Prediction of Protein-Protein Interaction Sites with Conditional Random Fields

Dissertation

Zur Erlangung des wissenschaftlichen Doktorgrades

"Doktor rerum naturalium"

an der Georg-August-Universität Göttingen

vorgelegt von

Zhijie Dong

aus Jiangsu

Göttingen, 2012

1. Referent: Professor Dr. Stephan Waack
2. Korreferent: Professor Dr. Carsten Damm

Tag der mündlichen Prüfung: 27.04.2012

To my family

Acknowledgements

First and foremost I would like to thank my supervisor Professor Stephan Waack. This work can have never been done without his help and mentoring. Besides, it is indeed appreciated for his understanding and support since the birth of my son. I return thanks to my co-advisor Professor Carsten Damm for the proof-reading of this thesis and his inspiring suggestions. I am also grateful to Professor Mario Stanke for discussions and cooperations by developing the new algorithm. I would like to thank my work group colleagues Mehmet Gültas, Roman Asper who provided meaningful features for the prediction. Thanks should also be given to Moritz Maneke for the implementation of the inference algorithm. Special thank goes to Liang Shi for his suggestion about English writing.

I am grateful for the financial and scientific support by the DFG research training group 1023 "Identification in Mathematical Models: Synergy of Stochastic and Numerical Methods".

Especially, I would like to thank my husband Keyu Wang who always supports me in both my research and my daily life.

Contents

List of Figures

LIST OF FIGURES

List of Tables

Glossary

3D three dimensional

ASA accessible surface area

BFGS Broyden-Fletcher-Goldford-Shanno method

CRF conditional random field

DNA deoxyribonucleic acid

DSSP dictionary of secondary structure of proteins

FN the number of false negatives

FP the number of false positives

HMM hidden Markov model

i.i.d. independent and identically distributed

L-BFGS limited memory Broyden-Fletcher-Goldford-Shanno method

lCRF linear-chain conditional random field

MAP maximum a posteriori

MSA multiple sequence alignment

NER named entity recognition tasks

pCRF pairwise conditional random field

PDB protein data bank

GLOSSARY

RASA relative accessible surface area

SVM support vector machine

TN the number of true negatives

TP the number of true positives

Å angstrom: 1×10^{-10} metres

1

Introduction

Application of mathematics, statistics, and information theory to the field of molecular biology is an established field of science. This work employs the statistical models and computer science approach to study and analyze biological datasets with the help of powerful computers. The aim is to predict protein-protein interaction sites which play a central role in protein functions.

Numerical valued-based methods, like linear regression [24][29], support vector machine [3][6][10][15][25][31], neural network [4][20][23][41][50], have been widely used in protein-protein interface prediction. Apart from these, many groups have also applied probabilistic methods to the prediction, for example naive Bayesian [21], Bayesian networks [30], hidden Markov models [48], etc.

Our approach is based on conditional random fields (CRFs) which belongs to the probabilistic methods. Recently there has been an explosion of interest in CRFs, with successful applications including text processing [9][18][19], computer vision [5], and bioinformatics [17][32]. Besides, the linear-chain conditional random field which is the simplest example of the model has been used to predict protein interfaces [39]. However, a linear-chain cannot represent the three dimensional (3D) structure of a protein. In this work, we have been concentrating on pairwise conditional random fields based on general graphs, since it can also describe the spatial neighborhood of a protein.

1.1 Biological Background

Proteins are large organic compounded of 20 different amino acids arranged in a linear chain, folding themselves into a three-dimensional structure. Like other biological macromolecules, proteins are essential parts of organisms and participate in every process within cells. Proteins can also work together to achieve a particular function, and often get clustered to form stable protein complexes.

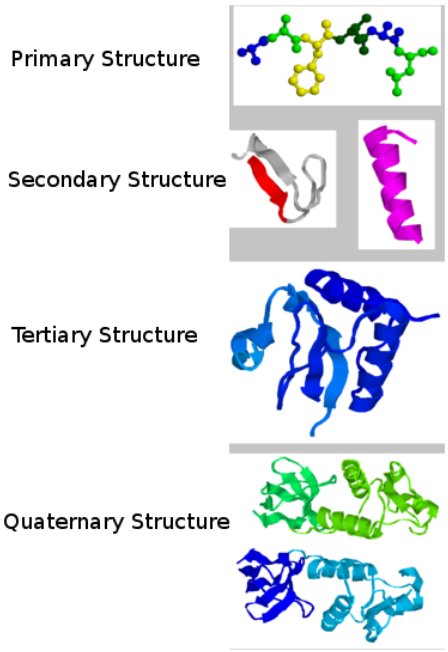

Figure 1.1: A protein from primary to quaternary structure.

In general, there are four levels of protein structures shown in Figure 1.1. The sequence of amino acids is called the primary structure which is considered as the backbone of a protein. An amino acid comprising one nitrogen and two carbon atoms is bound to various hydrogen and oxygen molecules (see Figure 1.2). The central carbon (C^{α}) is linked to the unit "R", single atom or a group of atoms. The R unit distinguishes different amino acids. Two amino acids bind together by releasing a water molecule and the remaining parts of the amino acids are called *amino acid residues*. Amino acids bind into polymer chains, forming the primary structure of proteins which are typically hundreds or thousands of amino acids long.

The next level is the secondary structure which is the substructure of local segments

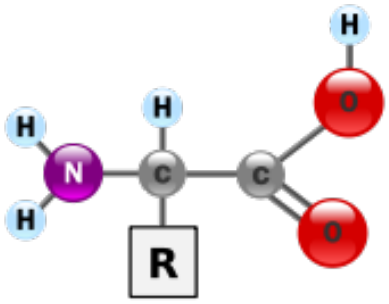

Figure 1.2: The general structure of an amino acid.

of proteins. Different secondary patterns might present in one single protein. Two common patterns are shown in Figure 1.1. On the right side, it is called $\alpha-$helix, whereas on the left side is $\beta-$sheet. However, secondary structure does not describe specific atomic positions in three-dimensional space which are considered in the tertiary structure. This level of structure defines the overall fold of a protein and finishes the folding with a 3D form. Some proteins might be functional as monomers, while more proteins assemble with others and build together a complex called the quaternary structure. Protein complexes are a form of quaternary structure which is shaped by protein-protein interactions.

Signals from the interactions between proteins play an important role in many biological processes. Identification of the interaction sites can improve our understanding of protein functions and provide a basis for the new therapeutic approaches to treat diseases (e.g. cancer).

As is mentioned earlier, a protein is composed of 20 types of amino acid residues. For simplicity, we call them residues. Identifying the interaction sites is to detect the residues involved in the interaction, which are called *interface residues*. If a protein interacts with another protein, its corresponding interface residues form an *interface* of this protein by this interaction.

Note that one protein can have more than one interfaces, because it can interact with different proteins. For example, a protein complex shown in Figure 1.3 consists of three proteins A, B and C. Protein B interacts with both protein A and C, resulting in two different interfaces: the one in red and the other in magenta. Different interfaces in one protein might overlap, while two proteins in the same complex can have no

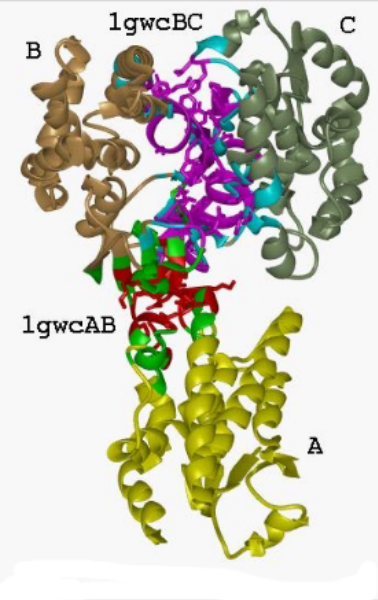

Figure 1.3: A protein complex consists of three proteins (taken from [40]).

interaction between each other. Protein A and C in the figure is an example. Although both of them interact with protein B, they are not close enough to form an interface.

In this work, we focus on the interactions between protein pairs which are also called *protein dimers*. Multiple interactions such as the example shown in Figure 1.3 will not be taken into account.

1.2 Protein-Protein Interaction Sites Prediction

Although protein-protein interaction sites can be determined experimentally, it is an expensive and time consuming process. Therefore, a large number of methods combining bioinformatics and structral biology have been developed to predict protein interfaces.

Since our task is to predict the interaction sites (interface residues) involved in an interaction between two proteins, the problem can be formulated as follows:

- Given a protein with three dimensional unbound structure, we assume that it forms a complex (or more exactly a dimer) when it interacts with another protein.
- The goal is to find the residues of the given protein, which participate in the interaction with the protein partner.

It is a typical labeling problem and *interface* is generally a binary variable: a residue is either in the interface, or not. Each residue in the given protein will be assigned a state label, either I (interface residue) or N (non-interface residue).

In reality, proteins are mainly folded into three dimensional structures. Hence the most important dependencies are the spatial relationships. Crucial is to make a meaningful prediction using the given information, especially the 3D structure of the protein. As we know, the 3D structure is represented by the atomic positions of the residues which can be obtained from the *Protein Data Bank*. A residue is usually described by a three dimensional point, the mass center of its heavy atoms. In this way, a protein can be modeled by a graph with nodes referred to the three dimensional residue positions. The edges in the graph represent the spatial neighborhood of the residues in the protein.

Due to the important spatial information, we use a graphical model to solve the labeling problem and we select the conditional random fields proposed by *Lafferty* [26]. CRF is a stochastic model based on a graph. One of the benefits is that the spatial information of a protein can be directly taken into the model. Moreover, CRF calculates the conditional distribution instead of the joint distribution, which means that the given protein is only considered as a condition in the model but not generated from the model. In practice, it is not easy to apply a general CRF directly, because training the model is intractable. Instead of the general CRF, we here develop a variant called the pairwise conditional random field (pCRF) and apply the pCRF to predict the protein interfaces.

2

Statistical Models

The models in this thesis are mostly of statistical nature. Simply speaking, a statistical model can be considered a pair (X, P) where X denotes the set of observations and P is the set of possible probability distributions on X and (or) some properties of X. Most of the statistical models are not *white box* models, which means they contain more or less parameters to be estimated. These parameters will be used to fit the model to the system it shall describe. The process of estimating the parameters in a model is called *training*. In order to evaluate the performance of a model, we usually divide the data into two disjointed sets. One set is used in the training, where the data are called *training data*. The other is used to test the quality of the model and the data are called *test data*. There are many different training principles and methods for the applications. Choosing an appropriate one is very important to obtain meaningful results.

To sum up, the following three main points should be taken into consideration by using a statistical model:

- Describing the possible distributions on the observations

  The discription of the distributions is given in a mathematical form and generally contains some unknown parameters.

- Using optimization techniques to estimate the parameters in the model

  The goal is to make the model best fit the training data.

- Evaluating the model on the test data

It will be checked based on some measures whether the model fits the system.

2.1 Some Useful Notations

Now we introduce some useful notations that will be used in this thesis. Let $\{Y_1, Y_2, \ldots, Y_n\}$ be a set of n discrete random variables and y_i a possible *realization* of the random variable $Y_i \in \mathcal{L}$, where $\mathcal{L}$ is a finite set of possible realizations. We consider the sequence $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_n)$ and a possible realization of this random sequence $\mathbf{y} = (y_1, y_2, \ldots, y_n)$.

The *joint probability mass function* $P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n)$ is simply written as $P(\mathbf{y})$. Analogously, $P(\mathbf{y}|\mathbf{x})$ denotes the *conditional probability* that the sequence is labeled $\mathbf{y}$ conditioned by a given observation $\mathbf{x}$.

We also mention here the common inference problem by using a statistical model, which is called *maximum a posteriori (MAP) assignment*. This problem is to find the maximal probability assignment by a given observation $\mathbf{x}$

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x}). \tag{2.1}$$

In the following chapters, I will introduce how to solve such an inference problem by using a conditional random field.

2.2 Graphical Models

Since a statistical model represents the behavior of an object through random variables and their associated probability distributions, the dependencies between the random variables should be considered. The simplest way to highlight these relationships is to get help from a graph. A statistical model represented via a graph is said to be a *graphical model* which is widely used in many applications, because they naturally accommodate the need to fuse multiple sources of information.

A graph is usually represented mathematically by a pair $G = (V, E)$, consisting of a set V of nodes and a set E of edges. G is called a *directed graph*, if the edges in G have orientation; Otherwise G is an *undirected graph*. According to this, graphical models can be classified into *directed graphical models* and *undirected graphical models*.

Let $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_n)$ denote a sequence of random variables indexed by the set $V = \{1, 2, \ldots, n\}$. The corresponding graph of the model $P(\mathbf{Y})$ is $G = (V, E)$, where

$(i, j) \in E$ indicates the dependency between random variables Y_i and Y_j. Therefore, the graph G also represents the conditional independencies between the variables.

We focus on the undirected graphical models and start with the definition of *Markov property*:

Definition 2.1 (Markov, 1954)**.**
A system is said to have the **Markov property**, *if it holds*

$$P(y_i|\{y_k, k \neq i\}) = P(y_i|\{y_k, k \in \mathcal{N}(i)\}), \tag{2.2}$$

where $\mathcal{N}(i)$ *denotes the set of neighbors of node* i *in* G.

The Markov property is used to limit the dependencies to a small number of variables, otherwise it could be intractable to compute the distribution if the local probability is represented over a large number of random variables. Besides, we introduce some other definitions from the graph theory.

Definition 2.2.
Let $G = (V, E)$ *be an undirected graph. A* **clique** C *in* G *is a subset of the node set* $C \subseteq V$, *so that for every pair of nodes in* C, *there exists an edge connecting the two. A clique* C *is said to be a* **maximal clique**, *if every node not in* C *is missing an edge to at least one node in* C.

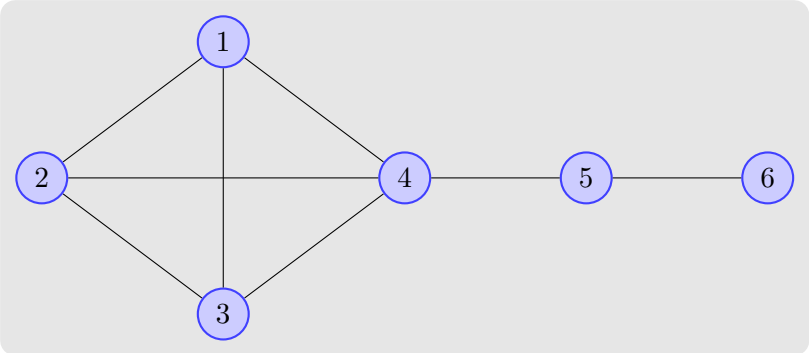

Figure 2.1: An undirected graph with 3 maximal cliques.

A graph may have many maximal cliques of different sizes. Figure 2.1 shows that $\{1, 2, 3, 4\}$ is a maximal clique with four nodes. Besides, $\{4, 5\}$ and $\{5, 6\}$ are the other maximal cliques in the same graph, of size two respectively.

A commonly used representation of the distributions of an undirected graphical model is factorized over the cliques of the graph G.

$$P(\mathbf{y}) = \frac{1}{Z} \prod_{C} \Phi_C(\mathbf{y}_C), \tag{2.3}$$

where C denotes a clique in G, and Φ_C is referred to a *potential function* over the clique C with

$$\Phi_C : \mathcal{L}^{|C|} \to \mathbb{R},$$

where $|C|$ denotes the size of C.

Z in Equation 2.3 is a constant, ensuring that the distribution sums to 1. This kind of constant is called *normalization factor*.

Remark. *Some references confine Equation 2.3 only maximal cliques C. But it does not mean that we can ignore those potential functions whose arguments are not indexed by a maximal clique. The maximal cliques are used to simplify the form of the distribution, because they contain all of the cliques in the graph.*

2.3 Hidden Markov Models

A hidden Markov model (HMM)[36] is a stochastic model based on sequential data. An HMM consists of an alphabet of labels and emission symbols. It is assumed that the labels are hidden from the observer, while only the emission symbols are observable. An HMM is also a typical directed graphical model which can be described in the following figure.

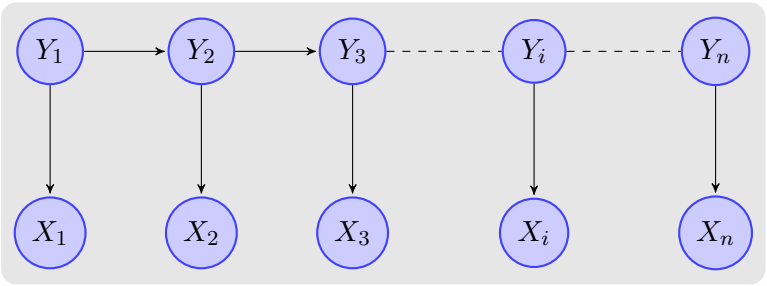

Figure 2.2: Architecture of an HMM with labels Y_i and observations X_i.

We denote $\mathcal{L}$ the set of labels and Σ the set of emission symbols or rather *observations*. An HMM is designed to make inference on the labels $Y_i \in \mathcal{L}$ through the

observation of emissions $X_i \in \Sigma$. Besides, it is shown in Figure 2.2 that not only the labels Y_i but also the observations X_i are generated from the model.

A hidden Markov model obeys the Markov property that the local probability of a random variable depends only on its parents, which means:

1. Each label Y_i (excluding Y_1) depends only on the previous label Y_{i-1}, $i = 2, \ldots, n$;
2. Each observation X_i depends only on the current label Y_i, $i = 1, \ldots, n$.

In this way, the parameters of an HMM are three types, namely *starting probabilities* $P(y_1)$, *transition probabilities* $P(y_i|y_{i-1})$, $i = 2, \ldots, n$, and *emission probabilities* $P(x_i|y_i)$, $i = 1, \ldots, n$. Usually, the Baum-Welch algorithm [35] is used to find the unknown parameters of a hidden Markov model.

The probability that an observation sequence $\mathbf{x}$ is labeled by a label sequence $\mathbf{y}$, can be written as follows

$$P(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{n} P(y_i|y_{i-1})P(x_i|y_i), \tag{2.4}$$

with $P(y_1|y_0) := P(y_1)$.

The next problem is how to use an HMM to make inference on the labels through the given observations. This problem can be solved by the well-known *Viterbi algorithm* [7]. The Viterbi algorithm is a dynamic programming algorithm which efficiently finds the most likely sequence of hidden labels. The complexity of this algorithm is $O(n \times |\mathcal{L}|^2)$.

Hidden Markov models are widely applied in pattern recognition and the field of bioinformatics. However, sequence models are not suitable for graphical data. In the following section, we will introduce the conditional random field which is a model based on a general graph.

2.4 Conditional Random Fields

A conditional random field (CRF) is one of the notable undirected graphical models, which is often used to solve the labeling problems explained as follows.

Let $\mathbf{X}$ be an input variable known as the *observation*. We use $\mathbf{x}$ to represent a possible realization of the observations. The labeling problem is to assign a most probable label to each input variable x_i. In other words, we are looking for a most

probable *label sequence* for the observation. We use $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_n)$ to denote a random output label sequence and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ a possible realization with $y_i \in \mathcal{L}$, where $\mathcal{L}$ is a finite set of labels.

In a CRF with respect to a graph $G = (V, E)$, the output variables are indexed by V and each Y_i is considered to be conditioned upon the observation $\mathbf{X}$.

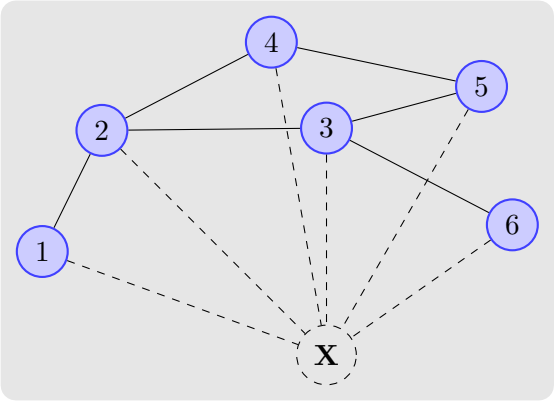

Figure 2.3: Graphical structure of a CRF with observation $\mathbf{X}$.

In Figure 2.3, every node $i \in \{1, 2, 3, 4, 5, 6\}$ is referred to as an output variable Y_i. Although all the output variables are conditioned upon the observation $\mathbf{X}$, but $\mathbf{X}$ itself is not generated from the model. For clarity, we draw this relation with dashed lines, which means that the node $\mathbf{X}$ is actually not included in the graph.

The remarkable characteristic of a CRF is to directly model the conditional distribution $P(\mathbf{Y}|\mathbf{X})$ instead of the joint distribution $P(\mathbf{X}, \mathbf{Y})$. This is a great benefit compared to the traditional graphical models, because representing the joint distribution involves simulating the dependencies among input variables, which can lead to intractable models.

Now we introduce the original definition of a conditional random field which was proposed by *John Lafferty* [26] in 2001.

The random pair $(\mathbf{X}, \mathbf{Y})$ is a **conditional random field**, when conditioned on an observation $\mathbf{X}$, the random variable Y_i obeys the Markov property with respect to the graph:

$$P(y_i|\mathbf{x}, \{y_k, k \neq i\}) = P(y_i|\mathbf{x}, \{y_k, k \in \mathcal{N}(i)\}), \tag{2.5}$$

where $\mathcal{N}(i)$ denotes the set of the neighbors of i.

This is an abstract definition of a CRF with a general graphical structure. As mentioned before, a CRF calculates the conditional distribution $P(\mathbf{Y}|\mathbf{X})$, and we can describe a CRF with an explicit form as follows:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp(\sum_{C \in \mathcal{C}} \langle \mathbf{\Lambda}_C, \mathbf{f}_C(\mathbf{y}_C, \mathbf{x}) \rangle), \tag{2.6}$$

where $\mathcal{C}$ denotes the set of all cliques in the related graph and $Z(x)$ is a normalization factor with

$$Z(\mathbf{x}) = \sum_{\tilde{\mathbf{y}}} \exp(\sum_{C \in \mathcal{C}} \langle \mathbf{\Lambda}_C, \mathbf{f}_C(\tilde{\mathbf{y}}_C, \mathbf{x}) \rangle). \tag{2.7}$$

Similarly to the potential functions in Equation 2.3, we use feature functions $\mathbf{f}_C$ in a CRF. Generally, a feature function is a mapping from every possible label of the clique to a real value. Since the observation $\mathbf{x}$ is a condition in the model, $\mathbf{x}$ behaviors as a parameter in the feature functions. In addition, feature functions are evaluated by a weight vector $\mathbf{\Lambda}$ which represents the importance of the corresponding feature functions.

Using the scalar product in Equation 2.6, all of the weighted feature functions add up over all cliques in the graph. Since the exponential function obeys the basic exponentiation identity, $\exp(a + b) = \exp(a) \cdot \exp(b)$, this conditional distribution can be reformulated as the general representation in Equation 2.3:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C \in \mathcal{C}} \Phi_C(\mathbf{y}_C, \mathbf{x}),$$

with the specifically defined potential functions

$$\Phi_C(\mathbf{y}_C, \mathbf{x}) := \exp(\langle \mathbf{\Lambda}_C, \mathbf{f}_C(\mathbf{y}_C, \mathbf{x}) \rangle). \tag{2.8}$$

In the application of a CRF, we should firstly define some feature functions through the data analysis. The weight vector remains unknown in the model, which can be obtained by training a CRF. In fact, one of the difficulties in applying a general CRF is that training a CRF is intractable, because computing either the marginal distributions or the normalization factor is very expensive. In the following chapters, we introduce two variants of CRFs, namely linear-chain conditional random field and pairwise conditional random field. In a linear-chain CRF, the related graph is restricted to a linear chain and a dynamic computing technique is used in the model training.

A pairwise CRF is however based on a general graph, which retains almost all spatial information as a general CRF. In this case, we select an approximate training method which splits the model into pieces, training independently, and combining the learned parameters into a whole model. In applications, a suitable CRF-variant will be used to solve different problems.

3

Linear-Chain Conditional Random Fields

Firstly, we talk about the simplest but the most important example of CRFs, the linear-chain conditional random field (lCRF), a sequence model based on a graphical structure which is a linear chain. The corresponding graph $G = (V, E)$ in a linear-chain CRF has a particular shape with $V = \{1, 2, \ldots, n\}$ and $E = \Big\{\{i-1, i\}, i = 2, 3, \ldots, n\Big\}$.

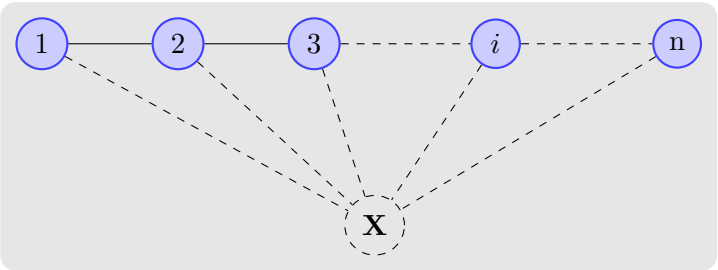

Figure 3.1: Graphical structure of a linear-chain CRF.

There are two types of cliques in such a linear chain, node cliques and edge cliques. It is shown in Figure 3.1 that the graph contains n node cliques $\{i\}, i = 1, 2, \ldots, n$ and $n - 1$ edge cliques $\{i - 1, i\}, i = 2, 3, \ldots, n$.

Like HMMs, linear-chain CRFs are also based on sequential data. However, a CRF is an undirected model and the observation $\mathbf{x}$ is not generated from the model, which is an advantage over an HMM, because $\mathbf{x}$ is anyway given in the application.

3.1 Definition of Linear-Chain CRFs

As we discussed in the section of general CRFs, feature functions over different types of cliques should be defined first. In a linear-chain CRF, feature functions are defined over node cliques and edge cliques, which are called *state feature functions* and *transition feature functions*, respectively.

Let $\mathcal{L}$ denote a finite set of labels and $\mathbf{x} \in \Sigma$ an observation, which will be labeled by a $\mathbf{y} \in \mathcal{L}^n$.

We assume that there are J state feature functions defined on nodes and let s_j denote some state feature function, $j = 1, 2, \ldots, \mathrm{J}$:

$$\begin{aligned} s_j :\ & \mathcal{L} \times \Sigma \to \mathbb{R} \\ & y \times \mathbf{x} \mapsto s_j(y, \mathbf{x}). \end{aligned}$$

Analogously, let t_l be one of the L transition feature functions defined over edges, $l = 1, 2, \ldots, \mathrm{L}$:

$$\begin{aligned} t_l :\ & \mathcal{L} \times \mathcal{L} \times \Sigma \to \mathbb{R} \\ & y \times y' \times \mathbf{x} \mapsto t_l(y, y', \mathbf{x}). \end{aligned}$$

Furthermore, we use ω_j and μ_l to denote the weight of state feature function s_j and transition feature function t_l, respectively.

According to Equation 2.6, the conditional distribution of a linear-chain CRF can be written as

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\Big(\sum_{i=1}^{n}\sum_{j=1}^{\mathrm{J}} \omega_j s_j(y_i, \mathbf{x}) + \sum_{i=2}^{n}\sum_{l=1}^{\mathrm{L}} \mu_l t_l(y_{i-1}, y_i, \mathbf{x})\Big).$$

The formula above can be simplified in terms of edge cliques that are maximal cliques in the graph. First of all, we introduce an extra node 0 to the graph and define

$$t_l(y_0, y_1, \mathbf{x}) := 0, \quad \forall l = 1, 2, \ldots, \mathrm{L}.$$

Since the extra node y_0 does not influnce the model, the conditional distribution can be reformulated as

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\Big(\sum_{i=1}^{n}\Big(\sum_{j=1}^{\mathrm{J}} \omega_j s_j(y_i, \mathbf{x}) + \sum_{l=1}^{\mathrm{L}} \mu_l t_l(y_{i-1}, y_i, \mathbf{x})\Big)\Big).$$

Now we consider a set $\mathcal{F}$ of feature functions over a pair $\{y_{i-1}, y_i\}$ conditioned on $\mathbf{x}$. This set is comprised of state feature functions and transition feature functions, that is

$$\mathcal{F} = \{s_j, j = 1, 2, \ldots, \mathrm{J}\} \cup \{t_l, l = 1, 2, \ldots, \mathrm{L}\}.$$

The size of set $\mathcal{F}$ is then $\mathrm{K} = \mathrm{J} + \mathrm{L}$ and each $f : \mathcal{L} \times \mathcal{L} \times \Sigma^n \to \mathbb{R}, f \in \mathcal{F}$ refers either to a state feature function $f(y_{i-1}, y_i, \mathbf{x}) = s_j(y_i, \mathbf{x})$ or to a transition feature function $f(y_{i-1}, y_i, \mathbf{x}) = t_l(y_{i-1}, y_i, \mathbf{x})$. Then we use f_k to index each feature function in $\mathcal{F}$ and $\lambda_k \in \mathbb{R}$ to denote its weight.

Using the above notations, a linear-chain CRF is defined as follows:

Definition 3.1 (Lafferty, 2001)**.**
Let $\mathbf{x}$ *be an observation over data and* $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ *one of the possible label sequences. Moreover,* $\mathcal{F} = \{f_k, k = 1, 2, \ldots, \mathrm{K}\}$ *denotes a set of real-valued feature functions with a weight vector* $\mathbf{\Lambda} = \{\lambda_k\}_{k=1}^{\mathrm{K}}$*. Then a* **linear-chain conditional random field** *takes the form*

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\Big(\sum_{i=1}^{n} \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_{i-1}, y_i, \mathbf{x})\Big), \tag{3.1}$$

where the normalization factor

$$Z(\mathbf{x}) = \sum_{\tilde{\mathbf{y}} \in \mathcal{L}^n} \exp\Big(\sum_{i=1}^{n} \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(\tilde{y}_{i-1}, \tilde{y}_i, \mathbf{x})\Big). \tag{3.2}$$

Since linear-chain CRFs and HMMs are both based on sequential data, it is interesting to make a comparison between them. The main difference between a (linear-chain) CRF and an HMM is that one computes the conditional distribution, whereas the other calculates the joint distribution. Actually, it is the main difference between a *discrimiative model* and a *generative model*. A model based on the conditional distribution (e.g. CRF) is considered to be discriminative, while a model based on the joint distribution (e.g. HMM) is called generative including a representation of the distribution of the observation sequences.

In practice, most real-world observations have multiple interacting features and long-range dependencies and it is often difficult to model the distribution of $P(\mathbf{x})$. Some generative models, such as HMMs, use the independence assumption to make the modeling easier, which is however not warranted. Therefore, we prefer to use a

discriminative model in our application. An important advantage of a discriminative model is that we can remain agnostic about the form of $P(\mathbf{x})$ which is not interesting to the prediction, because the observations will be given anyway.

Nevertheless, a linear-chain CRF is not always contrary to an HMM. An HMM can loosely be understood as a linear-chain CRF with very specific feature functions. Likewise, a linear-chain CRF can be interpreted as a generalization of an HMM that makes the constant transition probabilites into aritrary functions depending on the observation sequence [12]. Besides, the inference algorithms for HMMs have direct analogues to linear-chain CRFs, which will be discussed in the following sections.

3.2 Prediction with Viterbi Algorithm

We have mentioned two inference problems in Section 2.1. In this section, we focus on the MAP assignment problem for a linear-chain CRF, which is to find a most likely output label sequence $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})$ for a given observation $\mathbf{x}$. Using the form in Definition 3.1, we have

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmax}} \frac{1}{Z(\mathbf{x})} \exp\Big(\sum_{i=1}^{n} \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_{i-1}, y_i, \mathbf{x})\Big). \tag{3.3}$$

Since the normalization factor $Z(\mathbf{x})$ is a constant for a given $\mathbf{x}$ and the exponential function is monotonically increasing, Equation 3.3 is equivalent to

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmax}} \Big(\sum_{i=1}^{n} \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_{i-1}, y_i, \mathbf{x})\Big). \tag{3.4}$$

Like HMMs, we can use Viterbi algorithm to find a most probable label sequence $\mathbf{y}^*$ in Equation 3.4. As usual, let $\mathcal{L}$ denote a finite set of possible labels and every output variable $y_i \in \mathcal{L}$, $\forall i = 1, 2, \ldots, n$. The dynamic variables $\delta_{l,i}$ used in this computation are called *Viterbi variables*, where $l \in \mathcal{L}$ denotes every possible label for the node i.

First of all, we calculate the Viterbi variables for the first node. Each possible label $l \in \mathcal{L}$ is taken into consideration and the sum of all weighted feature functions based on the label l is stored in the corresponding Viterbi variable respectively:

$$\delta_{l,1} := \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_0, l, \mathbf{x}), \quad l \in \mathcal{L}. \tag{3.5}$$

Then, for $i = 2, 3, \ldots, n$:

$$\delta_{l,i} := \max_{l' \in \mathcal{L}} (\delta_{l',i-1} + \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(l', l, \mathbf{x}), \quad l \in \mathcal{L}. \tag{3.6}$$

The Viterbi variables of the first node are explicitly defined in Equation 3.5 and in the further calculations, the previous Viterbi variables are used to compute the current ones with the form 3.6. In this way, the Viterbi variables are computed efficiently by a *dynamic programming*.

Finally, we obtain $\delta_{l,n}$ for every possible label $l \in \mathcal{L}$. Among those possible labels, a best label is yielded with $l^* = \mathrm{argmax}_{l \in \mathcal{L}} \, \delta_{l,n}$. The corresponding Viterbi variable $\delta_{l^*,n}$ is called the *Viterbi score* with

$$\delta_{l^*,n} := \max_{l \in \mathcal{L}} \delta_{l,n}. \tag{3.7}$$

Now the problem is to find a most probable label sequence $\mathbf{y}^*$ through the Viterbi variables. This can be realized by the *Viterbi recursion*. For the last node, we have already got the best label which maximizes the Viterbi score:

$$y_n^* = l^* = \operatorname*{argmax}_{l \in \mathcal{L}} \delta_{l,n}. \tag{3.8}$$

Starting from this end point, each best label y_i^* can be obtained by the following recursion which is also called *backtracking*:

$$y_i^* = \operatorname*{argmax}_{l \in \mathcal{L}} \big(\delta_{l,i} + \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(l, y_{i+1}^*, \mathbf{x})\big), i = n-1, n-2, \ldots, 1. \tag{3.9}$$

In general, solving the MAP assignment problem is computationally difficult, because each possible label sequence must be taken into account and the exhaustive search requires $O(|\mathcal{L}|^n)$ calculations, where $|\mathcal{L}|$ denotes the cardinality of the label set. The Viterbi algorithm uses dynamic programming technique and has a computational complexity $O(n \cdot |\mathcal{L}|^2)$.

Now we present the Viterbi algorithm that finds a best label sequence with a linear-chain CRF. In the following algorithm, we assume that $\mathcal{L}$ is a finite set of Q possible labels, that is $\mathcal{L} = \big\{l_q, q \in \{1, 2, \ldots, \mathrm{Q}\}\big\}$.

Algorithm 1 Viterbi Algorithm in a Linear-Chain CRF

Input: observation $\mathbf{x}$, set of labels $\mathcal{L}$, feature functions f_k with weights λ_k
Output: $\mathbf{y}^*$ with $\mathbf{y}^* = \text{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})$

1. Initialize Viterbi variables at the first position

   for $q = 1, 2, \ldots, \text{Q}$ **do**
   $\delta_{l_q,1} = \sum_{k=1}^{\text{K}} \lambda_k f_k(y_0, l_q, \mathbf{x})$
   end for

2. Compute all the other Viterbi variables dynamically

   for $i = 2, 3, \ldots, n$ **do**
   for $q = 1, 2, \ldots, \text{Q}$ **do**
   $\delta_{l_q,i} = \max_{q'} \big(\delta_{l_{q'},i-1} + \sum_{k=1}^{\text{K}} \lambda_k f_k(l_{q'}, l_q, \mathbf{x})\big)$
   end for
   end for

3. Backtracking

   $y_n^* = \text{argmax}_q \, \delta_{l_q,n}$
   for $i = n-1, n-2, \ldots, 1$ **do**
   $y_i^* = \text{argmax}_q \big(\delta_{l_q,i} + \sum_{k=1}^{\text{K}} \lambda_k f_k(l_q, y_{i+1}^*, \mathbf{x})\big)$
   end for

4. Output

   return $\mathbf{y}^* = (y_1^*, y_2^*, \ldots, y_n^*)$

3.3 Parameter Estimation

A statistical model often contains some unknown parameters which can be calculated through the training. In a linear-chain CRF, the weights λ_k are the parameters which should be estimated to make the model fit the data.

It is assumed that we have a set of training data $\mathcal{D} = \{(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}), m = 1, 2, \ldots, \text{M}\}$, which are independent and identically distributed (i.i.d.). Each pair $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})$ in $\mathcal{D}$ is comprised of an observation $\mathbf{x}^{(m)}$ and its actual label sequence $\mathbf{y}^{(m)} = (y_1^{(m)}, y_2^{(m)}, \ldots, y_{n_m}^{(m)})$, where n_m denotes the length of $\mathbf{y}^{(m)}$.

One of the best known estimation methods is *maximum likelihood*. Since a CRF models the conditional distribution and is defined with log-linear potential functions

(2.8), it is appropriate to maximize the so-called *conditional log likelihood* $l : \mathbb{R}^{\mathrm{K}} \to \mathbb{R}$,

$$l(\mathbf{\Lambda}) := \sum_{m=1}^{\mathrm{M}} \log P(\mathbf{y}^{(m)}|\mathbf{x}^{(m)}). \tag{3.10}$$

We can get a more detailed form using the conditional distribution (3.1):

$$\begin{aligned} l(\mathbf{\Lambda}) &= \sum_{m=1}^{\mathrm{M}} \log \frac{1}{Z(\mathbf{x}^{(m)})} \exp\Big(\sum_{i=1}^{n_m} \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_{i-1}^{(m)}, y_i^{(m)}, \mathbf{x}^{(m)})\Big) \\ &= \sum_{m=1}^{\mathrm{M}} \sum_{i=1}^{n_m} \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_{i-1}^{(m)}, y_i^{(m)}, \mathbf{x}^{(m)}) - \sum_{m=1}^{\mathrm{M}} \log Z(\mathbf{x}^{(m)}). \end{aligned} \tag{3.11}$$

Lemma 3.2.
The log-sum-exp function $g : \mathbb{R}^n \to \mathbb{R}$ with

$$g(\alpha) = \log \sum_{i=1}^{n} \exp \alpha_i$$

is convex in $\alpha \in \mathbb{R}^n$.

Proof. See in *Convex Optimization* [Boyd & Vandenberghe, 2004]. □

Theorem 3.3.
The conditional log-likelihood function $l(\mathbf{\Lambda})$ described in (3.10) of a linear-chain CRF is concave in $\mathbf{\Lambda} \in \mathbb{R}^n$.

Proof. The first term in (3.11) is linear and concave in $\mathbf{\Lambda}$.

Now we consider the second term $\sum_{m=1}^{\mathrm{M}} \log Z(\mathbf{x}^{(m)})$. According to the definition of the normalization factor (3.2), $\log Z(\mathbf{x}^{(m)})$ is a log-sum-exp function for every m, which is shown convex in Lemma 3.2. With the fact that the sum of convex functions is still convex, it follows that $\sum_{m=1}^{\mathrm{M}} \log Z(\mathbf{x}^{(m)})$ is convex in $\mathbf{\Lambda} \in \mathbb{R}^n$, and its negative $-\sum_{m=1}^{\mathrm{M}} \log Z(\mathbf{x}^{(m)})$ is concave in $\mathbf{\Lambda} \in \mathbb{R}^n$.

So far, we have shown that $l(\mathbf{\Lambda})$ is the sum of two concave functions. Hence it is concave in $\mathbf{\Lambda} \in \mathbb{R}^n$. □

Because of the concavity of $l(\mathbf{\Lambda})$, every local maximum is also a global maximum. In other words, if the gradient of the function is zero at a point $\mathbf{\Lambda}^*$, the $\mathbf{\Lambda}^*$ is a global maximum of $l(\mathbf{\Lambda})$.

Now we compute the partial derivatives $\frac{\partial l}{\partial \lambda_k}, k = 1, 2, \ldots, \mathrm{K}$:

$$\begin{aligned}\frac{\partial l}{\partial \lambda_k} &= \sum_{m=1}^{\mathrm{M}} \sum_{i=1}^{n_m} f_k(y_{i-1}^{(m)}, y_i^{(m)}, \mathbf{x}^{(m)}) \\ &\quad - \sum_{m=1}^{\mathrm{M}} \sum_{i=1}^{n_m} \sum_{y_{i-1}, y_i \in \mathcal{L}} P(y_{i-1}, y_i | \mathbf{x}^{(m)}) \cdot f_k(y_{i-1}, y_i, \mathbf{x}^{(m)}).\end{aligned} \tag{3.12}$$

The first term in (3.12) is the expected value of $\sum_i f_k$ under the empirical distribution of the training data in $\mathcal{D}$, whereas the second term interprets the expectation of $\sum_i f_k$ under the distribution of a linear-chain CRF. Setting this derivative to zero means that the two expectations are equal.

Nevertheless, we cannot solve $\mathbf{\Lambda}$ in a closed form. Instead, numerical optimization methods are often used, such as iterative scaling [26] and some gradient-based methods [18].

3.4 Optimization Techniques for Training the Model

In this section, we will introduce a second-order technique which solves unconstrained nonlinear optimization problems. The *quasi-Newton* method is based on Newton's methods and is to find the stationary points of a function, where the gradient is zero.

Let $F : \mathbb{R}^n \to \mathbb{R}$ be an objective function depending on a parameter vector $\mathbf{x} \in \mathbb{R}^n$. Optimization requires to find the minimum of the function $\min_{\mathbf{x} \in \mathbb{R}^n} F(\mathbf{x})$. The quasi-Newton method solves the problem as follows:

First we calculate the gradient $\nabla F(\mathbf{x})$ of the objective function. Then we solve the equations $\nabla F(\mathbf{x}) = 0$ in a high dimensional field by using Newton's method which is as far as I know the best method to find successively better approximations to the roots of a real-valued function. The main idea is to update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$ with $\mathbf{d}_k = -\mathbf{H}(\mathbf{x}_k)^{-1} \cdot \nabla F(\mathbf{x}_k)$, where $\mathbf{H}$ is the *Hessian matrix* of the objective function. The element of the Hessian matrix are the second derivatives of F.

In general, it is not easy to calculate the Hessian matrix $\mathbf{H}$. One of the important advantages of quasi-Newton methods is that they do not require to compute the Hessian matrix exactly. Instead, we use an approximation and update them by analyzing successive gradient vectors. There are several variants of quasi-Newton methods and one of the most common algorithms is the *BFGS* method [14].

The BFGS algorithm starts from some approximate inverse of the Hessian matrix, say $\mathbf{B}_0$. $\mathbf{B}_0$ is often taken to be the unit matrix or a scaled unit matrix. A *Newton's step* $\mathbf{d}_k$ is calculated using the current approximation $\mathbf{B}_k$,

$$\mathbf{d}_k = -\mathbf{B}_k \cdot \nabla F(\mathbf{x}_k). \tag{3.13}$$

Then we use *line-search* method to find a step length $\alpha_k \in \mathbb{R}$ in the direction $\mathbf{d}_k$ and update the solution

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \cdot \mathbf{d}_k. \tag{3.14}$$

If it is necessary, we update $\mathbf{B}_{k+1}$ with the vectors

$$\mathbf{s}_k := \mathbf{x}_{k+1} - \mathbf{x}_k, \quad \mathbf{y}_k := \nabla F(\mathbf{x}_{k+1}) - \nabla F(\mathbf{x}_k), \tag{3.15}$$

so that

$$\mathbf{B}_{k+1} = \mathbf{V}_k^T \mathbf{B}_k \mathbf{V}_k + \rho_k \mathbf{s}_k \mathbf{s}_k^T, \tag{3.16}$$

where

$$\rho_k := \frac{1}{\mathbf{y}_k^T \mathbf{s}_k}, \quad \mathbf{V}_k := \mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^T. \tag{3.17}$$

A line-search is mentioned in the algorithm to find a step length, so that the function

$$\varphi(\alpha) := F(\mathbf{x} + \alpha \cdot \mathbf{d})$$

is minimized for fixed $\mathbf{x}$ and $\mathbf{d}$.

Generally, we cannot minimize φ exactly. Instead, we solve this problem loosely by asking for a sufficient decrease in φ. The *Wolfe conditions* are commonly used in quasi-Newton methods.

Definition 3.4 (Nocedal and Wright, 1999)**.**
We are given a function $F : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{x} \mapsto F(\mathbf{x})$. For some fixed $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{d} \in \mathbb{R}^n$, we define

$$\begin{aligned} \varphi : \mathbb{R} &\to \mathbb{R} \\ \alpha &\mapsto \varphi(\alpha) := F(\mathbf{x} + \alpha \cdot \mathbf{d}). \end{aligned}$$

A step length α is said to satisfy the **Wolfe conditions***, if these two inequalities hold:*

1. $F(\mathbf{x} + \alpha \cdot \mathbf{d}) \leq F(\mathbf{x}) + c_1 \cdot \alpha \cdot \mathbf{d}^T \cdot \nabla F(\mathbf{x})$,
2. $\mathbf{d}^T \cdot \nabla F(\mathbf{x} + \alpha \cdot \mathbf{d}) \geq c_2 \cdot \mathbf{d}^T \cdot \nabla F(\mathbf{x})$,

Algorithm 2 BFGS Algorithm

Input: $F : \mathbb{R}^n \to \mathbb{R}$ and $\nabla F : \mathbb{R}^n \to \mathbb{R}^n$. It is also given a tolerance $\epsilon > 0$.
Output: $\mathbf{x}^*$, which is a local minimum (global minimum if F convex) of F.

1. Initialize $\mathbf{x}_0$ and $\mathbf{B}_0$. Set iteration counter $k = 0$.
2. Calculate $\mathbf{d}_k$ in (3.13).
3. Perform a line-search to find α_k in direction $\mathbf{d}_k$ and update $\mathbf{x}_{k+1}$ in (3.14).
4. Check if the tolerance is reached.

   if $\|\nabla F(\mathbf{x}_{k+1})\| < \epsilon \cdot \max\{\|\mathbf{x}_{k+1}\|, 1\}$ **then**
   return $\mathbf{x}^* = \mathbf{x}_{k+1}$.
   else
   Update $\mathbf{B}_{k+1}$ in (3.16),
   $k = k + 1$,
   and go to step 2.
   end if

with $0 < c_1 < c_2 < 1$.

Now we represent the BFGS algorithm in pseudocode:

If the optimization problem has a large number of parameters, it is recommended to use the *limited memory BFGS* (L-BFGS) method [28]. We have seen that $\mathbf{B}_k$ is completely fixed by $\mathbf{B}_0$ and the vector pairs $\{\mathbf{s}_0, \mathbf{y}_0\}, \{\mathbf{s}_1, \mathbf{y}_1\}, \ldots, \{\mathbf{s}_{k-1}, \mathbf{y}_{k-1}\}$ in (3.15). Moreover, $\mathbf{B}_k$ is only used to get the direction $\mathbf{d_k}$ in Equation 3.13 in each iteration and it is possible to calculate $\mathbf{d_k}$ explicitly with the k vector pairs [28]. Therefore, instead of storing $\mathbf{B}_k$ completely, we can only store the k vector pairs.

Modified versions of $\mathbf{B}_k$ are updated from the most recent iterations $k-m, k-m+1, \ldots, k-1$, thus it only requires storage for

$$\{\mathbf{s}_{k-m}, \mathbf{y}_{k-m}\}, \{\mathbf{s}_{k-m+1}, \mathbf{y}_{k-m+1}\}, \ldots, \{\mathbf{s}_{k-1}, \mathbf{y}_{k-1}\}.$$

The number $m \in \mathbb{N}$ is suggested to be chosen between 3 and 7.

The implementation of the L-BFGS algorithm is more efficient, because it does not store $\mathbf{B}_k$ which can be quite expensive. Especially for the modified versions, it maintains a history of only the past m updates and saves much storage. In this sense, L-BFGS method is very suitable for large-scale problems.

3.5 Forward-Backward Algorithm

After discussing the second-order technique, we review the conditional log-likelihood defined in (3.11)

$$l(\mathbf{\Lambda}) = \sum_{m=1}^{\mathrm{M}} \sum_{i=1}^{n_m} \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_{i-1}^{(m)}, y_i^{(m)}, \mathbf{x}^{(m)}) - \sum_{m=1}^{\mathrm{M}} \log Z(\mathbf{x}^{(m)}).$$

Estimating the parameter vector $\mathbf{\Lambda}$ is to solve the following optimization problem:

$$\max_{\mathbf{\Lambda} \in \mathbb{R}^{\mathrm{K}}} l(\mathbf{\Lambda}) \Longleftrightarrow - \min_{\mathbf{\Lambda} \in \mathbb{R}^{\mathrm{K}}} -l(\mathbf{\Lambda})$$

Using the BFGS algorithm to minimize $-l(\mathbf{\Lambda})$, we get a maximum of $l(\mathbf{\Lambda})$. Since $l(\mathbf{\Lambda})$ has been proved concave in Theorem 3.3, every local maximum is also a global maximum of $l(\mathbf{\Lambda})$. We have seen in the previous section that the BFGS algorithm requires computing of the gradient of the objective function, which has been analyzed in (3.12)

$$\begin{aligned} \frac{\partial l}{\partial \lambda_k} = & \sum_{m=1}^{\mathrm{M}} \sum_{i=1}^{n_m} f_k(y_{i-1}^{(m)}, y_i^{(m)}, \mathbf{x}^{(m)}) \\ & - \sum_{m=1}^{\mathrm{M}} \sum_{i=1}^{n_m} \sum_{y_{i-1}, y_i \in \mathcal{L}} P(y_{i-1}, y_i | \mathbf{x}^{(m)}) \cdot f_k(y_{i-1}, y_i, \mathbf{x}^{(m)}). \end{aligned}$$

The difficulty of computing $l(\mathbf{\Lambda})$ and $\frac{\partial l}{\partial \lambda_k}$ lies in calculating $Z(\mathbf{x}^{(m)})$ and $P(y_{i-1}, y_i | \mathbf{x}^{(m)})$ efficiently, which involves one of the inference problems mentioned in Section 2.1, mamely the marginalization.

Now we concentrate on how to compute the marginal distributions $P(y_{i-1}, y_i | \mathbf{x}^{(m)})$ for each edge $\{i-1, i\}$, $i = 1, 2, \ldots, n$. There exists a *forward-backward* algorithm [36] computing the marginal distributions efficiently for a linear chain. This algorithm also performs a dynamic programming similar to the Viterbi algorithm. The main idea is to define the *forward variables* and *backward variables* recursively.

We assume that there are Q possible labels in $\mathcal{L}$ with $\mathcal{L} = \{l_p, p \in \{1, 2, \ldots, \mathrm{Q}\}\}$. Then we define a set of forward variables $\boldsymbol{\alpha}_i$ with

$$\boldsymbol{\alpha}_i = \begin{pmatrix} \alpha_{i,1} \\ \alpha_{i,2} \\ \vdots \\ \alpha_{i,\mathrm{Q}} \end{pmatrix} \in \mathbb{R}^{\mathrm{Q}}, \quad i = 0, 1, \ldots, n.$$

$\boldsymbol{\alpha}_0$ is initialized with $\boldsymbol{\alpha}_0 := (1, 1, \ldots, 1)^T \in \mathbb{R}^{\mathrm{Q}}$. $\boldsymbol{\alpha}_i$ is computed as follows:

$$\boldsymbol{\alpha}_i := \begin{pmatrix} \sum_p \boldsymbol{\alpha}_{i-1,p} \cdot \Phi(l_p, l_1, \mathbf{x}) \\ \sum_p \boldsymbol{\alpha}_{i-1,p} \cdot \Phi(l_p, l_2, \mathbf{x}) \\ \vdots \\ \sum_p \boldsymbol{\alpha}_{i-1,p} \cdot \Phi(l_p, l_{\mathrm{Q}}, \mathbf{x}) \end{pmatrix}, \quad i = 1, 2, \ldots, n, \tag{3.18}$$

where $\Phi(l, l', \mathbf{x}) = \exp \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(l, l', \mathbf{x})$.

Analogously, we define the backward variables $\boldsymbol{\beta}_i$ with

$$\boldsymbol{\beta}_i = \begin{pmatrix} \beta_{i,1} \\ \beta_{i,2} \\ \vdots \\ \beta_{i,\mathrm{Q}} \end{pmatrix} \in \mathbb{R}^{\mathrm{Q}}, \quad i = n, n-1, \ldots, 1.$$

We initialize the backward variables at the last position with $\boldsymbol{\beta}_n := (1, 1, \ldots, 1)^T \in \mathbb{R}^{\mathrm{Q}}$. Then the backward recursion is written as:

$$\boldsymbol{\beta}_i := \begin{pmatrix} \sum_p \Phi(l_1, l_p, \mathbf{x}) \cdot \boldsymbol{\beta}_{i+1,p} \\ \sum_p \Phi(l_2, l_p, \mathbf{x}) \cdot \boldsymbol{\beta}_{i+1,p} \\ \vdots \\ \sum_p \Phi(l_{\mathrm{Q}}, l_p, \mathbf{x}) \cdot \boldsymbol{\beta}_{i+1,p} \end{pmatrix}, \quad i = n-1, n-2, \ldots, 1. \tag{3.19}$$

The marginal distributions $P(y_{i-1}, y_i|\mathbf{x})$ can be computed by forward- and backward-variables in the following Lemma:

Lemma 3.5.
It is given that $y_{i-1} = l_q \in \mathcal{L}, y_i = l_{q'} \in \mathcal{L}$. Then it holds for $i = 1, 2, \ldots, n$

$$P(y_{i-1}, y_i|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \cdot \boldsymbol{\alpha}_{i-1,q} \cdot \Phi(l_q, l_{q'}, \mathbf{x}) \cdot \boldsymbol{\beta}_{i,q'}. \tag{3.20}$$

Proof. According to the definition of the marginal distribution (**??**), we have

$$P(y_{i-1}, y_i|\mathbf{x}) = \sum_{\mathbf{y} \setminus \{y_{i-1}, y_i\}} P(\mathbf{y}|\mathbf{x}).$$

$P(\mathbf{y}|\mathbf{x})$ can be formulated as follows:

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) &= \frac{1}{Z(\mathbf{x})} \exp\Big(\sum_{t=1}^{n} \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_{t-1}, y_t, \mathbf{x})\Big) \\ &= \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{n} \Phi(y_{t-1}, y_t, \mathbf{x}). \end{aligned}$$

It follows

$$P(y_{i-1}, y_i|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{y}\setminus\{y_{i-1},y_i\}} \prod_{t=1}^{n} \Phi(y_{t-1}, y_t, \mathbf{x}). \tag{3.21}$$

On the other hand, we compute $\boldsymbol{\alpha}_{i-1,q}$ and $\boldsymbol{\beta}_{i,q'}$ with forward recursion (3.18) and backward recursion (3.19).

$$\begin{aligned}
\boldsymbol{\alpha}_{i-1,q} &= \sum_{p} \boldsymbol{\alpha}_{i-2,p} \cdot \Phi(l_p, l_q, \mathbf{x}) \\
&= \sum_{p} \sum_{p'} \boldsymbol{\alpha}_{i-3,p'} \cdot \Phi(l_{p'}, l_p, \mathbf{x}) \cdot \Phi(l_p, l_q, \mathbf{x}) \\
&\vdots \\
&= \sum_{y_1, y_2, \ldots, y_{i-2}} \Phi(y_0, y_1, \mathbf{x}) \cdot \Phi(y_1, y_2, \mathbf{x}) \cdot \ldots \cdot \Phi(y_{i-2}, l_q, \mathbf{x}).
\end{aligned} \tag{3.22}$$

$$\begin{aligned}
\boldsymbol{\beta}_{i,q'} &= \sum_{p} \Phi(l_{q'}, l_p, \mathbf{x}) \cdot \boldsymbol{\beta}_{i+1,p} \\
&= \sum_{p} \sum_{p'} \Phi(l_{q'}, l_p, \mathbf{x}) \cdot \Phi(l_p, l_{p'}, \mathbf{x}) \cdot \boldsymbol{\beta}_{i+2,p'} \\
&\vdots \\
&= \sum_{y_{i+1}, y_{i+2}, \ldots, y_n} \Phi(l_{q'}, y_{i+1}, \mathbf{x}) \cdot \Phi(y_{i+1}, y_{i+2}, \mathbf{x}) \cdot \ldots \cdot \Phi(y_{n-1}, y_n, \mathbf{x}).
\end{aligned} \tag{3.23}$$

Then we compute the term on the right side $(*)$ of Equation (3.20):

$$\begin{aligned}
(*) &= \frac{1}{Z(\mathbf{x})} \Big(\sum_{y_1, y_2, \ldots, y_{i-2}} \Phi(y_0, y_1, \mathbf{x}) \cdot \Phi(y_1, y_2, \mathbf{x}) \cdot \ldots \cdot \Phi(y_{i-2}, l_q, \mathbf{x}) \Big) \cdot \Phi(l_q, l_{q'}, \mathbf{x}) \\
&\quad \cdot \Big(\sum_{y_{i+1}, y_{i+2}, \ldots, y_n} \Phi(l_{q'}, y_{i+1}, \mathbf{x}) \cdot \Phi(y_{i+1}, y_{i+2}, \mathbf{x}) \cdot \ldots \cdot \Phi(y_{n-1}, y_n, \mathbf{x}) \Big) \\
&= \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{y}\setminus\{y_{i-1},y_i\}} \prod_{t=1}^{n} \Phi(y_{t-1}, y_t, \mathbf{x}) \\
&\overset{(3.21)}{=} P(y_{i-1}, y_i|\mathbf{x}).
\end{aligned}$$

□

Note that the normalization factor $Z(\mathbf{x})$ can also be computed with forward variables or backward variables.

Lemma 3.6.
We can use the forward variables to compute the normalization factor $Z(\mathbf{x})$ as follows:

$$Z(\mathbf{x}) = \sum_{q=1}^{\mathrm{Q}} \alpha_{n,q}. \tag{3.24}$$

Proof. We use Equation (3.22) with application to $\alpha_{n,q}$, $q = 1, 2, \ldots, \mathrm{Q}$:

$$\alpha_{n,q} = \sum_{y_1, y_2, \ldots, y_{n-1}} \Phi(y_0, y_1, \mathbf{x}) \cdot \Phi(y_1, y_2, \mathbf{x}) \cdot \ldots \cdot \Phi(y_{n-1}, l_q, \mathbf{x}).$$

The sum of all $\alpha_{n,q}$, $q = 1, 2, \ldots, \mathrm{Q}$ yields:

$$\begin{aligned}
\sum_{q=1}^{\mathrm{Q}} \alpha_{n,q} &= \sum_{q=1}^{\mathrm{Q}} \sum_{y_1, y_2, \ldots, y_{n-1}} \Phi(y_0, y_1, \mathbf{x}) \cdot \Phi(y_1, y_2, \mathbf{x}) \cdot \ldots \cdot \Phi(y_{n-1}, l_q, \mathbf{x}) \\
&= \sum_{y_1, y_2, \ldots, y_n} \Phi(y_0, y_1, \mathbf{x}) \cdot \Phi(y_1, y_2, \mathbf{x}) \cdot \ldots \cdot \Phi(y_{n-1}, y_n, \mathbf{x}) \\
&= \sum_{\mathbf{y}} \prod_{i=1}^{n} \Phi(y_{i-1}, y_i, \mathbf{x}).
\end{aligned} \tag{3.25}$$

On the other hand, we compute $Z(\mathbf{x})$ as follows:

$$\begin{aligned}
Z(\mathbf{x}) &= \sum_{\mathbf{y}} \exp \sum_{i=1}^{n} \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_{i-1}, y_i, \mathbf{x}) \\
&= \sum_{\mathbf{y}} \big(\exp \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_0, y_1, \mathbf{x})\big) \cdot \big(\exp \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_1, y_2, \mathbf{x})\big) \cdot \ldots \cdot \big(\exp \sum_{k=1}^{\mathrm{K}} \lambda_k f_k(y_{n-1}, y_n, \mathbf{x})\big) \\
&= \sum_{\mathbf{y}} \prod_{i=1}^{n} \Phi(y_{i-1}, y_i, \mathbf{x}) \\
&\overset{(3.25)}{=} \sum_{q=1}^{\mathrm{Q}} \alpha_{n,q}.
\end{aligned}$$

□

Remark. *Similarly, $Z(\mathbf{x})$ can be calculated by backward variables with*

$$Z(\mathbf{x}) = \sum_{q=1}^{Q} \beta_{1,q}.$$

From above we have seen that the objective function $l(\mathbf{\Lambda})$ (3.11) can be computed efficiently, because $Z(\mathbf{x}^{(m)})$ can be obtained using every $\boldsymbol{\alpha}_n$ defined for each $\mathbf{x}^{(m)}$. Besides, the marginal distribution $P(y_{i-1}, y_i, \mathbf{x}^{(m)})$ can also be calculated efficiently. Thus the computation of the derivatives (3.12) is not a problem any more.

As a summary of this section, both $l(\mathbf{\Lambda})$ and $\nabla l(\mathbf{\Lambda})$ can be computed exactly and efficiently with the forward-backward algorithm. Let n be the maximal length of the observations in the training data. $Z(\mathbf{x}^{(m)})$ and $P(y_{i-1}, y_i, \mathbf{x}^{(m)})$ can be obtained efficiently, which uses a computational complexity $O(n\mathrm{Q}^2)$ respectively. We need to run forward-backward for each training instance $\mathbf{x}^{(m)}$ and hence need a cost of $O(n\mathrm{Q}^2\mathrm{M})$ to get $l(\mathbf{\Lambda})$ and $\nabla l(\mathbf{\Lambda})$. Then, BFGS algorithms can be applied to estimate a best weight vector $\mathbf{\Lambda}$. We assume that G is the number of iterations required by the optimization procedure and the training uses a total computational complexity $O(n\mathrm{Q}^2\mathrm{MG})$. If the optimization problem has a large number of parameters, we shall prefer to choose L-BFGS algorithms.

3.6 Reducing Overfitting — Regularization

As mentioned before, L-BFGS algorithms can reduce the complexity, if an optimization problem has a large number of parameters. However, if the optimization problem is formulated to estimate the parameter vector in the model with a large number of dimensions, *overfitting* generally occurs. This may lead to a poor predictive performance. In this case, additional techniques are often used to avoid overfitting, such as *regularization*.

Regularization is usually a penalty for complexity, such as restrictions for smoothness or bounds on the vector space norm. Here we consider the regularization as a penalty on weight vector whose norm is too large. Instead of maximizing $l(\mathbf{\Lambda})$ directly, we regularize it first:

$$l^R(\mathbf{\Lambda}) := l(\mathbf{\Lambda}) - \sum_{k=1}^{\mathrm{K}} \frac{\lambda_k^2}{2\sigma^2}. \tag{3.26}$$

The penalty in (3.26) is based on the quadratic Euclidean norm of $\mathbf{\Lambda}$ multiplied by a *regularization parameter* $1/2\sigma^2$, where σ is a free parameter determining how much to penalize the weight vector. The good point is that the accuracy of the model is generally not sensitive to changes in σ. σ is often chosen to equal one [12].

Because quadratic functions are convex, the appending term $-\sum_{k=1}^{\mathrm{K}} \frac{\lambda_k^2}{2\sigma^2}$ is concave. Therefore, the regularized objective function $l^R(\mathbf{\Lambda})$ in (3.26) retains the concavity. The partial derivatives of $l^R(\mathbf{\Lambda})$ have the form:

$$\frac{\partial l^R}{\partial \lambda_k} = \frac{\partial l}{\partial \lambda_k} - \frac{\lambda_k}{\sigma^2}, \qquad k = 1, 2, \ldots, \mathrm{K}. \tag{3.27}$$

The optimization problem $\max_{\mathbf{\Lambda} \in \mathbb{R}^{\mathrm{K}}} l^R(\mathbf{\Lambda})$ is equivalent to solving the problem

$$-\min_{\mathbf{\Lambda} \in \mathbb{R}^{\mathrm{K}}} -l^R(\mathbf{\Lambda}).$$

This regularized problem can be solved just like the original problem, because the appending terms in (3.26) and (3.27) can be easily computed .

4

Pairwise Conditional Random Fields

We have learned from the previous section that a linear-chain CRF has many limitations, since simply be seen as a linear structure will lead to a lack of data, especially in our application. To this end, we developed a variant of CRF called the pairwise conditional random field (pCRF) presenting the useful dependencies in a real graph.

Firstly, we describe how to get the idea of the pairwise CRF model. As mentioned in Section 2.4 that a general CRF is an undirected graphical model. One of its benefits is that all dependencies between the nodes can be presented in the model. Nevertheless, training for many large undirected models is intractable, because computing marginal distributions of the model is very expensive. The more intricate dependencies we bring into a CRF, the more difficult it is to train the model. Sometimes, it is even a tough work to find the best label sequence if a CRF is too complicated. So this involves a trade-off: On the one hand, we wish to retain as many as possible information in a CRF. On the other hand, the model must be feasible in both prediction and training.

In a pCRF, we consider the dependencies between two nodes. The cliques with the cardinality over two are not taken into account. That means, a pCRF is constructed by only node cliques and edge cliques.

Definition 4.1.
Let $\mathbf{X}$ *be a random observation over data and* $\mathbf{Y}$ *one of the possible label sequences modeled by an undirected graph* $G = (V, E)$*. Moreover,* $\mathcal{C} = V \cup E$ *denotes the set of cliques that are taken into consideration, and* $s(\mathbf{y}_C, \mathbf{x})$ *denotes a scoring function for*

a clique $C \in \mathcal{C}$ with different possible labels. Then a **pairwise conditional random field** (pCRF) *takes the form*

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \sum_{C \in \mathcal{C}} s(\mathbf{y}_C, \mathbf{x}), \tag{4.1}$$

where

$$Z(\mathbf{x}) = \sum_{\tilde{\mathbf{y}}} \exp \sum_{C \in \mathcal{C}} s(\tilde{\mathbf{y}}_C, \mathbf{x}). \tag{4.2}$$

is the normalization factor.

Compared to a linear-chain CRF, a pairwise CRF is more flexible, because it is not limited to a linear graph. In contrast, we can model our problem with an arbitrary graph. In this way, the edges can represent not only the neighborhood in a linear chain but also many other dependencies, like spatial neighborhood.

For simplicity, here we introduce another concept, namely scoring functions. As we know, several feature functions are defined on a clique and each feature function has a weight. In order to simplify the form, the clique with possible labels is evaluated through a scoring function which is the sum of the weighted feature functions defined on the clique. In a pairwise CRF, it involves node scoring function and edge scoring function.

Let $\psi_\varsigma(i, y_i, \mathbf{x})$ denote a node feature function, $\phi_\tau(i, j, y_i, y_j, \mathbf{x})$ an edge feature function with the edge $e = (i, j) \in E$, and $\lambda^{(n)}, \lambda^{(e)}$ are the corresponding weight vectors, respectively. The scoring function of a node $i \in V$ or an edge $e = (i, j) \in E$ is defined by

$$s(y_i, \mathbf{x}) = \sum_{\varsigma} \lambda_\varsigma^{(n)} \psi_\varsigma(i, y_i, \mathbf{x}), \tag{4.3}$$

or

$$s(y_i, y_j, \mathbf{x}) = \sum_{\tau} \lambda_\tau^{(e)} \phi_\tau(i, j, y_i, y_j, \mathbf{x}). \tag{4.4}$$

The scoring function $s(\mathbf{y}_C, \mathbf{x})$ in Definition 4.1 is referred to a node scoring function or an edge scoring function.

Remark. *A linear-chain CRF is a special case of a pairwise CRF whose related graph is a linear chain.*

4.1 Generalized Viterbi Algorithm

In this section, we are going to solve the MAP problem for a pairwise CRF. The goal is to find one of the most probable label sequences of an observation $\mathbf{x}$:

$$\begin{aligned}\mathbf{y}^* &= \underset{\mathbf{y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x}) \\ &= \underset{\mathbf{y}}{\operatorname{argmax}}\, \frac{1}{Z(\mathbf{x})} \exp \sum_{C\in\mathcal{C}} s(\mathbf{y}_C, \mathbf{x}) \\ &= \underset{\mathbf{y}}{\operatorname{argmax}} \sum_{C\in\mathcal{C}} s(\mathbf{y}_C, \mathbf{x}).\end{aligned} \tag{4.5}$$

We have seen that the Viterbi algorithm provides us with an efficient dynamic programming method to find the solution to the MAP problem of a linear-chain CRF. The dynamic process proceeds along the direction of the linear chain. However, the corresponding graph of a pairwise CRF is unrestricted, such that applying the original Viterbi algorithm directly to a pairwise CRF is not suitable.

In order to find the most probable label sequence with a pairwise CRF, we use the generalized Viterbi algorithm due to *Mario Stanke*. The generalized Viterbi algorithm extends the idea of dynamic programming to a pairwise CRF. The difference is that the dynamic programming proceeds not along a linear any more, but along the so-called "history sets" which will be introduced in the following notation.

Notation 4.2.
Let $G = (V, E)$ be an undirected graph.

- *$H \subseteq V$ which is a connected subset of nodes in V, is referred to as a* **history set**.
- *$B_H = \{v \in H | \exists \{v, w\} \in E : w \notin H\}$ is called the* **boundary** *of history set H.*
- *The dynamic programming variables (***Viterbi variables***) are defined as*

$$\gamma_H(\mathbf{y}_{B_H}) = \max_{\mathbf{y}_{H\setminus B_H}} \sum_{\substack{C\in\mathcal{C},\\ C\subseteq H}} s(\mathbf{y}_C, \mathbf{x}). \tag{4.6}$$

Taking an example in Figure 4.1, we have $H = \{a, b, c, d\}$ with boundary $B_H = \{a, d\}$. The nodes $v \in H \setminus B_H$, which belong to the "kernel" of H, have no contact with other nodes outside H. Therefore, they are independent of other history sets. In contrast, B_H can be thought as the "antenna" which represents its history set H to

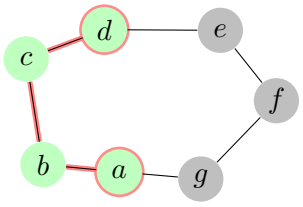

Figure 4.1: Example graph with $H = \{a, b, c, d\}$ (green) and $B_H = \{a, d\}$ (red circled).

communicate with outside nodes. Assuming that the label set $\mathcal{L} = \{0, 1\}$ and $\mathbf{y}_{B_H} = \{y_a, y_d\} \in \mathcal{L}^2$, the dynamic programming variable $\gamma_H(\mathbf{y}_{B_H})$ of H with $\mathbf{y}_{B_H} = \{0, 0\}$ is then calculated as follows:

$$\begin{aligned}\gamma_H(\mathbf{y}_{B_H} = \{0,0\}) = \max_{\{y_b,y_c\}} \big(&s(y_a,\mathbf{x}) + s(y_b,\mathbf{x}) + s(y_c,\mathbf{x}) + s(y_d,\mathbf{x}) \\ &+ s(y_a,y_b,\mathbf{x}) + s(y_b,y_c,\mathbf{x}) + s(y_c,y_d,\mathbf{x})\big) \\ = \max_{\{y_b,y_c\}} \big(&s(0,\mathbf{x}) + s(y_b,\mathbf{x}) + s(y_c,\mathbf{x}) + s(0,\mathbf{x}) \\ &+ s(0,y_b,\mathbf{x}) + s(y_b,y_c,\mathbf{x}) + s(y_c,0,\mathbf{x})\big).\end{aligned}$$

$\gamma_H(\mathbf{y}_{B_H} = \{0, 0\})$ represents the best score of H under the condition $y_a = 0$ and $y_d = 0$. This score can be achieved if we choose y_b^*, y_c^*, so that

$$\{y_b^*, y_c^*\} = \operatorname*{argmax}_{\{y_b,y_c\}} \gamma_H(\mathbf{y}_{B_H} = \{0,0\}).$$

Analogously, we can compute the other three Viterbi variables with $\mathbf{y}_{B_H} = \{0, 1\}, \{1, 0\}$ and $\{1, 1\}$. They will be memorized in $\gamma_H(\mathbf{y}_{B_H} = \{0, 1\})$, $\gamma_H(\mathbf{y}_{B_H} = \{1, 0\})$ and $\gamma_H(\mathbf{y}_{B_H} = \{1, 1\})$, respectively.

It is clear that the nodes in $H \setminus B_H$ are irrelevant in the further dynamic calculations. We put their information in the Viterbi variables and let the boundary represent this whole history set in the further calculation. This idea is similar to the original Viterbi algorithm. Computing current Viterbi variables at position i, we only take into consideration the previous Viterbi variables at position $i - 1$. The reason is that only node $i - 1$ belongs to the boundary of the unique history set $\{1, \ldots, i - 1\}$.

Now, we present the generalized Viterbi algorithm for a pairwise CRF with the following schema. The history sets are indexed by $j = 1, 2, \ldots, m$ and we simplify the notation of boundary B_{H_j} with B_j.

Algorithm 3 Generalized Viterbi Algorithm

Input:

- observation $\mathbf{x}$ and label set $\mathcal{L}$
- a connected undirected graph $G = (V, E)$
- scoring functions $s(\mathbf{y}_C, \mathbf{x}), \forall C \in \mathcal{C}$

Output: $\mathbf{y}^*$ with $\mathbf{y}^* = \text{argmax}_{\mathbf{y}} \sum_{C \in \mathcal{C}} s(\mathbf{y}_C, \mathbf{x})$.

1. Initialize the number of history sets $m = 0$.
2. Compute and update Viterbi variables:

   choose a node $v \in V \setminus (H_1 \cup H_2 \cup \ldots \cup H_m)$,
   update the history sets according to the undermentioned **case 1** or **case 2**,
   and update $\gamma_{H_1}(\mathbf{y}_{B_1}), \gamma_{H_2}(\mathbf{y}_{B_2}), \ldots, \gamma_{H_m}(\mathbf{y}_{B_m})$ in the corresponding case.

   This process will be repeated until $H_1 = V$.
3. Compute $\mathbf{y}^*$ by backtracking.

In the following two sections, we are going to discuss the details involving case 1 and case 2 mentioned in the second step.

4.2 Case 1: The Case of Isolated Node

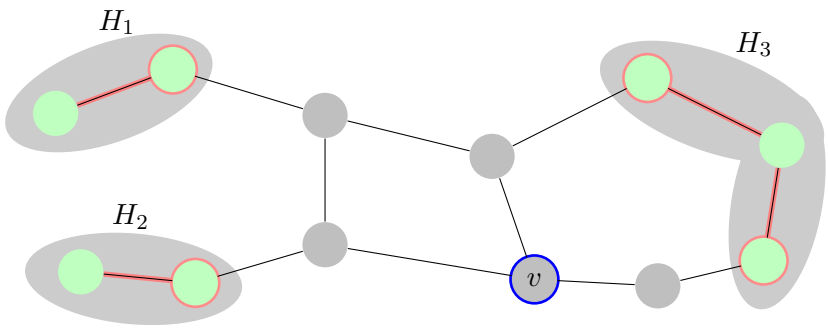

Figure 4.2: Case 1: the selected node v is isolated from all existing history sets.

As is shown in Figure 4.2, the node v in step 2 is isolated from all history sets H_1, H_2, H_3, which means there is no direct connection between v and the existing

history sets. In this case, we can simply build a new history set with this single element v. The number of history sets will then be increased by 1. $H_4 = \{v\}$ in Figure 4.3 is such a new history set.

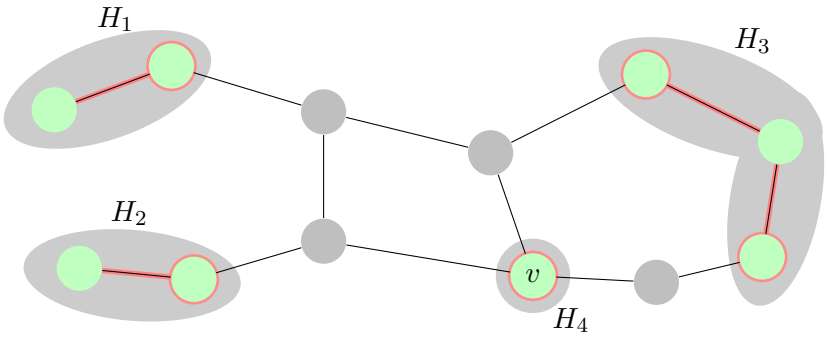

Figure 4.3: Case 1: the isolated v (left) built itself a history set H_4.

This case is quite simple. We build a further history set with this single node, and the boundary is also itself. The Viterbi variables for the other history sets keep unchanged. We only need to compute the Viterbi variables for the new history set, which can be obtained by the node scoring functions.

Algorithm 4 Case 1 of Algorithm 3 in Step 2

if $\nexists\{v, w\} \in E : w \in H_1 \cup H_2 \cup \ldots \cup H_m$ **then**
  $m = m + 1$
  $H_m = \{v\}$
  $B_m = \{v\}$
  for all possible assignments of $y_v \in \mathcal{L}$ **do**
    $\gamma_{H_m}(y_v) = s(y_v, \mathbf{x})$
  end for
end if

Remark. *Because the input graph is connected, i.e. each node cannot be isolated in the graph, the boundary of the new history set* $\{v\}$ *will never be empty and always* $\{v\}$ *itself.*

4.3 Case 2: The Case of Non-Isolated Node

Alternatively, v can be connected with some existing history sets. Without loss of generality, we assume that v is neighbor of $H_1, \ldots, H_k$, $1 \leq k \leq m$. In this case, all of the involving history sets will be merged together via the node v to form a new history set, but computing its Viterbi variables is much more complicated.

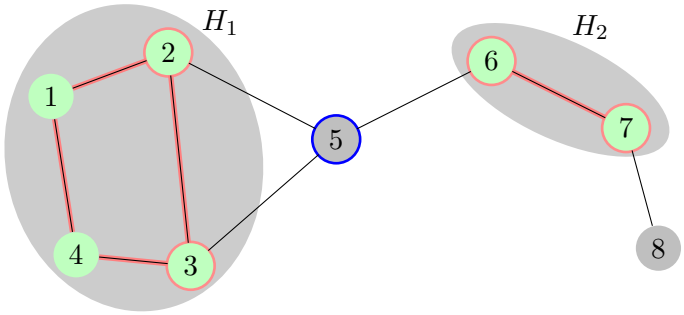

Figure 4.4: Case 2: the selected node has direct connections with some existing history sets.

It is shown in Figure 4.4 an example of case 2. The existing history sets are $H_1 = \{1, 2, 3, 4\}$ and $H_2 = \{6, 7\}$ with $B_1 = \{2, 3\}$ and $B_2 = \{6, 7\}$. The selected node 5 is directly connected with H_1 through edges $\{2, 5\}$ and $\{3, 5\}$. Meanwhile, edge $\{5, 6\}$ connects node 5 and history set H_2.

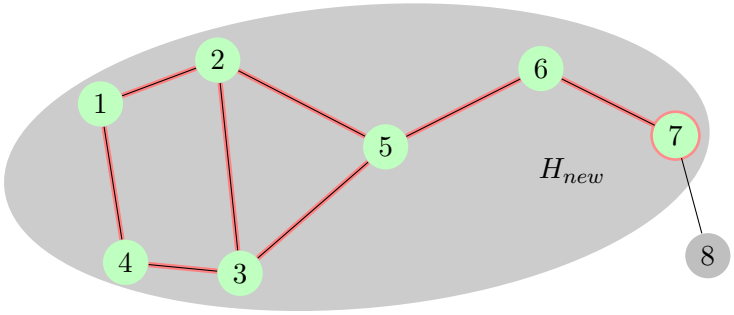

Figure 4.5: Case 2: the history sets H_1 and H_2 are merged together via node 5.

We now merge all these history sets and the selected node into a new history set $H_{new} = \{1, 2, 3, 4, 5, 6, 7\}$ and determine its boundary $B_{new} = \{7\}$ (Figure 4.5).

The difficulty is to calculate the Viterbi variables for the new history set. Let us compute them step by step. First, we observe only the history set H_1. If H_1 is merged

with the node v, all nodes in H_1 including those in B_1 will be buried in the kernel shown in Figure 4.6. That means, the nodes in B_1 will not be representative any more.

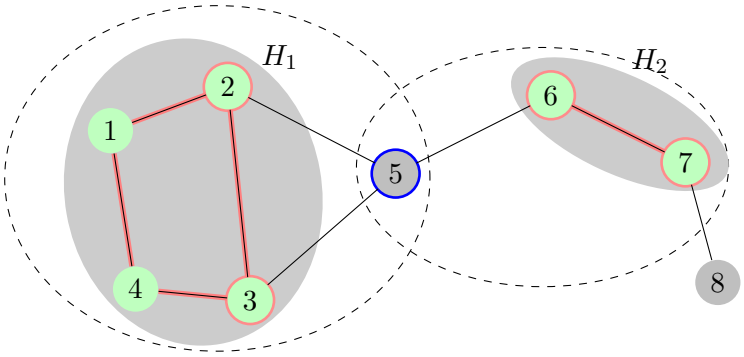

Figure 4.6: Calculation the Viterbi variables in case 2.

Since the nodes in B_1, namely node 2 and 3, will be buried in the kernel of the new history set, their information will be put in the Viterbi variables. Firstly, we consider the function

$$\gamma_{H_1}(\mathbf{y}_{B_1}) + s(y_2, y_5, \mathbf{x}) + s(y_3, y_5, \mathbf{x}),$$

which is the sum of the Viterbi variable of H_1 and the edge scoring function evaluated on two connection edges, $\{2, 5\}$ and $\{3, 5\}$. Then, the function above is maximized over y_2 and y_3 for every possible label of $y_5 \in \mathcal{L}$ and the best score is saved in $tmp_1(y_5)$:

$$tmp_1(y_5) = \max_{y_2, y_3} \big(\gamma_{H_1}(\mathbf{y}_{B_1}) + s(y_2, y_5, \mathbf{x}) + s(y_3, y_5, \mathbf{x})\big).$$

Similarly, we compute tmp_2 according to H_2. After the union of H_2 and node 5, node 7 is survived in the boundary B_{new}, while node 6 is buried in the kernel. We obtain thus $tmp_2(y_5, y_7)$ for all possible assignments of $\{y_5, y_7\} \in \mathcal{L}^2$

$$tmp_2(y_5, y_7) = \max_{y_6} \big(\gamma_{H_2}(\mathbf{y}_{B_2}) + s(y_5, y_6, \mathbf{x})\big).$$

To generalize, the set of nodes that turn from the boundary of H_j to the kernel in H_{new} can be represented by $B_j \setminus B_{new}$. The nodes in $B_{new} \cap B_j$ remain as representative of H_{new}. Then we compute tmp_j with arguments $\mathbf{y}_{B_{new} \cap B_j \cup \{v\}}$. The node v will be discussed at the end of the calculation. Its assignment appears temporarily in the arguments of every tmp_j.

In general form, we compute for every H_j, $j = 1, \ldots, k$:

$$tmp_j(\mathbf{y}_{B_{new} \cap B_j \cup \{v\}}) = \max_{\mathbf{y}_{B_j \setminus B_{new}}} \big(\gamma_{H_j}(\mathbf{y}_{B_j}) + \sum_{\substack{e=(v,w) \in E, \\ w \in B_j}} s(\mathbf{y}_e, \mathbf{x})\big). \tag{4.7}$$

So far, the node scoring function $s(y_v, \mathbf{x})$ has not been taken into account. That is because all of the involving history sets are merged together via node v and we add this node scoring function once to the sum of tmp_j in order to avoid duplicating. In the next step, we obtain the "quasi-Viterbi variables" with arguments $B_{new} \cup \{v\}$ instead of B_{new}:

$$\tilde{\gamma}_{H_{new}}(\mathbf{y}_{B_{new} \cup \{v\}}) = s(y_v, \mathbf{x}) + \sum_{j=1}^{k} tmp_j(\mathbf{y}_{B_{new} \cap B_j \cup \{v\}}). \tag{4.8}$$

Further, we need to discuss two cases in the last step. If v is an element in B_{new}, we have $B_{new} \cup \{v\} = B_{new}$ and the Viterbi variables of the new history set is

$$\gamma_{H_{new}}(\mathbf{y}_{B_{new}}) = s(y_v, \mathbf{x}) + \sum_{j=1}^{k} tmp_j(\mathbf{y}_{B_{new} \cap B_j \cup \{v\}}). \tag{4.9}$$

However, it is also possible that v is not in the boundary B_{new}, like node 5 in Figure 4.5. In this case, v is not a representative of the new history set H_{new} and we do not need to retain its assignment in the arguments of $\gamma_{H_{new}}$:

$$\gamma_{H_{new}}(\mathbf{y}_{B_{new}}) = \max_{y_v}\big(s(y_v, \mathbf{x}) + \sum_{j=1}^{k} tmp_j(\mathbf{y}_{B_{new} \cap B_j \cup \{v\}})\big). \tag{4.10}$$

Looking back to the example, $tmp_1(y_5)$ and $tmp_2(y_5, y_7)$ are calculated as in Equation 4.7. Here, the selected node 5 is not an element in $B_{new} = \{7\}$ and the Viterbi variables of the new history set H_{new} is computed according to Equation 4.10, yielding

$$\gamma_{H_{new}}(y_7) = \max_{y_5}\big(s(y_7, \mathbf{x}) + tmp_1(y_5) + tmp_2(y_5, y_7)\big).$$

After this iteration, the v neighboring history sets $H_1, H_2, \ldots, H_k$ are merged together into a new one and the number of history set is reduced to $m - k + 1$. The other $m - k$ history sets are unchanged. So are their Viterbi variables.

To sum up, we present case 2 in the following pseudocode:

Algorithm 5 Case 2 of Algorithm 3 in Step 2

if v is neighbor of $H_1, \ldots, H_k$ **then**
  /* *merge the history sets via* v */
  $H_{new} = H_1 \cup \ldots \cup H_k \cup \{v\}$
  B_{new} = boundary of H_{new}
  /* *calculate* tmp_j *for every* H_j */
  for $j = 1, \ldots, k$ **do**
    compute tmp_j according to 4.7 for every possible assignments of $\mathbf{y}_{B_{new} \cap B_j \cup \{v\}}$
  end for
  /* *discuss* v *in two cases* */
  if $v \in B_{new}$ **then**
    compute $\gamma_{H_{new}}$ according to 4.9 for every possible assignments of $\mathbf{y}_{B_{new}}$
  else
    compute $\gamma_{H_{new}}$ according to 4.10 for every possible assignments of $\mathbf{y}_{B_{new}}$
  end if
  /* *replace* H_1 *with the new history set* */
  $H_1 = H_{new}$
  $B_1 = B_{new}$
  $\gamma_{H_1}(\mathbf{y}_{B_1}) = \gamma_{H_{new}}(\mathbf{y}_{B_1})$, for every possible assignments of $\mathbf{y}_{B_1}$
  /* *shift index of unchanged history sets* */
  for $j = 2, \ldots, m - k + 1$ **do**
    $H_j = H_{j+k-1}$
  end for
  /* *update the number of history sets* */
  $m = m - k + 1$
end if

Remark. *Because the input graph in Algorithm 3 is connected, we will deal with case 2 in the last iteration of the algorithm and get a single history set* $H_1 = V$ *with* $B_1 = \varnothing$. *The* **Viterbi score** γ^* *is simply* $\gamma^* = \gamma_{H_1}$. *According to Equation 4.10, the optimal label of the last selected node* v *is obtained by*

$$y_v^* = \underset{y_v}{\operatorname{argmax}}\big(s(y_v, \mathbf{x}) + \sum_{j=1}^{k} tmp_j(y_v)\big). \tag{4.11}$$

4.4 Backtracking in Generalized Viterbi Algorithm

In order to find an optimal label sequence $\mathbf{y}^*$, we should memorize the best labels of $\mathbf{y}_{B_{H_j} \setminus B_{new}}$ which maximize the tmp_j in Equation 4.7, as well as the best label of y_v in Equation 4.10. Additionally, an one-to-one mapping between the best labels of $\mathbf{y}_{B_{H_j} \setminus B_{new}}$ and the arguments $\mathbf{y}_{B_{new} \cap B_j \cup \{v\}}$ of tmp_j in (4.7), and between y_v and $\mathbf{y}_{B_{new}}$ in (4.10) must be constructed.

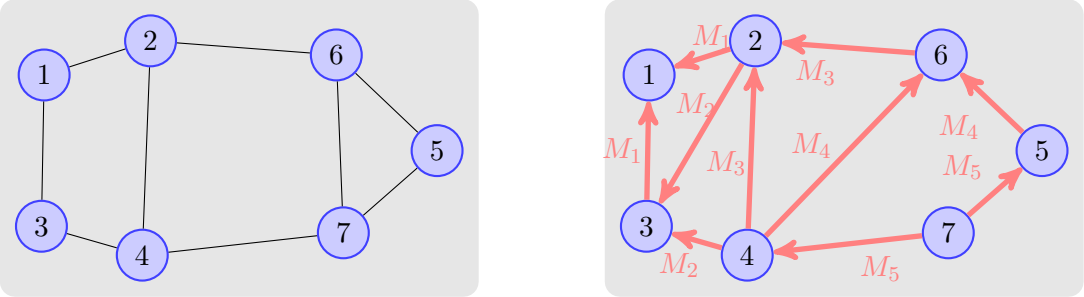

Figure 4.7: An example of backtracking (red arrow) of Algorithm 3.

Let's take the example shown in Figure 4.7. We assume that Algorithm 3 runs with a processing node order $1, 2, \ldots, 7$ and the method can be accomplished within 7 iterations. Since we are interested in backtracking in this section, the calculations of Viterbi variables will not be presented here. We concentrate on the mappings $\mathbf{y}_{B_{H_j} \setminus B_{new}} \longleftrightarrow \mathbf{y}_{B_{new} \cap B_j \cup \{v\}}$, and $y_v \longleftrightarrow \mathbf{y}_{B_{new}}$.

1. The number of history sets $m = 1$.
   $H_1 = \{1\}, B_1 = \{1\}$

2. The number of history sets $m = 1$.
   $H_1 = \{1, 2\}, B_1 = \{1, 2\}$

3. The number of history sets $m = 1$.
   $H_1 = \{1, 2, 3\}, B_1 = \{2, 3\}$
   Mapping: $y_1 \longleftrightarrow y_2, y_3$ (**M1**)

4. The number of history sets $m = 1$.
   $H_1 = \{1, 2, 3, 4\}, B_1 = \{2, 4\}$
   Mapping: $y_3 \longleftrightarrow y_2, y_4$ (**M2**)

5. The number of history sets $m = 2$.
   H_1 and B_1 are not changed. $H_2 = \{5\}, B_2 = \{5\}$

6. The number of history sets $m = 1$.
   $H_1 = \{1, 2, 3, 4, 5, 6\}, B_1 = \{4, 5\}$
   Mapping: $y_2 \longleftrightarrow y_4, y_6$ (**M3**), $y_6 \longleftrightarrow y_4, y_5$ (**M4**)

7. The number of history sets $m = 1$.
   $H_1 = V, B_1 = \varnothing$
   Mapping: $y_4, y_5 \longleftrightarrow y_7$ (**M5**)

The optimal y_7^* can be simply obtained by (4.11) and we can track along the retrorse direction of the mappings to get other optimal labels. The backtracking starts from y_7^* to y_4^*, y_5^*, as is shown with the red colored *M5* in Figure 4.7. Then y_6^* is obtained from y_4^*, y_5^* according to (**M4**). This process will continue along the retrorse direction of mappings (**M3**), (**M2**), until (**M1**). Consequently, an optimal label sequence $\mathbf{y}^*$ is received.

An implementation of generalized Viterbi algorithm including backtracking is presented in [38].

4.5 Complexity of Generalized Viterbi Algorithm

The labeling problem for a general graph is NP-hard. Although the generalized Viterbi algorithm provides a dynamic programming method, it is generally an exponential time algorithm.

It is clear that the complexity of the second case discussed in Section 4.3 dominates the other calculations in the generalized Viterbi algorithm. How many loops need to be executed in this case depends on the number of existing history sets that are neighbors of the observed node. However, it is not easy to know the number preliminarily as they vary for different processing node orders. In order to analyze the complexity, we use the upper bound N, which denotes the maximal degree of a node in the graph, to represent the number of loops.

The generalized Viterbi algorithm is accomplished in $n = |V|$ iterations. We suppose $|B_1^{(1)}|, |B_1^{(2)}|, \ldots, |B_1^{(n)}|$ to be the sequence of boundary size of history set H_1 in every iteration. Furthermore, let d denote the number of possible labels in the label set, i.e.

$d = |\mathcal{L}|$. Using these notations, an upper bound of the running time for the generalized Viterbi algorithm can be expressed as follows:

Lemma 4.3.
The generalized Viterbi algorithm has a complexity of $O(N \sum_{i=1}^{n} d^{|B_1^{(i)}|})$.

Proof. See in *Implementierung einer Verallgemeinerung des Viterbi-Algorithmus zur Dekodierung von Conditional Random Fields* [Maneke, 2009]. □

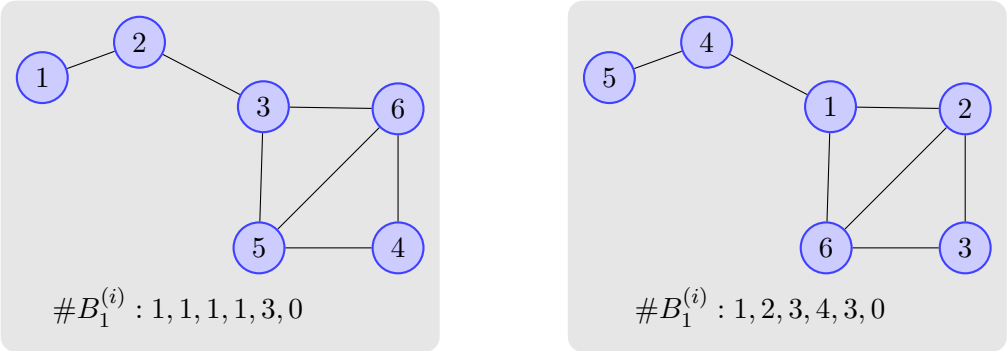

Figure 4.8: Executing Algorithm 3 with different processing node orders.

As we know, the processing node order influences the running time of the generalized Viterbi algorithm. An example is shown in Figure 4.8, which presents two possible processing node orders for the generalized Viterbi algorithm on the same graph.

If the algorithm is proceeded along the ordered nodes shown in the left graph, the sequence of boundary set size $|B_1^{(i)}|, i = 1, 2, \ldots, 6$ is

$$1, 1, 1, 1, 3, 0.$$

Instead, the sequence of $|B_1^{(i)}|, i = 1, 2, \ldots, 6$ for the other processing node order shown in the right graph becomes

$$1, 2, 3, 4, 3, 0.$$

It can be proved that the generalized Viterbi algorithm runs much faster with the processing node order in the left graph than that in the right one.

Since the processing node order plays an important role on the running time of the generalized Viterbi algorithm, it is meaningful to find an efficient one before executing the algorithm. In the next section, a method will be introduced, which finds an efficient node order for the generalized Viterbi algorithm.

4.6 Finding an Efficient Node Order

Developing a method to find an efficient node order is a problem in graph theory. In this section, we focus on the *Delaunay triangulation*[8] graph, because in our application a protein is modeled by such a graph and the details will be presented in the next chapter.

The Delaunay triangulation constructs a set of points into a plane graph with triangle as maximal clique. The benefit is that it maximizes the minimum angle of all the angles of the triangles, such that every two adjacent nodes are not too far away from each other. In this way, the important spatial information like the neighborhood of spatial nearby residues can be presented in the graph, which shows great importance in our application.

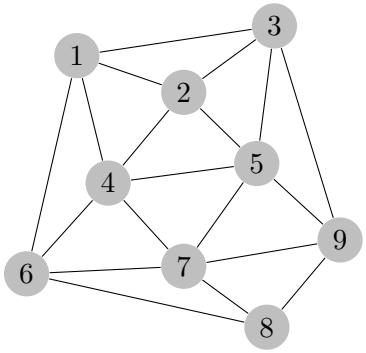

Figure 4.9: A delaunay triangulation graph.

In a Delaunay triangulation graph, each edge is an incident to maximal two triangles as is shown in Figure 4.9. Our method takes pattern from the breadth-first search and uses two queues, a working queue for the edges and an output queue for the nodes. Furthermore, the coloring technique is used to ensure that every node and every edge is added to the corresponding queue exactly once.

Edge Queue	Node Queue
$\{4, 5\}$	4, 5

Table 4.1: Edge queue and node queue in the first iteration.

Now we demonstrate the method with an example. First, the method randomly chooses an edge. This edge is then added to the edge queue and its incident nodes to

the node queue. Here we choose edge $\{4, 5\}$ from the graph shown in Figure 4.9 and add it to the edge queue. At the same time, its two incident nodes are added to the node queue in Table 4.1. In Figure 4.10, edge $\{4, 5\}$ is colored green and so are the two incident nodes.

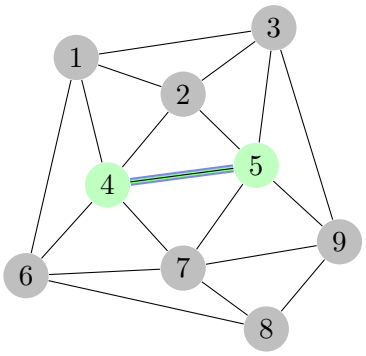

Figure 4.10: The first iteration in the method finding an efficient node order.

As long as the edge queue is not empty, the top edge is dequeued and its incident triangles are taken into consideration. Then, the nodes involved in the triangles will be put into the node queue, if they have not been in it, whether the nodes are in the node queue can be determined quickly by the node color. Similarly, the involved edges will be added to the edge queue, if they have not been colored green. Thereafter, a new top edge will be selected. In the example, edge $\{4, 5\}$ is dequeued, whose incident triangles are $\{2, 4, 5\}$ and $\{4, 5, 7\}$. Among the nodes, node 2 and 7 are not green and are added to the node queue. Analogously, four edges are added to the edge queue after removing edge $\{4, 5\}$. We select a new top edge $\{2, 4\}$ and highlight it with blue color in Table 4.2.

Edge Queue	Node Queue
$\{2, 4\}$	4, 5, 2, 7
$\{2, 5\}$	
$\{4, 7\}$	
$\{5, 7\}$	

Table 4.2: Edge queue and node queue in the second iteration.

In the graph, node 2 and 7 are colored green and so are the four edges. In order to

identify the top edge in Figure 4.11, it is displayed by a thicker green line.

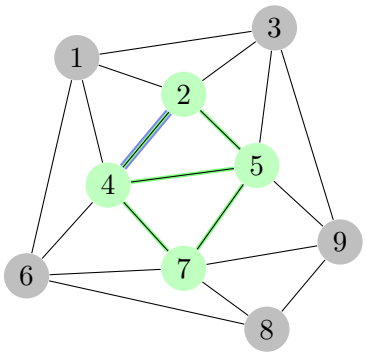

Figure 4.11: The second iteration in the method finding an efficient node order.

This process should be repeated until every edge has been selected once or rather the edge queue is empty. The node order stored in the node queue is the output. If we go on with the example, the output node order will be $4, 5, 2, 7, 1, 3, 6, 9, 8$.

An appropriate processing node order can reduce the running time of the generalized Viterbi algorithm. With the help of the method introduced in this section, we can find an efficient node order in the application. Another benefit is, if we are not satisfied with one order, there is the opportunity to compute another one, because the first edge is chosen uniformly at random.

4.7 Parameter Estimation for Pairwise CRFs

In this section, we solve the training problem for a pairwise CRF, which involves the estimation of the weight vector $\lambda^{(n)}$ for the node feature functions and the weight $\lambda^{(e)}$ for the edge feature functions.

As we discussed before, training a large undirected model is quite difficult. Therefore, approximate training methods are often used instead of exact training. Here we adopt the idea of *piecewise training* [13] which is to divide the whole model into pieces which can be trained independently and paralleled, combining the learned parameters from each local training into a global model. Our approach is to divide the graphs into node set and edge set. Maximizing objective functions based on node set and on edge set, we can get an optimal $\lambda^{(n)}$ and $\lambda^{(e)}$ respectively.

Firstly, we estimate the weight vector of node feature functions over the node set. Assume that we have a training data $\mathcal{D} = \{(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}$. The weight vector can be received by maximizing the following objective function:

$$l_n(\lambda^{(n)}) := \sum_m \sum_i \sum_\varsigma \lambda_\varsigma^{(n)} \psi_\varsigma(i, y_i^{(m)}, \mathbf{x}^{(m)}) - \sum_m \log Z_n(\mathbf{x}^{(m)}, \lambda^{(n)}), \tag{4.12}$$

where $Z_n(\mathbf{x}^{(m)}, \lambda^{(n)})$ is the *local normalization factor*

$$Z_n(\mathbf{x}^{(m)}, \lambda^{(n)}) = \sum_{\mathbf{y}} \exp\big(\sum_i \sum_\varsigma \lambda_\varsigma^{(n)} \psi_\varsigma(i, y_i, \mathbf{x}^{(m)})\big). \tag{4.13}$$

Next, we define the objective function for the edge training as follows:

$$l_e(\lambda^{(e)}) := \sum_m \sum_{(i,j)\in E} \sum_\tau \lambda_\tau^{(e)} \phi_\tau(i, j, y_i^{(m)}, y_j^{(m)}, \mathbf{x}^{(m)}) - \sum_m \log Z_e(\mathbf{x}^{(m)}, \lambda^{(e)}), \tag{4.14}$$

with the corresponding local normalization factor

$$Z_e(\mathbf{x}^{(m)}, \lambda^{(e)}) = \sum_{\mathbf{y}} \exp\big(\sum_{(i,j)\in E} \sum_\tau \lambda_\tau^{(e)} \phi_\tau(i, j, y_i, y_j, \mathbf{x}^{(m)})\big). \tag{4.15}$$

In this way, the weight vectors of node feature functions and edge feature functions can be trained independently over node set and edge set. Moreover, it can be proved that both of the objective functions are concave such that using (L-)BFGS method is also suitable in our training.

Since BFGS method requires the input of the gradient, we calculate the partial derivatives of $l_n(\lambda^{(n)})$ and $l_e(\lambda^{(e)})$ respectively.

$$\begin{aligned}\frac{\partial l_n}{\partial \lambda_\varsigma^{(n)}} &= \sum_m \sum_i \psi_\varsigma(i, y_i^{(m)}, \mathbf{x}^{(m)}) \\ &\quad - \sum_m \frac{1}{Z_n(\mathbf{x}^{(m)}, \lambda^{(n)})} \sum_{\mathbf{y}} \Big(\exp\big(\sum_i \sum_q \lambda_q^{(n)} \psi_q(i, y_i, \mathbf{x}^{(m)})\big) \cdot \big(\sum_i \psi_\varsigma(i, y_i, \mathbf{x}^{(m)})\big)\Big).\end{aligned} \tag{4.16}$$

$$\begin{aligned}\frac{\partial l_e}{\partial \lambda_\tau^{(e)}} &= \sum_m \sum_{(i,j)} \phi_\tau(i, j, y_i^{(m)}, y_j^{(m)}, \mathbf{x}^{(m)}) \\ &\quad - \sum_m \frac{1}{Z_e(\mathbf{x}^{(m)}, \lambda^{(e)})} \sum_{\mathbf{y}} \Big(\exp\big(\sum_{(i,j)} \sum_q \lambda_q^{(e)} \phi_q(i, j, y_i, y_j, \mathbf{x}^{(m)})\big) \cdot \big(\sum_{(i,j)} \phi_\tau(i, j, y_i, y_j, \mathbf{x}^{(m)})\big)\Big).\end{aligned} \tag{4.17}$$

The local normalization factors are computationally much easier than the original one. L-BFGS method will be used to train the two weight vectors independently. Finally, all of the parameters will be combined into a single pairwise CRF to make a global prediction.

5

Protein Data

Simply speaking, proteins are composed of (amino acid) residues and a residue consists of several atoms. In this chapter, some distance-based definitions will be introduced. The main point is to define the distance of two atoms.

Atom Type	**Symbol**	**radius (Å)**
Hydrogen	H	1.20
Carbon	C	1.70
Nitrogen	N	1.55
Oxygen	O	1.52
Fluorine	F	1.47
Phosphorus	P	1.80
Sulfur	S	1.80
Chlorine	Cl	1.75
Copper	Cu	1.40

Table 5.1: Van der Waals radii (taken from [1]).

An atom can be considered as a ball which can be described by its three dimensional center position (x,y,z) and a certain radius r, the *van der Waals radius* [1]. Let at_1 and at_2 denote two atoms with centers (x_1, y_1, z_1), (x_2, y_2, z_2) and radii r_1, r_2, respectively. Now, we focus on two atom-distance definitions from the literature. One of them defines the distance of two atoms simply with the Euclidean norm of theirs centers [23][39]:

$$dist(at_1, at_2) := \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}. \tag{5.1}$$

The other definition takes the van der Waals radius into consideration [22][40]:

$$dist(at_1, at_2) := \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} - r_1 - r_2. \tag{5.2}$$

The van der Waals radii are listed in Table 5.1. The three dimensional information of the atoms can be obtained from the so-called PDB file introduced in the following section.

5.1 A Protein Data Bank

A reliable data set is essential to the research. The Protein Data Bank (PDB) is such a data set which is avaliable online (*www.pdb.org*) for studying biological macromolecules. The PDB contains information about experimentally determined structures of proteins, nucleic acids, and complex assemblies.

Every protein complex deposited in the PDB has a unique ID. A PDB-ID consists of 4 characters, the first of which is a digit in the range 0 - 9; the remaining 3 are alpha-numeric, where letters are upper case only. A protein ID is a alpha-numeric character which represents different proteins with upper case letter and lower case letter. Using a PDB-ID, we can identify a protein complex and obtain its information from the PDB by downloading the corresponding PDB file named *PDB-ID.pdb* which is a textual file describing the 3D structures of the proteins in this complex.

A typical PDB file is presented in lines, such like the following example shown in Table 5.2. Each line begins with a record type, e.g. HEADER, SEQRES, ATOM and so on. The most important information for our research is contained in ATOM records which contains the atomic coordinates for standard amino acid residues in the protein.

More detailly, the first ATOM record in protein complex with PDB-ID *1QDM* describes atom N contained in residue *VAL* (Valine) of protein A. The first three floating point numbers are the x, y and z coordinate of the center position of atom N, which are in units of angstrom. The 6-th column of ATOM record is of our interest as well. It is the residue ID consisting of a sequence number and sometimes with an alphabetic insertion code. With this ID, we can identify every residue in a protein. Thus residue *VAL* with the ID of *6P* in protein chain A is composed of 7 atoms which can be read from the first 7 ATOM records in the PDB file.

```
HEADER    HYDROLASE                                19-MAY-99   1QDM
TITLE     CRYSTAL STRUCTURE OF PROPHYTEPSIN, A ZYMOGEN OF A BARLEY
TITLE    2 VACUOLAR ASPARTIC PROTEINASE.
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: PROPHYTEPSIN;
COMPND   3 CHAIN: A, B, C;
  :              :
  :              :
REMARK 465 THE FOLLOWING RESIDUES WERE NOT LOCATED IN THE
REMARK 465 EXPERIMENT. (M=MODEL NUMBER; RES=RESIDUE NAME; C=CHAIN
REMARK 465 IDENTIFIER; SSSEQ=SEQUENCE NUMBER; I=INSERTION CODE.)
  :              :
  :              :
DBREF  1QDM A    6P  308  UNP    P42210   ASPR_HORVU      31    508
DBREF  1QDM B    6P  308  UNP    P42210   ASPR_HORVU      31    508
DBREF  1QDM C    6P  308  UNP    P42210   ASPR_HORVU      31    508
SEQRES   1 A  478  VAL ARG ILE ALA LEU LYS LYS ARG PRO ILE ASP ARG ASN
SEQRES   2 A  478  SER ARG VAL ALA THR GLY LEU SER GLY GLY GLU GLU GLN
SEQRES   3 A  478  PRO LEU LEU SER GLY ALA ASN PRO LEU ARG SER GLU GLU
  :                                  :
  :                                  :
ATOM      1  N   VAL A   6P     34.377  17.703  -3.619  1.00 47.86           N
ATOM      2  CA  VAL A   6P     33.699  16.952  -4.717  1.00 32.58           C
ATOM      3  C   VAL A   6P     32.380  16.431  -4.183  1.00 19.83           C
ATOM      4  O   VAL A   6P     32.361  15.748  -3.171  1.00 24.36           O
ATOM      5  CB  VAL A   6P     34.538  15.760  -5.152  1.00 32.00           C
ATOM      6  CG1 VAL A   6P     33.999  15.187  -6.444  1.00 32.82           C
ATOM      7  CG2 VAL A   6P     35.978  16.181  -5.298  1.00 41.70           C
ATOM      8  N   ARG A   7P     31.292  16.725  -4.883  1.00 20.22           N
  :                                  :
  :                                  :
ATOM    157  N   GLU A   2      27.407 -15.748  11.053  1.00 73.55           N
ATOM    158  CA  GLU A   2      28.756 -15.727  11.703  1.00 82.41           C
  :                                  :
  :                                  :
END
```

Table 5.2: A part of PDB file of protein complex 1QDM.

Remark. *The angstrom (Å) is an internationally recognized unit of length equal to* 10^{-10} *meters. It is commonly used in structural biology like expressing the size of atoms.*

5.2 Nussinov Database and Data Set *PlaneDimers*

Since our research is to investigate the interaction sites of two proteins, a data set of protein-protein interacting pairs is foremost required. Large-scale experiments reveal pairs of interacting proteins, among which there are many similar interfaces. Using structure comparison algorithm, a non-redundant database has been compiled by the group of *R.Nussinov* [40].

```
 .-- PDB ID
 |    .-- ID of the first interacting protein
 |    |.-- ID of the second interacting protein
 |    ||
1APM  EI
1NC3  AB
1AOC  AC
1QDL  AB
 : :
 : :
1ILU  HM
```

Figure 5.1: A piece of Nussinov database.

Each entry in Nussinov database records the PDB-ID of a protein complex and the IDs of two interacting proteins. The protein ID used in Nussinov database is identical to the protein denomination in the PDB file, making the structure information very conveniently accessible. For example, the first line in Nussinov database shown in Figure 5.1 states that protein E interacts with protein I in protein complex *1APM*.

Interacting protein pairs are also called the *dimers* which can be classified as *homodimers* and *heterodimers*. A pair of interacting proteins is called homodimer, if the two proteins have the same residue sequences; Otherwise it is called heterodimer. Nussinov database contains entries of both homo- and heterodimeric protein pairs.

The data set *PlaneDimers* provided by *Zellner et al.* [22] is our basis of comparison. *PlaneDimers* is a subset of Nussinov database and contains of redundancy-free homodimers with flat protein-protein interfaces. It is publicly available (see *http://www-bioinf.uni-regensburg.de/.*)

5.3 Surface Residues in Proteins

Proteins fold themselves into a three dimensional structure such that some residues are folded in the core as interior residues, whereas the rest of residues remain on the protein surface.

It is believed that residues on the protein surface are more useful in assigning biochemical function than the use of the whole protein and interfaces are formed mostly by residues that are exposed to the solvent if the partner protein is removed. We trimm our data by removing the interior residues from the proteins and leave the surface residues in the data set, which is also done in some other groups [22][23][39].

However, there is no uniform definition of surface residues. Most of the definitions are based on the solvent exposure of a residue, which measures to what extent the residue is accessible to the solvent. This solvent exposure can be numerically described by some measures, e.g. the so-called *accessible surface area* (**asa**) which is quoted in square angstrom ($Å^2$).

For any residue a, the $\mathbf{asa}(a)$ can be deduced from the 3D structure of the protein under study. We compute it by means of the software library *BALL* [2]. The *relative accessible surface area* of a is defined as

$$\mathbf{rasa}(a) := \frac{\mathbf{asa}(a)}{\mathbf{asa}_{max}(a)}, \tag{5.3}$$

where $\mathbf{asa}_{max}(a)$ is the nominal maximum area of a (see Table 5.3).

We classify a residue a as part of the protein surface, if its relative accessible surface area $\mathbf{rasa}(a)$ is at least 5% [4]. Using this criterion, some of the residues are removed from our data set. The fact is that only a little fraction of interface residues defined in the following section do not pass the surface area threshold, which proves the aforementioned argument.

Remark. *In the **asa** calculation, only coordinates of the particular protein is used. All other proteins in the PDB file should be stripped. Otherwise, the accessible surface*

Residue Type	Abbr.	asa_{max}	Residue Type	Abbr.	asa_{max}
Alanine	ALA	113	Methionine	MET	204
Cysteine	CYS	140	Asparagine	ASN	158
Aspartic acid	ASP	151	Proline	PRO	143
Glutamic acid	GLU	183	Glutamine	GLN	189
Phenylalanine	PHE	218	Arginine	ARG	241
Glycine	GLY	85	Serine	SER	122
Histidine	HIS	194	Threonine	THR	146
Isoleucine	ILE	182	Valine	VAL	160
Lysine	LYS	211	Tryptophan	TRP	259
Leucine	LEU	180	Tyrosine	TYR	229

Table 5.3: Nominal maximum area ($Å^2$) of 20 amino acid residues (taken from [22]).

areas of the residues that eventually form the interface with another proteins may be incorrectly calculated.

5.4 Interface Residues

Although pairs of interacting proteins can be experimentally characterized, there is no high-resolution experimental information to capture interface residues. Generally, there are two common methods to define interface residues: one is simply based on the distance of each residue to the partner molecule [20][39], while the other on the loss of surface accessibility when the proteins are separated [11][33].

We use the distance-based approach to define the interface residues. As we mentioned at the beginning of this chapter, this kind of definition is based on the distance of two atoms. Using different atom-distance definitions, two classifications of interface residues from *Li et al.* [39] and *Nussinov et al.* [40] can be obtained as follows:

A surface residue is considered to be an *interface residue*,

- (*Li Def*) if the distance (defined in Equation 5.1) between any of its heavy atom and any heavy atom of its interacting partner is less than 5Å;
- (*Nuss Def*) if the distance (defined in Equation 5.2) between any of its atom and any atom of its interacting partner is less than 0.5Å.

The set of all interface residues in one protein is then called the *interface* of this protein.

	Li Def	*Nuss Def*
no. of protein pairs	63	63
no. of surface residues	22369	22369
no. of interface residues	3424	2402
no. of non-interface residues	18945	19967

Table 5.4: Data overview in data set *PlaneDimers*.

In the data set *PlaneDimers*, for example, there are 63 protein homodimers involving 22369 surface residues. Among them, 3424 and 2402 surface residues are classified as part of the protein interfaces according to *Li* definition and *Nussinov* definition, respectively. Hence, one of the difficulties in protein-protein interface prediction is that there are much more non-interface residues with both definitions (see Table 5.4), which makes the model incline to predict a residue as non-interface residue.

5.5 Spatial Neighborhood and Protein Surface Graphs

In order to use a CRF to predict protein interfaces, a protein should be converted into a graph whose nodes represent the surface residues in the protein. By the application of a linear-chain CRF [39], the edges are referred to as the "backbone" neighbors according to the primary structure of the protein. However, in practice, it is better to model a protein without neglecting its 3D structure. To this end, the edges represent the spatial neighborhood between the residues, which leads to a more meaningful prediction than "backbone" neighbors.

Several neighborhood notions for residues worth to be reminded here. One of them is atom-distance-based definition: two surface residues in a protein are considered *spatial neighbors*, if the distance (defined in Equation 5.1) between their alpha carbon atoms is under 6Å [40]. Another one that relies on the Delaunay triangulation of the protein surface, as is mentioned in Section 4.6.

Delaunay triangulations have already been shown useful in analyzing cavities in proteins [27][42]. In our application, we represent a surface residue in a protein with the mass center of its heavy atoms. Taking the centers-of-mass set of all surface residues

as the input, we obtain a *Delaunay triangulation graph* by using Fortune's algorithm [46]. Since the Delaunay triangulation maximizes the minimum angle of all the angles of the triangles, the adjacent nodes in the graph represent mostly the spatial neighbors in the protein. In general, the Delaunay triangulation yields a graph with a sizeable number of edges. This is an inconvenience to the generalized Viterbi algorithm, because a large number of edges directly affects the complexity of the algorithm. Based on this observation, we get the *protein surface graphs* by removing nonsignificant edges from the triangulation graphs. The edge reduction principle will be explained in the following chapter.

6

Results and Discussion

In this chapter, we apply pairwise CRFs to predict protein-protein interaction sites. Firstly, we define feature functions from protein characteristics, followed by the leave-one-out cross-validation experiments on the data set *PlaneDimers*. The performance of the predictions is discussed in the end.

6.1 Protein Characteristics Used in CRF Feature

We start by reviewing some protein characteristics in literature. The relative accessible surface areas are commonly used. For example, this residue property was turned into node feature functions by *Li et al.* in their linear-chain CRF [39].

Besides, multiple sequence alignments (MSAs) are an essential tool for protein structure and function prediction. An MSA of a protein is a sequence alignment of protein sequences with the goal that residues in a given column are homologous or play a common functional role. Sometimes, one mutation in a certain column of an MSA influences a compensating mutation in another column, which means that the two involved sites are coevolved. The residues involved in the compensatory mutations may form key sites for the interaction between proteins [16][34][43]. In order to use this signal, the first task is to detect the coevolved positions in MSAs.

We used the dictionary of secondary structure of proteins database (DSSP) [49] to get the MSA of each protein. Let $i \neq j$ be two sites of protein surface. Under the study of the associated colunms in the MSA, we calculated the normalized measure of

mutual information according to [43]:

$$\mathbb{U}(i,j) := 2 \cdot \frac{\mathbb{H}(i) + \mathbb{H}(j) - \mathbb{H}(i,j)}{\mathbb{H}(i) + \mathbb{H}(j)}, \tag{6.1}$$

where $\mathbb{H}(j)$, $\mathbb{H}(j)$ and $\mathbb{H}(i,j)$ are the entropy and joint entropy of the empirical amino acid residue (pair) distribution of column i, of column j and of column pair $(i,\ j)$, respectively. The larger the $\mathbb{U}(i,j)$-value is, the more i and j are coevolved. Position pair $(i,\ j)$ is considered as *important*, if their $\mathbb{U}(i,j)$-value is above a threshold which is chosen according to [37].

This information was then taken into the node by assigning each node in the protein surface graph with a boolean variable:

$$\text{coevolved}(i) := \begin{cases} 1, & \text{if there is a } j \neq i \text{ with } (i,j) \text{ is important;} \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

Another interface residue property of interest is based on the *patch classifier* [44]. The main idea is to distinguish between a random surface area of the protein (random surface patch) and an area which consists mainly of the interface residues (interface patch). For a node i, we generate a patch which is the node set with the first and the second level neighborhood of i.

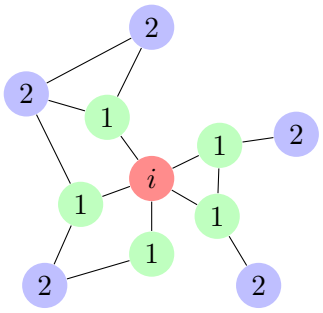

Figure 6.1: Patch of i with the first and the second level neighbors.

The first level neighbors marked with 1 in Figure 6.1 refer to the node k whose corresponding residue a_k is a spatial neighbor of a_i according to the distance-based definition in Section 5.5. The second level neighbors marked with 2 are the spatial neighbors of the first level neighbors.

By using the patch classifier, we can distinguish the interface patch from the random surface patch. Similar to the definition in Equation 6.2, we put the patch information

into a node i:

$$\text{patch}(i) := \begin{cases} 1, & \text{if the patch of } i \text{ is an interface patch;} \\ 0, & \text{otherwise.} \end{cases} \tag{6.3}$$

In addition, we use the *presCont score* which can be obtained from the webserver *PresCont* [22]. The presCont score with the range of 0 to 1 represents the posterior probability for the interface classification of a node i, which we refer to as presCont(i). A higher presCont score means a more probable interface residue prediction.

6.2 Feature Functions in pCRF

According to [23] we average every raw residue characteristics over the neighborhood $\mathcal{N}(a_i)$ stabilizing the signals on grounds of the weak law of large numbers, where $\mathcal{N}(a_i)$ is the set of the spatial neighbors of residue a_i due to the definition in Section 5.5. Assume that we have a protein surface graph $G = (V, E)$. For every node $i \in V$, we define the average surrounding characteristics with consideration of the aforementioned values of its neighboring residues:

$$\text{Rasa}^n(i, \mathbf{x}) := \frac{1}{\#\mathcal{N}(a_i)} \sum_k \mathbf{rasa}(a_k), \tag{6.4}$$

$$\text{Coevolved}^n(i, \mathbf{x}) := \frac{1}{\#\mathcal{N}(a_i)} \sum_k \text{coevolved}(k), \tag{6.5}$$

$$\text{Patch}^n(i, \mathbf{x}) := \frac{1}{\#\mathcal{N}(a_i)} \sum_k \text{patch}(k), \tag{6.6}$$

$$\text{PresCont}^n(i, \mathbf{x}) := \frac{1}{\#\mathcal{N}(a_i)} \sum_k \text{presCont}(k), \tag{6.7}$$

where k ranges over all nodes such that $a_k \in \mathcal{N}(a_i)$.

In our application, we employ a standard step function technique to define the CRF feature functions. Initially, we focus on the node features and take some node characteristic C into consideration, e.g. $\text{Rasa}^n, \text{Coevolved}^n, \text{PresCont}^n$. We subdivide the range of the characteristic C into γ intervals, where γ is not less than two. Let $s_0 < s_1 < \ldots < s_\gamma$ be the corresponding sampling points. It is reasonable to assume that $s\iota$ is the $\iota/\gamma-$quantile of the empirical distribution of C with $C(i, \mathbf{x}) \in [s_0, s_\gamma]$.

Then we define for each $i \in V$ the following 2γ node feature functions associated with the characteristic C:

$$\psi_{y,\iota}^{(C)}(i, y_i, \mathbf{x}) := \begin{cases} 1, & \text{if } y_i = y \text{ and } C(i, \mathbf{x}) \in [s_\iota, s_{\iota+1}]; \\ -1, & \text{otherwise;} \end{cases} \tag{6.8}$$

where $y \in \mathcal{L} = \{\mathrm{I}, \mathrm{N}\}$ and $\iota = 0, 1, \ldots, \gamma - 1$.

In order to define edge feature functions, we must make use of edge characteristics which can be transformed from every node characteristic C into a uniform manner. Let $(i, j) \in E$ be an edge in the protein surface graph. As before, we consider the average characteristics over the neighborhood $\mathcal{N}(i, j)$, which is the set of nodes k such that residue a_k is an element in $\mathcal{N}(a_i)$ or in $\mathcal{N}(a_j)$ Then, an edge characteristic D is defined as

$$D(i, j, \mathbf{x}) := \mathrm{crop}_l^u\Big(\frac{1}{\#\mathcal{N}(i, j)} \sum_{k \in \mathcal{N}(i,j)} C(k, \mathbf{x})\Big), \tag{6.9}$$

where crop_l^u is the *crop function* defined by $\mathrm{crop}_l^u(s) = \mathbb{1}_{s \in [l,u]} \cdot s$, where $[l, u]$ is referred to as the *crop interval*. An appropriate crop function is sometimes effective to sharpen the signal. The default crop interval is set to $[0, 1]$.

As mentioned in Section 5.5, a protein surface graph is obtained by reducing non-significant edges from its triangulation graph. In this work, we classify an edge as *significant*, if its characteristic based on the presCont score is over 0.65. The corresponding $PresCont^e$ characteristic is defined as follows:

$$\mathrm{PresCont}^e(i, j, \mathbf{x}) := \mathrm{crop}_l^u\Big(\frac{1}{\#\mathcal{N}(i, j)} \sum_{k \in \mathcal{N}(i,j)} \mathrm{PresCont}^n(k, \mathbf{x})\Big), \tag{6.10}$$

where the crop interval is set to $[0.65, 1]$. In this way, all nonsignificant edges are removed from the surface graphs.

Two additional applied edge characteristics are based on $\mathrm{Coevolved}^n$ and Patch^n with the default crop interval.

Analogously, we obtain for each $(i, j) \in E$ the following 4γ edge feature functions based on an edge characteristic D:

$$\phi_{yy',\iota}^{(D)}(i, j, y_i, y_j, \mathbf{x}) := \begin{cases} 1, & \text{if } y_i y_j = yy', \text{ and } D(i, j, \mathbf{x}) \in [s_\iota, s_{\iota+1}]; \\ -1, & \text{otherwise;} \end{cases} \tag{6.11}$$

where $yy' \in \{\mathrm{II}, \mathrm{IN}, \mathrm{NI}, \mathrm{NN}\}$, and $\iota = 0, 1, \ldots, \gamma - 1$.

6.3 Performance of the Prediction

In order to evaluate the performance of our pairwise CRF, we introduce firstly some evaluation measures. Generally speaking, the prediction should cover as many of the actual interface residues as possible. More important is that the model should predict as few false positives as possible. We denote TP and FP the number of true and false positives, which is the number of correctly and incorrectly labeled interface residues, respectively. For correctly and incorrectly labeled non-interface residues, we use the notation TN and FN. In the literature, the following measures are taken into account [23][39][47]:

1. $precision = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$
2. $recall = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$
3. $accuracy = (\mathrm{TP} + \mathrm{TN})/(\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN})$

The most important measure in our application is *precision* which represents the ratio of correctly labeled interface residues to all labeled interface residues including false positives. A higher *precision* means that fewer interface residues are incorrectly labeled. Apart from this, *recall* returns the percentage of interface residues recognized from all real interface residues. The more real interface residues are predicted, the higher the *recall*. These two measure the performance for labeling interface residues. The last one *accuracy* is used to evaluate the performance for the whole test data including both interface and non-interface residues.

As mentioned earlier, the data set is unbalanced with respect to the number of interface and non-interface residues. Therefore, it is necessary to manipulate the ratio of these to parameters. Here we enhance the influence of the positive examples, rather than select various sets of training data by deleting negative ones as done in [39].

Let ν_I, ν_N, ν_{II} and ν_{NN} be the number of interface nodes, the number of non-interface nodes, the number of interface-interface edges, and the number of non-interface-non-interface edges in a training data $\mathcal{D} = \{(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}$, respectively. We define the following *enhancer functions* for all nodes i and for all edges (i, j).

$$\eta_n(i) := \begin{cases} \frac{\nu_N}{\nu_I} - 1, & \text{if } y_i = \mathrm{I}; \\ 0, & \text{if } y_i = \mathrm{N}. \end{cases} \tag{6.12}$$

$$\eta_e(i,j) := \begin{cases} \frac{\nu_{NN}}{\nu_{II}} - 1, & \text{if } y_i = y_j = \text{I}; \\ \frac{\nu_N}{\nu_I} - 1, & \text{if } y_i \neq y_j; \\ 0, & \text{if } y_i = y_j = \text{N}. \end{cases} \tag{6.13}$$

To uniformly govern the influence of the enhancers, an *enhancer control parameter* $\eta_c \in [0,1]$ is used to control the trade-off between precision and recall. Then, the log-likelihood objective functions are set up by

$$\begin{aligned} l_n(\lambda^{(n)}, \eta_c) : = & \sum_m \sum_i (1 + \eta_c \eta_n(i)) \sum_\varsigma \lambda_\varsigma^{(n)} \psi_\varsigma(i, y_i^{(m)}, \mathbf{x}^{(m)}) \\ & - \sum_m \log Z_n(\mathbf{x}^{(m)}, \lambda^{(n)}, \eta_c), \end{aligned} \tag{6.14}$$

$$\begin{aligned} l_e(\lambda^{(e)}, \eta_c) : = & \sum_m \sum_{(i,j)\in E} (1 + \eta_c \eta_e(i,j)) \sum_\tau \lambda_\tau^{(e)} \phi_\tau(i, j, y_i^{(m)}, y_j^{(m)}, \mathbf{x}^{(m)}) \\ & - \sum_m \log Z_e(\mathbf{x}^{(m)}, \lambda^{(e)}, \eta_c), \end{aligned} \tag{6.15}$$

where the local normalization factors

$$Z_n(\mathbf{x}^{(m)}, \lambda^{(n)}, \eta_c) = \sum_{\mathbf{y}} \exp\big(\sum_i (1 + \eta_c \eta_n(i)) \sum_\varsigma \lambda_\varsigma^{(n)} \psi_\varsigma(i, y_i, \mathbf{x}^{(m)})\big) \tag{6.16}$$

and

$$Z_e(\mathbf{x}^{(m)}, \lambda^{(e)}, \eta_c) = \sum_{\mathbf{y}} \exp\big(\sum_{(i,j)\in E} (1 + \eta_c \eta_e(i,j)) \sum_\tau \lambda_\tau^{(e)} \phi_\tau(i, j, y_i, y_j, \mathbf{x}^{(m)})\big). \tag{6.17}$$

The above defined objective functions are concave and are maximized by using L-BFGS method to get the parameters.

Besides, leave-one-out cross-validation experiments are performed on our pairwise CRF with the data set *PlaneDimers*. In every round, a protein dimer is selected and removed from the data set. The remaining data are used to train the model which in the next step predict the interacting sites of the selected protein dimer. This process is repeated until each dimer has been tested once. The benefit of this strategy is that it can avoid the overfitting problem.

The observed node and edge characteristics are (a) Rasa^n, (b) Coevolved^n, (c) PresCont^n, (d) Patch^n, (e) Coevolved^e, (f) PresCont^e and (g) Patch^e. By defining node feature functions with Rasa^n and Coevolved^n, we use the step functions with two

intervals (see Equation 6.8) according to the 1/2 quantile of the empirical distribution of Rasa^n and Coevolved^n, respectively. The two intervals used in definition of PresCont^n-features are set by $s_0 = 0, s_1 = 0.82$ and $s_2 = 1$. Since the most nodes are evaluated with 0 by Patch^n, it is not necessary to assign the nodes into different intervals. We simply define for $y \in \{\mathrm{I}, \mathrm{N}\}$:

$$\psi_y^{(\mathrm{Patch}^n)}(i, y_i, \mathbf{x}) := \begin{cases} \mathrm{Patch}^n(i, \mathbf{x}), & \text{if } y_i = y; \\ -1, & \text{otherwise.} \end{cases} \tag{6.18}$$

We now turn to the edge feature definition. Firstly, $\phi_{yy',\iota}^{(\mathrm{Coevolved}^e)}$ are defined by Equation 6.11 with three intervals according to the 1/3 quantile of the empirical distribution of Coevolved^e. We use two intervals with $s_0 = 0.65, s_1 = 0.80, s_2 = 1$ to define the PresCont^e-step feature functions. With the same argument in Patch^n-feature definition, we define edge feature functions with Patch^e and $yy' \in \{\mathrm{II}, \mathrm{IN}, \mathrm{NI}, \mathrm{NN}\}$:

$$\phi_{yy'}^{(\mathrm{Patch}^e)}(i, j, y_i, y_j, \mathbf{x}) := \begin{cases} \mathrm{Patch}^e(i, j, \mathbf{x}), & \text{if } y_i y_j = yy'; \\ -1, & \text{otherwise.} \end{cases} \tag{6.19}$$

Our basic data set is *PlaneDimers*. The following tables list the performance of our pCRF with some combination of features. The enhancer control parameter η_c is set to 0.3. We tabulate two sets of results according to the interface residue definition by *Li et al.* and by *Nussinov et al.*, respectively.

	Node Chara.	Edge Chara.	*precision*	*recall*	*accuracy*
1.	(c)	(f)	44.89%	9.51%	88.83%
2.	(a)(c)	(f)	45.13%	12.29%	88.97%
3.	(a)(b)	(e)	23.88%	68.00%	72.97%
4.	(a)(c)(d)	(f)(g)	**45.30%**	**12.41%**	**88.98%**
5.	(a)(b)(c)(d)	(e)(f)(g)	41.67%	17.96%	88.48%

Table 6.1: Performance of pCRF with *Nuss Def.*

The first row in Table 6.1 and 6.2 shows the performance of the pCRF using only feature functions based on the *presCont* score. By adding features of Rasa^n and Patch^n, Patch^e, the pCRF achieves the best performance with the bold-faced evaluation values. Another interesting feature combination is presented in the third row. More than 60% interface residues can be recognized from the data set, which is largely due to the

	Node Chara.	Edge Chara.	*precision*	*recall*	*accuracy*
1.	(c)	(f)	50.67%	7.29%	84.72%
2.	(a)(c)	(f)	51.57%	9.53%	84.79%
3.	(a)(b)	(e)	32.65%	63.14%	73.87%
4.	(a)(c)(d)	(f)(g)	**51.86%**	**9.65%**	**84.80%**
5.	(a)(b)(c)(d)	(e)(f)(g)	48.77%	14.23%	84.58%

Table 6.2: Performance of pCRF with *Li Def.*

coevolved-features. In case that the application requires a high *recall*, it would be a good choice to use this feature combination in a pCRF.

6.4 Model Modification

We have seen that it is not easy to obtain a high *precision* of the prediction. One of the reasons is that the edges with label IN or NI carry much noisy signal over the characteristics. In this section, we follow the idea of interface-residue-connectedness [40] that the neighbors of an interface residue very possibly also belong to the interacting region. In this sense, only II- and NN-edge feature functions will be taken into account. The edge feature functions defined in Equation 6.11 are limited to $yy' \in \{\mathrm{II}, \mathrm{NN}\}$.

We still use the data set *PlaneDimers* with enhancer control parameter $\eta_c = 0.3$. Since there are no IN- or NI-edge feature functions defined in our pCRF, we should handle the edges with label IN or NI in the training. Due to the interface-residue-connectedness, IN- and NI-edges are considered to be II-edges in the training. Since our basic data set is the same one as the webserver *PresCont*, it is reasonable to make a comparison of the performances. To be impartial, the features used in the pCRF are only PresCont^n and PresCont^e.

Method	Interface Def	*precision*	*recall*	*accuracy*
Webserver *PresCont*	*NussDef*	44.14%	20.51%	88.70%
Modified pCRF	*NussDef*	**48.37%**	11.18%	89.13%
Webserver *PresCont*	*LiDef*	51.30%	16.65%	84.69%
Modified pCRF	*LiDef*	**53.92%**	8.71%	84.76%

Table 6.3: Performances of webserver *PresCont* and modified pCRF.

The result in Table 6.3 shows that our modified pCRF obtains a higher *precision* with 48.37% by *Nussinov* definition and 53.92% by *Li* definition. As discussed before, *precision* is the most important measure in our application, to this end, it is worth a lower *recall*. Moreover, a slightly higher *accuracy* is achieved by our modified pCRF.

We have seen in the previous section that our pairwise CRF with the feature combination of Rasa^n, PresCont^n, Patch^n, PresCont^e and Patch^e obtained the best performance (Table 6.1 & 6.2). It is interesting to use this combination in the modified pairwise CRF. Nevertheless, this feature combination seems not so suitable for the modified pairwise CRF with a *precision* of 47.04% (52.41%) and a *recall* of 9.59% (7.47%) by *Nussinov* (*Li*) definition. A possible reason is that we reduced the number of edges in the protein surface graphs according to the presCont score (see Equation 6.10). The removed nonsignificant edges can be significant for the other features. An II-edge in the remaining edges has probably a relativly large PresCont^e-value, while an NN-edge has probably a relativly small PresCont^e-value. This relation does not necessarily exist in other features. In this situation, the modified pCRF with the feature combination of PresCont^n and PresCont^e fits the data better. In our further work, the computing capability of our generalized Viterbi algorithm should be enhanced. The goal is to use the Delaunay triangulation graphs directly without edge reductions.

7

Conclusion

Protein-protein interactions appear in almost every biological process. Proteins are mainly folded into three dimensional structures, which requires that a meaningful prediction should take consideration of the important spatial relationships of the amino acid residues. In this regard, many research groups use the spatial neighborhood information to evaluate the residues in their predictions, but the model itself do not consider the dependencies between the spatial neighboring residues. Our contribution is modeling the spatial neighborhood information of a protein directly into a graphical model based on the approach of conditional random fields.

CRF is a stochastic graphical model used for solving segmenting and labeling problems. Because training a general CRF is intractable, we developed pairwise CRFs in our application. In order to find the best label sequence using a pairwise CRF, a generalized Viterbi algorithm was proposed. This algorithm extends the idea of dynamic programming to a general graph. One of the best label sequences can be obtained by processing the algorithm with an efficient node order. Furthermore, the parameters in a pairwise CRF were estimated by dividing the graphs into nodes and edges. The weights of node feature functions and of edge feature functions were trained separately.

In the protein-protein interaction sites prediction, we selected several protein properties in the CRF-feature definition. Each property was represented by the related node and (or) edge feature functions. Due to the unbalanced data set, we manipulated the ratio of the number of interface and non-interface residues with enhancer functions and an enhancer control parameter. In addition, we applied a modified pairwise CRF with the feature functions based on the same protein properties as the webserver *PresCont*.

By comparison, our pairwise CRF showed a higher precision evidencing its abilities in handling the data of graphical structures.

Since the generalized Viterbi algorithm has a limited computing power, the number of edges in the protein surface graphs should be reduced. Because of the edge reduction according to the presCont score, the performance of our pCRFs is adversely affected. In the future work, the computing capability of the generalized Viterbi algorithm will be enhanced so that the edge reduction can be prevented. Besides, the triangle cliques will be taken into account in the model. To this end, a new CRF variant is to be developed. Meanwhile, more signals from the data analysis are required in order to define more reasonable feature functions in the CRF, which is expected to result in a better performance in the protein-protein interaction sites prediction.

Bibliography

[1] Bondi A. van der waals volumes and radii. *J. Phys. Chem.*, 68(3):441451, 1964.

[2] Hildebrandt A, Dehof AK, Rurainski A, et al. Ball - biochemical algorithms library 1.3. *BMC Bioinformatics*, 11:531, 2010.

[3] Koike A and Takagi T. Prediction of protein-protein interaction sites using support vector machines. *Protein Eng.Des.Sel.*, 17:165–173, 2004.

[4] Porollo A and Meller J. Prediction-based fingerprints of protein-protein interactions. *Proteins*, 66:630–645, 2007.

[5] Quattoni A, Collins M, and Darrell T. Conditional random fields for object recognition. *MIT Press*, pages 1097–1104, 2004.

[6] Bordner AJ and Abagyan R. Statistical analysis and prediction of protein-protein interfaces. *Proteins*, 60:353–366, 2005.

[7] Viterbi AJ. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[8] Delaunay B. Sur la sphere vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[9] Settles B. Abner: an open source tool for automaically tagging genes,proteins,and other entity names in text. *Bioinformatics*, 21(14):3191–3192, 2005.

[10] Wang B, Chen P, Huang DS, et al. Predicting protein interaction sites from residue spatial sequence profile and evolution rate. *FEBS Lett.*, 580:380–384, 2006.

[11] Chothia C and Janin J. Principles of protein-protein recognition. *Nature*, 256:705–708, 1975.

[12] Sutton C and McCallum A. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2006.

[13] Sutton C and McCallum A. Piecewise pseudolikelihood for efficient crf training. In *International Conference on Machine Learning (ICML)*, pages 863–870. ACM Press, 2007.

[14] Broyden CG. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:7690, 1970.

[15] Yan CH, Honava V, and Dobbs D. Identification of interface residues in protease-inhihitor and antigen-antibody complexes: a support vector machine approach. *Neural.Comput.Appl.*, 13:123–129, 2004.

[16] Yeang CH and Haussler D. Detecting coevolution in and among protein domains. *PLoS ComputBiol*, 3(11):e211, 2007.

[17] DeCaprio D, Vinson JP, Pearson MD, et al. Gene prediction using conditional random fields. *Genome Res.*, 17:1389–1398, 2007.

[18] Sha F and Pereira F. Shallow parsing with conditional random fields. *HLT-NAACL*, pages 213–220, 2003.

[19] Peng FC, Feng FF, and McCallum A. Chinese segmentation and new word detection using conditional random fields. *COLING*, pages 562–568, 2004.

[20] Chen H and Zhou HX. Prediction of interface residues in protein-protein complexes by a consensus neural network mehtod: test against nmr data. *Proteins*, 61:21–35, 2005.

[21] Neuvirth H, Raz R, and Schreiber G. Promate: a structure based prediction program to identify the location of protein-protein binding sites. *J.Mol.Biol.*, 338:181–199, 2004.

[22] Zellner H, Staudigel M, Trenner M, Bittkowski M, Wolowski V, Icking M, and Merkl R. Prescont: Predicting protein-protein interfaces utilizing four residue properties. *Proteins*, 80(1):154–68, 2012.

[23] Zhou HX and Shan Y. Prediction of protein interaction sites from sequence profile and residue neighbor list. *Proteins*, 44:336–343, 2001.

[24] Kufareva I, Budagyan L, Raush E, et al. Pier: protein interface recognition for structural proteomics. *Proteins*, 67:400–417, 2007.

[25] Res I, Mihalek I, and Lichtarge O. An evolution based classifier for prediction of protein interfaces without using protein structures. *Bioinformatics*, 21:2496–2501, 2005.

[26] Lafferty J, McCallum A, and Pereira F. Conditinal random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th International Conf. on Machine Learning*, pages 282–289, 2001.

[27] Liang J, Edelsbrunner H, and Woodward C. Anatomy of protein pockets and cavities: measurement of binding site geometry and implications for ligand design. *Protein Sci*, 7:18841897, 1998.

[28] Nocedal J. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.

[29] Li JJ, Huang DS, Wang B, et al. Identifying protein-protein interfacial residues in heterocomplexes using residue conservation scores. *Int.J.Biol.Macromol.*, 38:241–247, 2006.

[30] Bradford JR, Needham CJ, Bulpitt AJ, et al. Insights into protein-protein interfaces unsing a bayesian network prediction method. *J.Mol.Biol.*, 362:365–386, 2006.

[31] Bradford JR and Westhead DR. Improved prediction of protein-protein binding sites using a support vector machines approach. *Bioinformatics*, 21:1487–1494, 2005.

[32] Sato K and Sakakibara Y. Rna secondary structural alignment with conditional random fields. *Bioinformatics*, 21:237–242, 2005.

[33] Lo Conte L, Chothia C, and Janin J. The atomic structure of protein-protein recognition sites. *J Mol Biol*, 285:2177–2198, 1999.

[34] Martin LC, Gloor GB, Dunn SD, and Wahl LM. Using information theory to search for co-evolving residues in proteins. *Bioinformatics*, 21(22):4116–4124, 2005.

[35] Baum LE, Petrie T, Soules G, and Weiss N. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Ann. Math. Statist.*, 41:164171, 1970.

[36] Rabiner LR. A tutorial on hidden markov models and selected applications in speech recognition. *IEEE*, 77(2):257–286, 1989.

[37] Gültas M, Haubrock M, Tüysüz N, and Waack S. Coupled Mutation Finder: A new entropy-based method quantifying phylogenetic noise for the detection of compensatory mutations. *BMC Bioinformatics*, 13:225, 2012.

[38] Maneke M. Implementierung einer verallgemeinerung des viterbi-algorithmus zur dekodierung von conditional random fields. Master's thesis, Georg-August-Universität Göttingen, Germany, 2009.

[39] Li MH, Lin L, Wang XL, et al. Protein-protein interaction site prediction based on conditional random fields. *Bioinformatics*, 23:597–604, 2007.

[40] Keskin O, Tsai CJ, Wolfson H, and Nussinov R. A new, structurally nonredundant, diverse data set of protein-protein interfaces and its implications. *Protein Science*, 13(4):1043–1055, 2004.

[41] Fariselli P, Pazos F, Valencia A, et al. Prediction of protein-protein interaction in heterocomplexes with neural networks. *Eur.J.Biochem.*, 269:1356–1361, 2002.

[42] Medek P, Benes P, and Sochor J. Computation of tunnels in protein molecules using delaunay triangulation. *Journal of WSCG*, pages 107–114, 2007.

[43] Merkl R and Zwick M. H2r: Identification of evolutionary important residues by means of an entropy based analysis of multiple sequence alignments. *BMC Bioinformatics*, 9(1):151, 2008.

[44] Asper RY. Classifiers for discrimination of significant protein residues and protein-protein interaction using concepts of information theory and machine learning. Master's thesis, Georg-August-Universität Göttingen, Germany, 2011.

[45] Boyd S and Vandenberghe L. *Convex Optimization*. Cambridge University Press, 2004.

[46] Fortune S. *Voronoi diagrams and Delaunay triangulations*. CRC Press, Inc., 1997.

[47] Qin S and Zhou H. meta-ppisp: a meta web server for protein-protein interaction site prediction. *Bioinformatics*, 23:3386–3387, 2007.

[48] Friedrich T, Pils B, Dandekar T, et al. Modelling interaction sites in protein domains with interaction profile hidden markov models. *Bioinformatics*, 22:2851–2857, 2006.

[49] Kabsch W and Sander C. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–637, 1983.

[50] Ofran Y and Rost B. Predicted protein-protein interaction sites from local sequence information. *FEBS Lett.*, 544:236–239, 2003.