The k-best paths in Hidden Markov Models. Algorithms and Applications to Transmembrane Protein Topology Recognition

by

Daniil Golod

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2009

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Traditional algorithms for hidden Markov model decoding seek to maximize either the probability of a state path or the number of positions of a sequence assigned to the correct state. These algorithms provide only a single answer and in practice do not produce good results. The most mathematically sound of these algorithms is the Viterbi algorithm, which returns the state path that has the highest probability of generating a given sequence. Here, we explore an extension to this algorithm that allows us to find the k paths of highest probabilities. The naive implementation of k best Viterbi paths is highly space-inefficient, so we adapt recent work on the Viterbi algorithm for a single path to this domain. Out algorithm uses much less memory than the naive approach. We then investigate the usefulness of the k best Viterbi paths on the example of transmembrane protein topology prediction. For membrane proteins, even simple path combination algorithms give good explanations, and if we look at the paths we are combining, we can give a sense of confidence in the explanation as well. For proteins with two topologies, the k best paths can give insight into both correct explanations of a sequence, a feature lacking from traditional algorithms in this domain.

Acknowledgements

I would like to thank my supervisor, Daniel G. Brown, for all the help with this work. I would also like to thank Jakub Truszokowski for helpful discussion. Finally, I thank my readers Therese Biedl and Pascal Poupart for their helpful comments.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Hidden Markov Models (HMMs) are an analysis tool used in a variety of areas, including bioinformatics. One of their main uses is assigning annotations to long sequences to denote features of that sequence. The features of the sequence are usually encoded into the HMM as labels for states: while every state must have a single label, some labels may be assigned to multiple states. Usually decoding algorithms, such as Viterbi [8], posterior decoding [8], and the 1-best algorithm [14], provide a single (presumably correct) annotation of the sequence.

Here, we explore an alternative way to provide good annotations of sequences: finding the k-best Viterbi paths and using them to find annotations. We introduce a memory-efficient algorithm for finding k-best paths in Chapter 3 based on a recent compressed tree approach for the Viterbi algorithm [24]. We implement this algorithm and run it on a standard HMM model from the transmembrane protein topology predictor called Phobius [12]. We show that in practice the new algorithm is significantly more memory-efficient, without incurring a large time penalty (Section 5.2). We additionally investigate another way to potentially improve memory usage, which involves detecting coalescence points. A coalescence point is a point in the sequence where a given symbol is guaranteed to be emitted from a particular state. We present algorithms for detecting coalescence points for both one path and the k-best paths (Chapter 4).

We investigate the amount of information in the k-best paths by looking at how often the k-best paths contain at least one path that is correct. Such situations are more common than the number of correct predictions for the original 1-best algorithm for the Phobius model (Section 5.1). Then we show how to extract good annotations from groups of predictions in Chapter 6 and conclude that using simple ways to extract prediction we can do better then Viterbi algorithm, but worse than the 1-best decoding method.

Finding a good annotations is not the only use of the k-best paths: we can use them to extract other information about the sequence. Probably the most useful additional information that we have investigated is the confidence in the prediction. We show that we can access confidence in two different ways. In cases when

all of the predicted annotations have the same number of helices we have higher confidence than in the cases when multiple different numbers of predicted helices is found. Additionally we show that there is a correlation between the amount of total probability that the top k-best paths occupy and how often we predict correctly (Sections 6.2, 6.1). Here, having larger total probability corresponds to higher confidence.

We also investigate whether different predictions or groups of predictions can tell us something about correct alternative explanations (Section 6.4) and show that we sometimes can predict biologically meaningful alternative annotations for a special subset of transmembrane proteins called dual topology proteins. Dual topology proteins are the proteins for which it can be shown that they assume one of two different topologies within a cell. Here, we show that sometime we can predict both topologies in a meaningful manner.

Chapter 2

Background and previous work

Before describing the k-best algorithm and its applications we need to introduce some background material. We first present Hidden Markov Models (HMMs) together with some common algorithms designed to be used with them. Next, we give a short introduction to transmembrane proteins, transmembrane protein topology and ways to predict this topology using sequence information.

2.1 Hidden Markov Model

HMMs are a probabilistic model for sequences that is often used for recognizing features of a sequence. HMMs are a common tool in bioinformatics due to their ease of design and their fast and mathematically-justified training and decoding algorithms. Here we introduce the overall structure and two decoding algorithms for HMMs, as well as ways to calculate the probability of a sequence given a model (called the "forward algorithm"). We also present an algorithm that allows one to find the probability of a labelling given a model and sequence.

A hidden Markov model is a probabilistic generative automaton that creates a sequence, over a finite alphabet, Σ, while traversing stochastically through a finite set of states. Let m be the number of states in the model. An HMM is defined by a collection of parameters: an initial probability vector I, a set of transition parameters a_{ij}, a set of emission parameters b_i, and, potentially, a final probability vector F. The initial probability vector I identifies the probability of starting in each state (from 1 to m). The final probability vector F, for each state i, defines the probability that the HMM will stop emitting during a time step in state i. The transition probability a_{ij} gives the probability that the model is in state j at step t of its execution if it is in state i at step $t-1$, assuming $t \geq 1$. The emission probability $b_i(e)$ gives the probability that the model emits the symbol e at steps when it is in state i. Sometimes when designing an HMM we want to ensure that two or more states have exactly the same probability distribution for emissions; such emissions are called *tied emissions*.

Additionally an HMM can have a set of labels L associated with it. In such cases each state is assigned a label (*i.e.* there is a function $f : (1 \dots m) \to L$ which returns a label for each state). Sometimes we may use $L(v)$ to denote the label of state v. A path is a sequence of states through the HMM; for a path π, let $\pi(t)$ be the state that emits the t-th symbol of a sequence. A labelling of a sequence of symbols (emitted sequence) is a sequence of labels, one per symbol. Each path though the HMM gives a single labelling of a sequence. However, for some HMMs, a single labelling may be generated using multiple paths.

Without loss of generality, the initial probability vector I can have all of its mass at a single state, meaning that there is a designated start state in the model. If F is not defined, then the HMM generates sequences of infinite length. However, if we want to generate a finite length sequence and define the final probability vector F, then we can generate an equivalent HMM with a single final state, sometimes called an *end state* or *final state*.

An HMM can be seen as a biased random walk on a directed graph with m nodes that emit symbols: we set an arc from state i to j when a_{ij} is positive, with weight a_{ij}. Let $d(i)$ be the outdegree of state i. The HMM is called dense if the underlying graph is dense (has $\Theta(m^2)$ edges), and the HMM is called sparse when the underlying graph is sparse (just $\Theta(m)$ edges). In many biological applications the HMMs used are sparse [12, 25, 23].

The HMM emits symbols, as described above: the ith symbol emitted has the probability distribution for state $\pi(i)$. If we let the model run for n steps, the output sequence is $X = x_1 x_2 \dots x_n$. Let x_t^u be the substring $x_t \dots x_u$. For standard HMMs, the model usually has few states, compared to the lengths of the sequences, so a runtime that is "linear" is linear in n, not necessarily in m. In practice, HMMs used for biological sequence analysis might have a couple hundred states [12, 18], so a runtime or space usage cubic in this parameter may be costly, particularly for space usage.

2.1.1 Viterbi algorithm

When we decode an HMM, we assign states or labels to the symbols of a sequence presumed to have been generated by it. The Viterbi algorithm decodes an HMM by finding the optimal path through the model for a given sequence. It computes a single path maximizing the joint probability of the state path and the sequence. The Viterbi algorithm is a straightforward dynamic programming procedure, as the maximum probability path has maximum probability subpaths. As with any dynamic programming problem, the Viterbi algorithm has two distinct parts. The first part involves calculating probabilities $v[t, i]$ of the optimal path for v_1^t ending in state i. The optimal subpath rule lets us compute this by $v[t, i] = \max_j(v[t - 1, j]a_{ji}e_i[x_t])$. We can either use a full nm-size matrix to store the probabilities or just store the information about the previous column, using $\Theta(m)$ space. The runtime for this part of the Viterbi algorithm is $O(n \sum_i d(i))$. The calculation of

the probabilities step is the bottleneck step of the whole Viterbi algorithm, making the Viterbi runtime $O(n \sum_i d(i))$.

The second part of Viterbi involves keeping track of the decisions made. A classical implementation of the Viterbi algorithm uses a $\Theta(nm)$-size matrix of back pointers. This matrix gives the predecessor state for each (state, position) pair. The classical approach is very memory intensive, thus alternative approaches have been developed to heuristically reduce space without asymptotic runtime decrease. We present one such approaches now.

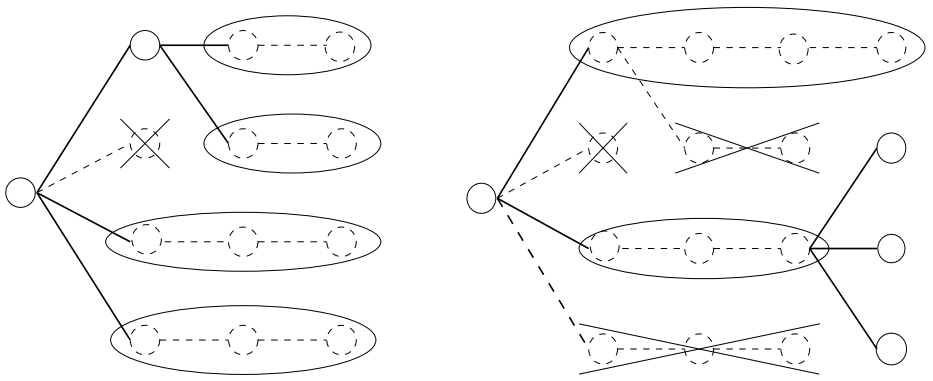
It should be mentioned that if one is willing to increase runtine as a trade off for saving space several approaches are available. Checkpointing can be used to decrease space to $\Theta(\sqrt{n})$ at a cost of doubling the runtime. Further refinement of checkpointing can decrease space usage to $\Theta(n^{1/L})$ at the cost of factor of L slowdown [10]. Alternatively one can use the divide and conquer approaches to achieve $\Theta(\log n)$ space in exchange for a factor $\log(n)$ slowdown.

2.1.2 Compressed tree approach to Viterbi algorithm

The compressed tree approach reduces the space usage required by the back pointers from the $O(nm)$ worst case closer to $O(m \log n)$ in practice (in theory this approach may not actually reduce space usage). The implementation we describe below has the same overall effect as the original compressed tree [24], but is simpler in its data structure and operations.

First we describe the mental construction of the compressed tree, and then we will talk about efficient data structure and efficient on-line maintenance. We can think of the compressed tree as a final result of the following operations. First, we create an m by n grid of nodes where each node corresponds to a cell of the Viterbi matrix (a position-state pair). We think of each column as m cells corresponding to the m paths ending at a particular sequence position (one cell per state). Then we create an edge between node v_t of column t and node u_{t+1} of column $t+1$ if the Viterbi path to the position-state pair corresponding to the node $(t+1, u_{t+1})$ has state v at position t. Thus all edges connect consecutive columns (corresponding to sequence positions) of the grid; we will order the graph so that cells in column t point to cells in column $t-1$. The nodes that correspond to the beginning of the sequence are at the beginning of the graph, which will be the location of the root of the tree, and the nodes corresponding to the end of the sequence are the end of the graph (leaves of the tree). Because there is exactly one previous Viterbi path state for each position-state pair each node has exactly one parent. Next we remove all nodes which are not reachable from the end row of the graph. As described before, we can assume that the HMM has a single start state, so the removal will leave exactly one node corresponding to the beginning of the sequence, the root of the tree.

The tree described in the previous paragraph records all the Viterbi paths though the HMM up to each sequence location. To compress the tree, we compress



(a) A sample compressed tree

(b) compressed tree in (a) extended by a level

Figure 2.1: An example of extending compressed tree by one level, showing underlying virtual structure. The dotted items are virtual, while solid ones can be found in actual compressed tree. The node on the extreme left denotes the at before sequence start. The crossed items indicate nodes removed during processing.

all nodes with exactly one parent. An example of a compressed tree (oriented horizontally) is given in Figure 2.1a. After compression, each node in the compressed tree corresponds to the sequence of states which emits a particular substring of the given string, found in a potential Viterbi path. Note that during the process of construction of the compressed tree, there are always m potential Viterbi paths, one for each state, which need to be maintained.

Now we will describe a possible implementation of the compressed tree. The data structure consists of nodes linked to each other using pointers. Each node stores the sequence of states it corresponds to; linked lists are an appropriate structure for this, since they can be concatenated in $O(1)$ time. We keep track of m current children and of the root node. We also need to solve the problem of tying the probabilities of the path to the actual paths themself. A simple way to do this is to store pointers to children (there exists only one leaf per state of HMM, thus an m-sized vector will suffice) as a vector of values and enforce that the index in the vector of probabilities corresponding to state j is the same as an index of pointer to node corresponding to state j in the vector of children pointers.

The updating process is done on line as we advance through the sequence X. If we have created the tree for x_1^t, we must update it to include x_{t+1}. This involves adding leaves to the tree for each possible state at position $t+1$, connecting them with the nodes for their predecessor in the Viterbi path (which are available as the previous set of leaves), identifying paths in the tree that no longer reach a new leaf, deleting those paths, and compressing the internal nodes with outdegree 1 in the updated tree (Figure 2.1b).

First, we identify states at position t that can be found in only one optimal path ending at position $t+1$; these are nodes at level t with exactly one child in the new level. These nodes are removed, and the state path from the deleted node

is prepended to the state of the new leaf.

Then we finish updating the tree. For each leaf from position t which has no children among the new leaves at position $t+1$, in sequence, we delete that node and examine its parent. If its parent now has only one child, we delete the parent node, and concatenate (using the $O(1)$ list concatenation method) the paths for the two joined nodes into its remaining child. Later, we will call a very similar operation a *merge* of two nodes. After each update, the only nodes left in the tree with one child are the parents of leaves from position t that we have not yet updated. Note that because deleting a child happens on the valid compressed tree (due to merging) each parent has two or more children, thus removing the case of cascading deletes.

At the end of this procedure, what remains is a new valid tree: it has m leaves, corresponding to the m model states, each of which corresponds to a possible value of $\pi(t+1)$. The algorithm does at most m node deletions and at most m node merges (compressions); both require $O(1)$ time, for a total of $O(m)$, so the data structure update time is asymptotically less than the Viterbi path probability calculation.

The added asymptotic runtime for this procedure, then, is no more than the Viterbi calculation; in a sparse HMM, they may be on the same order. The algorithm can still require $O(nm)$ space, if the paths do not meet, but experiments by the original authors [24] suggest that in practice the space requirement is more like $O(m \log n)$ when we do not count the length of the path in the root of the tree.

A variation on this approach was independently presented by Keibler *et. al* [13]; these authors call it the Treeterbi algorithm. Treeterbi does not perform tree compression (though it still does identify when all paths share a common prefix).

2.1.3 1-best algorithm

The 1-best algorithm is a heuristic decoding algorithm presented by Krogh [14] in an attempt to address several drawbacks of the Viterbi algorithm. More precisely Krogh was interested in finding the most probable labelling of a sequence for a particular HMM. Finding the most probable labelling for a sequence is NP-hard [5], so the algorithm Krogh presented is a heuristic to approximate it.

The idea behind this approach is for each state to keep a list of labellings which are consistent with the last symbol of the sequence, so far, being emitted from the state (the state with this label could have produced last symbol). The probability of each labelling is kept and updated with each iteration. Let each labelling for each state be called a hypothesis, and let us denote it with h. Note that zero probability hypotheses (for example ones where a hypothesis for state v does not have the label of v as the last symbol) can be disregarded. The algorithm itself (as described by Krogh) is as follows:

1. Propagate the empty hypothesis forward to all states (sequence position $i =$

1). At this stage number of hypotheses is $|L|$ and their probability for each state v is $I(v)b_v(x_a)$ and for all non-zero probability states $\pi_1 = v$.

2. Propagate the hypotheses forward yielding L (size of the label set) new hypotheses for every old one. For each state v the probability of the new hypothesis hY, where h is a preexisting hypothesis and Y is a label.

$$\Pr(hY, \pi_{i+1} = v) = \begin{cases} (\sum_u a_{uv} \Pr(h, \pi_i = u)) b_v(x_{i+1}) & \text{if the label of } v \text{ is } Y \\ 0 & \text{otherwise} \end{cases}$$

3. In each state, choose the hypothesis with the highest probability. Discard all hypotheses that were not chosen in any state. If we have not reached the end of the sequence, go to step 2.

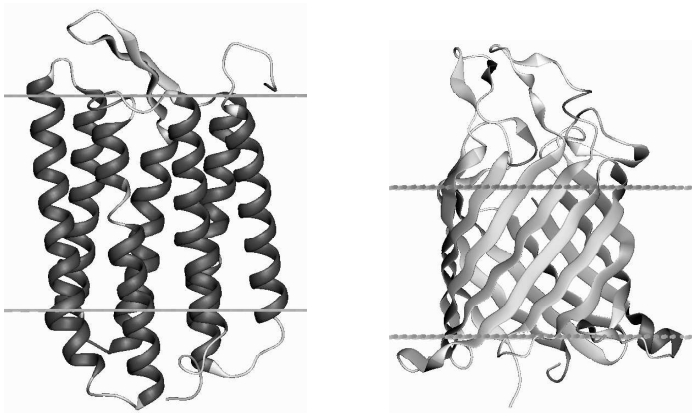
4. Find the final probability for each hypothesis by summing over the states (that admit the same hypothesis) and return the hypothesis with the highest probability.

Implemented as described this algorithm does not provide the most probable labelling in cases when at some location of the sequence the most probable overall labelling does not correspond to overall best labelling for that state-location pair. In other words, this algorithm suffers from local maxima. It is worth noting that the CPU usage (and potentially space usage) for this algorithm is very large. To reduce the resources used one needs to use threshold to disregard very low probability hypotheses, which in turn can cause problems with finding a "correct" answer.

2.1.4 Forward algorithm and the probability of a labelling

Finding the most probable path for a given sequence is only one of the possible tasks that we can perform on an HMM. An alternative task is finding the probability of the sequence given the model. The dynamic programming algorithm that performs the task is called the forward algorithm and is described, for example, in [8]. The main idea behind the algorithm is that for each position, in order, we calculate, for each state, the probability of emitting the sequence so far and ending in the particular state. For a single state u, this can be done by adding the forward probabilities, multiplied by the respective transition and emission probabilities, for all the possible immediate predecessor states. If a state does not emit the current symbol in the sequence, the forward probability for the state at the sequence position is zero.

Another probability which interests us is the probability of a given labelling, λ, for the sequence. We can find the probability a labelling using a slight variation of the forward algorithm: for each state u at position i, record the probability if and only if the label of u is λ_i, otherwise, zero probability is recorded.



(a) α-helical transmembrance protein. PDB ID 1VGO

(b) β-barrel membrane protein. PDB ID 1TLY

Figure 2.2: Examples of different classes of transmembrane proteins. The pictures are taken from [1].

2.2 Transmembrane proteins and their topology

Transmembrane proteins are a biologically important subset of proteins that composes about a quarter of all proteins in the cell [20]. As the name implies, transmembrane proteins are proteins which have domains spanning a membrane. The reason for the abundance of transmembrane proteins is fairly simple: in the cell the membrane is used as a semi-permeable barrier which serves the dual function of keeping everything inside of the cell in and protecting a cell from outside influence, and, in eukaryotes, to separate parts of the cell from each other. However, the cell needs to communicate and exchange chemicals with the outside world (potentially between compartments), and transmembrane proteins are used for that purpose. Transmembrane proteins are involved in signal transduction, ion transmission and bioenergetics, and many other processes.

There are two types of transmembrane proteins: α-helices and β-barrels (Figure 2.2). The difference between two classes can be reduced to the way that their transmembrane domains are structured: α-helices use hydrophobic helices to traverse the membrane, while β-barrels use β-sheets woven in a pore like manner for membrane traversal. Due to their structure, the signals that can be used to predict β-barrel proteins are non-local, making prediction significantly harder [2, 3]. For the rest of the thesis we will be talking about α-helical proteins exclusively, unless otherwise specified. Thus any mention of a transmembrane protein will refer to an α-helical transmembrane protein, and a transmembrane helix, or just a "helix", will refer to a membrane-spanning α-helix of such a protein.

The term transmembrane protein *topology* refers to the number of the transmem-

brane segments, their location, and their sidedness (which side of the membrane each non-membrane amino acid segment is on). There are several ways in which knowing transmembrane protein topology can help. First, knowing protein topology can be used as a stepping stone to prediction of the three dimensional structure of the protein. Second, knowing transmembrane protein topology can help to predict protein function, as knowing which side of the protein active domains (or even simply long non-membrane segments) are can provide clues to protein function.

Multiple features of transmembrane proteins can be used to help recognize their topology; let us talk about some of them. The first, and the most prominent feature, are the transmembrane helices themselves. Each transmembrane helix is a sequence of 15 to 30 [20] consecutive hydrophobic amino acids. In globular (non-transmembrane) proteins such long stretches of hydrophobic residues are very rare, therefore seeing such a stretch gives a good indication that this particular proteins is transmembrane. The second distinguishing characteristic of a transmembrane helix is the gradient of hydrophobicity among its residues. The amino acids closer to the centre of the helix are much more likely to be strongly hydrophobic then ones closer to the borders of the helix [11]. The third feature of transmembrane proteins, one that helps to pinpoint the locations of the transmembrane helix boundaries, is called the *aromatic ring* property: transmembrane helices often have aromatic amino acids at the boundaries of the helix (the areas where transmembrane domain ends and non-membrane domain starts) [20]. The fourth feature is the *positive-inside rule*: there is a propensity to have high concentration of positively charged Lysines (K) and Arginines (R) on the inside (cytoplasmic side) of the protein. Using the positive-inside rule helps to determine the sidedness of the protein: which non-membrane segments are inside or outside of the cell? For more in-depth discussion of the features mentioned and some other characteristics of transmemebrane proteins one should consult Rapp [20].

Since transmembrane proteins are very hard to crystallize, computational methods are often used for topology prediction. There are several different protein topology predictors which should be mentioned. Some of them use neural networks like TOPPRED [6], MEMSAT [16], and PHDhtm [22]. Some use HMMs as their tool of choice, like TMHMM [15], HMMTOP [27], and Phobius [12]. More recently, a combined method SPOCTOPUS [28] was introduced.

There is a very large variability in the definitions of correctness for predicting topology of the transmembrane proteins. In some cases, predicting the right number of helices and sidedness is considered correct. In other cases the correctness takes into account the overlap between the correct and predicted helix locations, and sometimes the locations of the borders are used as a correctness criteria. Finally, sometimes the per-residue correctness is judged: what fraction of residues are correct? Such variability, coupled with the fact that the data set of transmembrane proteins with known 3D structures is relatively small, results in wide variety estimates of overall correctness rates of predictions. On some data sets, with lax correctness measures, accuracy can reach 95% [15, 17, 7]. On the other hand, when small data set is taken into account and correctness is measured strictly, the

estimates can fall to 50% [20, 7] or even lower.

Of the predictors mentioned above, Phobius can be distinguished as one HMM based predictor which tries to be good at separating transmembrane proteins from globular ones.

2.2.1 Phobius

Phobius [12] is an HMM-based transmembrane topology predictor. As stated above it is designed to reduce the amount of false positives, while still predicting structure well. This is achieved by incorporating the prediction of the signal peptide into topology prediction. A signal peptide is a segment of a protein (always located at the beginning of the protein), which indicates that the amino acids following it are to be located on the outside of the membrane. In its structure, a signal peptide is similar to a structure of transmembrane helix, so they may be confused with each other. However, if the signal peptide is predicted correctly, it helps to predict the sidedness of the transmembrane protein, as the model knows where to place the first non-signal amino acid of the protein.

Now let us describe the Phobius model. It is a concatenation of two well known HMM models for topology prediction (TMHMM) and signal peptide recognition (Signal-P). It consists of 188 states, the exact topology of which can be found in Figure 2.3. The model has a very large number of tied emission parameters, at least partially with the purpose of avoiding overfitting. In the original paper the 1-best algorithm (Section 2.1.3) is used for decoding of the sequences.

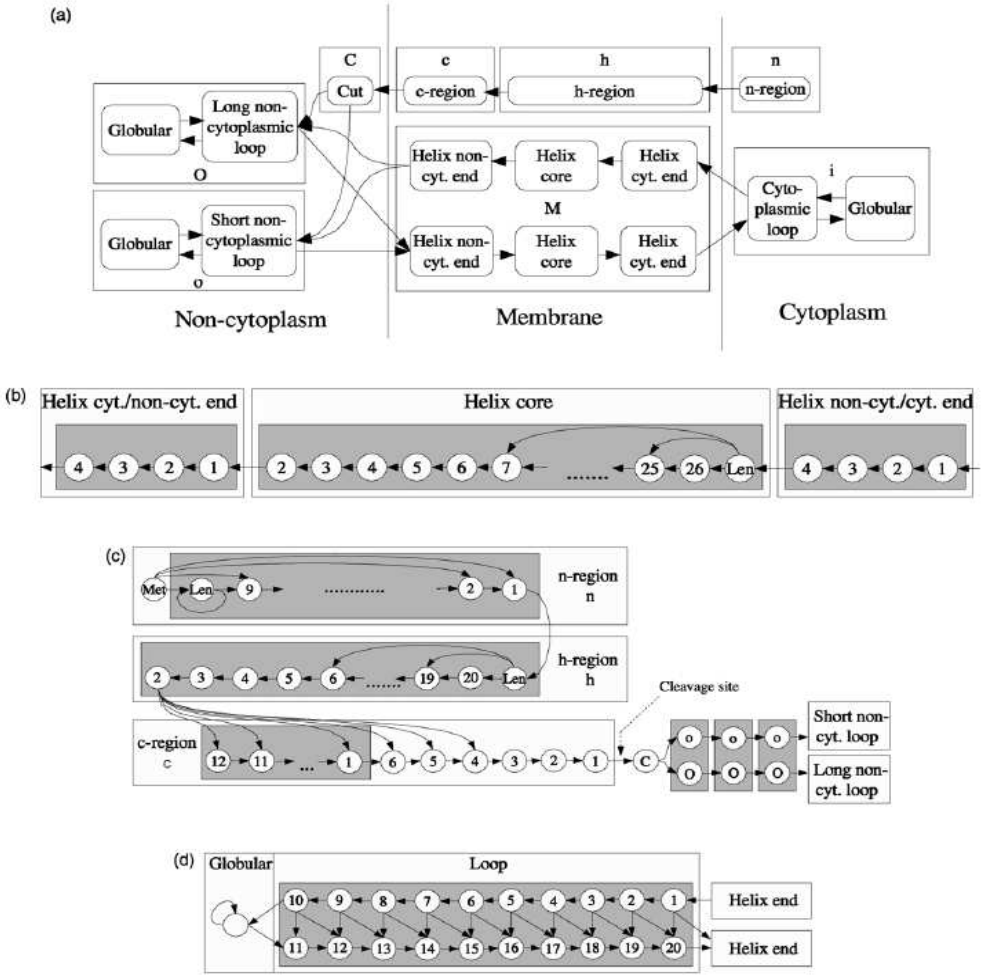


Figure 2.3: Phobius Model, taken from (Kall *et al.*) [12]. States with labels n, h, c, and C correspond to signal peptide prediction. States with labels o, O, M, and i correspond to transmembrane topology prediction.

Chapter 3

K-best Viterbi algorithm

Here we will describe algorithms and data structures necessary for computing the k-best paths in an HMM. The Viterbi algorithm and the procedures for finding the k-best paths consist of two different calculations: computing the probabilities of the best path (or k paths) to every state for every prefix x_1^i of a given sequence X, and also storing the back pointers necessary to reconstruct those paths. We will first describe how to compute probabilities for the k-best Viterbi path. Next, we will present two different ways to store back pointers.

3.1 Computing the probabilities

Consider the computation of the k-best probabilities to each state at position i. The key observation is that the k highest probability paths for s_1^i that end in state v_i all have to be from the k-best paths to each of the states for the sequence from s_1 to s_{i-1}. Suppose this were not true, and π, one of the k-best paths for $s_1 \ldots s_i$ ended in state v_i and was in state v_{i-1} at the previous position, but was *not* among the k-best paths to state v_{i-1}. Then all of the k-best paths to state v_{i-1}, concatenated with v_i would have higher probabilities than π, contradicting that π was one of the k-best paths for s_1^i ending in v_i. The k-best probabilities can be found by finding all possible ways to get to state v (there are at most $k * Pred(v)$ of them) and then picking k with highest probability.

Alternatively, with the same result, we keep a sorted list of the k-best path probabilities to each state at position $i-1$. Then, if we are considering a state a whose possible predecessors in the HMM are $Pred(a)$, we can find the k-best probabilities for state a at postion i by performing an operation very similar to the first k steps of a $|Pred(a)|$-way merge sort. The Viterbi probability of the ℓth best path to state v is:

$$\max_{c \in Pred(v)} \left(\max_{\text{path } k \text{ to } c \text{ not used in } l-1 \text{ best paths}} (\Pr(k) a_{cv} b_v(s_i)) \right).$$

By keeping index of the last path used in the prediction, for each list of the k most probable paths for each possible predecessor state to v, we can easily compute the k-best paths probabilities in $O(km)$ time. It is an interesting algorithmic question whether this can be sped up heuristically, since all paths to state v that were in state c at position $i-1$ will have their probabilities multiplied by the same constant, $a_{cv}b_v(s_i)$.

This calculation, then, takes k times the cost of a standard Viterbi calculation (a naive implementation might require k^2 times the cost), and $\Theta(mk)$ space. We note that this approach has been used by speech recognition experts as long ago as 1993 [19].

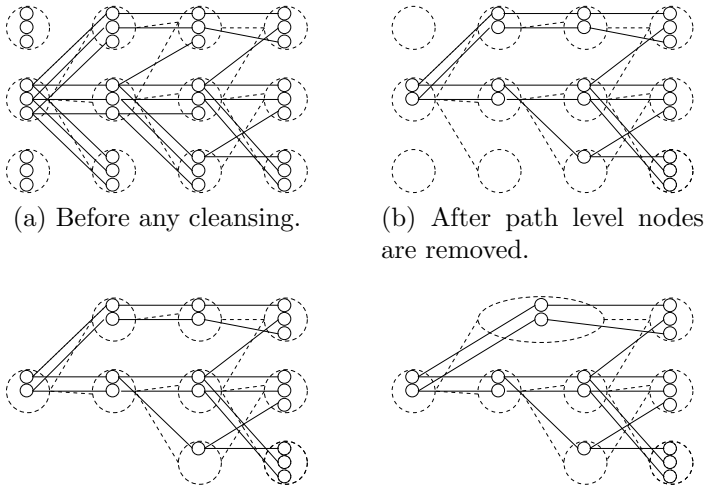
3.2 Naive Viterbi path storage

There is a natural way to extend the classical (matrix based) Viterbi implementation to finding the k-best paths in the HMM: store the k highest-scoring paths for each state-position pair. This observation leads to a Viterbi-like algorithm whose runtime is k times the runtime of the Viterbi algorithm, and which requires $\Theta(kmn)$ storage for the backtracking matrix and $\Theta(km)$ storage for the moving probability front. Unfortunately, the space requirements of this method make it infeasible for finding the k-best paths for large values of k on substantial HMMs for long sequences; see the experimental results in Chapter 5.

3.3 Tree based path storage

Now we will describe different data structure used to store the k-best paths. First, we will give a basic idea behind the data structure and, after that, we will describe how to implement it efficiently. We can mentally construct a k-best path tree by the following algorithm. For each state-position pair create one node, which we will call a *vertex level* node. We can arrange the vertex level nodes in a grid, with rows corresponding to positions in the sequence, in a way very similar to how the compressed tree was constructed. We define beginning and end vertex level nodes according to the sequence positions. In each vertex level node, we create k nodes representing the k-best paths to that vertex level node, and we call those nodes the *path level* nodes. For convenience, we store path level nodes in order of the sorted probabilities of their path, which allows for easy mapping between path level nodes and the k-best paths probabilities. We have already shown that only k probabilities for each position of the previous row are needed to calculate k best paths, therefore k path level nodes are sufficient at each vertex level node.

Next, add an edge between a path level node v_a in vertex level node v and a path level node u_b in vertex level node u, if u is on the column previous to v (u and v correspond to consecutive symbols in the string, with u corresponding to position



(a) Before any cleansing.

(b) After path level nodes are removed.

(c) After vertex level nodes are removed.

(d) Final data-structure (after node merging) .

Figure 3.1: K-best tree construction visualization. The solid lines represent path level nodes and edges, while dotted lines represent vertex level nodes and edges.

before v) and the a-th best path to the position-state pair (i, v) goes though the b-th best path for $(i-1, u)$ (Figure 3.1a). Note that always $a \leq b$. An edge between two vertex level nodes exists if any of their path level nodes are connected by an edge.

Next we describe how to make the storage of this structure more efficient by removing and merging nodes. First, we remove all the path level nodes that cannot be reached from the path level nodes in the vertex level nodes corresponding the the last position in the sequence (Figure 3.1b). Then, we remove all vertex level nodes which do not have path level nodes associated with them (Figure 3.1c). Finally, we merge all vertex level nodes that have a single vertex level edge connecting them in their row levels (Figure 3.1d). The exact conditions in which vertex merges happen are described in the next section.

3.3.1 Updating the path structure

For each path level node, we need to store the pointer to its vertex level node, a parent pointer, which links to the path node exactly one step closer to the root, and the number of children the path node has, as non-leaf path nodes should be deleted when they have zero children. The vertex level node stores the list of states associated with it (which is the actual state path through the model for that vertex level node), the list of at most k path nodes that it includes, pointers to its children, and the number of vertex nodes that are its parents. When needed, we can find the parents of a vertex node by traversing though its associated path nodes and requesting their parent's vertex nodes. A vertex level node is deleted if it has no

path level nodes associated with it. It is merged with another vertex node if that node is its only parent and it is the only child of its parent; here, all paths that include the sequence of states $\pi_i \dots \pi_j$ for the subsequence $s_i \dots s_j$ are followed by the same set of states $\pi_{j+1} \dots \pi_k$ for $s_{j+1} \dots s_k$, so we can merge the subpaths together.

On-line maintenance of path structure

This data structure is perhaps easier to understand when explained in terms of how it is maintained.

Suppose we are about to incorporate a sequence letter, s_i. We will describe how to find the k-best paths for a vertex node at the next sequence position, corresponding to the position-state pair (i, c). Recall that in each vertex node, the path nodes are stored in sorted order by probability, and the way we calculate the k-best paths probabilities for (i, c) is similar to first k steps of an (at most) m-way mergesort.

For the ℓth path node, corresponding to the ℓth best path probability ending at state c, we calculate the probability. Suppose that this probability was derived from the probability associated with vertex node v at position b. Then the ℓth path node of the current vertex node that we are building the child of the bth path node of the the vertex node a, updating all counters and potentially adding a vertex level child pointer to b, in case b is a new child for a.

After performing this set of operations for all the new vertex nodes corresponding to sequence position s_i (at all possible model states), we need to prune the data structure of the path nodes which currently do not form part of one of the k-best paths to a leaf, and we must then delete vertex nodes with no path nodes associated with them. We also need to merge appropriate vertex nodes.

This set of deletions and merges can be done by creating a list of all the path nodes with zero count (meaning they are never used on paths to a leaf). Note that in the very beginning the "removal list" will consist only of the paths nodes associated with vertex nodes which were leaves in the previous iteration, thus making them easy for find; this is because we started from a valid k-best tree. For each path node in this removal list, we remove the path node from its vertex node, and update the appropriate counters. If its parent's child counter reaches zero, then the parent is moved to the removal list as well, as this corresponds to the case where the parent corresponds to a path subsection that is never used in one of the k-best paths to a leaf.

There are several interesting cases that could happen to the vertex nodes of the removed path nodes. If a path node removed was the last path node for that vertex node, then the vertex node is removed. Alternatively, if a vertex node becomes the only child of a vertex node with a single parent, then they are merged. We perform these operations until the removal list is empty, meaning that all path level nodes

remaining are those that actually participate in one of the k-best paths to a leaf. The number of nodes touched depends only on the length of the removed paths which are unique to those paths, which is not necessarily proportional to the length of the sequence or the length of the paths removed.

In effect, this structure allows us to store the DAG of the k-best paths in the most efficient way possible, by focusing on regions of the sequence where we reuse the same states for the same symbols. In the worst case, the algorithm can still use $\Omega(kmn)$ space, but in practice, the space usage is dramatically lower, as seen in our experimental results in Chapter 5.

3.3.2 Recovering the paths

Once we have produced the final structure, we must extract the k paths with highest probability. At the end of the traversal, the k path probabilities in each of the m leaves are the probabilities of the best paths to those states. From these km paths, we must select the k with the highest probability. Again, done as a first k steps of the merge operation in the m-way mergesort (we have m lists of probabilities in the sorted order) this operation will take $O(km)$ time, and we can construct the k-best paths then in $O(kn)$ time after the merging by following the back pointers.

A surprising fact is that we may have computed more than the k-best paths while trying to produce the top k. That is, suppose that the discovered k-best paths do not all terminate at a single state. Then if we continue with the mergesort operation, the next-highest probability of a path, which we will discover next, must be the actual $k+1$-st best path probability; this is because we technically are merging the top $k+1$ paths to each node, yet the actual mergesort operation does not need to know what the $k+1$-st best path probability to any state is. Specifically, we can continue the merging operation until we reach the end of any of the m lists of k probabilities. After that point, we have no guarantee of having the proper probability for the next path. This procedure, of course, does not work when we have a designated final state: there, the mergesort is actually a 1-way merge.

Chapter 4

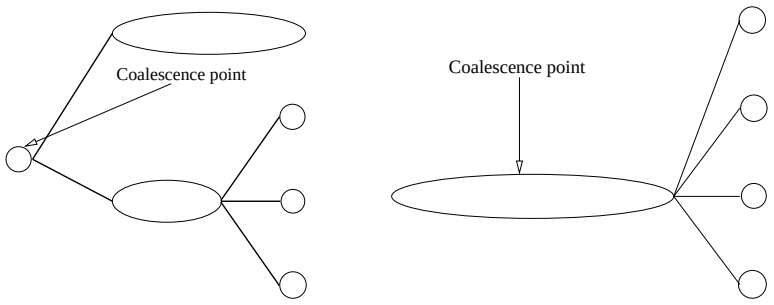
Coalescence points

In practical applications additional saving of space in tree-based approaches can be achieved through the use of coalescence points [24, 13]. Note that here by space we mean RAM, the resource which is commonly limiting when attempting to perform calculation. A *coalescence point* is a point in the sequence where it can be guaranteed that in every potential Viterbi path the symbol at that location is emitted from a single known state. For example, in an HMM with a fixed start state, the location before the first emitted symbol is a coalescence point (Figure 4.1a), albeit not an interesting one.

4.1 Coalescence points in compressed tree

In the Viterbi path, fixing the state at position i in the sequence also fixes the states from positions 1 to $i-1$ (since the optimal path is unique). This allows us to identify the Viterbi path up to coalescence points with a guarantee that it will not get modified by the further steps of the algorithm. In a compressed tree, detecting a coalescence point is an easy task. Assuming a fixed start state the root of the tree is a coalescence point. By the compression property we know that the root node has at least two children. If a node has at least two children this means that there are at least two distinct paths which diverge at the location indicated by the end of the node. But a coalescence point guarantees that there will be unique path left after it. Therefore, the root of the compressed tree is the only coalescence point of the tree.

A simple example of a coalescence point is a state which emits a unique symbol (the symbol is emitted in that state and that state only). After encountering such a symbol in the sequence, all Viterbi paths (and, indeed, all valid paths) use the state thus forcing a coalescence point at the location immediately previous to leaves. Figures 4.1a and 4.1b illustrate how a compressed tree with such a coalescence point would look like right before and right after encountering such a uniquely emitted symbol.



(a) A compressed tree before uniquely emitted symbol incorporation.

(b) The compressed tree in (a) after uniquely emitted symbol incorporation.

Figure 4.1: A schematic illustration of the coalescence point. The uniquely emitted letter is incorporated into the compressed tree resulting in root node encompassing all the states safe leaves. In part (b) circles correspond to leaves. Note that the location on the start node was moved to improve presentation.

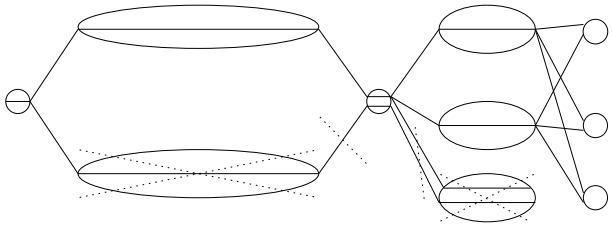


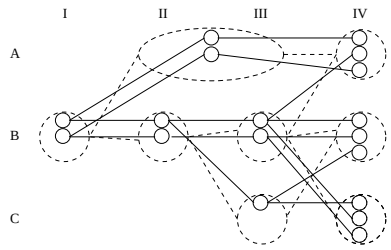
Figure 4.2: An example of the case where paths can be removed after a coalescence point. Each solid line represents a path, the dotted lines signal the removal of either vertexes or edges.

4.2 Coalescence points in k-best tree

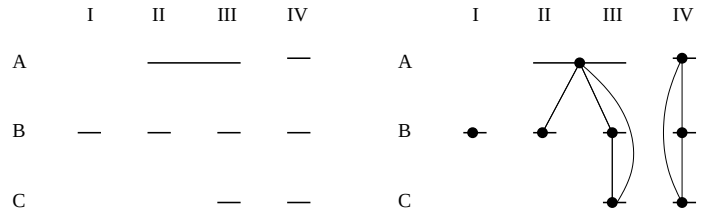
Coalescence points for the k-best Viterbi algorithm do not provide the same level of guarantees as in 1-path Viterbi: the paths may diverge before a coalescence point. Moreover, with k-best paths, there is no guarantee that one of the paths before the coalescence point will not get removed after some later modification to the tree structure; see Figure 4.2 for an example. In the k-best tree coalescence points provide hints in decoding. Overall the problem of finding the coalescence points in the k-best tree strikes us as interesting.

4.2.1 Interval graph representation of k-best tree

The algorithms for detecting coalescence points in the k-best tree are most readily explained when we represent the data structure as an interval graph (either explicitly or implicitly). We use the interval graph as a tool to help us think about the



(a) A k-best tree.



(b) Intervals associated with (a)

(c) Interval graph associated with (a)

Figure 4.3: A k-best tree, the interval representation implied by it and its interval graph.

problem. Interval graphs are a family of graphs which can represent intervals on the real line: each interval corresponds to a vertex, and there is an edge between vertices if the corresponding intervals intersect. More information about interval graphs can be found in [9]. The k-best tree lands itself naturally to an interval graph representation, if we consider only the vertex level nodes. Each vertex node corresponds to a sequence of state position pairs annotating a sub-sequence x_i^j of the input sequence. If we consider each vertex node that annotates x_i^j to be the interval $[i, j]$ (see Figure 4.3), we can construct the interval graph representation.

The k-best tree has several operations which need to be incorporated into the interval graph as adjustments during its maintenance. There are three operation that we perform on the k-best tree. For the purpose of this discussion, we will assume that the operation of adding leaves to an interval graph and then using the node merge operation (when required) is done consecutively, thus there is no need to worry about merging leaves as a separate operation. Addition of leaves is an operation of adding a clique of size m to the graph, and this clique is disconnected from the rest of the interval graph. All new leaves share the same location initially (their corresponding vertices intersect); this will change during merging and pruning, but is originally true due to our assumption that all operations are done consecutively. Deletion corresponds to the removal of a vertex in the interval graph.

The most interesting change to the interval representation happens during the node merge operation. Understanding of the node merge operation in the interval graph requires understanding of two distinct constraints: the constraint on the interval graph due to the fact that it represents a valid k-best tree and the constraint

that the conditions of the node merge operation impose on the corresponding interval graph operation. The fact that an interval graph is a representation of the k-best tree ensures several things. The first of those is that no two isolated nodes (nodes without siblings) whose representations are consecutive intervals can occur: such nodes would have been merged. The second is that for every end of an interval (except for the leaves) the next position has at least one start of the interval, as a path needs to exist between adjacent intervals. The conditions leading to node merging require that a node merge can occur only to the vertices that correspond to intervals that are exactly one position apart. During the node merge no new sequence locations are included in the interval and no locations are lost. In terms of the interval graph this means that the set of neighbours of the new (merged) node is a union of the sets of neighbours of the nodes on which merging was performed.

In this interval graph the coalescence point corresponds to a vertex with no neighbours.

4.2.2 Coalescence points detection algorithms in k-best tree

The naive way to detect coalescence points in the k-best tree is to explicitly construct the interval graph described above. Then we can check if a particular vertex has no neighbours and thus detect coalescence points. Unfortunately, these interval graphs are dense and non-trivial to maintain explicitly, meaning that this approach is very expensive both in terms of memory and run time.

Alternatively one can represent the interval graph implicitly, by storing a list of endpoints of the intervals. The idea behind this approach was given by Wang [29]. We store the end points of the intervals in the order we encounter them when going from the start of the sequence to the end of the sequence. This list, L, will have twice as many elements as the number of vertex level nodes. A coalescence point can be easily detected as a location in the list where the start of the interval is immediately followed by the end of the interval. Our interval graph is a representation of the k-best tree, so we are guaranteed that there will be no intervals which will either start or end at the same points in the sequence as the coalescence interval, as nodes which are children and parents of vertex node in question (there are at least two of each unless one is a root node) have corresponding intervals start or end locations at adjacent sequence positions.

The list L is easily maintained. We start with the list containing two elements, start of the root node and the end of the root node. For each new set of leaves we first append start points for all the leaf vertex level nodes to the list followed by the end points for the leaf nodes(ensuring that starts and ends are inserted in the same order). Every time a particular vertex level node is deleted we delete the start and end points associated with this vertex level node. Note that deletion is the only operation which can lead to a formation of a coalescence point (in node merge the interval still exists and if a location is not singly covered it will not be singly covered after node merge). After node deletion, we check the neighbours of the

deleted start and end points for creation of a coalescence point. In this structure, the node merge operation is not complex. Suppose we are merging vertex level nodes u and v, and u is the one that we want to retain (because it is closer to the root of the tree). We remove the end point of u from the list, remove the start point of v from the list, and substitute the end point of v by the end point of u.

Chapter 5

Experimental results

We implemented our space-efficient k-best HMM paths algorithm in C++ to see if it was, in fact, substantially more space-efficient than the more naive $\Theta(knm)$-space algorithm, and to see how fast it is compared to naive approach.

We were also very interested to see how much we could learn by exploring the k-best paths through an HMM for moderate or large values of k. In particular, we studied the effect of examining k paths in the prediction of transmembrane protein topology, a problem where Viterbi-style decoding has not, traditionally, been especially successful. Our hypothesis is that while the best HMM path may not always be an especially good decoding of a sequence, somewhere in the best 10 or 100 or 1000 paths might lurk a very good analysis of the sequence. We begin with this analysis.

5.1 Prediction accuracy

We have used the recent membrane topology prediction software Phobius [12] to test the usefulness of finding k-best Viterbi paths. The original decoding algorithm for Phobius is the 1-best algorithm [14], developed precisely because the Viterbi algorithm does not provide satisfactory decodings. We used the data set presented by the same paper as Phobius [12] as the data source for our experiments. The two data sets used here are 45 transmembrane proteins which have a signal peptide tag (data set I) and 247 transmembrane proteins which do not (data set II).

Topology prediction is somewhat imprecise because the actual boundary of the membrane-spanning segments is itself a bit inexact, but the boundaries of the membrane-spanning regions can be identified to within a residue or so based on solved protein structures [26]. The original quality measure used by the authors of the Phobius paper describes a prediction as correct if that prediction identifies the correct number of helices, the correct sidedness, and if each true helix overlaps with corresponding predicted helix in at least five positions. Given that helical regions tend to be around twenty two residues long, this measure is lax.

We have additionally studied a different correctness measure, which we call the border boundary measure. This measure has a parameter τ: in the border boundary measure (BBM-τ), a prediction is correct if the sidedness and number of helices are both correct, and if the predicted boundaries of helical regions are all no more than τ residues away from the true boundary. (This measure can also be used to find a distance between two annotations, which is the smallest τ for which the prediction is correct; for predictions with the wrong sidedness or number of helices, this measure is infinite.) Note that the border boundary measure is strict: if even one helix boundary is shifted by $\tau + 1$ residues, while everything else is correctly predicted, the prediction fails.

When evaluating predictions we are evaluating labellings of the sequence. Multiple paths can lead to the same labelling, which means that evaluating 100 paths is not the same as evaluating 100 different predictions. We will mention some differences between the two when talking about relative probability masses of paths and labellings in Section 5.4.

When dealing with more than one predicted path we need to define what do we mean by saying that prediction passes (is correct). Until stated otherwise, prediction is considered to be correct if *at least* one path evaluated is correct by a given measure. This is an indication that some post-processing could have potentially extracted the correct result from the returned result set. Of course, this is an interesting and challenging task, which we will explore in Chapter 6. For now, however, we are interested in whether the information is to be found at all.

Our results for the two data sets are shown in Tables 5.1a and 5.1b. From the results it is clear that, especially for tighter quality measures, the k-best paths contain more information than the 1-best prediction. However, when the strictness of quality measures is relaxed the difference between methods is dropping, as the slackness in the quality measure covers up the difference in the results. The results may be an artifact of the fact that in Phobius emission parameters are tied in a way which makes it hard for the model to recognize transmembrane helix boundaries. Additionally both Phobius and TMHMM, from which their transmembrane topology model was taken, were designed to be evaluated using the 1-best algorithm, which has a potential to skew the results. The training of the Phobius model was performed using complex variant of Baum-Welch training, which should ensure no biases for either method.

Still, we do note the interesting result that choosing the best prediction from the first ten is often much better than the 1-best algorithm's results, particularly for the more stringent measures of quality.

5.2 Performance metrics

Now we want to present some performance metrics associated with the algorithm as a validation of our assumption of memory usage reduction. The runtime and

	BBM-0	BBM-1	BBM-2	BBM-3	BBM-4	BBM-5	Phobius
Vit. 1	3	9	12	19	26	28	32
Vit. 10	11	15	18	26	28	29	33
Vit. 100	14	21	23	30	33	33	37
Vit. 1000	18	24	25	33	35	37	39
1-best	4	14	17	26	32	39	41

(a) Data set I results, containing 45 members.

	BBM-0	BBM-1	BBM-2	BBM-3	BBM-4	BBM-5	Phobius
Vit. 1	2	11	19	27	46	59	137
Vit. 10	9	21	33	44	67	83	157
Vit. 100	34	52	64	85	110	129	198
Vit. 1000	46	66	89	123	143	168	214
1-best	4	18	26	38	62	79	166

(b) Data set II results, containing 247 members.

Table 5.1: Prediction correctness measures. 'Vit. k' stands for k-best Viterbi paths, '1-best' stands for 1-best algorithms used originally for decoding Phobius model. Two correctness measures are presented: BBM-τ is a Border Boundary Measure and Phobius is a Phobius correctness measure. The numbers shown is the absolute number of correct predictions produced. The total number of proteins evaluated for each data set is provided in the caption under the respective tables.

memory usage for data set I are found in Figures 5.1 and 5.2 respectively. The runtime figure shows the total runtime over the whole data set, while the memory figure shows the maximum memory usage. The maximum memory usage for the execution is reached when evaluating the longest sequence contained in the data set. There are three different implementations presented: our tree-based approach, the naive matrix-based approach, and Viterbi without backtracking at all, where we only compute the probabilities of the paths, not their component paths.

Our implementation uses much less memory, while roughly doubling the runtime. There is almost no difference in the runtimes of the matrix implementation and the implementation which does not keep backtracking information at all. Note that one can slightly reduce the absolute memory of the matrix and tree based implementations. However, this will not change the pattern: the typical rate of growth as a function of k is much smaller for our algorithm than the naive one.

5.3 How many paths?

Our next set of experimental results concerns the number of paths recovered after a single run attempting to retrieve k paths. Recall from Section 3.3.2 that the way the k-best paths are revealed was a search through the space of mk probabilities

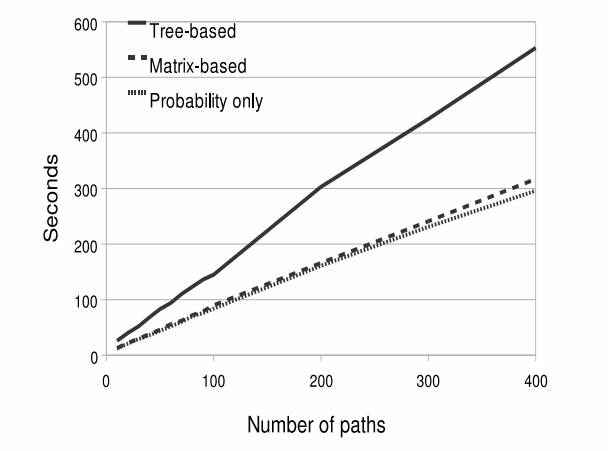


Figure 5.1: Runtimes of different k-best Viterbi implementations on data set I. The solid line represents the naive (matrix) representation, the dashed line represents our tree-based approach, and the dotted line represents no backtracking.

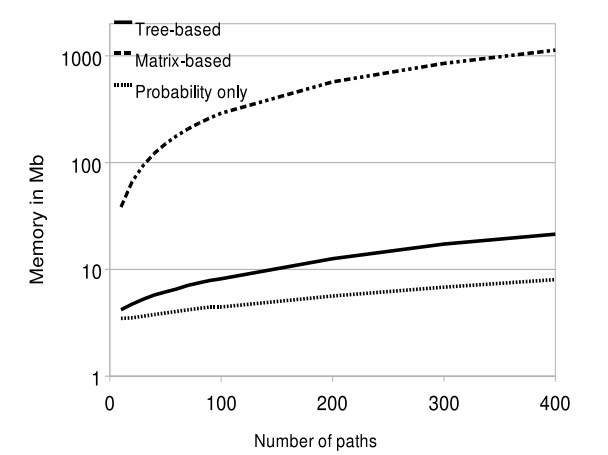
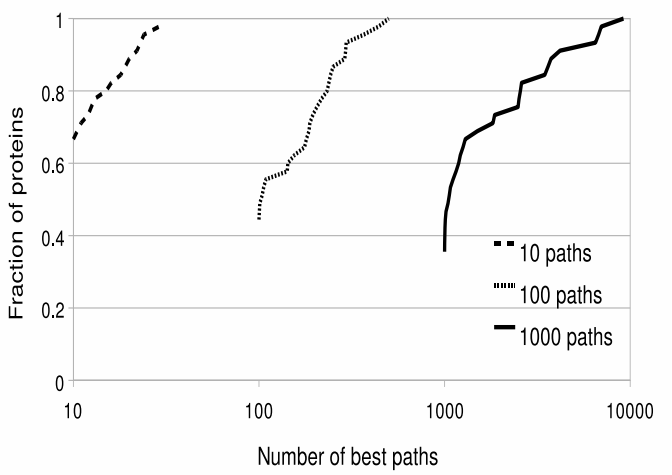
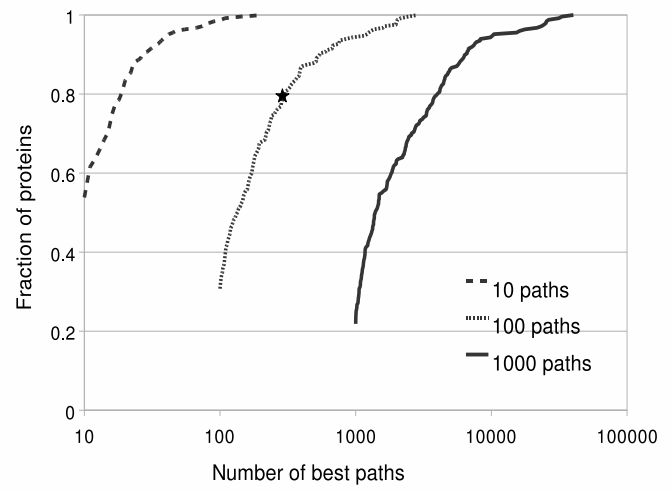


Figure 5.2: Memory usage of of different k-best Viterbi implementations on data set I. The solid line represents our tree-based approach, the dashed line represents naive (matrix) representation, and the dotted line represents no backtracking.



(a) Distribution of actual number of best paths which can be retrieved for data set I.



(b) Distribution of actual number of best paths which can be retrieved for data set II.

Figure 5.3: Cumulative distribution of number of recoverable paths. Dashed line represents ten searched paths, dotted line represents hundred searched paths, and solid line represents thousand searched paths. For example in part (b), the star in the figure shows that for 80% of proteins we found at most 300 available best paths, when 100 best paths were requested; for the other 20%, we have found more.

for the k largest ones. We noted there that we could continue the merging process until we finished looking at all k paths terminating in a single state; this process can result in far more than k paths, though the actual number is unpredictable. Figures 5.3a and 5.3b present the distributions of the number of path which one can recover for both of our data sets, respectively.

Our findings show that, at least for this application, we should expect to find exactly k paths when we execute the procedure that searches for k paths: in a large fraction of our tests, that is exactly what we found. However, if the number of path is not the same as we requested than it can vary wildly: the minimum is of course k, but the maximum can be far larger; this is an example of a distribution where the standard deviation exceeds the mean.

For a better understanding of the absolute numbers it should be noted that number of states which constitute a valid end state (for reasonably long sequences) in Phobius is 139, so if we are looking for the k-best paths, in practice, we can retrieve at most $138k$ paths. Naturally, this phenomenon depends heavily on the structure of the HMM being analyzed.

5.4 Probability mass of paths and labellings

For each path generated by the k-best algorithm we can easily find both the absolute probability of the path given the model (Viterbi probability), and the total probability of the labelling corresponding to that path. We also can compute the relative probabilities of both, in the space of all paths given the model and the sequence, by calculating the probability of the sequence given the model (forward probability) and then finding the fraction of that probability that the Viterbi probability of the path or probability of the labelling represents.

The data resulting from this analysis can be found in Tables 5.2a and 5.2b for data sets I and II respectively. It is interesting to see just how much of the probability mass the first paths or labellings are occupying. Each sequence admits an exponential number of both paths and labellings, yet on average, the first one thousand paths occupy 15% and 28% of all probability space in proteins with and without signal peptides respectively. For labellings the situation is even more concentrated (as expected, since the probability of a labelling is always at least as high as the probability of the path having this labelling). On average, more than a fifth of the total probability mass for proteins with a signal peptide and more than a third of all probability mass for proteins without a signal peptide is allocated to the labellings admitted by one of the one thousand best paths.

It is also interesting to investigate how the proportion of the probability mass allocated changes with the number of paths. A very interesting trend is showing: the probability mass used is about the same for first ten, the next ninety, and the next nine hundred paths; a similar finding seems to hold for labels. This suggests that the proportions of mass taken by paths and labels decreases exponentially

	1 path	10 paths	100 paths	1000 paths
Total probability of paths	0.01	0.05	0.10	0.15
Total probability of labels for paths	0.03	0.09	0.16	0.22

(a) Average proportion of total probability mass for data set I

	1 path	10 paths	100 paths	1000 paths
Total probability of paths	0.02	0.09	0.19	0.28
Total probability of labels for paths	0.04	0.14	0.28	0.37

(b) Average proportion of total probability mass for data set II

Table 5.2: Average proportion of total paths probabilities and total probability of labels encounted among paths, conditioned on each sequence. For example, the number 0.28 in part (b) means that in data set II, the total probability of the top 1000 paths on average occupies 28% of all the probability mass for the sequences.

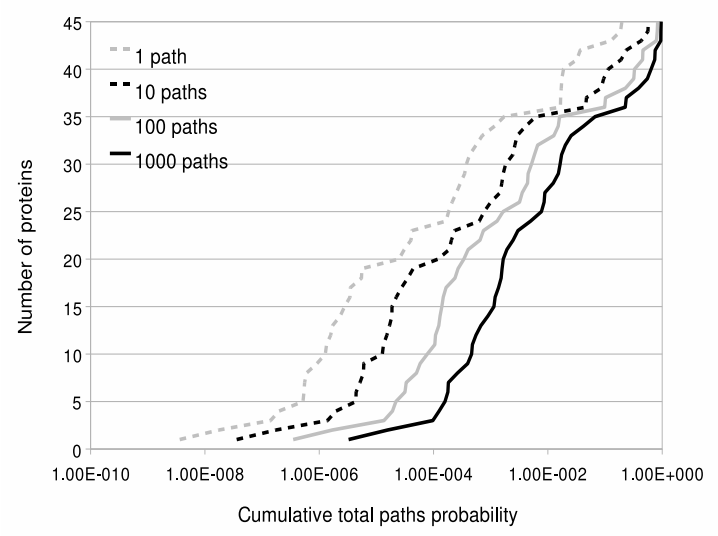
with the depth of the search, though we have no theory to support this; it would be fascinating to study this further.

The next question which arises is how the total probabilities of paths and labellings are distributed. The cumulative distribution plots for the probability masses for paths and labels can be found in Figure 5.4 for data set I and Figure 5.5 for data set II. The probability mass of most of the proteins in both data sets is relatively low. However, there are several proteins for which the probability mass of the paths calculated nears 1.

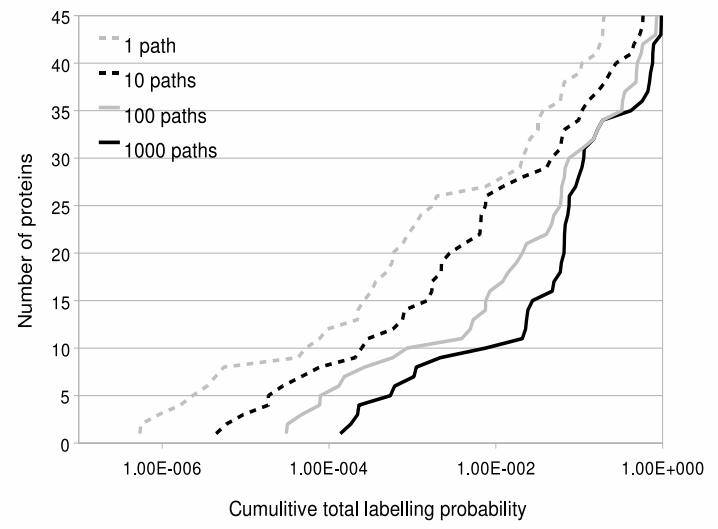
The labelling total probability follows the same general pattern, as the path probabilities. One interesting thing to note is that the graph for the thousand paths. In data set I and especially in data set II there is a non-negligible number of cases with the total probability close to 1. One other interesting detail is the steepness of the curves, especially in the case of 1000 paths. This indicates that the number of proteins in the range between 0.2 and 0.8 is small, which is strange considering that this range represents 60% of all the probability space. We do not have a theory to explain this observation.

5.5 Random sampling from the model

In the previous section, we have shown that sometimes the top paths take a very large proportion of the probability mass. Later, in section 6.1, we will show that there is a correlation between having a high proportion of the probability mass

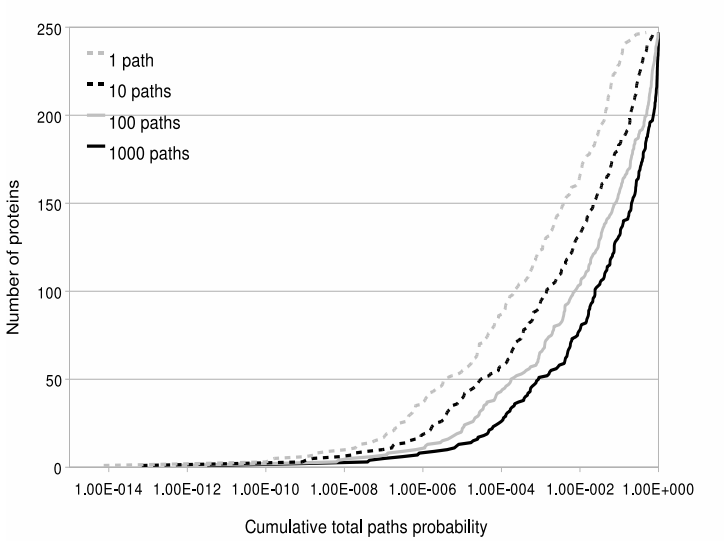


(a) Distribution of total probabilities of top 1, 10, 100, and 1000 paths

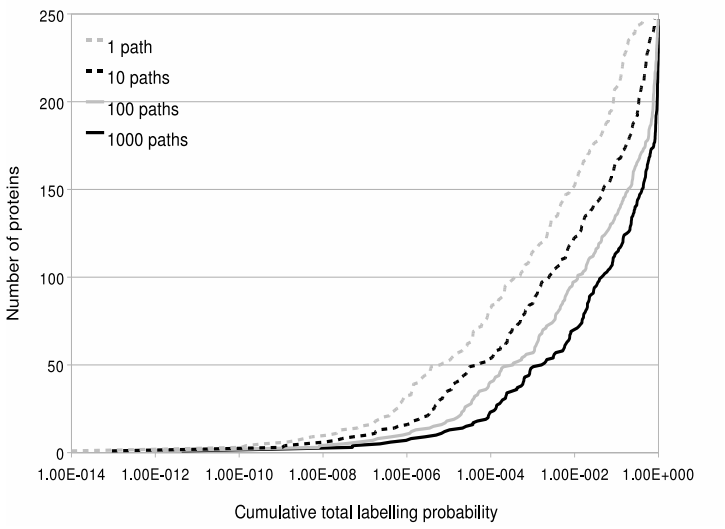


(b) Distribution of total labelling probabilities found in top 1, 10, 100, and 1000 paths

Figure 5.4: Distribution of total probabilities for data set I. For example, 10 proteins have the total probabilities of their top 100 paths total 10^{-4} or less. Note: the scales of the two graphs are different.



(a) Distribution of total probabilities of top 1, 10, 100, and 1000 paths



(b) Distribution of total labelling probabilities found in top 1, 10, 100, and 1000 paths

Figure 5.5: Distribution of total probabilities for data set II.

in the top paths and getting the prediction right. In this section, we will look at a similar problem: evaluating how well paths randomly sampled form the model (while respecting the sequence) behave. We have sampled paths from the probability distribution of all consistent paths, where the probability of picking a particular path is its relative probability in the space of all consistent paths.

The algorithm for the sampling procedure is fairly simple. For each location in the sequence we calculate the probability of each state given a sequence before that position (the forward probability) and the probability of each state such that the remainder of the sequence is emitted starting from it (the backward probability [8]). Given these probabilities, we can calculate the probability of taking each edge at each position at the sequence. The paths are constructed by randomly taking edges (according to this stated probability distribution). This way of calculating edge probabilities is also used in the Baum-Welch training algorithm [8].

We sampled paths without replacement, which lead to an interesting phenomenon. In some cases, when the relative probability of the top paths is high, more than k tries are required to return k distinct paths. To make the computation tractable we have limited the amount of attempts to $100k$ in cases when k random paths are requested. The number of attempts that were required to produce a requested number of paths can be found in Tables 5.3 and 5.4 for data set I and II respectively. In many cases for data set II, the limit on the amount of attempts is reached and fewer then k paths are returned. For that data set a significant proportion of proteins have most of their probability mass in their top paths (see Figure 5.5).

Next, we look at how well the sampling performed as a predictor. The results of those experiments can be found in Tables 5.5 and 5.6. These results indicate that in data set I there are some proteins that are relatively easy to predict, while the rest have a much more even distribution of probabilities of paths between alternative predictions. In this data set, the probability of the top paths is distributed comparatively evenly (see Section 5.4) and the correlation between the total of the top k-best paths probabilities and prediction correctness is much less prominent than in data set II (see Section 6.1), therefore an even distribution of probability between alternative explanations is not surprising.

For data set II, the number of correct predictions is growing steadily with the number of paths sampled, while starting from a relatively high initial number of correct samples. This, again, corresponds well with the distribution of total probabilities, which indicates that there is a fair fraction of easily predicted proteins with a high total probability mass in the top paths (see Figure 6.2). Still, the overall number of correct predictions is very low for sampling, making this approach not useful for prediction.

Next we wanted to verify that if the samples that are counted as correct (for the Phobius correctness measure) are more likely to come from the proteins with high probability in their top k-paths. We sorted all proteins in the sample in the order of the probability of their top paths (with larger probability being higher)

Number of attempts	Trial one	Trial two	Trial three
10	37	40	39
11-100	8	5	6
Total	45	45	45

(a) 10 paths requested

Number of attempts	Trial one	Trial two	Trial three
100	32	31	32
101-1000	11	12	11
2001-3000	0	1	0
3001-4000	1	1	2
4001-5000	1	0	0
Total	45	45	45

(b) 100 paths requested

Number of attempts	Trial one	Trial two	Trial three
1000	27	30	28
1001-10000	15	12	14
20001-30000	1	1	1
100001	2	2	2
Total	45	45	45

(c) 1000 paths requested

Table 5.3: Number of attempts needed to sample random paths without replacement for data set I. The results for three trial runs are given to provide more evidence that results are not outliers. For example in part a) the number 37 means that in 37 out of 45 cases random sampling produced 10 distinct paths after exactly 10 attempts. There was a limit on the number of attempts: a new sample was requested no more than 100 times the number of paths that we where trying to output.

Number of attempts	Trial one	Trial two	Trial three
10	194	193	201
11-100	49	49	40
101-200	1	1	3
201-300	1	0	0
301-400	0	0	1
401-500	0	1	0
601-700	0	1	0
1001	2	2	2
Total	247	247	247

(a) 10 paths requested

Number of attempts	Trial one	Trial two	Trial three
100	137	134	137
101-1000	91	94	91
1001-2000	2	2	2
2001-3000	1	1	0
3001-4000	0	0	1
9001-1000	0	0	1
10001	16	16	15
Total	247	247	247

(b) 100 paths requested

Number of attempts	Trial one	Trial two	Trial three
1000	91	87	89
1001-10000	113	117	115
10001-20000	10	10	10
20001-30000	2	2	2
30001-40000	1	1	1
40001-50000	1	2	1
50001-60000	2	1	2
100001	27	27	27
Total	247	247	247

(c) 1000 paths requested

Table 5.4: Number of attempts needed to sample random paths without replacement for data set II. There was a limit on the number of attempts: a new sample was requested no more than 100 times the number of paths that we where trying to output.

	BBM-0	BBM-1	BBM-2	BBM-3	BBM-4	BBM-5	Phobius
1 path	0	0	0	0	2	3	6
10 paths	0	0	0	3	5	5	7
100 paths	0	0	2	6	6	6	7
1000 paths	0	1	5	6	6	6	7

(a) Trial one

	BBM-0	BBM-1	BBM-2	BBM-3	BBM-4	BBM-5	Phobius
1 path	0	0	1	1	1	2	6
10 paths	0	0	2	4	5	6	7
100 paths	0	0	1	5	6	6	7
1000 paths	0	1	4	6	6	6	7

(b) Trial two

	BBM-0	BBM-1	BBM-2	BBM-3	BBM-4	BBM-5	Phobius
1 path	0	0	0	0	1	2	5
10 paths	0	0	1	2	5	5	7
100 paths	0	0	1	5	6	6	7
1000 paths	0	1	4	6	6	6	7

(c) Trial three

Table 5.5: Sampling results for the data set I. Each number indicates the number of proteins (out of 45) for which at least one sampled paths provided the correct explanation. The data for 3 different random paths trials is provided. For example, in the first sample, when 100 paths were requested 6 proteins where correct by $BBM-4$ correctness measure.

and identified the indexes of the proteins correctly predicted by sampling. The data for 100 sampled paths, for the data set II (in particular trial two), is in Table 5.7. Correct samples are overrepresented among the proteins with the highest probability mass of their top paths. This is consistent with the conclusion for Section 6.1, on correlation between probability mass of top paths and prediction correctness, that correct predictions tend to have higher mass for the top paths.

	BBM-0	BBM-1	BBM-2	BBM-3	BBM-4	BBM-5	Phobius
1 path	0	1	1	3	4	6	34
10 paths	0	2	3	6	9	14	38
100 paths	0	4	6	9	12	16	42
1000 paths	0	5	7	10	17	20	43

(a) Trial one

	BBM-0	BBM-1	BBM-2	BBM-3	BBM-4	BBM-5	Phobius
1 path	0	0	2	4	5	6	31
10 paths	0	1	3	7	10	13	39
100 paths	0	3	5	9	12	18	41
1000 paths	0	5	6	11	15	19	43

(b) Trial two

	BBM-0	BBM-1	BBM-2	BBM-3	BBM-4	BBM-5	Phobius
1 path	0	0	0	1	4	6	33
10 paths	0	2	3	6	10	16	39
100 paths	0	4	6	9	13	17	40
1000 pahts	0	5	6	9	16	19	44

(c) Trial three

Table 5.6: Sampling results for the data set II. Each number indicates the number of proteins (out of 247) for which at least one sampled paths provided the correct explanation. The data for 3 different random paths trials is provided.

	Top third	Mid third	Bottom third
1-best probability	20	11	9
10-best probabilities	20	12	8
100-best probabilities	22	10	8
1000-best probabilities	22	9	9

Table 5.7: Distribution of correctly sampled proteins among the indexes of all proteins sorted by total probability of their top paths. For example number 20 in row with label "10-best probabilities" and column "Top third" indicates that among 40 proteins which sampling predicted correctly 20 where among one third of the most probable (with indexes between 1 and 82) in the list of proteins sorted total probabilities for their top 10 paths.

Chapter 6

Extracting information from the k-best paths

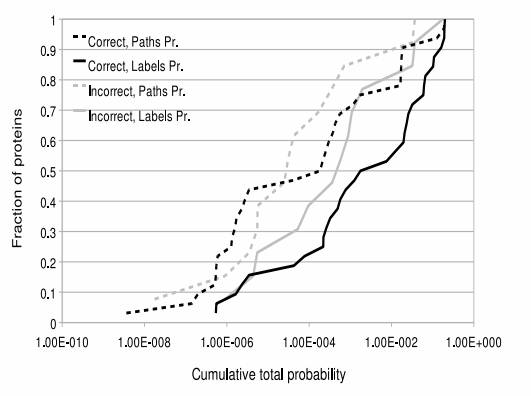
What do multiple high-probability paths show us? To study this, we broke predictions (labellings) into groups based on their proposed topology. Each group (cluster) has a distinct number of transmembrane helices and sidedness.

6.1 Connection between correctness and probability mass of paths and labellings

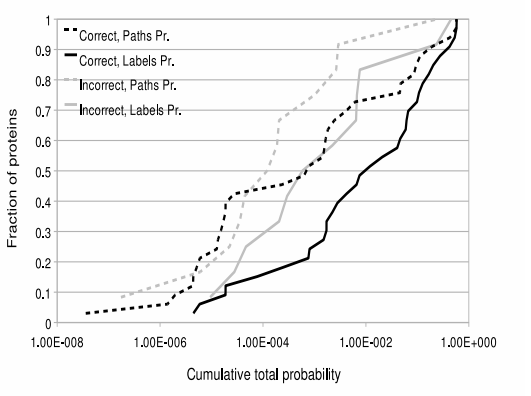
Here we investigate the relation between the probability mass that predicted paths occupy and whether they give a correct prediction. In other words we are looking at the degree of confidence in the prediction, based on the total probability mass of the top k-paths. We have graphed the relation between probability mass and correctness in a series of plots, presented in Figures 6.1, 6.2, 6.3, and 6.4. In each plot cumulative probabilities of paths and labellings for cases of both correct and incorrect predictions are shown. The figures themselves vary on the number of paths (1, 10, 100, and 1000), the correctness measure used (Phobius in Figures 6.1 and 6.2, BBM-5 in Figures 6.3, and 6.4), and the data set (I in Figures 6.1 and 6.3 and II in Figures 6.2 and 6.4).

Data set I is smaller, which leads to it being more prone to random variation. Also, in this data set, the number of correct predictions is very high, making comparison between correct and incorrect predictions hard. For data set I, most of the patterns discussed below are less vivid and noticeable. It is also harder to judge the significance of the patterns seen. Data set II, on the other hand, has 247 members, and often has a significant proportion of the predictions incorrect. The pattern seen in this data set are more trustworthy.

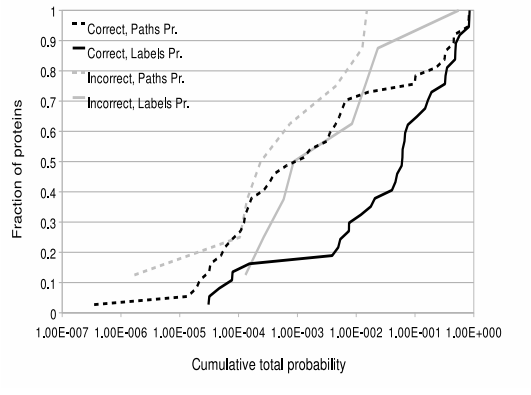
The most interesting comparison is in the difference between the correctness measures used. The more lax Phobius correctness measure leads to less difference



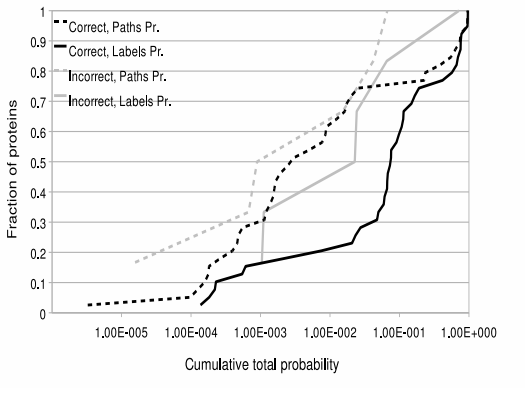
(a) 1 path. 32 Correct; 13 Incorrect.



(b) 10 paths. 33 Correct; 12 Incorrect.

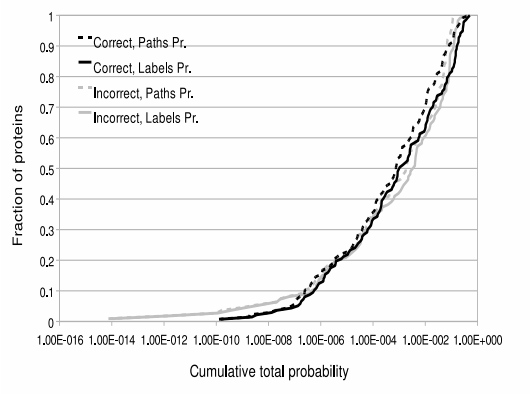


(c) 100 paths. 37 Correct; 8 Incorrect.

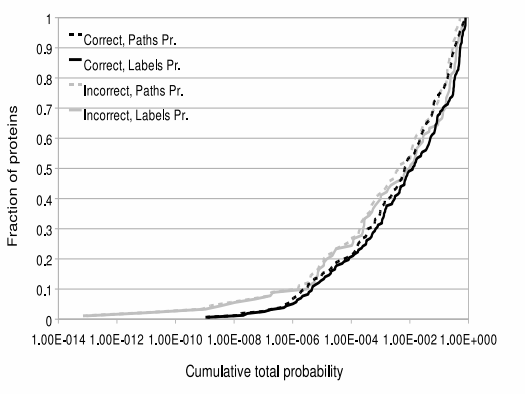


(d) 1000 paths. 39 Correct; 6 Incorrect.

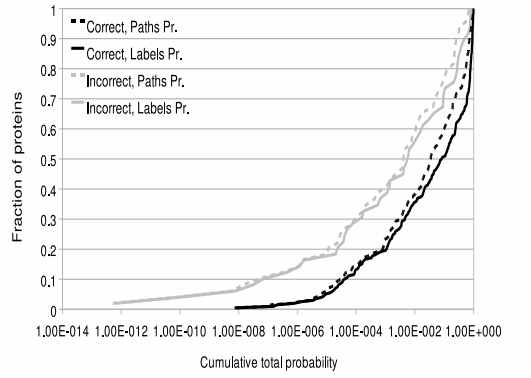
Figure 6.1: Cumulative total probability distribution separated by correctness. Here the Phobius correctness measure is used on the data set I. Larger percent of correct predictions at lower probability (line is closer to righthand lower corner) indicates that high probability mass is correlated with correct predictions. Note that the scales are different for the graphs.



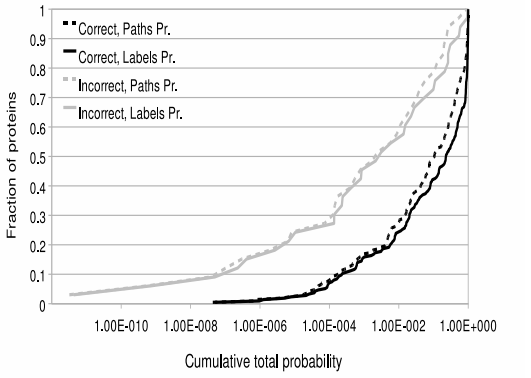
(a) 1 path.137 Correct; 110 Incorrect.



(b) 10 paths. 157 Correct; 90 Incorrect.

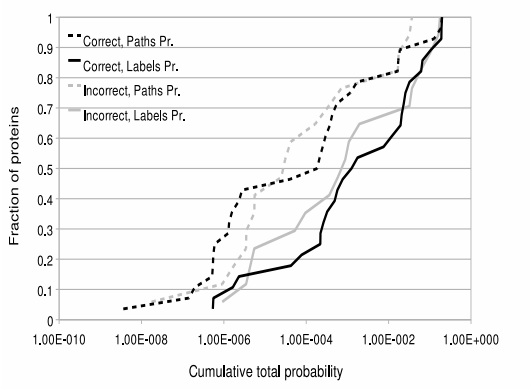


(c) 100 paths. 198 Correct; 49 Incorrect.

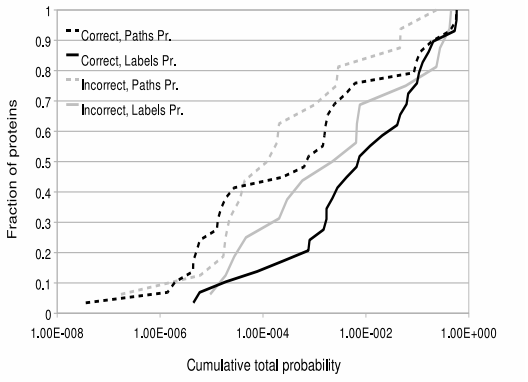


(d) 1000 paths. 214 Correct; 33 Incorrect.

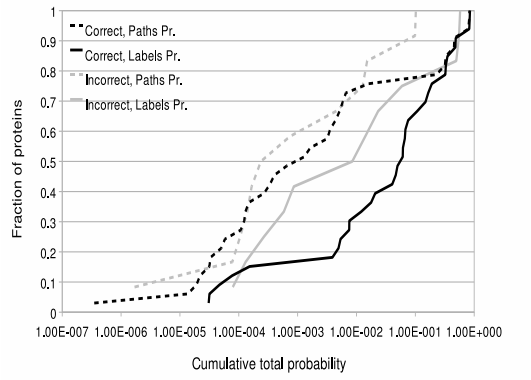
Figure 6.2: Cumulative total probability distribution separated by correctness. Here the Phobius correctness measure is used on the data set II. Note that the scales are different for the graphs.



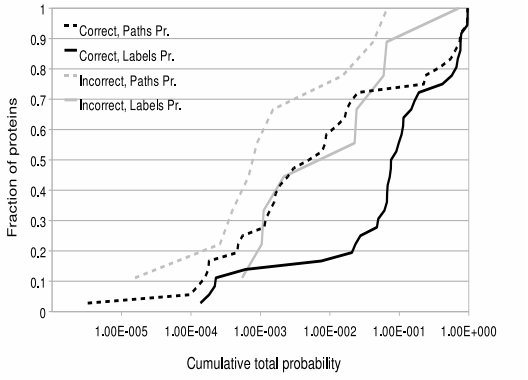
(a) 1 path. 28 Correct; 17 Incorrect.



(b) 10 paths. 29 Correct; 16 Incorrect.

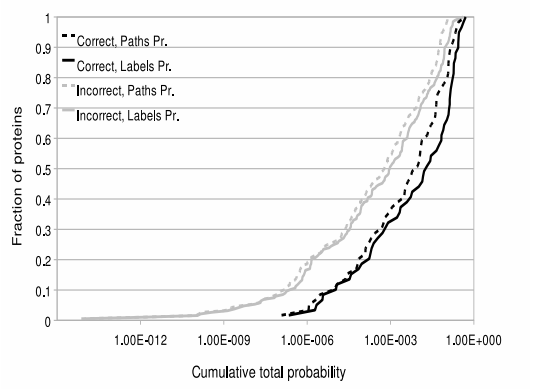


(c) 100 paths. 33 Correct; 12 Incorrect.

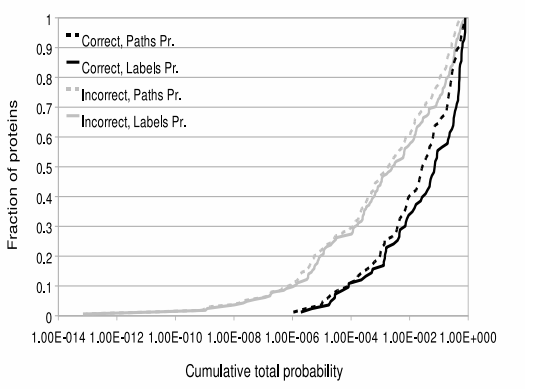


(d) 1000 paths. 36 Correct; 9 Incorrect.

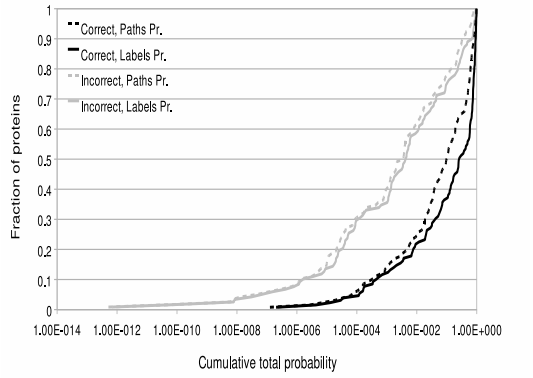
Figure 6.3: Cumulative total probability distribution separated by correctness. Here the BBM-5 border boundary measure is used to identify correct predictions on data set I. Note that the scales are different for the graphs.



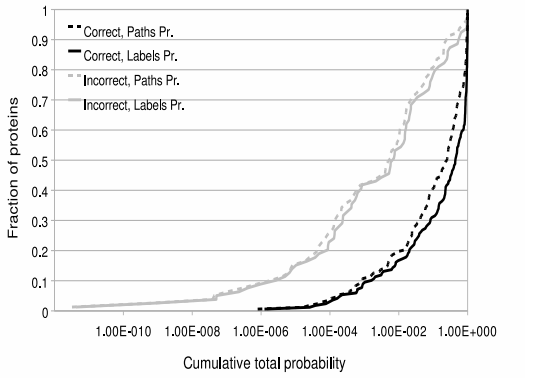
(a) 1 path. 59 Correct; 188 Incorrect.



(b) 10 paths. 83 Correct; 164 Incorrect.



(c) 100 paths. 129 Correct; 118 Incorrect.



(d) 1000 paths. 168 Correct; 79 Incorrect.

Figure 6.4: Cumulative total probability distribution separated by correctness. Here the BBM-5 border boundary measure is used to identify correct predictions on data set II. Note that the scales are different for the graphs.

between correct and incorrect predictions in terms of probability mass. On the other hand, our BBM-5 correctness measure makes a very clear distinction. It is very likely that the paths which are correct using this measure have a high relative probability mass, even in the cases of the low number of paths.

The overall trend is that a high percentage of the probability mass indicates higher confidence in the prediction. Proteins that have a high percentage of their probability mass allocated to the top path are easy to predict. This is emphasized by the difference between the correctness measure where the correlation between high mass and correct predictions is especially clean for the stricter BBM-5 correctness measure.

6.2 Gaining confidence in predictions based on agreement in their structure

In this section, we present the relationship between the number of paths investigated and the confidence in our prediction. All the predictions naturally fall into two distinct (and compliementary) categories: one where all of the predictions have the same number of helices (class one) and one where at least two different helix numbers whre predicted (class two). The natural assumption is that a more consistent class, one where all the predicted helix numbers are the same, would give better predictions. We now present the data for these two classes.

We first analyze the case where all of the predicted paths have the same number of helices: these form the content of Tables 6.1b and 6.2b. In both cases the trend is very clear: if more and more paths predict the same number of helices, it is more probable that the prediction is correct. At the same time, the total number of such proteins naturally decreases with number of paths predicted. As even a single path with a number of predicted helices distinct from the majority forces the proteins out of this class, the decrease in the total number of proteins in it with the growth of the number of paths produced is not surprising.

The number of correct predictions found in this class is high: these are the easy-to-predict proteins (this is similar to a relation between the proportion of the probability mass occupied by the k-best paths and the correctness of the prediction, described in Section 6.1). As the number of paths increases, the size of this class shrinks, and at the same time our confidence that the prediction is correct increases.

The second class that we investigate are all the cases where there are at least two different predicted helix numbers. This class is complementary to the one described previously: together, they cover the whole range of predictions. In this class, there are multiple groups which we can use for prediction; here we investigate the largest and the heaviest (highest probability) clusters as potentially predictive. The results for this class can be found in Tables 6.1c and 6.2c. The overall size of the class is rising as the number of predicted paths increases, as it is more likely that at

Number of paths	Class 1	Class 2
10	39	6
100	28	17
1000	21	24

(a) Number of potentially correct predictions

Type of correctness	10 paths (n = 39)	100 paths (n = 28)	1000 paths (n = 21)
Number of helices	31 (79.49%)	24 (85.71%)	19 (90.48%)
Overall topology	30 (76.92%)	24 (85.71%)	19 (90.48%)

(b) Distribution of correct predictions when all predicted paths have the same number of helices (class 1 from part (a)).

Type of correctness	10 paths (n = 6)	100 paths (n = 17)	1000 paths (n = 24)
Number of helices in largest cluster	5 (83.33%)	12 (70.59%)	18 (75.00%)
Number of helices in heaviest cluster	5 (83.33%)	12 (70.59%)	18 (75.00%)
Overall topology in largest cluster	4 (66.67%)	10 (58.82%)	17 (70.83%)
Overall topology in heaviest cluster	4 (66.67%)	10 (58.82%)	17 (70.83%)

(c) Distribution of correct predictions when two different helix counts were predicted (class 2 from part (a)).

Table 6.1: The distribution of the correct predictions in different classes of predictions for data set I.

Number of paths	Class 1	Class 2
10	189	58
100	120	127
1000	64	183

(a) Number of potentially correct predictions

Type of correctness	10 paths (n = 189)	100 paths (n = 120)	1000 paths (n = 64)
Number of helices	156 (82.54%)	107 (89.17%)	58 (90.63%)
Overall topology	123 (65.08%)	93 (76.67%)	56 (87.50%)

(b) Distribution of correct predictions in cases when all predicted paths have the same number of helices (class 1 in part (a)).

Type of correctness	10 paths (n = 58)	100 paths (n = 127)	1000 paths (n = 183)
Number of helices in largest cluster	28 (48.28%)	82 (64.57%)	125 (68.31%)
Number of helices in heaviest cluster	32 (55.17%)	81 (63.78%)	130 (71.04%)
Overall topology in largest cluster	16 (27.59%)	54 (42.52%)	97 (53.01%)
Overall topology in heaviest cluster	19 (32.76%)	51 (40.16%)	93 (50.82%)

(c) Distribution of correctness for in cases when two different helix counts were predicted (class 2 in part (a)).

Table 6.2: Distribution of correct predictions in different classes of predictions for the data set II.

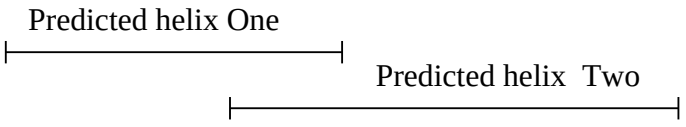
least two paths that have different number of predicted helices are encountered. Also, the number of correct predictions is rising with the number of paths, as more proteins which where previously in class one are moved to this class. The absolute numbers of correct predictions is small when the number of paths is small: these are the hardest proteins to predict. In no case is the prediction accuracy as large as the one for the case when all predicted paths have the same number of helices. Our confidence in the prediction is lower than in the cases when all paths have the same number of helices. Such pattern indicates that we have identified the cases hardest to predict in this class at the low number of paths predicted.

As the results of the patterns we see above we can talk about a confidence scale for the predictions based on the distribution of the k-best paths into these two groups. If the prediction has at least two distinct number of helices in the top ten paths, we have very little confidence in its correctness; these seem to form the hardest proteins. If the first one thousand paths all have the same number of predicted transmembrane helices, this indicates a high confidence in the prediction; these seem to be the easy ones. Overall, the moment at which the second value for the number of helices appears seems to indicate the degree of confidence in the prediction.

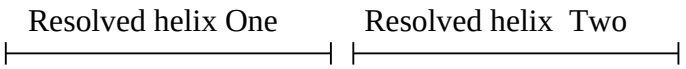
6.3 From many predictions to one prediction

In this section we will persent several different ways to agglomerate a cluster into a single prediction. The heaviest cluster is the one on which the agglomeration is performed, as there does not seem to be a significant difference between using the largest and heaviest clusters, based on the results of experiments in Section 6.2. The paths in the cluster are agglomerated using a method, and then we apply different correctness measures to evaluate the resulting prediction. There is a natural upper bound on the amount of cases in which agglomeration can succeed: if the heaviest cluster predicts the wrong sidedness or helix number, our agglomeration methods are sure to fail. The numbers can be found in Tables 6.1 and 6.2. For example, for data set II, predicting 100 paths, for the case when all paths predicted the same number of helices, 93 proteins can potentially be predicted correctly using agglomeration.

All of the experiments described in this section involve two types of cases. The first type does not take into account the probabilities associated with each of the path and all such experiment instances are called *unweighted*. In such cases the number of the proteins that can be found in a group will be referred to as the size of the group. In the second type, which we call *weighted*, we incorporated the probabilities of paths into the agglomeration. In such cases, the probability of the path is its weight and sometimes, for a single group, it will be referred as its mass.



(a) Majority voting conflict



(b) Majority voting conflict resolution

Figure 6.5: Example of a conflict created by majority voting and its resolution. The distance between two helices in the resolved case (b) is exactly one amino acid.

6.3.1 Averaging

This method of agglomerating paths in a cluster into a single prediction involves averaging the positions of the start and the end of each transmembrane helix. We used two types of averages: an unweighted average off all starts and ends and the weighted average. A major advantage of this way of agglomerating is that it is guaranteed not to lead to inconsistent results: start and end location positions naturally balance out, and helices never end before they begin. The results of the experiment on both data sets I and data set II can be found in Tables A.0 and A.1 respectively.

This method is fairly good at retrieving the information that is contained in the cluster. For the Phobius measure, at most three (of 93) proteins (data set II, at one thousand paths in the case when at least two different helices are predicted) were missed. It is striking how much the quality of prediction depends on the correctness measure used, which we have already observed in the Section 5.1, on the existence of information in predictions in the first place.

6.3.2 Majority voting

This agglomeration method involves keeping track of the start position and length for each helix. The values encountered most often are used for the prediction; in the weighted cases, we count paths with weight equal to their probability. The results for the unweighted and weighted majority voting agglomeration can be found in Tables A.2 and A.3 respectively.

One drawback of majority voting is that in some cases it can produce inconsistent results. This happens, for example, when there are two helices that are predicted close to each other have weak signals for their start and end positions. This may lead to the agglomeration predicting a pair of overlapping helices, as in Figure 6.5a. In our data set this happend only once, to the protein SecD. We smoothed the prediction by creating two helices with equal length; these helices

	10 paths	100 paths	1000 paths
Weighted	1	4	1
Unweighted	1	5	3

Table 6.3: Number of locations at which proteins in data set II needed smoothing.

cover the same positions as the prediction, except for one amino acid in the exact middle (Figure 6.5b).

The results produced by the majority voting are practically the same as the ones predicted by the average voting scheme. Again, for the Phobius measure, we have have predicted correctly either all of the proteins for which the biggest cluster could yield correct results, or almost all of them. As before, incorporating weight into prediction did not influence the results noticeably.

6.3.3 Per-position voting

In our per-position voting scheme, at each position, the agglomeration is given the value of the most common label at that position among the set of paths. The values for predictions of the data set I can be found in Table A.4, while the values for the data set II can be found in Table A.5. The per position voting scheme inherently produces inconsistent results due to lack of accounting for global information. We check the resulting prediction for consistency and smooth it when needed.

In the data sets used, only four proteins, out of almost three hundred, required any smoothing of this sort. They were PgdR, Quinoprotein glucose dehydrogenase (DHG_ECOLI), SecD, and Mec4. Some of these need to be adjusted in more than one location: the distribution of the number of changes for data set II can be found in Table 6.3. In data set I a single prediction requires smoothing, which is a single unweighted agglomeration of one hundred paths. There seems to be an increase in the number of predictions requiring smoothing in the case of one hundred paths. Perhaps, when the number of predicted paths is small, there are fewer distinct, conflicting predictions trying to influence the final result, while when the number of paths is large, similar predictions outvote the outliers. In the middle, which 100 paths represents, the outliers influence the final prediction.

The absolute numbers for per position voting are very similar to both average and majority voting. Both the weighted and unweighted cases follow this pattern.

6.3.4 Averaging using label wights

Next we want to investigate weather using label probabilities instead of path probabilities will influence the results. We have performed the average agglomeration using label probabilities instead of paths probabilities as weight, with the results

shown in Tables A.7 and A.2 for data sets I and II respectively. When dealing with label probabilities one needs to be careful to avoid double counting, as there may be fewer labels in the data then there are paths. The first step of agglomeration is creating a list of unique labels, with attached probabilities. Then weighted averaging was performed using label probabilities as weights.

As can be seen for the data, there is no significant difference between label probabilities and path probabilities for the agglomeration.

6.3.5 Summary

The three methods described give comparable results on both data sets. This seems to indicate that inside of one predictive cluster predicted results are very similar to each other. Moreover, it seems that there is no noticeable difference between the weighted, weighted by labelling probabilities, and the unweighted cases, again indicating the inner cluster results similarity. The Averaging method has an added advantage of always producing consistent results, thus is likely to be a method of choice if a predictor is implemented. As compared to other decoding algorithms, agglomeration performs better than the Viterbi algorithm, but not as well as the 1-best decoding.

6.4 Dual topology proteins

Dual topology proteins form an interesting subset of transmembrane proteins. These proteins have two different topologies they can assume. The difference between the two topologies is the sidedness. The existence of dual topology proteins was first confirmed by Rapp *et al.* in 2006 [21]. In their paper Rapp *et al.* mention five proteins which they confirm have dual topologies. Those five proteins are EmrE, SugE, CrcB, YdgC, and YnfA. All five proteins are very short, have four transmembrane helices and a very weak sidedness signal. As we stated in Section 2.2, the signal for the sidedness of the protein is the amount of the positive amino acids which tends to appear on the inside of the protein ('positive inside' rule). In the dual topology proteins, there is no significant difference in positive amino acids on different sides of the transmemebrane helices. Here, we would like to investigate how the k-best predictions in the Phobius model perform on those proteins. The most interesting question is if the model can recognize the duality of the topology.

The results for the correctness prediction of Phobius on the five dual topology proteins can be found in Table 6.4. A correct prediction in this case is defined as the prediction that can be found in Uniprot [4], which is only one of the two topologies. Because this data set contains only five proteins and all of the proteins have small length, we calculated up to ten thousand paths. Often, the information about the correct topology can be found within the first ten thousand paths, but it is not readily available in the most probable, Viterbi, prediction. Moreover, looking

	1 path	10 paths	100 paths	1000 paths	10000 paths
BBM-0	0	0	0	0	0
BBM-1	0	0	0	0	0
BBM-2	0	0	0	2	2
BBM-3	0	1	1	2	3
BBM-4	0	2	2	3	4
BBM-5	1	2	2	3	4
Phobius	1	2	2	3	4

Table 6.4: Results of running Phobius on dual topology data

	1 path	10 paths	100 paths	1000 paths	10000 paths
BBM-0	0	0	0	0	0
BBM-1	0	0	0	0	1
BBM-2	0	0	0	3	4
BBM-3	1	1	2	4	5
BBM-4	2	4	4	4	5
BBM-5	3	4	4	4	5
Phobius	3	4	4	4	5

Table 6.5: Results of running Phobius without signal prediction on dual topology data

at the first one hundred paths only shows the correct prediction for two proteins, which is a lower percentage than we would expect based on previously seen data.

Having looked at the data manually to find out the reason for such poor showing of top paths, we found that the main reason for such poor prediction was that Phobius predicted a signal peptide instead of the first transmembrane helix. This is not entirely surprising: these dual topology proteins have a very small distance before the start of the first transmembrane helix, have small length and have a weak inside/outside signal (as they can be in two topologies). The lack of predictive signals can easily lead to misprediction of the first helix. We have tried predicting dual topology proteins after disabling signal prediction in Phobius, with the results presented in Table 6.5. Without the signal prediction, the Phobius model is equivalent to TMHMM's membrane protein topology model. The correct prediction is seen much earlier in this case, and, for the laxer quality measures, even the single highest probability path gives a prediction that is correct in 3 of 5 cases. Moreover, somewhere within the first ten thousand best paths we can find a correct solution for all five proteins, for one of the two topologies of each protein.

The next step is to investigate the properties of the k-best paths. In cases when the two top groups correspond to the two different correct topologies for the protein, using k-best paths gives an easy way to predict the dual topology. We

	10 paths	100 paths	1000 paths
EmrE	2	3	5
YnfA	1	2	2
YdgC	1	2	3
CrcB	1	4	5
SugE	2	3	4

Table 6.6: Number of groups predicted for each of the dual topology proteins.

		number %	weight %	helices
EmrE	largest	0.8	0.79	3
	2nd largest	0.2	0.21	4
YnfA	largest	N/A		
	2nd largest	N/A		
YdgC	largest	N/A		
	2nd largest	N/A		
CrcB	largest	N/A		
	2nd largest	N/A		
SugE	largest	0.6	0.59	4
	2nd largest	0.4	0.41	4

(a) 10 paths

		number %	weight %	helices
EmrE	largest	0.61	0.68	3
	2nd largest	0.27	0.26	4
YnfA	largest	0.63	0.77	4
	2nd largest	0.37	0.23	4
YdgC	largest	0.87	0.92	4
	2nd largest	0.13	0.08	3
CrcB	largest	0.83	0.89	4
	2nd largest	0.09	0.05	4
SugE	largest	0.49	0.57	4
	2nd largest	0.50	0.43	4

(b) 100 paths.

		number %	weight %	helices
EmrE	largest	0.5	0.62	3
	2nd largest	0.24	0.25	4
YnfA	largest	0.52	0.63	4
	2nd largest	0.48	0.37	4
YdgC	largest	0.79	0.88	4
	2nd largest	0.19	0.12	3
CrcB	largest	0.72	0.80	4
	2nd largest	0.13	0.09	4
SugE	largest	0.57	0.51	4
	2nd largest	0.32	0.42	4

(c) 1000 paths

Table 6.6: Information about two largest predicted groups for each of the dual topology proteins. 'N/A' signifies that only a single group has been predicted. Number of helices gives an indication of the topology. When the top two groups both have 4 helices, both valid topologies are predicted (as there only two valid topologies with 4 transmembrane helices).

break the resulting predictions into groups having same sidedness and number of helices, the number of the groups along with several statistics about the two largest groups can be found in Table 6.6. For one thousand paths and one hundred paths, in three out of the five cases, the two top clusters actually show the two different topologies. Unfortunately, the relative difference between the masses or sizes of the groups does not seem to be a predictor. For example for EmrE the two largest clusters are more balanced then in CrcB, but its largest cluster predicts the number of helices incorrectly.

The last intersting question we want to address is the distribution of probability mass in the dual topology proteins. The results of this experiments can be found in Table 6.7. The probability mass of the predictions is very high. This is likely due to strong transmembrane signals and the fact that all of the dual topology proteins used are of a very short length, allowing for only small variation in the length of between helix distance.

Overall, in cases when one is looking to find topologies for the dual topology proteins, is seems that k-best paths provides a simple way to find both topologies. However, due to the small sample size the conclusion is preliminary.

	1 path	10 paths	100 paths	1000 paths	10000 paths
EmrE	0.040	0.16	0.38	0.66	0.88
YnfA	0.017	0.091	0.26	0.57	0.87
YdgC	0.047	0.19	0.44	0.74	0.93
CrcB	0.021	0.081	0.23	0.49	0.76
SugE	0.0039	0.034	0.15	0.40	0.73
Avg	0.026	0.11	0.29	0.57	0.83

(a) paths

	1 path	10 paths	100 paths	1000 paths	10000 paths
EmrE	0.061	0.20	0.44	0.71	0.90
YnfA	0.017	0.091	0.27	0.59	0.89
YdgC	0.047	0.19	0.47	0.76	0.95
CrcB	0.021	0.090	0.25	0.52	0.78
SugE	0.0039	0.046	0.18	0.46	0.78
Avg	0.030	0.12	0.32	0.61	0.86

(b) labels

Table 6.7: Total probability mass of paths and labellings for dual topology proteins.

Chapter 7

Conclusion

This thesis consists of two major parts: algorithmic and experimental. In the algorithmic part we describe a new algorithm for finding the k-best paths in an HMM, as well as several algorithms for finding coalescence points in tree based approaches to the Viterbi paths storage. In the experimental part of this thesis, we describe what information can be extracted from the k-best paths, including extraction of the correct prediction, confidence in the prediction and alternative explanations.

The naive extention of the Viterbi algorithm to finding k-best paths in the HMM is very memory inefficient. We present a new, tree based, k-best algorithm that keeps minimum required information for the k-best paths. We show that this algorithm achieves significant space savings at the cost of approximately doubling the runtime.

We also present a theoretical way to further improve space efficiency by finding coalescence points. We present several algorithms for finding coalescence points in the k-best tree and argue that the algorithm based on simple storage of end points of the intervals of annotation is both simple and very efficient.

In the experimental section we first look at the quality of the predictions of the k-best paths. We conclude that for the Phobius model for transmembrane topology prediction, k-best paths contain more information, for high values of k, than the native 1-best decoding. However, when we attempt to extract this information using groupings of paths the results are not encouraging. The largest and the heaviest groups do not perform better then 1-best algorithm, independent of the agglomeration method used on them. Additionally we find that within a single group the paths are very similar to each other, thus agglomeration method is used to find consensus prediction does not influence the results significantly.

The k-best paths contain more information then the prediction itself. We show that we can also use them to judge our confidence in the prediction, moreover we show two distinct ways to do that. We show that the probability mass that top paths occupy helps to estimate confidence. As expected, on average, more of the

probability mass is allocated to the top paths with correct predictions than to paths giving the incorrect predictions. The second way to estimate confidence is to look at the composition of the top paths. In cases when all of the top paths, for large values of k, predict the same number of helices the prediction is likely to be correct. For this method of confidence estimation the key point is the number of paths that predict the same number of helices. We show that the moment when the first path with the different number of predicted helices appears indicates the degree of the confidence in the prediction provided by these top paths.

We also explore the ability of the k-best paths to provide alternative explanations. To achieve this we look at the results for the dual topology proteins, known for their ability to assume two different topologies on the membrane. We show that, in cases when we are specifically looking for the alternative explanations, we fairly often do find them in the two largest groups. However, the data set for these experiments is very small, resulting in low confidence in the pattern.

7.1 Future work

There are several interesting extensions to this work. The most promising one is the creation of new tree based decoding algorithms for the General HMMs (GHMMs). General HMMs are HMMs with arbitrary probability distribution allowed in the states. Due to arbitrary distribution in states they have $O(n^2)$ run time in the worst case. However, in the most common use of GHMMs, gene finding, the decoding "usually" takes $O(n)$ time. $O(n)$ decoding implies that the general states on average "touch" constant number of previous positions, meaning that a tree-based approach likely will work well.

One reason the k-best paths for gene finding is so interesting is the alternative explanations that k-best paths can provide. In gene finding, alternative explanations could correspond to alternative splicing of a gene. Alternative splicing [30], a process which leads to creation of different proteins form a single peace of DNA, has a large role in providing protein variability and specificity, making their prediction a very practical and important problem.

On the theoretical side, we have presented different algorithms for coalescence point detection. An interesting question is to study this probabilistically: what is the expected distance between coalescence points? It is easy to show that the location of coalescence points depends on all the components of HMM decoding: the HMM structure, its transition probabilities, its emission probabilities and the sequence itself. It is interesting to study the dependence of coalescence point location on each of these, as well as developing a method which, given an HMM, would compute the expected distance between coalescence points for the average sequence emitted by this HMM.

Bibliography

[1] Orientations of Proteins in Membranes (OPM) database. http://opm.phar.umich.edu/?images=all.

[2] P.G. Bagos, T.D. Liakopoulos, and S.J. Hamodrakas. Evaluation of methods for predicting the topology of β-barrel outer membrane proteins and a consensus prediction method. *BMC Bioinformatics*, 6(1):7, 2005.

[3] P.G. Bagos, T.D. Liakopoulos, I.C. Spyropoulos, and S.J. Hamodrakas. A Hidden Markov Model method, capable of predicting and discriminating β-barrel outer membrane proteins. *BMC Bioinformatics*, 5(1):29, 2004.

[4] A. Bairoch, R. Apweiler, C.H. Wu, W.C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, et al. The universal protein resource (UniProt). *Nucleic Acids Research*, 33(Database Issue):D154, 2005.

[5] B. Brejova, D.G. Brown, and T. Vinar. The most probable labeling problem in HMMs and its application to bioinformatics. In *Proceedings of WABI*, pages 426–437, 2004.

[6] M.G. Claros and G. von Heijne. TopPred II: an improved software for membrane protein structure predictions. *Bioinformatics*, 10(6):685–686, 1994.

[7] J.M. Cuthbertson, D.A. Doyle, and M.S.P. Sansom. Transmembrane helix prediction: a comparative evaluation and analysis. *Protein Engineering Design and Selection*, 18(6):295–308, 2005.

[8] R. Durbin, S.R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge University Press, Cambridge, UK, 1998.

[9] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

[10] J.A. Grice, R. Hughey, and D. Speck. Reduced space sequence alignment. *Bioinformatics*, 13(1):45–53, 1997.

[11] T. Hessa, H. Kim, K. Bihlmaier, C. Lundin, J. Boekel, H. Andersson, I.M. Nilsson, S.H. White, and G. von Heijne. Recognition of transmembrane helices by the endoplasmic reticulum translocon. *Nature*, 433:377–381, 2005.

[12] L. Kall, A. Krogh, and E.L.L. Sonnhammer. A combined transmembrane topology and signal peptide prediction method. *Journal of Molecular Biology*, 338(5):1027–1036, 2004.

[13] E. Keibler, M. Arumugam, and M.R. Brent. The Treeterbi and Parallel Treeterbi algorithms: efficient, optimal decoding for ordinary, generalized and pair HMMs. *Bioinformatics*, 23(5):545–554, 2007.

[14] A. Krogh. Two methods for improving performance of a HMM and their application for gene finding. In *Proceedings of ISMB*, pages 179–186, 1997.

[15] A. Krogh, B.E. Larsson, G. von Heijne, and E.L.L. Sonnhammer. Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes. *Journal of Molecular Biology*, 305(3):567–580, 2001.

[16] L.J. McGuffin, K. Bryson, and D.T. Jones. The PSIPRED protein structure prediction server. *Bioinformatics*, 16(4):404–405, 2000.

[17] S. Moller, M.D.R. Croning, and R. Apweiler. Evaluation of methods for the prediction of membrane spanning regions. *Bioinformatics*, 17(7):646–653, 2001.

[18] K. Munch and A. Krogh. Automatic generation of gene finders for eukaryotic species. *BMC Bioinformatics*, 7(1):263, 2006.

[19] L. Rabiner and B.H. Juang. *Fundamentals of speech recognition*. PTR Prentice Hall, 1993.

[20] M. Rapp. *The Ins and Outs of membrane proteins*. PhD thesis, 2006. Stockholm University, Faculty of Science, Department of Biochemistry and Biophysics Stockholm.

[21] M. Rapp, E. Granseth, S. Seppaa, and G. Von Heijne. Identification and evolution of dual-topology membrane proteins. *Nature structural and molecular biology*, 13(2):112–116, 2006.

[22] B. Rost, P. Fariselli, and R. Casadio. Topology prediction for helical transmembrane proteins at 86% accuracy. *Protein Science*, 5(8):1704–1718, 1996.

[23] E.L.L. Sonnhammer, S.R. Eddy, and R. Durbin. Pfam: a comprehensive database of protein domain families based on seed alignments. *Proteins: Structure, Function, and Genetics*, 28(3):405–420, 1997.

[24] R. Sramek, B. Brejova, and T. Vinar. On-line Viterbi algorithm for analysis of long biological sequences. In *Proceedings of WABI*, pages 240–251, 2007.

[25] M. Stanke, R. Steinkamp, S. Waack, and B. Morgenstern. AUGUSTUS: a web server for gene finding in eukaryotes. *Nucleic acids research*, 32(Web Server Issue):W309, 2004.

[26] G.E. Tusnady, Z. Dosztanyi, and I. Simon. Transmembrane proteins in the Protein Data Bank: identification and classification. *Bioinformatics*, 20(17):2964–2972, 2004.

[27] G.E. Tusnady and I. Simon. Principles governing amino acid composition of integral membrane proteins: application to topology prediction. *Journal of Molecular Biology*, 283(2):489–506, 1998.

[28] H. Viklund, A. Bernsel, M. Skwark, and A. Elofsson. SPOCTOPUS: a combined predictor of signal peptides and membrane protein topology. *Bioinformatics*, 24(24):2928–2929, 2008.

[29] C.S. Wang and R.S. Chang. Parallel maximal cliques algorithms for interval graphs with applications. *Proceedings of ISPAN*, pages 89–96, 1994.

[30] J. D. Watson, T. A. Baker, S. P. Bell, A. Gann, M. Levine, and R. Losick. *Molecular Biology of the Gene (6th Edition)*. Benjamin Cummings, 2007.

Appendix A

Agglomeration methods data

Correctness measure	10 paths (of possible 30)	100 paths (of possible 24)	1000 paths (of possible 19)
0-off	1	1	1
1-off	8	8	8
2-off	14	13	11
3-off	19	18	13
4-off	23	20	15
5-off	25	21	16
Phobius	29	23	18

(a) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Unweighted average

Correctness measure	10 paths (of possible 30)	100 paths (of possible 24)	1000 paths (of possible 19)
0-off	2	0	0
1-off	10	9	8
2-off	13	14	11
3-off	18	16	13
4-off	22	19	15
5-off	25	21	16
Phobius	29	23	18

(b) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Weighted average

Correctness measure	10 paths (of possible 4)	100 paths (of possible 10)	1000 paths (of possible 17)
0-off	1	0	0
1-off	2	2	6
2-off	3	4	9
3-off	4	6	12
4-off	4	7	14
5-off	4	8	15
Phobius	4	10	17

(c) Distribution of correctness in cases when two different helix counts where predicted. Unweighted average.

Correctness measure	10 paths (of possible 4)	100 paths (of possible 10)	1000 paths (of possible 17)
0-off	1	0	0
1-off	2	2	6
2-off	3	3	8
3-off	4	6	13
4-off	4	7	14
5-off	4	8	15
Phobius	4	10	17

(d) Distribution of correctness for in cases when two different helix counts where predicted. Weighted average.

Table A.0: For data set I, generating a consensus prediction using averaging. "Of possible" indicates the number of proteins for which the information is available, meaning the number of cases where the overall number of helices and their sidedness matches the correct ones.

Correctness measure	10 paths (of possible 123)	100 paths (of possible 93)	1000 paths (of possible 56)
0-off	3	2	1
1-off	7	6	2
2-off	15	11	3
3-off	20	18	5
4-off	40	29	12
5-off	54	40	22
Phobius	122	92	55

(a) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Unweighted average

Correctness measure	10 paths (of possible 123)	100 paths (of possible 93)	1000 paths (of possible 56)
0-off	3	2	1
1-off	8	5	2
2-off	15	11	3
3-off	19	17	6
4-off	38	29	12
5-off	59	41	20
Phobius	122	92	55

(b) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Weighted average

Correctness measure	10 paths (of possible 19)	100 paths (of possible 51)	1000 paths (of possible 93)
0-off	0	1	1
1-off	2	9	11
2-off	7	12	22
3-off	8	15	30
4-off	10	23	44
5-off	11	29	59
Phobius	17	49	90

(c) Distribution of correctness for in cases when two different helix counts where predicted. Unweighted average.

Correctness measure	10 paths (of possible 19)	100 paths (of possible 51)	1000 paths (of possible 93)
0-off	0	2	2
1-off	4	9	13
2-off	8	12	21
3-off	8	15	29
4-off	9	22	45
5-off	10	29	57
Phobius	17	49	90

(d) Distribution of correctness for in cases when two different helix counts where predicted. Weighted average.

Table A.1: For data set II, generating a consensus prediction using averaging. "Of possible" indicates the number of proteins for which the information is available, meaning the number of cases where the overall number of helices and their sidedness matches the correct ones.

Correctness measure	10 paths (of possible 30)	100 paths (of possible 24)	1000 paths (of possible 19)
0-off	2	1	1
1-off	8	8	8
2-off	11	11	10
3-off	17	15	13
4-off	23	19	15
5-off	25	21	16
Phobius	29	23	18

(a) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Unweighted average

Correctness measure	10 paths (of possible 30)	100 paths (of possible 24)	1000 paths (of possible 19)
0-off	2	2	1
1-off	8	8	8
2-off	10	10	10
3-off	16	14	13
4-off	23	19	15
5-off	25	21	16
Phobius	29	23	18

(b) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Weighted average

Correctness measure	10 paths (of possible 4)	100 paths (of possible 10)	1000 paths (of possible 17)
0-off	1	1	2
1-off	2	2	4
2-off	3	3	7
3-off	4	6	11
4-off	4	7	13
5-off	4	8	15
Phobius	4	10	17

(c) Distribution of correctness in cases when two different helix counts where predicted. Unweighted average.

Correctness measure	10 paths (of possible 4)	100 paths (of possible 10)	1000 paths (of possible 17)
0-off	4	10	17
1-off	1	1	4
2-off	2	2	6
3-off	3	3	7
4-off	4	6	12
5-off	4	7	14
Phobius	4	8	15

(d) Distribution of correctness for in cases when two different helix counts where predicted. Weighted average.

Table A.2: For data set I, generating a consensus prediction using majority voting for the start and length of the segment. "Of possible" indicates the number of proteins for which the information is available, meaning the number of cases where the overall number of helices and their sidedness matches the correct ones.

Correctness measure	10 paths (of possible 123)	100 paths (possible 93)	1000 paths (of possible 56)
0-off	0	2	1
1-off	4	3	1
2-off	11	8	3
3-off	20	13	4
4-off	40	29	13
5-off	51	35	15
Phobius	122	93	55

(a) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Unweighted average

Correctness measure	10 paths (of possible 123)	100 paths (possible 93)	1000 paths (of possible 56)
0-off	1	2	1
1-off	5	4	1
2-off	12	8	3
3-off	19	12	4
4-off	38	29	13
5-off	49	35	15
Phobius	122	92	55

(b) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Weighted average

Correctness measure	10 paths (of possible 19)	100 paths (of possible 51)	1000 paths (of possible 93)
0-off	2	1	4
1-off	4	8	10
2-off	7	13	17
3-off	8	14	24
4-off	10	20	38
5-off	11	25	49
Phobius	17	49	91

(c) Distribution of correctness in cases when two different helix counts where predicted. Unweighted average.

Correctness measure	10 paths (of possible 19)	100 paths (of possible 51)	1000 paths (of possible 93)
0-off	2	1	2
1-off	5	7	11
2-off	8	14	20
3-off	8	15	24
4-off	10	20	40
5-off	12	26	51
Phobius	17	50	91

(d) Distribution of correctness in cases when two different helix counts where predicted. Weighted average.

Table A.3: For data set II, generating a consensus prediction using majority voting for the start and length of the segment. "Of possible" indicates the number of proteins for which the information is available, meaning the number of cases where the overall number of helices and their sidedness matches the correct ones.

Correctness measure	10 paths (of possible 30)	100 paths (of possible 24)	1000 paths (of possible 19)
0-off	2	2	1
1-off	8	8	8
2-off	11	11	10
3-off	17	16	13
4-off	23	20	15
5-off	25	21	16
Phobius	29	23	18

(a) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Unweighted average

Correctness measure	10 paths (of possible 30)	100 paths (of possible 24)	1000 paths (of possible 19)
0-off	2	2	1
1-off	8	8	8
2-off	11	11	10
3-off	17	15	13
4-off	23	20	15
5-off	25	21	16
Phobius	29	23	18

(b) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Weighted average

Correctness measure	10 paths (of possible 4)	100 paths (of possible 10)	1000 paths (of possible 17)
0-off	1	1	2
1-off	2	2	6
2-off	3	3	8
3-off	4	5	12
4-off	4	6	13
5-off	4	8	15
Phobius	4	10	17

(c) Distribution of correctness for in cases when two different helix counts where predicted. Unweighted average.

Correctness measure	10 paths (of possible 4)	100 paths (of possible 10)	1000 paths (of possible 17)
0-off	1	1	2
1-off	2	2	5
2-off	3	3	6
3-off	4	5	11
4-off	4	6	13
5-off	4	8	15
Phobius	4	10	17

(d) Distribution of correctness for in cases when two different helix counts where predicted. Weighted average.

Table A.4: For data set I, generating a consensus prediction using per position voting. "Of possible" indicates the number of proteins for which the information is available, meaning the number of cases where the overall number of helices and their sidedness matches the correct ones.

Correctness measure	10 paths (out of 189, of possible 123)	100 paths (out of 120, of possible 93)	1000 paths (out of 64, of possible 56)
0-off	0	3	1
1-off	6	6	1
2-off	12	8	3
3-off	19	13	5
4-off	39	28	13
5-off	53	39	19
Phobius	122	92	55

(a) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Unweighted average

Correctness measure	10 paths (out of 189, of possible 123)	100 paths (out of 120, of possible 93)	1000 paths (out of 64, of possible 56)
0-off	1	2	1
1-off	7	5	1
2-off	13	8	3
3-off	19	14	5
4-off	39	29	13
5-off	2	39	18
Phobius	122	92	55

(b) Distribution of correct consensus in cases when all predicted paths have the same number of helices. Weighted average

Correctness measure	10 paths (out of 58, of possible 19)	100 paths (out of 127, of possible 51)	1000 paths (out of 183, of possible 93)
0-off	1	1	2
1-off	4	8	13
2-off	6	13	20
3-off	8	14	27
4-off	10	20	42
5-off	12	27	53
Phobius	17	49	91

(c) Distribution of correctness in cases when two different helix counts where predicted. Unweighted average.

Correctness measure	10 paths (out of 58, of possible 19)	100 paths (out of 127, of possible 51)	1000 paths (out of 183, of possible 93)
0-off	1	1	1
1-off	4	7	12
2-off	7	12	18
3-off	8	15	27
4-off	10	19	39
5-off	12	26	51
Phobius	17	49	90

(d) Distribution of correctness in cases when two different helix counts where predicted. Weighted average.

Table A.5: For data set II, generating a consensus prediction using per position voting. "Of possible" indicates the number of proteins for which the information is available, meaning the number of cases where the overall number of helices and their sidedness matches the correct ones.

Correctness measure	10 paths (of possible 30)	100 paths (of possible 24)	1000 paths (of possible 19)
0-off	1	1	1
1-off	8	8	8
2-off	15	13	12
3-off	19	18	13
4-off	23	20	15
5-off	25	21	16
Phobius	29	23	18

(a) Distribution of correct consensus in cases when all predicted paths have the same number of helices.

Correctness measure	10 paths (of possible 4)	100 paths (of possible 10)	1000 paths (of possible 17)
0-off	1	0	0
1-off	2	2	7
2-off	3	4	9
3-off	4	6	12
4-off	4	7	14
5-off	4	8	15
Phobius	4	10	17

(b) Distribution of correctness for in cases when two different helix counts where predicted.

Table A.7: For data set I, generating consensus using averaging on the weights of the labels. "Of possible" indicates the number of proteins for which the information is available, meaning the number of cases where the overall number of helices and their sidedness matches the correct ones.

Correctness measure	10 paths (of possible 123)	100 paths (of possible 93)	1000 paths (of possible 56)
0-off	3	2	1
1-off	6	5	2
2-off	14	11	3
3-off	19	18	5
4-off	42	29	12
5-off	54	40	22
Phobius	122	92	55

(a) Distribution of correct consensus in cases when all predicted paths have the same number of helices.

Correctness measure	10 paths (of possible 19)	100 paths (of possible 51)	1000 paths (of possible 93)
0-off	0	1	2
1-off	2	7	9
2-off	7	12	23
3-off	8	15	32
4-off	10	23	44
5-off	11	29	58
Phobius	17	49	90

(b) Distribution of correctness for in cases when two different helix counts where predicted.

Table A.2: For data set II, generating a consensus prediction using averaging on the weights of the labels. "Of possible" indicates the number of proteins for which the information is available, meaning the number of cases where the overall number of helices and their sidedness matches the correct ones.