             runAnalysis();
151         }
152     };
153
154     // Graph and data switch button actions
155     private final Action actionModeSwitch = new AbstractAction() {
```

```
156         private static final long serialVersionUID = 1L;
157 
158         @Override
159         public void actionPerformed(final ActionEvent e) {
160             final boolean showData = (e.getSource() == dataModeButton);
161             ((CardLayout) modePanel.getLayout()).show(modePanel, showData ?
                    "dataMode"
162                     : "graphMode");
163             dataModeButton.setEnabled(!showData);
164             graphModeButton.setEnabled(showData);
165         }
166     };
167 
168     // Save as LaTeX button action
169     private final Action actionSave = new AbstractAction() {
170         private static final long serialVersionUID = 1L;
171         private final JFileChooser fileChooser = new JFileChooser();
172         {
173             fileChooser.setFileFilter(new FileFilter() {
174                 @Override
175                 public boolean accept(File f) {
176                     if (f.isDirectory())
177                         return true;
178                     return f.getName().endsWith(".tex");
179                 }
180 
181                 @Override
182                 public String getDescription() {
183                     return "LaTeX files (*.tex)";
184                 }
185             });
186         }
187 
188         @Override
189         public void actionPerformed(final ActionEvent e) {
190             if (fileChooser.showSaveDialog(getMainFrame()) ==
                    JFileChooser.APPROVE_OPTION) {
191                 boolean okay = true;
192                 if (fileChooser.getSelectedFile().exists()) {
193                     okay = (JOptionPane.showConfirmDialog(getMainFrame(),
194                             "File already exists:\n"
195                                     + fileChooser.getSelectedFile().getName()
196                                     + "\nAre you sure to overwrite?\n", getMainFrame()
197                                     .getTitle(), JOptionPane.YES_NO_OPTION,
198                             JOptionPane.WARNING_MESSAGE) == JOptionPane.YES_OPTION);
199                 }
200                 if (okay) {
201                     LaTeXFormatter.writeFile(fileChooser.getSelectedFile(),
202                             analyseResultSum);
203                 }
204             }
205         }
206     };
```

```
207
208
209     /**
210      * Creates the main Protein Window implemented version
211      *
212      * @throws Exception in case of parse errors
213      */
214     public ProteinWindowImpl() throws Exception {
215         super();
216         setupActions();
217         /* add graph to graph mode */
218         graphPanel.add(createChart());
219
220         /* make more intelligent grid layout for pies */
221         piePanel.setLayout(new GridLayout() {
222             private static final long serialVersionUID = 1L;
223             @Override
224             public void layoutContainer(Container parent) {
225
226                 if (parent.getComponentCount() > 0) {
227                     // Form layout after space available
228                     final int cols = Math.max(1, (int)
                            Math.round(Math.sqrt(parent.getComponentCount() / ((double)
                            piePanel.getSize().height / piePanel.getSize().width))));
229                     final int rows =
                            (int)Math.ceil((double)parent.getComponentCount()/cols);
230                     setColumns(cols);
231                     setRows(rows);
232                 }
233
234                 super.layoutContainer(parent);
235             }
236         });
237
238         updateContentOfComponents();
239         getMainFrame().setVisible(true);
240     }
241
242     private void setupActions() {
243         saveMenuItem.addActionListener(actionSave);
244         selectFolder.addActionListener(actionSelectFolder);
245         analyseButton.addActionListener(actionAnalyse);
246         dataModeButton.addActionListener(actionModeSwitch);
247         graphModeButton.addActionListener(actionModeSwitch);
248         folderList.addListSelectionListener(new ListSelectionListener() {
249             @Override
250             public void valueChanged(final ListSelectionEvent e) {
251                 updateProteinList();
252             }
253         });
254
255         sequenceList.addListSelectionListener(new ListSelectionListener() {
256             @Override
```

```
257             public void valueChanged(final ListSelectionEvent e) {
258                 analyseResultSum = null;
259                 /*
260                  * if only one is selected we will run the analysis automatically
261                  */
262                 if (!disableAutoAnalysis && sequenceList.getSelectedValues().length
                        == 1) {
263                     runAnalysis();
264                 } else {
265                     analyseButton.setEnabled(sequenceList.getSelectedValues().length >
                            0);
266                 }
267                 updateTypeLists();
268 
269             }
270         });
271 
272         final ActionListener updateResults = new ActionListener() {
273             @Override
274             public void actionPerformed(final ActionEvent e) {
275                 updateProteinResults();
276             }
277         };
278 
279         selectObsType.addActionListener(updateResults);
280         selectPredType.addActionListener(updateResults);
281 
282         /* graph panel */
283         final ActionListener updateChart = new ActionListener() {
284             @Override
285             public void actionPerformed(final ActionEvent e) {
286                 updateChart();
287             }
288         };
289 
290         selectObsTypeGraph.addActionListener(updateChart);
291         chartByEval.addActionListener(updateChart);
292 
293         graphTabs.addChangeListener(new ChangeListener() {
294             @Override
295             public void stateChanged(ChangeEvent e) {
296                 chartByEval.setEnabled(((JTabbedPane)e.getSource()).getSelectedComponent()
                        == graphPanel);
297             }
298         });
299 
300         // Setting default button.
301         getMainFrame().getRootPane().setDefaultButton(analyseButton);
302     }
303 
304     private void updateContentOfComponents() {
305         /* update all parts of GUI */
306         updateFolderList(null);
```

```
307         updateProteinList();
308         updateTypeLists();
309         updateProteinResults();
310     }
311
312     /**
313      * This updates the list of proteins sequences
314      */
315     protected void updateProteinList() {
316         try {
317             if (folderList.getSelectedIndex() != -1) {
318                 proteinCollection = new ProteinMapCollection(((URLView) folderList
319                         .getSelectedValue()).url);
320                 analyseResultSum = null;
321                 sequenceList.setListData(proteinCollection.getProteinIdentifiers()
322                         .toArray());
323
324                 // Select all by default
325                 final int[] selected = new int[sequenceList.getModel().getSize()];
326                 for (int i = 0; i < selected.length; i++) {
327                     selected[i] = i;
328                 }
329                 if (selected.length > 1) {
330                     disableAutoAnalysis = true;
331                 }
332                 sequenceList.setSelectedIndices(selected);
333                 disableAutoAnalysis = false;
334             } else {
335                 sequenceList.setListData(new Object[] {});
336             }
337         } catch (final IOException e) {
338             // TODO: Display warning
339             e.printStackTrace();
340         }
341     }
342
343     /**
344      * Updates list of available folders
345      *
346      * @param otherDirectory Custom selected folder to also add.
347      */
348     protected void updateFolderList(final URL otherDirectory) {
349         final List<URLView> folders = new ArrayList<URLView>();
350         final URLView otherDir = otherDirectory != null ? new
                URLView(otherDirectory)
351                 : null;
352
353         try {
354             for (final URL file : Resources.getDirectoryList(
355                     ProteinMapCollection.DEFAULT_DATA_PATH, new ResourceFilter() {
356
357                         @Override
358                         public boolean accept(final ResourceWrap resource) {
```

```
359                     return resource.isDirectory();
360                 }
361
362             })) {
363                 folders.add(new URLView(file));
364             }
365         } catch (final IOException e) {
366             // FIXME: Throw something
367             e.printStackTrace();
368         }
369
370         if (otherDir != null) {
371             folders.add(otherDir);
372         }
373
374         folderList.setListData(folders.toArray());
375
376         if (otherDir != null) {
377             folderList.setSelectedValue(otherDir, true);
378         }
379
380     }
381
382     /**
383      * Updates comboboxes for prediction and observed types
384      */
385     void updateTypeLists() {
386         /* try to preserve selected items */
387         if (selectObsType.getSelectedIndex() != -1) {
388             previousSelectedObs = selectObsType.getSelectedItem();
389         }
390         if (selectPredType.getSelectedIndex() != -1) {
391             previousSelectedPredict = selectPredType.getSelectedItem();
392         }
393         if (selectObsTypeGraph.getSelectedIndex() != -1) {
394             previousSelectedObsGraph = selectObsTypeGraph.getSelectedItem();
395         }
396
397         selectObsType.removeAllItems();
398         selectPredType.removeAllItems();
399         selectObsTypeGraph.removeAllItems();
400         if (proteinCollection != null && analyseResultSum != null) {
401             final Set<SequenceTranslated> allTypes = new
                  TreeSet<SequenceTranslated>();
402             /*
403              * make map of available types looking at which protein that are selected
404              */
405             for (final Object sequence : sequenceList.getSelectedValues()) {
406                 allTypes.addAll(proteinCollection.getProteinMap((String) sequence)
407                         .keySet());
408             }
409
410             for (final SequenceTranslated st : allTypes) {
```

```
411             if (st.getSequenceType().isObservation()) {
412                 selectObsType.addItem(st);
413                 selectObsTypeGraph.addItem(st);
414             }
415             if (st.getSequenceType().isPrediction()) {
416                 selectPredType.addItem(st);
417             }
418         }
419
420         /* preserve selection */
421         if (previousSelectedObs != null
422                 && previousSelectedObs instanceof SequenceTranslated) {
423             /*
424              * ignored if no longer in combobox
425              */
426             selectObsType.setSelectedItem(previousSelectedObs);
427         }
428         if (previousSelectedPredict != null
429                 && previousSelectedPredict instanceof SequenceTranslated) {
430             /*
431              * ignored if no longer in combobox
432              */
433             selectPredType.setSelectedItem(previousSelectedPredict);
434         }
435         if (previousSelectedObsGraph != null
436                 && previousSelectedObsGraph instanceof SequenceTranslated) {
437             /*
438              * ignored if no longer in combobox
439              */
440             selectObsTypeGraph.setSelectedItem(previousSelectedObsGraph);
441         }
442     }
443 }
444
445 /**
446  * This fills in all the result fields in the window.
447  */
448 void updateProteinResults() {
449     /* reset all fields first */
450     for (final JTextComponent field : new JTextComponent[] { sequenceId,
451             predictedType, observedType, detection1, detection2, predicted,
                observed,
452             primary, QSeqCount, QSeqLength, MCCSeqCount, MCCSeqLength, SMScore,
453             SOVSeqCount, SOVSeqLength, SMSeqCount, SMSeqLength, }) {
454         field.setText("");
455     }
456
457     /* reset all table values */
458     for (final ResultTable table : new ResultTable[] { QScoreTable,
            QScoreAccumTable,
459             MCCTable, SOVTable, StateAccTable, StateTable, StateAccPredTable,
460             StateAccObsTable }) {
461         final TableModel model = table.getModel();
```

```
462             for (int c = 0; c < model.getColumnCount(); c++) {
463                 for (int r = 0; r < model.getRowCount(); r++) {
464                     model.setValueAt(null, r, c);
465                 }
466             }
467         }
468 
469         if (proteinCollection != null && analyseResultSum != null
470                 && !analyseResultSum.isEmpty() && selectObsType.getSelectedIndex() !=
                        -1
471                 && selectPredType.getSelectedIndex() != -1) {
472 
473             final SequenceTranslated selectedObs = (SequenceTranslated) selectObsType
474                     .getSelectedItem();
475             final SequenceTranslated selectedPred = (SequenceTranslated)
                        selectPredType
476                     .getSelectedItem();
477 
478             final Object proteins[] = sequenceList.getSelectedValues();
479 
480             if (proteins.length != 1) {
481                 sequenceId.setText(proteins.length > 1 ? "[multiple selected]" : "");
482             } else {
483                 final String id = (String) proteins[0];
484                 final ProteinMap proteinMap = proteinCollection.getProteinMap(id);
485                 final Protein obsProtein = proteinMap.get(selectedObs);
486                 final Protein predProtein = proteinMap.get(selectedPred);
487                 sequenceId.setText(id);
488                 predictedType.setText(predProtein.getDescription());
489                 observedType.setText(obsProtein.getDescription());
490                 predicted.setText(predProtein.getCleanSequence());
491                 observed.setText(obsProtein.getCleanSequence());
492 
493                 /* Try to find primary sequence */
494                 final Protein primaryProtein = proteinMap.get(new SequenceTranslated(
495                         SequenceType.PRIMARY, TranslationType.RAW));
496                 if (primaryProtein != null) {
497                     primary.setText(primaryProtein.getSequence());
498                 }
499             }
500 
501             /* Fill in all the various tables */
502             final MultipleEvaluationsMap resultsPerAnalysis = analyseResultSum.get(
503                     selectedObs).get(selectedPred);
504             for (final AbstractEvaluation e : resultsPerAnalysis.values()) {
505                 final Map<String, Object> results = e.getResults();
506                 if (e instanceof QScore) {
507                     TableModel model = QScoreTable.getModel();
508                     ListModel rowModel = QScoreTable.getRowModel();
509                     for (int c = 0; c < model.getColumnCount(); c++) {
510                         final String colName = model.getColumnName(c);
511                         for (int r = 0; r < model.getRowCount(); r++) {
512                             final String rowName = (String) rowModel.getElementAt(r);
```

```
513                     model.setValueAt(results.get("Q-" + rowName + colName), r,
                            c);
514                 }
515             }
516
517             model = QScoreAccumTable.getModel();
518             rowModel = QScoreAccumTable.getRowModel();
519             for (int r = 0; r < model.getRowCount(); r++) {
520                 final String rowName = (String) rowModel.getElementAt(r);
521                 model.setValueAt(results.get(rowName), r, 0);
522             }
523
524             QSeqCount.setText(Integer
525                     .toString((Integer) results.get("sequences")));
526             QSeqLength.setText(Integer.toString((Integer)
                    results.get("length")));
527
528         } else if (e instanceof MCC) {
529
530             final TableModel model = MCCTable.getModel();
531             final ListModel rowModel = MCCTable.getRowModel();
532             for (int r = 0; r < model.getRowCount(); r++) {
533                 final String rowName = (String) rowModel.getElementAt(r);
534                 model.setValueAt(results.get(rowName), r, 0);
535             }
536
537             MCCSeqCount.setText(Integer.toString((Integer) results
538                     .get("sequences")));
539             MCCSeqLength.setText(Integer
540                     .toString((Integer) results.get("length")));
541
542         } else if (e instanceof SOV) {
543
544             final TableModel model = SOVTable.getModel();
545             final ListModel rowModel = SOVTable.getRowModel();
546             for (int r = 0; r < model.getRowCount(); r++) {
547                 final String rowName = (String) rowModel.getElementAt(r);
548                 model.setValueAt(results.get(rowName), r, 0);
549             }
550
551             SOVSeqCount.setText(Integer.toString((Integer) results
552                     .get("sequences")));
553             SOVSeqLength.setText(Integer
554                     .toString((Integer) results.get("length")));
555
556         } else if (e instanceof StateMachine) {
557
558             SMScore.setText(DECIMALFORMAT.format(results.get("State")));
559
560             /* fill in all sum tables */
561             final Map<ResultTable, String> tables = new HashMap<ResultTable,
                    String>();
562             tables.put(StateAccTable, "");
```

```
563                 tables.put(StateAccPredTable, "-Predicted");
564                 tables.put(StateAccObsTable, "-Observed");
565                 for (final Map.Entry<ResultTable, String> tableToPrefix : tables
566                       .entrySet()) {
567
568                    final ResultTable table = tableToPrefix.getKey();
569                    final String prefix = tableToPrefix.getValue();
570
571                    final TableModel model = table.getModel();
572                    final ListModel rowModel = table.getRowModel();
573                    for (int c = 0; c < model.getColumnCount(); c++) {
574                       final String colName = model.getColumnName(c);
575                       for (int r = 0; r < model.getRowCount(); r++) {
576                          String rowName = ((String) rowModel.getElementAt(r))
577                                .toUpperCase();
578
579                          /*
580                           * construct logical name from row name (e.g. Bridges =>
581                           * BRIDGE)
582                           */
583                          if (rowName.endsWith("S")) {
584                             rowName = rowName.substring(0, rowName.length() - 1);
585                          }
586
587                          String key;
588                          if (colName.equals("Total")) {
589                             key = "State-Sum-" + rowName;
590                          } else {
591                             key = "State-Sum-" + colName + prefix + "-" + rowName;
592                          }
593
594                          model.setValueAt(results.get(key), r, c);
595
596                       }
597                    }
598                 }
599
600                 /*
601                  * Detailed table as (Bridges/Gaps/Expansions/Reductions/Other)
602                  */
603                 final ConfusionParameter order[] = new ConfusionParameter[] {
604                       ConfusionParameter.BRIDGE, ConfusionParameter.GAP,
605                       ConfusionParameter.EXPANSION, ConfusionParameter.REDUCTION,
606                       ConfusionParameter.OTHER };
607                 final TableModel model = StateTable.getModel();
608                 final ListModel rowModel = StateTable.getRowModel();
609                 for (int c = 0; c < model.getColumnCount(); c++) {
610                    final String colName = model.getColumnName(c);
611                    for (int r = 0; r < model.getRowCount(); r++) {
612
613                       final String rowName = (String) rowModel.getElementAt(r);
614                       final String baseKey = "State-" + rowName + colName + "-";
615
```

```
616                     /* construct entry (b/g/e/r/o) */
617                     final StringBuffer entry = new StringBuffer();
618                     for (int i = 0; i < order.length; i++) {
619                        if (i > 0) {
620                           entry.append("␣/␣");
621                        }
622                        entry.append(results.get(baseKey + order[i].name()));
623                     }
624
625                     /* ignore diagonal fields */
626                     if (r == c && entry.toString().equals("0␣/␣0␣/␣0␣/␣0␣/␣0")) {
627                        model.setValueAt("-", r, c);
628                     } else {
629                        model.setValueAt(entry.toString(), r, c);
630                     }
631                  }
632               }
633
634               SMSeqCount.setText(Integer.toString((Integer) results
635                     .get("sequences")));
636               SMSeqLength
637                     .setText(Integer.toString((Integer) results.get("length")));
638
639               /* generate sequence type strings */
640               final String[] mismatchBySeq = ((StateMachine) e)
641                     .getMismatchesBySequence();
642               detection1.setText(mismatchBySeq[0]);
643               detection2.setText(mismatchBySeq[1]);
644
645            }
646         }
647
648         /*
649          * Place all carets at zero (so first parts of strings are shown/not
               last)
650          */
651         for (final JTextComponent field : new JTextComponent[] { sequenceId,
652               predictedType, observedType, predicted, observed, detection1,
653               detection2, primary }) {
654            field.setCaretPosition(0);
655         }
656
657         /* Enable save action */
658         saveMenuItem.setEnabled(true);
659      } else {
660         /* Empty: disable save action */
661         saveMenuItem.setEnabled(false);
662      }
663
664   }
665
666   /**
667    * This runs the analysis by comparing all combinations of observed types and
```

```
668      * predicted types for the selected sequences
669      */
670     public void runAnalysis() {
671         analyseButton.setEnabled(false);
672 
673         if (proteinCollection != null && sequenceList.getSelectedValues().length >
                0) {
674 
675             analyseResultSum = new ObservationPredictionMap();
676 
677             for (final Object sequence : sequenceList.getSelectedValues()) {
678 
679                 final ProteinMap pm = proteinCollection.getProteinMap((String)
                        sequence);
680                 final ObservationPredictionMap results = ProcessUtil
681                         .performAllTestsByType(pm, SequenceType.OBSERVATION_TYPES,
682                                 SequenceType.PREDICTION_TYPES);
683 
684                 ProcessUtil.summerizeAll(analyseResultSum, results);
685             }
686 
687             updateTypeLists();
688         } else {
689             JOptionPane.showMessageDialog(getMainFrame(),
690                     "Select directory and proteins first.", "Notice",
691                     JOptionPane.ERROR_MESSAGE);
692         }
693     }
694 
695     /**
696      * Creates a new bar chart view of the differences
697      *
698      * @return Chart
699      */
700     private static ChartPanel createChart() {
701         chartResults = ChartFactory.createBarChart(null,
702                 null, "Score (0-100)", null, PlotOrientation.VERTICAL, true,
703                 true, false);
704         final CategoryPlot plot = (CategoryPlot) chartResults.getPlot();
705         plot.setBackgroundPaint(Color.LIGHT_GRAY);
706         plot.setDomainGridlinePaint(Color.WHITE);
707         plot.setDomainGridlinesVisible(true);
708         plot.setRangeGridlinePaint(Color.WHITE);
709         NumberAxis numberAxis = (NumberAxis) plot.getRangeAxis();
710         numberAxis.setAutoRangeIncludesZero(false);
711         BarRenderer barRenderer = (BarRenderer) plot.getRenderer();
712         barRenderer.setDrawBarOutline(true);
713         barRenderer.setIncludeBaseInRange(false);
714         /* default colours / does not hold if grouping by evaluators */
715         GradientPaint mcc = new GradientPaint(0, 0, Color.GREEN, 0, 0,
716                 new Color(0, 64, 0));
717         GradientPaint q3 = new GradientPaint(0, 0, Color.BLUE, 0, 0,
718                 new Color(0, 0, 64));
```

```
719         GradientPaint sov3 = new GradientPaint(0, 0, Color.RED, 0, 0,
720                 new Color(64, 0, 0));
721         GradientPaint smcm = new GradientPaint(0, 0, Color.ORANGE, 0, 0,
722                 new Color(64, 64, 0));
723         barRenderer.setSeriesPaint(0, mcc);
724         barRenderer.setSeriesPaint(1, q3);
725         barRenderer.setSeriesPaint(2, sov3);
726         barRenderer.setSeriesPaint(3, smcm);
727         return new ChartPanel(chartResults, true);
728     }
729
730     /**
731      * Updated results for making up graph
732      */
733     protected void updateChart() {
734         final CategoryPlot plot = (CategoryPlot) chartResults.getPlot();
735
736         /* Split Pie plot into the different predictors */
737         piePanel.removeAll();
738
739         if (proteinCollection != null && analyseResultSum != null
740                 && !analyseResultSum.isEmpty() &&
                        selectObsTypeGraph.getSelectedIndex() != -1) {
741             final SequenceTranslated obs = (SequenceTranslated) selectObsTypeGraph
742             .getSelectedItem();
743
744
745             final Map<SequenceTranslated, MultipleEvaluationsMap> data =
                        analyseResultSum.get(
746                     obs);
747
748             /* Update Bar plot and Pie plot */
749             final DefaultCategoryDataset ds = new DefaultCategoryDataset();
750             final boolean byEval = chartByEval.isSelected();
751
752             chartResults.setTitle(obs.toString());
753
754             for (SequenceTranslated prediction: new
                        TreeSet<SequenceTranslated>(data.keySet())) {
755
756                 for (AbstractEvaluation eval: data.get(prediction).values()) {
757                     final Map<String, Object> results = eval.getResults();
758                     Double value = null;
759                     String type = null;
760
761                     if (eval instanceof MCC) {
762                         double stacked = ((Double) results.get("MCC-H") + (Double)
                                    results.get("MCC-E") + (Double) results.get("MCC-C")) / 3;
763                         value = (stacked + 1) / 0.02; // calculate percentage
764                         type = "MCC";
765                     } else if (eval instanceof QScore) {
766                         value = (Double) results.get("Q3");
767                         type = "Q3";
```

```
768                 } else if (eval instanceof SOV) {
769                     value = (Double)results.get("SOV3");
770                     type = "SOV3";
771                 } else if (eval instanceof StateMachine) {
772                     value = (Double)results.get("State");
773                     type = "SMCM";
774                 }
775 
776                 if (value != null) {
777                     if (byEval) {
778                         ds.addValue(value, prediction, type);
779                     } else {
780                         ds.addValue(value, type, prediction);
781                     }
782                 }
783 
784             }
785 
786             StateMachine sm = (StateMachine)
                    data.get(prediction).get(StateMachine.class.getName());
787 
788             if (sm != null) {
789                 final Map<String, Object> results = sm.getResults();
790                 /* Create pie charts for SMCM */
791                 final DefaultPieDataset pieData = new DefaultPieDataset();
792 
793                 for (String category: new String[] {"Bridges", "Gaps",
                        "Expansions", "Reductions", "Other"}) {
794 
795                     String catName = category.toUpperCase();
796                     if (catName.endsWith("S")) {
797                         catName = catName.substring(0, category.length()-1);
798                     }
799                     pieData.setValue(category, (Integer) results.get("State-Sum-" +
                            catName));
800                 }
801 
802                   final JFreeChart pieChart =
                        ChartFactory.createPieChart3D(prediction.toString(),
                        pieData, false, true, false);
803                   final PiePlot3D pie = (PiePlot3D)pieChart.getPlot();
804                   pie.setForegroundAlpha(0.6F);
805                   pie.setCircular(true);
806                   piePanel.add(new ChartPanel(pieChart));
807             }
808 
809         }
810         plot.setDataset(ds);
811         plot.getDomainAxis().setLabel(byEval ? "Evaluator" : "Predictor");
812     } else {
813         plot.setDataset(null);
814         chartResults.setTitle((String)null);
815         plot.getDomainAxis().setLabel(null);
```

```
816         }
817         piePanel.validate();
818         piePanel.repaint(); // workaround needed
819     }
820 
821     /**
822      * Private class used to display the folder name in the folder list Only
           displays the
823      * last folder entry.
824      */
825     private static class URLView {
826         public URL url;
827         private String file;
828 
829         public URLView(final URL url) {
830             this.url = url;
831             file = url.getFile();
832 
833             // cut off path unless at end of path
834             final int lastSlash = file.lastIndexOf("/", file.length() - 2);
835             if (lastSlash != -1) {
836                 file = file.substring(lastSlash + 1);
837             }
838         }
839 
840         @Override
841         public String toString() {
842             return file;
843         }
844     }
845 
846 }
```

B.17 dk.sdu.imada.jkkn04.Protein.gui.ResultTable

```
1  /**
2   *
3   */
4  package dk.sdu.imada.jkkn04.Protein.gui;
5 
6  import java.awt.Component;
7  import java.util.Arrays;
8 
9  import javax.swing.JLabel;
10 import javax.swing.JList;
11 import javax.swing.JScrollPane;
12 import javax.swing.JTable;
13 import javax.swing.ListCellRenderer;
14 import javax.swing.ListModel;
15 import javax.swing.ListSelectionModel;
16 import javax.swing.SwingConstants;
17 import javax.swing.UIManager;
18 import javax.swing.border.CompoundBorder;
19 import javax.swing.border.EmptyBorder;
```

```
20 import javax.swing.table.DefaultTableCellRenderer;
21 import javax.swing.table.DefaultTableModel;
22 import javax.swing.table.JTableHeader;
23 import javax.swing.table.TableModel;
24
25 /**
26  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
27  */
28 public class ResultTable extends JScrollPane {
29
30    private static final long serialVersionUID = 1L;
31
32    /**
33     * Default type used inside table fields (determinds alignment)
34     */
35    Class<?> defaultType = String.class;
36    private final JTable table;
37    private Object defaultValue = "";
38    private String[] headers = new String[0];
39    private String[] rowHeaders = new String[0];
40
41    /**
42     * Construct a result table, which also contains row header.
43     */
44    public ResultTable() {
45       super();
46
47       table = new JTable();
48       updateModel();
49
50       setViewportView(table);
51       table.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);
52       table.setCellSelectionEnabled(true);
53       table.getColumnModel().getSelectionModel().setSelectionMode(
54             ListSelectionModel.SINGLE_INTERVAL_SELECTION);
55       ((DefaultTableCellRenderer) table.getTableHeader().getDefaultRenderer())
56             .setHorizontalAlignment(SwingConstants.RIGHT);
57       // setPreferredSize(new Dimension(200,200));
58
59    }
60
61    /**
62     * Class for generating the row header for the table
63     */
64    private class RowHeaderRenderer extends JLabel implements ListCellRenderer {
65       private static final long serialVersionUID = 1L;
66
67       RowHeaderRenderer(final JTable table) {
68          final JTableHeader header = table.getTableHeader();
69          setOpaque(true);
70          setBorder(new
                CompoundBorder(UIManager.getBorder("TableHeader.cellBorder"),
71                new EmptyBorder(0, 20, 0, 20)));
```

```
72              setHorizontalAlignment(CENTER);
73              setForeground(header.getForeground());
74              setBackground(header.getBackground());
75              setFont(header.getFont());
76
77          }
78
79          public Component getListCellRendererComponent(final JList list,
80                  final Object value, final int index, final boolean isSelected,
81                  final boolean cellHasFocus) {
82              setText((value == null) ? "" : value.toString());
83              return this;
84          }
85      }
86
87      /**
88       * Change default value put into the table cells
89       *
90       * @param value new value
91       */
92      public void setDefaultValue(final Object value) {
93          defaultValue = value;
94          updateModel();
95      }
96
97      /**
98       * Change default type for the table cells
99       *
100      * @param type new class type
101      */
102     public void setDefaultType(final Class<?> type) {
103         defaultType = type;
104         updateModel();
105     }
106
107     /**
108      * Sets the column headers for the table
109      *
110      * @param headers column headers
111      */
112     public void setHeaders(final String[] headers) {
113         this.headers = headers;
114         updateModel();
115     }
116
117     /**
118      * Set the row headers for the table
119      *
120      * @param rowHeaders row headers
121      */
122     public void setRowHeaders(final String[] rowHeaders) {
123         this.rowHeaders = rowHeaders;
124         updateModel();
```

```
125     }
126
127     /**
128      * Creates an updated model for the table
129      */
130     private void updateModel() {
131
132         // Cast default value
133         if (!defaultType.isInstance(defaultValue)) {
134             try {
135                 defaultValue = defaultType.getConstructor(defaultValue.getClass())
136                         .newInstance(defaultValue);
137             } catch (final Exception e) {
138                 e.printStackTrace();
139                 System.exit(1);
140             }
141         }
142
143         final Object[][] defaults = new Object[rowHeaders.length][headers.length];
144
145         for (int r = 0; r < rowHeaders.length; r++) {
146             Arrays.fill(defaults[r], defaultValue);
147         }
148
149         table.setModel(new DefaultTableModel(defaults, headers) {
150             private static final long serialVersionUID = 1L;
151             private final Class<?> type = defaultType;
152
153             @Override
154             public Class<?> getColumnClass(final int columnIndex) {
155                 return type;
156             }
157
158             @Override
159             public boolean isCellEditable(final int rowIndex, final int columnIndex)
                    {
160                 return false;
161             }
162         });
163
164         // row header
165         final JList rowHeader = new JList(rowHeaders);
166         rowHeader.setFixedCellHeight(table.getRowHeight());
167         rowHeader.setCellRenderer(new RowHeaderRenderer(table));
168         rowHeader.setOpaque(false);
169         setRowHeaderView(rowHeader);
170
171     }
172
173     /**
174      * Return the currently active model
175      *
176      * @return table model
```

```
177      */
178     public TableModel getModel() {
179        return table.getModel();
180     }
181
182     /**
183      * Returns the list model for the row header of the table
184      *
185      * @return list model
186      */
187     public ListModel getRowModel() {
188        return ((JList) getRowHeader().getView()).getModel();
189     }
190
191  }
```

B.18 dk.sdu.imada.jkkn04.Protein.cmd.CompareAllCombinations

```
1   package dk.sdu.imada.jkkn04.Protein.cmd;
2
3   import java.io.IOException;
4   import java.util.Map;
5
6   import dk.sdu.imada.jkkn04.Protein.data.ProteinMap;
7   import dk.sdu.imada.jkkn04.Protein.data.ProteinMapCollection;
8   import dk.sdu.imada.jkkn04.Protein.data.SequenceTranslated;
9   import dk.sdu.imada.jkkn04.Protein.data.SequenceType;
10  import dk.sdu.imada.jkkn04.Protein.tests.AbstractEvaluation;
11  import dk.sdu.imada.jkkn04.Protein.util.ProcessUtil;
12  import dk.sdu.imada.jkkn04.Protein.util.ProcessUtil.MultipleEvaluationsMap;
13  import dk.sdu.imada.jkkn04.Protein.util.ProcessUtil.ObservationPredictionMap;
14
15  /**
16   * Commandline program to compare all combinations of predictors and observed
        sequences of
17   * a given type
18   *
19   * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
20   */
21  public class CompareAllCombinations {
22
23     /**
24      * Main method - this runs a comparition of all combinations of predictors and
25      * observed sequences
26      *
27      * @param args Not used
28      */
29     public static void main(final String[] args) {
30
31        try {
32           final ProteinMapCollection pmc = new ProteinMapCollection("cytokines/");
33
34           final ObservationPredictionMap sum = new ObservationPredictionMap();
35           /* list each protein for every file by protein */
```

```
36          for (final String proteinId : pmc.getProteinIdentifiers()) {
37
38              final ProteinMap pm = pmc.getProteinMap(proteinId);
39              /* first description */
40              final String description =
                    pm.values().iterator().next().getDescription();
41              System.out.println("Processing for \"" + description + "\":");
42              final ObservationPredictionMap results = ProcessUtil
43                      .performAllTestsByType(pm, SequenceType.OBSERVATION_TYPES,
44                              SequenceType.PREDICTION_TYPES);
45
46              printResults(results);
47              System.out.println();
48              ProcessUtil.summerizeAll(sum, results);
49          }
50
51          System.out.println("SUMMARIZE:");
52          printResults(sum);
53
54      } catch (final IOException e) {
55          e.printStackTrace();
56      }
57  }
58
59  private static void printResults(final ObservationPredictionMap results) {
60      for (final Map.Entry<SequenceTranslated, Map<SequenceTranslated,
            MultipleEvaluationsMap>> observed : results
61              .entrySet()) {
62          final SequenceTranslated observedType = observed.getKey();
63          for (final Map.Entry<SequenceTranslated, MultipleEvaluationsMap>
                prediction : observed
64                  .getValue().entrySet()) {
65              final SequenceTranslated predictionType = prediction.getKey();
66              for (final Map.Entry<String, AbstractEvaluation> evaluation :
                    prediction
67                      .getValue().entrySet()) {
68
69                  // ((StateMachine)evaluation.getValue()).printDebug();
70
71                  System.out.println(String.format("%s -> %s (%s):", observedType
72                          .name(), predictionType.name(), evaluation.getKey()));
73                  for (final Map.Entry<String, Object> result : evaluation.getValue()
74                          .getResults().entrySet()) {
75                      System.out.println(String.format("   %s = %s", result.getKey(),
76                              result.getValue().toString()));
77                  }
78              }
79          }
80      }
81  }
82
83  }
```

B.19 dk.sdu.imada.jkkn04.**Protein.cmd.ListAllProteins**

```
1  package dk.sdu.imada.jkkn04.Protein.cmd;
2
3  import java.io.IOException;
4
5  import dk.sdu.imada.jkkn04.Protein.data.Protein;
6  import dk.sdu.imada.jkkn04.Protein.data.ProteinMap;
7  import dk.sdu.imada.jkkn04.Protein.data.ProteinMapCollection;
8  import dk.sdu.imada.jkkn04.Protein.data.SequenceTranslated;
9  import dk.sdu.imada.jkkn04.Protein.data.SequenceType;
10 import dk.sdu.imada.jkkn04.Protein.data.Protein.TranslationType;
11
12 /**
13  * Commandline program to lists all protein sequences after translation
14  *
15  * @author Kristian Kræmmer Nielsen <jkkn@jkkn.dk>
16  */
17 public class ListAllProteins {
18
19    /**
20     * Main method - lists all protein sequences after translation
21     *
22     * @param args Not used
23     */
24    public static void main(final String[] args) {
25
26       try {
27          final ProteinMapCollection pmc = new
               ProteinMapCollection("chymotrypsins/");
28          // ProteinMapCollection pmc = new
29          // ProteinMapCollection("cytokines/");
30
31          /* list each protein for every file by protein */
32          for (final String proteinId : pmc.getProteinIdentifiers()) {
33             // String proteinId = "1INR__";
34
35             final ProteinMap pm = pmc.getProteinMap(proteinId);
36             for (final SequenceType seqType : SequenceType.values()) {
37                for (final TranslationType transType : TranslationType.values()) {
38                   final SequenceTranslated st = new SequenceTranslated(seqType,
39                         transType);
40                   if (pm.containsKey(st)) {
41                      final Protein p = pm.get(st);
42                      final String sequence = p.getSequenceType().isObservation()
                           ? p
43                                .getCleanSequence()
44                           : p.getSequence();
45                      System.out.println(String.format("%-34s %s", p
46                            .getIdentifier()
47                            + " ("
48                            + p.getSequenceType()
49                            + "/"
```

```
50                                  + p.getTranslationType() + "):", sequence));
51                          }
52                      }
53                  }
54
55                  System.out.println();
56              }
57
58          } catch (final IOException e) {
59              e.printStackTrace();
60          }
61
62      }
63
64  }
```