

**Propagating Interaction Logic
Toward Predictive Protein
Hypernetworks**

Johannes Köster

Algorithm Engineering Report
TR11-4-002
February 2011
ISSN 1864-4503

Diploma Thesis

**Propagating Interaction Logic Toward
Predictive Protein Hypernetworks**

Johannes Köster
August 27, 2010

Supervisors:

Prof. Dr. Sven Rahmann

(Technische Universität Dortmund)

Dr. Eli Zamir

*(Max Planck Institute of Molecular
Physiology Dortmund)*

Fakultät für Informatik
Algorithm Engineering (Ls11)
Technische Universität Dortmund
<http://ls11-www.cs.uni-dortmund.de>

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 2 | Mathematical Foundations | 5 |
| 2.1 | Graphs | 5 |
| 2.2 | Trees | 6 |
| 3 | Modal Logic | 9 |
| 3.1 | Syntax and Semantic | 9 |
| 3.2 | The Modal Logic Tableau | 13 |
| 3.2.1 | Idea | 13 |
| 3.2.2 | Optimization | 14 |
| 3.2.3 | Implementation | 17 |
| 4 | Protein Hypernetworks | 23 |
| 4.1 | Incorporating Interaction Logic | 24 |
| 4.2 | Data Mining in Protein Hypernetworks | 27 |
| 4.2.1 | Perturbation Effects | 28 |
| 4.2.2 | Minimal Network States | 29 |
| 5 | Prediction of Protein Complexes | 35 |
| 5.1 | Measuring Prediction Quality | 36 |
| 5.2 | The LCM Algorithm | 37 |
| 5.2.1 | Detecting local cliques | 37 |
| 5.2.2 | Merging dense regions | 39 |
| 5.2.3 | Implementation | 39 |
| 5.2.4 | Discussion | 41 |
| 5.3 | Complex Prediction with Protein Hypernetworks | 42 |
| 5.3.1 | Perform Perturbations | 42 |
| 5.3.2 | Network Based Complex Prediction | 42 |
| 5.3.3 | Compute Minimal Network States | 43 |
| 5.3.4 | Refine predicted complexes | 43 |
| 5.3.5 | Implementation | 49 |
| 5.3.6 | Discussion | 49 |
| 6 | Protein Hypernetwork Analysis | 53 |
| 6.1 | Prediction of Master Switches | 53 |
| 6.2 | Prediction of Functional Similarity | 57 |
| 7 | Conclusion and Outlook | 61 |
| | Further Informations | 65 |

Contents

| | |
|-------------------------|-----------|
| Figures | 69 |
| Tables | 71 |
| Algorithms | 73 |
| Bibliography | 75 |
| Acknowledgements | 77 |
| Erklärung | 79 |

1 Introduction

Proteins are fundamental building blocks of cells, the smallest unit of life. Most of the cellular functions are executed by proteins: among many other functions, they form channels in the cell membrane to allow the passage of small molecules, act as enzymes to promote chemical reactions and create or carry signals between adjacent cells or different parts of a single cell.¹

A protein consists of at least one chain of amino acids linked by *covalent bonds*. In general a protein is restricted to certain three-dimensional *conformations* by non-covalent bonds between parts of its chain (hydrogen bonds, electrostatic attractions and van der Waals attractions) and hydrophobicity of parts of certain amino acids (the property to be repelled by water molecules).

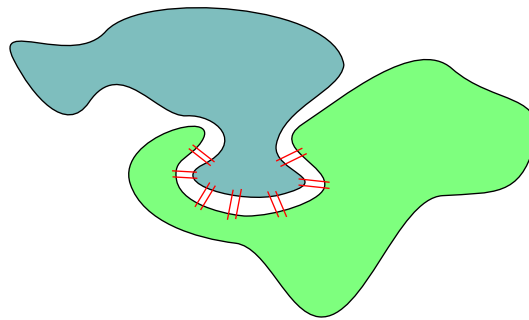


Figure 1.1: Binding of a protein to another molecule. A cavity in the proteins surface allows the molecule to fit tightly so that a large number of non-covalent bonds can be formed, providing a stable binding. Derived from Alberts et al. (2007).

The source of proteins' power is their ability to bind other molecules. This happens by non-covalent bonds between the surfaces. Since non-covalent bonds are much weaker than covalent ones, several of them are needed to provide a stable binding. Hence binding is only possible in an area of the protein surface the three dimensional conformation of which fits closely to a part of the binding molecule (Figure 1.1). Such an area is called *binding domain* or *binding site*. This kind of binding allows for a very high specificity so that only a few or even only one type of molecule may be able to bind to a certain domain. Of special interest in our case is the interaction between two proteins, which is in most cases provided by this kind of binding. Since the surface contour is essential for this, the binding between two proteins can be influenced by their conformation. A change in conformation can induce a specific surface contour – that may either allow or prevent binding. Furthermore the binding by itself may induce conformational changes that have effects on other parts of the protein. These effects are called *allosteric*. In the context of this thesis, we will consider *scaffold dependency* as a special case of allosteric effects: An interaction

¹This and the following paragraphs are based on Alberts et al. (2007)

1 Introduction

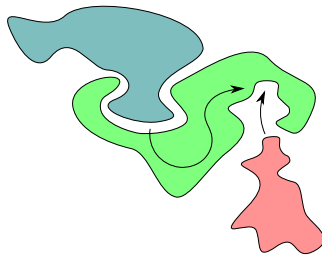


Figure 1.2: An important allosteric effect is a *scaffold dependent interaction*. The binding of one protein (blue) changes the conformation of a second protein (green) so that the binding of an additional protein (red) is possible.

between two proteins is scaffold dependent, if it depends on one of the proteins taking part in a second interaction (Figure 1.2). Apart from allosteric effects, two proteins may *compete* on the same binding domain of a third, thus mutually inhibiting their binding (Figure 1.3). Both scaffold dependency and competition on the same binding domain define logic relationships between interactions, which we call interaction logic.

By interacting in the described way, proteins form large networks with emergent system-level functions the execution of which is closely related to the propagation of conformational changes along different pathways. Due to the large complexity, abstraction is needed in order to capture the behaviour of such a network in a model. In contrast to quantitative approaches, which are based on protein concentrations and try to model individual chemical reactions based on the law of mass action (Monod, Wyman, and Changeux, 1965), graph based abstractions are popular now. This qualitative approach does not need detailed information about the chemical reactions and protein concentrations, which is mostly not available at the moment. Rather it argues about possible existences of proteins and interactions. However, interaction logic is not considered here. Hence, predictions that are made with these models suffer from inaccuracies. In a previous study (Jung et al., 2010), a first inclusion of effects induced by competition on the same binding domain was provided. In this thesis, the graph based protein network model is extended to higher dimensional *protein hypernetworks* which allow flexible formulation of arbitrary logic *constraints* on the existence of interactions and proteins. That is, they allow the formulation of interaction logic including scaffold dependency and competition on the same binding domain. On top of this model – using modal logic and corresponding tableau based

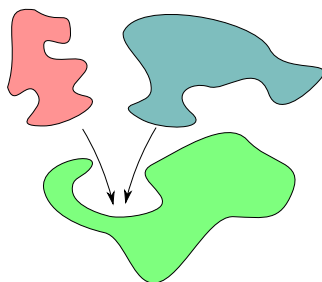


Figure 1.3: Competition of two proteins (blue, red) on the same binding domain of another protein (green).

methods in combination with graph theoretic tools – algorithms for data mining and analysis will be defined.

Biological analysis of protein networks often uses *perturbations* to determine the structure and properties of protein networks. A perturbation is an experimental intervention that changes the level or state of a protein or interaction. In this context, we prescind from this and assume a perturbation to entirely remove a protein or interaction. In general, a perturbation has side effects generated by the described interaction logic, so that it may result in additional removals. Utilizing the constraints of protein hypernetworks, it will be possible to predict such effects, hence complementing experimental perturbations.

Proteins tend to form large *complexes* by interacting with each other. Using the mechanism of conformation changes, these complexes perform impressive tasks, acting like molecular machines. A prominent example are ribosomes which execute protein synthesis (Alberts et al., 2007). The computational prediction of protein complexes can guide experiments to discover new real protein complexes that may give rise to a deeper understanding of the networks mechanisms. Based on graphs, so far clustering methods like the Markov Cluster Algorithm (Brohee and Helden, 2006), as well as specialized approaches like MCODE (Bader and Hogue, 2003) or LCMA (Li et al., 2005), were proposed for complex prediction. When used alone, these methods suffer from being unaware of effects like scaffold dependency or mutual exclusiveness of interactions. The latter were recently addressed by Jung et al. (2010), combining networks of simultaneously possible protein interactions and conventional complex prediction methods. In this thesis, a similarly combined approach ensures that conventionally predicted complexes respect all constraints of a protein hypernetwork. It will be shown that this improves the accuracy of protein complex prediction.

Since the most important functions in protein networks are executed by protein complexes, it can be assumed that a protein the perturbation of which has extensive effects on the occurring complexes is important for the network. This thesis proposes the master switch score which describes the functional importance of proteins and interactions, based on the simulation of perturbation effects on protein complexes.

Lastly, using protein complex prediction, the simulation of perturbations, and the master switch score, a method is provided to predict the functional similarity of proteins or interactions.

Structure

This thesis first introduces into mathematical foundations needed later (Chapter 2), followed by a chapter defining modal logic and the tableau algorithm (Chapter 3), the main tools we use for the protein hypernetwork model. After that, the protein hypernetwork model is defined and two methods for data mining the model are shown (Chapter 4). Based on the protein hypernetwork model, Chapter 5 describes the prediction of protein complexes, while Chapter 6 describes the prediction of master switches and functional similarities.

2 Mathematical Foundations

Here mathematical tools that will be needed in this thesis are presented. A short introduction into graph theory is followed by the definition of trees as a special case. We use the abbreviation “*iff*” for “*if and only if*” (that is, equivalence between two statements).

2.1 Graphs

Graphs are structures that are widely used throughout computer science and mathematics. A graph is a collection of nodes (also called vertices), which are connected to each other by edges. We will use two types of graphs:

2.1 Definition (Directed Graph). A pair (V, E) is a *directed graph*, if $V \neq \emptyset$ is a set of nodes and

$$E \subseteq V \times V$$

is a set of edges. An edge $(v, v) \in E$ with $v \in V$ is called *loop*.

2.2 Definition (Undirected Graph). A pair (V, E) is an *undirected graph* if $V \neq \emptyset$ is a set of nodes and

$$E \subseteq \{\{v_1, v_2\} \mid v_1, v_2 \in V\}$$

is a set of edges. An edge $\{v, v\} \in E$ with $v \in V$ is called *loop*.

Obviously we can abandon our definition of undirected graphs by interpreting E of a directed graph (V, E) as a relation between nodes and requiring symmetry, leading to

$$\forall (v_1, v_2) \in E : (v_2, v_1) \in E.$$

Thus we will use directed graphs in the following definitions.

2.3 Definition (Successor and Predecessor). Let (V, E) be a directed graph and $v \in V$ be a node. All nodes $v' \in V$ with $(v, v') \in E$ are called *successors* of v . We denote

$$\text{succ}_{(V,E)}(v) := \{v' \in V \mid (v, v') \in E\}$$

as the set of successors of v . Analogously we define a node v' to be a *predecessor* of v if $(v', v) \in E$, and denote $\text{pred}_{(V,E)}(v)$ as the set of predecessors. In an undirected graph, for each vertex v we call

$$n_{(V,E)}(v) := \text{succ}_{(V,E)}(v) \cup \text{pred}_{(V,E)}(v)$$

the *neighbourhood* and each element of $n_{(V,E)}(v)$ a *neighbour* of v .

2 Mathematical Foundations

Based on this we can define the *degree* of a node, which describes its interconnection to other nodes:

2.4 Definition (Degree). Let (V, E) be a graph and $v \in V$ be a node. The in-degree of the node is defined as

$$deg_{(V,E)}^{in}(v) := |pred_{(V,E)}(v)|$$

whereas the out-degree is analogously defined as

$$deg_{(V,E)}^{out}(v) := |succ_{(V,E)}(v)|.$$

If (V, E) is undirected, we call

$$deg_{(V,E)}(v) := |succ_{(V,E)}(v)| = |pred_{(V,E)}(v)|$$

the degree of the node.

A *path* is a sequence of nodes that connects one node with another. If there exists an edge between two nodes, it is the shortest path between them.

2.5 Definition (Path). Let (V, E) be a graph. A sequence of nodes

$$(v_1, \dots, v_n)$$

is called path if for all $i \in \{1, \dots, n-1\}$ there exists an edge $(v_i, v_{i+1}) \in E$.

Using paths, we can assess the connectivity of a graph:

2.6 Definition (Connectivity). Let (V, E) be a graph. (V, E) is called *connected* if there exists a path (v, \dots, v') for all nodes $v, v' \in V$. If (V, E) is not connected, it is called *disconnected*. (V, E) is *fully connected* or a *clique* if for all nodes $v, v' \in V$ there exist edges $(v, v'), (v', v) \in E$.

A subset of nodes can be used to build an *induced subgraph*, which includes all edges from its parent that connect any two nodes of the given subset.

2.7 Definition (Induced Subgraph). Let (V, E) be a graph. The induced subgraph for a set of nodes V' is defined as

$$S_{(V,E)}(V') = (V', E')$$

with $E' := \{(v, v') \mid (v, v') \in E, v, v' \in V'\}$.

2.2 Trees

A *tree* can be seen as a special directed graph. Each tree has a node without any predecessors, which is called *root* node. Nodes without successors are called *leaves*. Each node in a tree may have only one predecessor.

2.8 Definition (Tree). A directed graph (V, E) is a tree iff there exists a node $r \in V$ so that

$$\text{deg}_{(V,E)}^{\text{in}}(r) = 0$$

and for all nodes $v \in V \setminus \{r\}$ holds that

$$\text{deg}_{(V,E)}^{\text{in}}(v) = 1.$$

We denote

$$\text{children}_{(V,E)}(v) := \text{succ}_{(V,E)}(v)$$

and term each successor of $v \in V$ as a *child* of v . The predecessor of a node is called its *parent* and is denoted by

$$\text{parent}_{(V,E)}(v) := \text{pred}_{(V,E)}(v)$$

for $v \in V$. Additionally, the node

$$\text{root}((V, E)) := v \in V \text{ with } \text{deg}_{(V,E)}^{\text{in}}(v) = 0$$

is called the *root* of (V, E) .

2.9 Definition (Ancestor). Let (V, E) be a tree. Let $v, v' \in V$ be two nodes. The node v' is called *ancestor* of v iff v' is on the path between $\text{root}(V, E)$ and v . We denote

$$\text{ancestors}_{(V,E)}(v) := \{v' \in V \mid v' \text{ is ancestor of } v\}$$

as the set of ancestors of v .

3 Modal Logic

The basic tool we are using for this work is modal logic. It is an extension of propositional logic with additional operators. Modal logic provides high flexibility as one can extend it toward various directions, for example Temporal Logic or Probabilistic Logic (Chapter 7). For the purpose of this thesis, *basic modal logic* – from now on referred as *modal logic* for simplicity – is sufficient.

3.1 Syntax and Semantic

The modal logic defined here is also referred in the literature as “modal logic K ”. The following definitions are inspired from Doberkat (2009), as well as Kreuzer and Kühling (2006).

3.1 Definition (Propositional Logic). Let P be a set of *propositions*. We denote by $\mathfrak{Prop}(P)$ the set of all possible propositional logic formulae over the propositions P . We define $\mathfrak{Prop}(P)$ as the smallest set of formulae for which

$$\begin{aligned} P &\subseteq \mathfrak{Prop}(P), \\ \neg\phi &\in \mathfrak{Prop}(P), \\ \phi \wedge \phi' &\in \mathfrak{Prop}(P), \\ \phi \vee \phi' &\in \mathfrak{Prop}(P), \\ \phi \Rightarrow \phi' &\in \mathfrak{Prop}(P) \end{aligned}$$

for $\phi, \phi' \in \mathfrak{Prop}(P)$.

Now propositional logic can be used to define modal logic as its superset.

3.2 Definition (Modal Logic). Let P be a set of propositions. We denote by $\mathfrak{Mod}(P)$ the set of all possible modal logic formulae over the propositions P . We define $\mathfrak{Mod}(P)$ as the smallest set of formulae with

$$\begin{aligned} \mathfrak{Prop}(P) &\subseteq \mathfrak{Mod}(P), \\ \diamond\phi &\in \mathfrak{Mod}(P), \\ \square\phi &\in \mathfrak{Mod}(P) \end{aligned}$$

for $\phi \in \mathfrak{Mod}(P)$.

It can be seen that modal logic is extending propositional logic by adding two operators. The semantic of modal logic is determined by *Kripke models*. A Kripke model provides a set of *Kripke states* in combination with two relations.

3.3 Definition (Kripke Model). Given $\mathfrak{Mod}(P)$, a triple

$$(S, R, \Vdash)$$

with a set of Kripke states $S \neq \emptyset$, a *reachability relation* between states $R \subseteq S \times S$, and a *satisfiability relation* $\Vdash \subseteq S \times \mathfrak{Mod}(P)$ is called Kripke model iff for each state $s \in S$ the following holds:

$$s \Vdash \neg\phi \quad \text{iff } s \not\Vdash \phi, \quad (3.1)$$

$$s \Vdash \phi_1 \wedge \phi_2 \quad \text{iff } s \Vdash \phi_1 \text{ and } s \Vdash \phi_2, \quad (3.2)$$

$$s \Vdash \phi_1 \vee \phi_2 \quad \text{iff } s \Vdash \phi_1 \text{ or } s \Vdash \phi_2, \quad (3.3)$$

$$s \Vdash \phi_1 \Rightarrow \phi_2 \quad \text{iff } s \Vdash \neg\phi_1 \vee \phi_2, \quad (3.4)$$

$$s \Vdash \Box\phi \quad \text{iff } \forall s' \in S, (s, s') \in R : s' \Vdash \phi, \quad (3.5)$$

$$s \Vdash \Diamond\phi \quad \text{iff } s \not\Vdash \Box\neg\phi. \quad (3.6)$$

Further, the extension of a formula ϕ

$$\llbracket \phi \rrbracket_{(S, R, \Vdash)} := \{s \in S \mid s \Vdash \phi\}$$

is the set of states which satisfy the formula.

We say, a Kripke model (S, R, \Vdash) satisfies a formula iff $\llbracket \phi \rrbracket_{(S, R, \Vdash)} \neq \emptyset$.

Regarding the reachability relation $R \subseteq S \times S$, if $(s, s') \in R$, then state s' can be reached from state s . The satisfiability relation \Vdash relates each state $s \in S$ with the formulae it satisfies.

Rule (3.1) introduces a “closed world” assumption: If a state does not to satisfy a formula, then its negation is assumed to be satisfied.

For a formula $\phi = \phi_1 \circ \phi_2$ or $\phi = \circ \phi_1$ with \circ being one of the above defined operators we call ϕ_1 and ϕ_2 *subformulae* of ϕ . The operator \neg is called *negation*, a formula $\phi_1 \wedge \phi_2$ is called *conjunction* and $\phi_1 \vee \phi_2$ is called *disjunction*. The subformulae of the latter are called *disjuncts*, whereas the subformulae of $\phi_1 \wedge \phi_2$ are called *conjuncts*. The *implication* $\phi_1 \Rightarrow \phi_2$ is equivalent to $\neg\phi_1 \vee \phi_2$. The operator \Box is called *modal operator*. $s \Vdash \Box\phi$ means that the formula ϕ holds on every state s' that is reachable from state s ($(s, s') \in R$). \Diamond is called *dual modal operator*. A closer look at the dual modal operator reveals the following property:

3.4 Lemma (Dual modal operator). *Let (S, R, \Vdash) be a Kripke model. The statement*

$$s \Vdash \Diamond\phi \text{ iff } \exists s' \in S, (s, s') \in R : s' \Vdash \phi$$

holds for all $s \in S$.

Proof. Assuming $s \Vdash \Diamond\phi$ we gain that $s \not\Vdash \Box\neg\phi$. This is equivalent to

$$\exists s' \in S, (s, s') \in R : s' \not\Vdash \neg\phi.$$

Rule (3.1) leads to

$$\exists s' \in S, (s, s') \in R : s' \Vdash \phi.$$

For the opposite direction we assume that $\exists s' \in S, (s, s') \in R : s' \Vdash \phi$ holds. Then we know that the statement

$$\forall s' \in S, (s, s') \in R : s' \Vdash \neg\phi$$

does not hold. Hence the state s does not satisfy the formula $\Box\neg\phi$ ($s \not\Vdash \Box\neg\phi$), and from rule (3.6) follows $s \Vdash \Diamond\phi$. \blacksquare

Lemma 3.4 and rule (3.5) give the motivation for the naming and therefore the interpretation of the two modal operators we will use from now on. The modal operator \Box is called *necessity operator*, whereas the dual modal operator \Diamond is called *possibility operator*.

For our further usage of the modal logic it will be useful to work with a normalized type of formulae. This reduces the complexity of the used methods and data structures.

3.5 Definition (Negation normal form). The map $nn : \mathfrak{Mod}(P) \rightarrow \mathfrak{Mod}(P)$

$$nn(\phi) = \begin{cases} nn(\phi') & \text{if } \phi = \neg\neg\phi' \\ nn(\neg\phi_1) \vee nn(\neg\phi_2) & \text{if } \phi = \neg(\phi_1 \wedge \phi_2) \\ nn(\neg\phi_1) \wedge nn(\neg\phi_2) & \text{if } \phi = \neg(\phi_1 \vee \phi_2) \\ nn(\phi_1) \wedge nn(\phi_2) & \text{if } \phi = \phi_1 \wedge \phi_2 \\ nn(\phi_1) \vee nn(\phi_2) & \text{if } \phi = \phi_1 \vee \phi_2 \\ nn(\neg\phi_1) \vee nn(\phi_2) & \text{if } \phi = \phi_1 \Rightarrow \phi_2 \\ \Diamond nn(\neg\phi') & \text{if } \phi = \neg\Box\phi' \\ \Box nn(\neg\phi') & \text{if } \phi = \neg\Diamond\phi' \\ \Box nn(\phi') & \text{if } \phi = \Box\phi' \\ \Diamond nn(\phi') & \text{if } \phi = \Diamond\phi' \\ \phi & \text{else .} \end{cases}$$

translates an arbitrary modal logic formula into its negation normal form.

Negation normal form removes the implication (\Rightarrow) operator and moves all negations directly in front of the propositions.¹

3.6 Example (Negation normal form). A formula $\neg\Box(A \Rightarrow B)$ is first mapped to $\Diamond\neg(A \Rightarrow B)$, then to $\Diamond\neg(\neg A \vee B)$. Negation is moved into the disjunction leading to $\Diamond(\neg\neg A \wedge \neg B)$. Finally the mapping produces the simplified formula $\Diamond(A \wedge \neg B)$.

Now we show that using negation normal form does not decrease the expressive power of $\mathfrak{Mod}(P)$:

3.7 Lemma (Negation normal form). *Let $\phi \in \mathfrak{Mod}(P)$ be a modal logic formula and (S, R, \Vdash) be an arbitrary Kripke model. For each state $s \in S$ it holds that $s \Vdash \phi$ iff $s \Vdash nn(\phi)$.*

¹When using the tableau algorithm (Chapter 3.2), this prevents the need of two additional expansion rules.

3 Modal Logic

Proof. Let $p \in P$ be an arbitrary proposition. Assuming $\phi = p$, we know $nn(\phi) = p$, and for $\phi = \neg p$ we know $nn(\phi) = \phi$, so the lemma is valid for $\phi = p$ and $\phi = \neg p$. Let (S, R, \Vdash) be an arbitrary Kripke model and $s \in S$ be an arbitrary state. Let $\phi_1, \phi_2, \neg\phi_1, \neg\phi_2 \in \mathfrak{Mod}(P)$ be formulae for which the lemma is valid.

Case 1. Assuming $\phi = \phi_1 \wedge \phi_2$ we derive

$$\begin{aligned} s \Vdash \phi &\text{ iff } s \Vdash \phi_1 \wedge \phi_2 \\ &\text{ iff } s \Vdash nn(\phi_1) \wedge nn(\phi_2) \\ &\text{ iff } s \Vdash nn(\phi). \end{aligned}$$

An analogous argument can be applied for $\phi = \phi_1 \vee \phi_2$, $\phi = \Box\phi_1$ and $\phi = \Diamond\phi_1$.

Case 2. Assuming $\phi = \neg(\phi_1 \wedge \phi_2)$ we see that

$$\begin{aligned} s \Vdash \phi &\text{ iff } s \Vdash \neg(\phi_1 \wedge \phi_2) \\ &\text{ iff } s \not\Vdash \phi_1 \wedge \phi_2 \\ &\text{ iff } s \not\Vdash \phi_1 \text{ or } s \not\Vdash \phi_2 \\ &\text{ iff } s \Vdash \neg\phi_1 \text{ or } s \Vdash \neg\phi_2 \\ &\text{ iff } s \Vdash \neg\phi_1 \vee \neg\phi_2 \\ &\text{ iff } s \Vdash nn(\neg\phi_1) \vee nn(\neg\phi_2) \\ &\text{ iff } s \Vdash nn(\phi). \end{aligned}$$

These arguments can be applied in an analogous manner to $\phi = \neg(\phi_1 \vee \phi_2)$.

Case 3. Assuming $\phi = \neg\Box\phi_1$ the connection between the two modal operators leads to

$$\begin{aligned} s \Vdash \phi &\text{ iff } s \Vdash \neg\Box\phi_1 \\ &\text{ iff } s \not\Vdash \Box\phi_1 \\ &\text{ iff } s \Vdash \Diamond\neg\phi_1 \\ &\text{ iff } s \Vdash \Diamond nn(\neg\phi_1) \\ &\text{ iff } s \Vdash nn(\phi), \end{aligned}$$

and an analogous argument can be applied for $\phi = \neg\Diamond\phi_1$. ■

In order to work with modal logic formulae inside a software tool, they have to be represented in an appropriate data structure. The general solution is using a *formula tree* (Figure 3.1). The inner nodes of such a tree represent operators and may thus be of the type \neg, \wedge, \vee, \Box or \Diamond . Propositions occur as leaves. Inner nodes of the type \wedge and \vee may have more than one child, whereas all other types are restricted to one child. In the underlying formula the children of an inner node are connected by the corresponding operator.

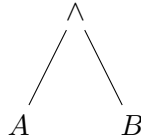


Figure 3.1: Example formula tree for $A \wedge B$.

3.2 The Modal Logic Tableau

A method to find a satisfying Kripke model for a modal logic formula is the *modal logic tableau*. The original purpose of tableau based methods is to prove the satisfiability of a given formula. Tableau based methods aim to generate a deductive tree. If successful, the method finds a possible model for a given formula, represented by a path – in this chapter, each mentioned path is assumed to reach from the root of the deductive tree to a leaf – without contradictions. If not, the formula is proven to be not satisfiable. We restrict modal logic formulae to be in negation normal form (Definition 3.5) since that is equivalent to general modal logic formula and reduces the complexity of the implementation.

3.2.1 Idea

For a given formula ϕ the tableau algorithm tries to find possible Kripke models (S, R, \Vdash) that satisfy ϕ . The tableau is created by recursively breaking ϕ into subformulae. It can be seen as a tree in which each node represents an *assumption* that can be made due to assumptions which were made in an ancestral node. The process of generating such an assumption is called *expansion*.

3.8 Definition (Assumption). Let ϕ be a modal logic formula. An assumption can be of the type $s \Vdash \psi$ for a subformula ψ of ϕ or a relation $(s, s') \in R$ meaning that two states $s, s' \in S$ are assumed to be in relation R .

A path in the tree is thus a sequence of assumptions which may be valid at the same time. Two assumptions can obviously contradict each other. This is called a *clash*. We say that a path is *open* if it does not contain any clash. A path that contains a clash is called *closed*. Hence an open path on which all nodes have been expanded, thus no further assumptions are possible, can be interpreted as a satisfying Kripke model for the formula ϕ . Closed paths then are obviously invalid models, even if they are not yet fully expanded. Thus a path can be ignored once it is closed.

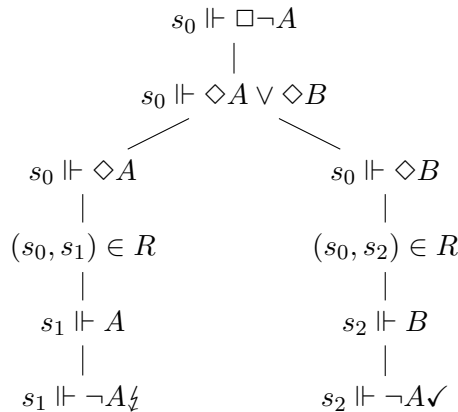


Figure 3.2: Example tableau for $(\Box \neg A) \wedge (\Diamond A \vee \Diamond B)$. The left path marked by $\not\checkmark$ is closed, because it contains a clash between the assumptions $s \Vdash \neg A$ and $s \Vdash A$. The right path marked by \checkmark does not contain any clash, hence it is open.

| Assumption | $s \Vdash \phi_1 \wedge \phi_2$ | $s \Vdash \Box \phi$ | $s \Vdash \Diamond \phi$ |
|-------------------|---|--|---|
| Expansion | $\begin{array}{c} \\ s \Vdash \phi_1 \\ \\ s \Vdash \phi_2 \end{array}$ | $\forall (s, s') \in R :$ $\begin{array}{c} \\ s \Vdash \phi \end{array}$ | $\begin{array}{c} \\ (s, s') \in R \\ \\ s \Vdash \phi \\ \forall s \Vdash \Box \psi \in \text{ancestors}(s \Vdash \Diamond \phi) : \\ \\ s' \Vdash \psi \end{array}$ |

Table 3.1: Deterministic expansion rules. In order to satisfy Rule (3.5) of Definition 3.3, the third rule for expanding the assumption $s \Vdash \Diamond \phi$ creates a new Kripke state s' and thus needs to add $s' \Vdash \psi$ to the open paths for all assumptions $s \Vdash \Box \psi$ that were made before on this path.

New assumptions are generated by applying expansion rules on nodes. When applying an expansion rule on a node v , the results of the expansion are added to every open path containing node v . There exist two types of expansion rules: *deterministic* (Table 3.1) and *non-deterministic* (Table 3.2). A deterministic expansion leads to only one possible result. For example, when expanding $\Diamond A$ in Figure 3.2 we can assume that $(s_0, s_1) \in R$ and $s_1 \Vdash A$. A non-deterministic expansion leads to more than one possible result. In Figure 3.2, expanding $s_0 \Vdash \Diamond A \vee \Diamond B$ means, that $s_0 \Vdash \Diamond A$ or $s_0 \Vdash \Diamond B$ can be true. When building the tableau this means that a deterministic expansion expands all paths it lies on by adding nodes. A non-deterministic expansion splits each path into several, each of them ending with one of the results as leaf. A tableau which does not contain any open paths nor any unexpanded nodes proves the unsatisfiability of the formula it was generated from. Accordingly a tableau with at least one open path on which all nodes are expanded, is a proof of the satisfiability of the formula.

3.2.2 Optimization

The possible size of a tableau is exponentially big, and the computation time of the method can be shown to be in the complexity class NEXPTIME (Li, 2008).

| Assumption | $s \Vdash \phi_1 \vee \phi_2$ |
|-------------------|--|
| Expansion | $\begin{array}{ccc} & & \\ & \diagdown & \diagup \\ s \Vdash \phi_1 & & s \Vdash \phi_2 \end{array}$ |

Table 3.2: Non-deterministic expansion rule. Expansion does only make minimal assumptions necessary for satisfiability here. Thus it does for example neither assume $s \Vdash \neg \phi_2$ nor $s \Vdash \phi_2$ in the left case. For a disjunction with n disjuncts, this ensures a search space with a size linear in the number of disjuncts (n expansions) in contrast to $2^n - 1$ different expansions.

This means, that the solution space, as well as the number of operations needed to investigate a possible solution, may be exponentially big in the size of the input. However the method can be optimized in order to show acceptable performance in practice.

Backtracking

The relation between the satisfiability and the existence of an open, fully expanded path, which was shown above, gives rise to an optimization approach. Since one path is sufficient to show satisfiability, the tableau can stop after it has found the first fully expanded path. Further, it can even work in a *depth-first* manner: When a non-deterministic expansion rule is applied to a disjunction, only one of the disjuncts is investigated. This means that the tableau method does not split paths any more and only investigates one single path.

If this leads to a clash, the tableau needs to find another path. Therefore it has to *backtrack* its decisions: A disjunction that has a not yet investigated disjunct has to be found. From there, the tableau has to take a different path. Hence the misleading

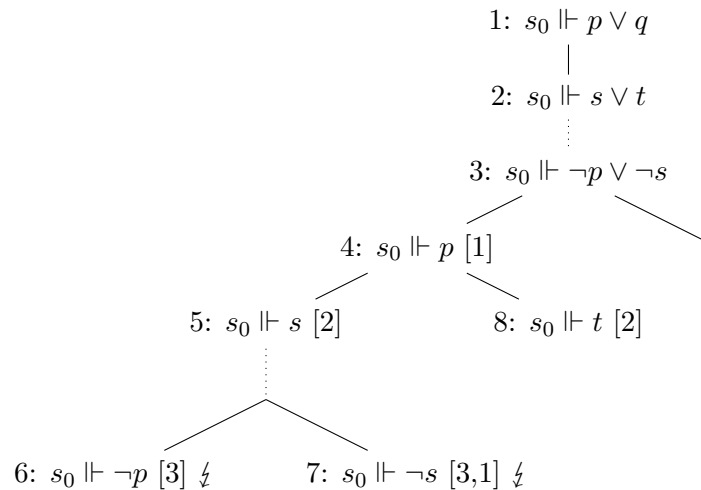


Figure 3.3: Backjumping example (Li, 2008). Dependency sets are denoted in brackets, assumptions are numbered. The path between the assumptions 1 and 3 is resulting from the expansion of a conjunction containing the formulae of the path as conjuncts. After that, expansion of 1 causes 4 to be appended to the current leaf of the path (which is 3 at that time). Expansion of 2 leads to appending assumption 5 to 4. Expansion of assumption 3 leads to assumption 6. This causes a clash with assumption 4. Backjumping to the first assumption in the dependency set of 6 – which is 3 – causes the second disjunct of 3 to be investigated. Therefore, assumption 7 is appended to the path. The dependency set of 7 not only contains its parent disjunct but the cause of the clash of its sibling disjunct (4). As 7 causes a clash again, the dependency sets of 7 and the clashing assumption 5 have to be investigated: [3,1] and [2]. Since assumption 3 is fully explored and 4 is not a disjunction, assumption 2 is investigated so that it leads to the addition of the node 8.

Let the dotted line between 2 and 3 represent additional disjunctions and the dotted line below 5 represent their expansions. With chronological backtracking after the clash at 7, each disjunction between 2 and 3 would be investigated until the true source of the problem (2) is found.

disjunct chosen so far is called *blocked* and another not yet blocked disjunct is chosen. This backtracking procedure has to be repeated until a fully expanded path without clashes is found.

In literature, several different backtracking mechanisms have been elaborated (Li, 2008). The naive approach is called *chronological backtracking*. The disjunctions are investigated chronologically: Once a clash is found, a not blocked disjunct of the the last possible disjunct will be investigated. This happens even if the expansion does not have any influence on the clash.

Chronological backtracking obviously causes unnecessary overhead as it could investigate multiple paths that have the same clash. *Backjumping* tries to reduce this unneeded effort by only investigating disjunctions that are a direct or indirect cause of a clash. Therefore, a *dependency set* is recorded for each assumption which includes its ancestral disjunctions in the formula tree. Additionally, if a disjunct is selected because of a previous clash, the dependency set contains the assumption which was responsible for the clash. Figure 3.3 shows an example.

Dynamic backtracking (Li, 2008) is an alternative approach. It uses *elimination explanations* to record the reasons for a disjunct to be misleading. If a disjunct ψ is decided to be misleading, all disjunctions which are ancestors of the clashed subformulae inside the formula tree (except it's own disjunction) are recorded as its elimination explanation $E(\psi)$. When choosing another disjunct, all disjuncts

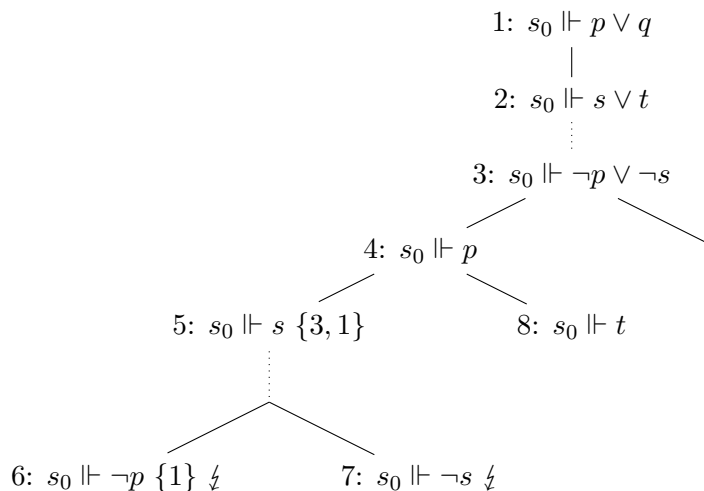


Figure 3.4: Dynamic backtracking example (Elimination explanations are denoted in braces, assumptions are numbered). Expansion of assumption 3 leads to assumption 6. This causes a clash with assumption 4. Dynamic backtracking reveals that disjunction 3 has another disjunct which is not blocked. Thus, all assumptions that were made because of 6 are undone and 3 is expanded to 7. This produces a clash again between 7 and 5. This time, there is no not blocked disjunct left in 3 so that the next parent disjunction of the clashed formulae is investigated: Disjunction 2 has a disjunct that is not blocked. The current active disjunct of 2 and everything resulting from it is removed from the active path and 2 is expanded to 8. Assumption 7 now no longer produces a clash.

Consider the dotted lines between 2 and 3 as additional assumptions and the dotted lines under 5 as their expansions. As we did not remove the full path under 5, the work on these assumptions has not to be done twice.

that were blocked before become unblocked if their elimination explanation contains the disjunction of ψ . In other words, if the reason for the decision that a disjunct ρ is misleading turns out to be misleading itself, then ρ is no longer assumed to be misleading. This mechanism gives rise to the key difference between dynamic backtracking and backjumping: Since it provides the possibility to undo previous blocks there is no need to undo all subsequent assumptions if it selects another disjunct. Only assumptions that are derived from the misleading disjunct have to be removed from the path. Thus, in contrast to backjumping, dynamic backtracking does not need to repeat work that was not influenced by a clash.

Li (2008) shows that dynamic backtracking is, compared to the other approaches, a fast method that ensures high efficiency even for pathologically structured formulae. A detailed comparison between the different approaches can be found there.

3.2.3 Implementation

The implementation uses a formula tree as described above. A formula is represented by a pointer to a node of the tree. The algorithms make use of the following data structures and functions (Li, 2008):

| | |
|-------------------------------|--|
| (s, ϕ) | A labelled formula. For $s \in S$, $(s, \phi) \in P$ means that $s \Vdash \phi$ is assumed on the current path P . |
| P | A sequence of labelled formulae. Represents the currently investigated path. |
| <i>unknown/active/blocked</i> | Each labelled formula has a state (initially <i>unknown</i>). A formula is set to <i>active</i> if it is in the current path. It is set to <i>blocked</i> if it caused a contradiction. |
| <i>expanded</i> | Each labelled formula is flagged <i>expanded</i> when an expansion rule has been applied to that formula. |
| $Unexpanded(P)$ | A function that returns a labelled formula $(s, \phi) \in P$ which is not flagged as expanded. The selection can be based on various heuristics. Returns <i>null</i> if all formulae are expanded. |
| D | A sequence of labelled disjunctive formulae. It contains all disjunctions on the current path which have been expanded. |
| S | A set of states. Represents the set of states of the Kripke model. |
| R | A set of pairs (s, s') with $s, s' \in S$. This represents the reachability relation R of the Kripke model. |
| $Succ_R(s)$ | A function that returns a set of states, |

$$Succ_R(s) = \{s' \mid (s, s') \in R\}.$$

3 Modal Logic

Box_s A set of formulae. For $s \in S$

$$Box_s = \{\phi \mid (s, \Box\phi) \in P\}$$

gives all formulae ϕ which necessarily have to be true in states reachable from s .

Dia_s A labelled formula $\Diamond\phi$. The expansion of $\Diamond\phi$ led directly to the generation of state $s \in S$.

$Disjs(\phi)$ A function that returns all disjunctions that are ancestors of ϕ in the formula tree.

$Clashes(P)$ A function that returns a set of labelled formulae:

$$Clashes(P) = \{(s, \phi) \mid (s, \phi), (s, \neg\phi) \in P\}.$$

$E(\phi)$ A set of disjunctive formulae. The *active* formula of each disjunction in here is a cause for ϕ to be blocked, respectively clashing.

The implemented algorithm (Algorithm 3.1) subsequently expands formulae on the current path as long as no clash is detected. The currently investigated path is recorded in the sequence P . If no clashes are found on the current path, an unexpanded labelled formula is selected and the corresponding expansion rule (Algorithm 3.2) is applied.

If the current path has no unexpanded formula, it represents a satisfying Kripke model. The formula is therefore satisfiable. If a clash is detected, the method tries to “backtrack” the decisions (Algorithm 3.3). This is implemented using *dynamic backtracking* (Section 3.2.2). All parent disjunctions of the clashed formulae are hold

Input: formula ϕ

Output: satisfiability of ϕ

```

1: create state  $s$  and set  $S := \{s\}$ 
2:  $P := \{(s, \phi)\}$ 
3:  $D := \emptyset$ 
4: while  $Clashes(P) \neq \emptyset$  or  $Unexpanded(P) \neq null$  do
5:   if  $Clashes(P) = \emptyset$  then
6:      $(s, \phi) := Unexpanded(P)$ 
7:      $expand(s, \phi)$ 
8:     set  $(s, \phi)$  to be expanded
9:   else
10:    if not  $backtrack(Clashes(P))$  then
11:      return “unsatisfiable”
12:    end if
13:  end if
14: end while
15: return “satisfiable”

```

Algorithm 3.1: The modal logic tableau (Li, 2008).

Input: labelled conjunction (s, ϕ)

- 1: **for all** conjuncts ψ of ϕ **do**
- 2: $P := P \cup \{(s, \psi)\}$
- 3: set (s, ψ) to be *active*
- 4: **end for**

Input: labelled disjunction (s, ϕ)

- 1: $\psi :=$ an *unknown* disjunct of ϕ
- 2: $P := P \cup \{(s, \psi)\}$
- 3: $D := D \cup \{(s, \psi)\}$
- 4: set (s, ψ) to be *active*

Input: labelled possibility $(s, \diamond\psi)$

- 1: create state s' and set $S := S \cup \{s'\}$ and $R := R \cup (s, s')$
- 2: $Dia_s := (s, \diamond\psi)$
- 3: $P := P \cup \{(s', \psi)\}$
- 4: set (s', ψ) to be *active*
- 5: **for all** $\rho \in Box_s$ **do**
- 6: $P := P \cup \{(s', \rho)\}$
- 7: **end for**

Input: labelled necessity $(s, \Box\psi)$

- 1: **for all** $s' \in Succ_R(s)$ **do**
- 2: $P := P \cup \{(s', \psi)\}$
- 3: set (s', ψ) to be *active*
- 4: **end for**
- 5: $Box_s := Box_s \cup \psi$

Algorithm 3.2: The modal logic tableau (formula expansion) (Li, 2008).

in the set T . We iterate over T in reverse order of their appearance in the current path.

If a disjunction $\psi \in T$ has an unknown disjunct (one that is not expanded on the current path nor blocked because it would cause another clash) we perform backtracking on this disjunction. If not, we extend the search space for backtracking points by including all elimination explanations of the disjuncts of ψ in the set T and including all parent disjunctions of ψ . Further, we add the parent disjunctions of the formula that caused the current state s to appear.

Performing backtracking on a disjunction ψ works as follows: The elimination explanation of the active disjunct is set to contain T without the disjunction ψ since it is the direct parent of the disjunct and cannot be a contradiction. The disjunct is flagged as blocked, and all disjuncts that were blocked before because of the disjunction ψ are flagged unknown again since the next selected disjunct of ψ probably is not a contradiction to these any more. We remove everything that results from the newly blocked disjunct and expand ψ again (this time skipping the blocked disjunct). If backtracking is not successful, a clash cannot be avoided. Hence all paths end in a clash, there may not be a satisfying Kripke model and the formula is unsatisfiable.

3 Modal Logic

Input: set of clashing formulae, called *clashes*

Output: success of backtracking

```

1:  $T := \{parentDisj(\phi) \mid (s, \phi) \in clashes\}$ 
2: for  $(s, \psi) \in D$  in reverse order do
3:   if  $\psi \in T$  then
4:     if  $\psi$  has unknown disjuncts then
5:        $(s, \phi) := active$  disjunct of  $\psi$ 
6:        $E(\phi) := T \setminus \{\psi\}$ 
7:       set  $(s, \phi)$  to be blocked
8:       for all  $(s, \gamma)$  with  $\psi \in E(\gamma)$  do
9:         set  $(s, \gamma)$  to be unknown
10:         $E(\gamma) := \emptyset$ 
11:       end for
12:       remove  $(s, \phi)$  and every labelled formula, state and relation resulting from
          $(s, \phi)$  from  $P, D, Dia, Box, S, R$ .
13:        $expand(s, \psi)$ 
14:       return true
15:     end if
16:     for all  $\rho$  such that  $\rho$  is a disjunct of  $\psi$  do
17:        $T := T \cup E(\rho)$ 
18:     end for
19:      $T := T \cup Disjs(\psi) \cup Disjs(Dia_s)$ 
20:      $T := T \setminus \{\psi\}$ 
21:   end if
22: end for
23: return false

```

Algorithm 3.3: The modal logic tableau (dynamic backtracking) (Li, 2008).

Finding Kripke models

In this thesis, the tableau algorithm will be used to find Kripke models for a formula. It will be seen that for this it is important to be sure that for each disjunction the leftmost disjunct that does not lead to a clash, is expanded:

3.9 Definition (Strictly Left Expanding Tableau). A tableau algorithm is called *strictly left expanding* if it ensures, that for all disjunctions the leftmost not blocked disjunct is expanded.

To some extent the implementation of the tableau algorithm mentioned above already shows the desired behaviour. For a formula

$$\alpha \vee \beta \tag{3.7}$$

in negation normal form, the tableau first tries to expand α , and only if that fails due to a clash, dynamic backtracking leads to the expansion of β and blocks the disjunct α . This happens because the tableau algorithm always tries to expand the leftmost disjunct first. However in the lines 8 to 11 of the dynamic backtracking

```

1: for all  $(s, \gamma)$  with  $\psi \in E(\gamma)$  do
2:   set  $(s, \gamma)$  to be unknown
3:    $E(\gamma) := \emptyset$ 
4:    $(s, \chi) :=$  parent disjunct of  $\gamma$ 
5:    $(s, \xi) :=$  active disjunct of  $(s, \chi)$ 
6:   remove  $(s, \xi)$  and every labelled formula, state and relation resulting from
        $(s, \xi)$  from  $P, D, Dia, Box, S, R$ .
7:    $expand(s, \chi)$ 
8: end for

```

Algorithm 3.4: The modal logic tableau (modification of dynamic backtracking) These lines replace the lines 8 to 11 of the dynamic backtracking algorithm (Algorithm 3.3). The changed behaviour ensures that for each disjunction the leftmost possible disjunct is expanded.

algorithm (Algorithm 3.3), a blocked disjunct γ is flagged as unknown if its elimination explanation contains the currently backtracked disjunction ψ . For above formula (3.7) consider the situation that the disjunct α is blocked, and β is expanded. If another backtracking process leads to α being flagged as unknown, the path still contains an expanded β , which is not the leftmost possible disjunct of formula (3.7).

In fact, the above behaviour illustrates the strength of dynamic backtracking to do only necessary work. We will however provide a way to ensure a strictly left expanding tableau without the need to abandon dynamic backtracking.

Algorithm 3.4 is a replacement for the lines 8 to 11 of the dynamic backtracking algorithm (Algorithm 3.3). It provides a way to ensure the tableau algorithm to be strictly left expanding. Each disjunct γ that was blocked due to the currently backtracked disjunction ψ is flagged as unknown and its elimination explanation is cleared. Then the currently active and expanded disjunct ξ of the disjunction χ of γ is removed from all data structures together with all resulting formulae. Afterwards the disjunction χ is expanded again. As normal expansion of disjunctions was already shown to behave in the way that the leftmost not blocked disjunct is selected, this ensures the desired behaviour.

4 Protein Hypernetworks

This chapter will describe an approach for incorporating interaction logic into the description of protein networks. In the literature it is common practice to model protein networks as graphs (Chapter 2.1). They consist of proteins, and binary interactions between them.

4.1 Definition (Protein Network). An undirected graph (P, I) with a vertex $p \in P$ for each protein and an undirected edge $\{p_1, p_2\} \in I$ for each possible interaction is a protein network.

Obviously this structure is an abstraction from substance concentration and even individual protein instances. If it contains a node representing a particular protein, one can only say that the protein exists at some time point somewhere in the cell, no assumption can be made on the amount or even different particular instances of the protein. An interaction between two proteins in the graph only gives the information that there might be two instances of the proteins in reality, which are interacting in the described way at some point in time at a specific location in the cell.

In the literature, structures of these graphs have been mapped to biological meaningful properties: Dense regions are considered to be protein complexes (Chapter 5). Overrepresented patterns, in other words small subgraphs, are investigated as network motifs in order to find functional properties of the network (for example by Milo et al. (2002)).

Here, a fundamental problem occurs because the information gained by these traditional analyses of protein networks cannot take the dynamics of interactions – the interaction logic (Chapter 1) – into account. This leads to incomplete and probably wrong predictions. Two proteins can compete on the same binding domain (Figure 1.3) of a third protein, so that the two interactions cannot appear at the same time because they are mutually inhibiting each other. Further, as a special case of allosteric effects, an interaction may depend on a certain scaffold of interacting proteins to be available (scaffold dependency, Figure 1.2). Protein complexes, for example, are assumed to consist of proteins that are interacting at the same time and place (Spirin and Mirny, 2003), thus they may not contain mutual exclusive interactions. Jung et al. (2010) show that accounting knowledge about mutual exclusive interactions improves the computational prediction of protein complexes. Our *protein hypernetwork* approach is an extension of the above protein network model by arbitrary propositional logic *constraints* on the existence of proteins and interactions. In this chapter, it will be shown that interaction logic can be captured by this. The approach gives rise to improvements in traditional analysis of protein networks as well as allowing new methods of analysis. Possible usages are shown in Chapter 6.

4 Protein Hypernetworks

Sometimes there are several interactions between two proteins, which can be distinguished by different used binding sites. Therefore, a finer grained model seems appropriate. This can be achieved by introducing a variant of the protein network, which considers interactions between binding domains of proteins.

4.2 Definition (Protein Domain Network). A pair (P, I) of proteins $p \in P$ and domain interactions $\{(p_1, d_1), (p_2, d_2)\} \in I$ is a *protein domain network*. A protein domain can be an arbitrary identifier or a wildcard “?”, indicating that no domain can be specified. For each protein domain network (P, I) there exists a protein network (P, I') with $I' := \{\{p_1, p_2\} \mid \{(p_1, d_1), (p_2, d_2)\} \in I\}$.

A protein domain network is not a graph. Hence, the mentioned translation to a protein network, which is an undirected graph, will be necessary whenever a graph based analysis is intended.

4.1 Incorporating Interaction Logic

Since we want to combine a protein network with logical information, for both of its entities – proteins and interactions – a logical representation has to be found. A constraint will be modelled as a logic formula of interactions or proteins. Thus a constraint can be seen as a third type of entity, which expands the two dimensions of the protein network, turning it into a protein hypernetwork. The logic of choice will be modal logic (Chapter 3). Although we will not always use the full expressional power – the modal operators – in our constructs, modal operators cannot be avoided completely.

On the one hand, as the system should be able to distinguish between interactions on the level of protein domains, we use protein domain interactions $i \in I$ as atomic units. On the other hand, proteins $p \in P$ will be used as we need to have a single representation for each. Thus, the atomic units are represented as logic propositions $P \cup I$. Considering a Kripke model (S, R, \Vdash) for an arbitrary modal logic formula $\phi \in \mathfrak{M}\text{od}$, we can now provide an interpretation:

4.3 Definition (Proteins and Interactions). Let (S, R, \Vdash) be a Kripke model for a modal logic formula $\phi \in \mathfrak{M}\text{od}(P \cup I)$. A protein $p \in P$ is said to be *possible* in a Kripke state $s \in S$ iff

$$s \Vdash p.$$

An interaction $i \in I$ is said to be possible in a Kripke state $s \in S$ iff

$$s \Vdash i.$$

All proteins and interactions satisfied by the Kripke state are assumed to be possible *simultaneously*.

A Kripke state represents a situation that can occur in the cell at some location and some point in time. All proteins and interactions satisfied by the Kripke state thus may possibly appear at that location and time. Hence they are said to be possible simultaneously.

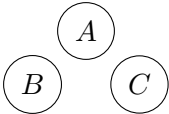
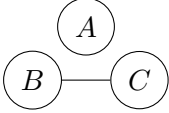
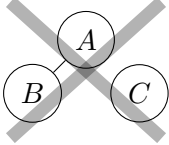
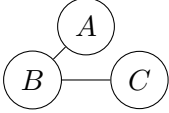
| $\{A, B\}$ | $\{C, B\}$ | $\{A, B\} \Rightarrow \{C, B\}$ | Protein network |
|------------|------------|---------------------------------|--|
| 0 | 0 | 1 |  |
| 0 | 1 | 1 |  |
| 1 | 0 | 0 |  |
| 1 | 1 | 1 |  |

Table 4.1: *Truth table* for a constraint modelling scaffold dependency. Interaction between protein A and B shall be only possible if protein C interacts with B . Each row of the truth table for the corresponding constraint $\{A, B\} \Rightarrow \{C, B\}$ represents either a satisfying – if the entry for the constraint is 1 – or a not satisfying Kripke state. Appended to the truth table is the representation as a protein network. The protein network that violates the constraint is crossed out.

This makes the goal of our modelling obvious: All formulae describing (a part of) the protein hypernetwork have to be designed in a way that ensures a satisfying solution (the states of a Kripke model) to represent a combination of interactions and proteins, which is assumed to be possibly appearing simultaneously in reality. Having that in mind, we provide the definition of a constraint and reveal its properties.

4.4 Definition (Constraint). A constraint is a logic formula of the form

$$q \Rightarrow \psi$$

with $q \in P \cup I$ and $\psi \in \mathfrak{Prop}(P \cup I)$. With $\mathfrak{C}(P \cup I) \subseteq \mathfrak{Prop}(P \cup I)$ we denote the set of all possible constraints. Further we call a constraint $q \Rightarrow \psi$ *active* in a Kripke state s iff $s \Vdash (q \Rightarrow \psi)$, $s \Vdash q$ and $s \Vdash \psi$.

We want constraints to capture the interaction logic and thus restrict the simultaneous possibility of proteins and interactions. However, since arbitrary implications are allowed and not only interactions but also proteins may be constrained, we are in theory not limited to the description of interaction logic. Accordingly, proteins and interactions are treated the same in the following. Formulae describing the hypernetwork will use a conjunction of the constraints to ensure that assumptions about simultaneous possibility respect the interaction logic.

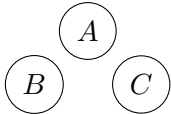
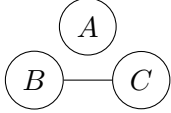
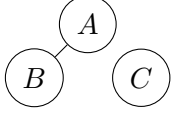
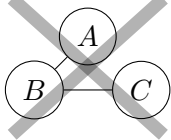
| $\{A, B\}$ | $\{C, B\}$ | $(\{A, B\} \Rightarrow \neg\{C, B\}) \wedge (\{C, B\} \Rightarrow \neg\{A, B\})$ | Protein network |
|------------|------------|--|---|
| 0 | 0 | 1 |  |
| 0 | 1 | 1 |  |
| 1 | 0 | 1 |  |
| 1 | 1 | 0 |  |

Table 4.2: Truth table for constraints modelling competition on domain. The two interactions are mutually exclusive. Each row of the truth table for the conjunction of constraints $(\{A, B\} \Rightarrow \neg\{C, B\}) \wedge (\{C, B\} \Rightarrow \neg\{A, B\})$ represents either a satisfying – if the entry for the conjunction is 1 – or a not satisfying Kripke state. Appended to the truth table is the representation as a protein network. The protein network that violates the constraint is crossed out.

Taking a deeper look at a constraint $q \Rightarrow \psi$ uncovers that it restricts the satisfiability of q by the satisfiability of ψ . In other words: if q is satisfied in a Kripke state, then the same has to apply for ψ . Equivalently the contraposition has to hold for a satisfying state: If ψ is not satisfied, then q may not either. From Definition 3.3 we gain the insight that q being not satisfied in a Kripke state puts no limitations on the satisfiability of ψ . The following two examples show that this behaviour is desired. For better readability we denote an interaction $\{(p, ?), (p', ?)\}$ as $\{p, p'\}$.

4.5 Example (Constraint for scaffold dependency). Consider a protein network (P, I) with $P = \{A, B, C\}$ and $I = \{\{A, B\}, \{C, B\}\}$. Assume that the interaction between protein A and B depends on the existence of the interaction between C and B (e.g. the binding of C is introducing a conformational change to protein B (Figure 1.2) that enables a binding domain for protein A). This behaviour can be reflected by the constraint

$$\{A, B\} \Rightarrow \{C, B\}.$$

Satisfying Kripke states for this formula can be directly translated to situations, which are assumed to appear in reality: On the one hand, if protein A binds to B , then there has to be an interaction between C and B at the same time. On the other hand, if C does not bind to B , then A cannot interact with B . Table 4.1 shows all Kripke states for this constraint in a *truth table*. It can be seen that every Kripke state except the third one satisfies the constraint. The representation of

the third state as a protein network accordingly shows a situation that violates the scaffold dependency: the interaction $\{A, B\}$ is assumed to be possible alone. Each Kripke state that satisfies a formula which contains the above constraint describes a situation that does not violate this scaffold dependency.

4.6 Example (Constraints for competition on domain). Consider a protein network (P, I) with $P = \{A, B, C\}$ and $I = \{\{A, B\}, \{C, B\}\}$. This time we assume that both proteins A and C are competing on the same binding domain of protein B (Figure 1.3). Thus the two interactions of this network are mutually exclusive. This can be achieved by the conjunction of two constraints:

$$(\{A, B\} \Rightarrow \neg\{C, B\}) \wedge (\{C, B\} \Rightarrow \neg\{A, B\})$$

Again, the satisfying Kripke states for this formula reflect the desired behaviour (Table 4.2): the two interactions cannot appear at the same time, but any other combination – including no interaction at all – is allowed. The latter is the reason why one cannot simply use equivalence here:

$$\{A, B\} \Leftrightarrow \neg\{C, B\}$$

shows the same properties, except that an absence of both interactions would lead to an unsatisfied constraint, which is certainly not desired.

Now a protein hypernetwork can be defined as a combination of proteins, domain interactions and a set of constraints. In order to ensure consistency among interactions and their proteins, we force a default constraint for each interaction.

4.7 Definition (Protein hypernetwork). Let P be a set of proteins, and I be a set of domain interactions. A triple

$$(P, I, C)$$

with a set of constraints $C \subseteq \mathfrak{C}(P \cup I)$ is called protein hypernetwork. Additionally C has to contain a set $D(C)$ with a default constraint $i \Rightarrow p_1 \wedge p_2$ for each interaction $i = \{(p_1, d_1), (p_2, d_2)\} \in I$.

In the following we assume that the constraints C of a protein hypernetwork do not contradict each other, so that $q \wedge \bigwedge_{c \in C} c$ has to be satisfiable for all $q \in P \cup I$.

4.2 Data Mining in Protein Hypernetworks

One can think of many ways of information retrieval out of a protein hypernetwork. Presented here are methods based on the modal logic tableau, which follow a simple recipe:

1. Construct a (modal) logic formula, using the constraints.
2. Use the tableau algorithm (Chapter 3.2) to find a satisfying Kripke model.
3. Post process the Kripke model in order to retrieve the desired information.

Based on this recipe, we now present an approach to calculate perturbation effects. Later, *minimal network states* are introduced, which represent for an interaction all proteins and interactions that need to be simultaneously possible and those that are not possible simultaneously.

4.2.1 Perturbation Effects

For the prediction of perturbation effects (Chapter 1), we use all three structures of the protein hypernetwork to define a logic formula, which shows the wanted behaviour. We manipulate that formula by adding additional subformulae which model a perturbation. Retrieval of a satisfying Kripke model by the tableau algorithm will provide the information about the effects of perturbation. In our context, they can be summarized by *directly* and *indirectly* perturbed proteins or interactions. A directly perturbed protein or interaction is the equivalent of experimentally removing the protein in the real network. An indirectly perturbed protein or interaction disappears because of logic constraints propagating the effect of a perturbation. For example, an interaction $\{A, B\}$ can be scaffold dependent on a perturbed interaction $\{C, B\}$ (Example 4.5). Accordingly, both interactions are impossible to appear in the network, while $\{C, B\}$ is directly and $\{A, B\}$ is indirectly perturbed.

4.8 Definition (Perturbation formula). Let (P, I, C) be a protein hypernetwork. The *perturbation formula* is given by

$$\mathcal{P}_{(P,I,C)} := \bigwedge_{q \in P \cup I} \left(\diamond \left(q \wedge \bigwedge_{c \in C(q)} c \right) \vee \square \neg q \right)$$

with $C(q) := \{c \in C \mid c = (q \Rightarrow \psi)\}$.

For each interaction or protein q the first disjunct $\diamond(q \wedge \bigwedge_{c \in C(q)} c)$ models the situation that – if all of its constraints are satisfiable – there exists a Kripke state in which q is possible. The second disjunct $\square \neg q$ describes that no such state exists. Obviously a constraint for q can enforce an interaction or protein r to be negated. Due to the possibility operator \diamond this happens in an isolated Kripke state and has no effect on the general possibility of r . That is, there cannot appear any conflict between the conjuncts for different proteins or interactions of the perturbation formula. In contrast to the first disjunct, the satisfaction of the second disjunct $\square \neg q$ has global a effect and propagates the impossibility of interaction q to all Kripke states. The usage of $C(q)$ instead of C in conjunction with q speeds up the computation with the tableau algorithm without changing the solution. This is because effects of other constraints $(q' \Rightarrow \psi) \in C \setminus C(q)$ are propagated by the satisfaction of the corresponding second disjunct $\square \neg q'$.

The perturbation formula is satisfiable since for each interaction or protein either the first or the second disjunct is satisfiable. If the tableau algorithm is applied onto this formula, for each interaction or protein the first disjunct will be tried to be satisfied. If the first disjunct is unsatisfiable, the second disjunct will be expanded.

This reflects that an interaction is assumed to be impossible iff the requirements for it to be possible are not met – in other words: if its constraints are not satisfiable.

Biologically, the formula describes that an interaction or protein is either possible to appear at some point in time somewhere in the cell (accordingly a Kripke state satisfying it has to exist), or cannot appear at all. The possibility to appear at some point is linked to the satisfiability of the constraints, hence may not violate the interaction logic.

Using the perturbation formula, a perturbation can now be simulated by adding an additional subformula.

4.9 Theorem (Perturbation). *Let (P, I, C) be a protein hypernetwork. Consider (S, R, \Vdash) to be a satisfying Kripke model derived from the application of the tableau algorithm onto*

$$\left(\bigwedge_{q \in P_{\downarrow} \cup I_{\downarrow}} \Box \neg q \right) \wedge \mathcal{P}_{(P, I, C)}.$$

Then the set of all proteins that are not directly or indirectly perturbed after the perturbation of proteins $P_{\downarrow} \subseteq P$ and interactions $I_{\downarrow} \subseteq I$ is

$$Q := \{q \in P \cup I \mid \exists s \in S : s \Vdash q\}.$$

Proof. The conjunctive formula does not contain any non-modal proposition. Accordingly the first state $s_0 \in S$ produced by the tableau algorithm does not satisfy any proposition and can be ignored. For each other state $s \in S \setminus \{s_0\}$, it holds that $(s_0, s) \in R$ giving s_0 the role of an empty base state.

We show that the set Q of not perturbed proteins and interactions does not contain any directly or indirectly perturbed interaction or protein. Assume a protein or interaction $q \in P \cup I$ to be perturbed. If $q \in P_{\downarrow} \cup I_{\downarrow}$, then the conjunction of $\Box \neg q$ ensures that there is no state $s \in S$ that satisfies q ($\forall s \in S : s \not\Vdash q$). If q is indirectly perturbed ($q \in (P \cup I) \setminus (P_{\downarrow} \cup I_{\downarrow})$), then the perturbation of q has to depend on a directly or indirectly perturbed protein or interaction $q' \in P \cup I$. Thus there has to exist a constraint $(q \Rightarrow \phi) \in C(q)$ which is not satisfiable in conjunction with $\Box \neg q'$. The unsatisfiable constraint turns the first disjunct of $\Diamond(q \wedge \bigwedge_{c \in C(q)} c) \vee \Box \neg q$ to be unsatisfiable. Accordingly the tableau algorithm has to choose the second disjunct $\Box \neg q$ so that there is no state $s \in S$ satisfying q and thus q is not contained in the set.

Next we show that the set contains all interactions and proteins that are not directly or indirectly perturbed. Assume a protein or interaction $q \in P \cup I$ to be not perturbed. Then $q \notin P_{\downarrow} \cup I_{\downarrow}$, so that there is no conjunct $\Box \neg q$ in the formula. As q is assumed to be not perturbed, all of its constraints $c \in C(q)$ have to be satisfiable in conjunction with q . Thus the tableau chooses the first disjunct of $\Diamond(q \wedge \bigwedge_{c \in C(q)} c) \vee \Box \neg q$ and creates a Kripke state $s_q \in S$ with $s_q \Vdash q$ in particular. Hence q is contained in the set. ■

4.2.2 Minimal Network States

Biologically, logic constraints may restrict the possibility of an interaction by imposing dependencies on proteins or interactions. For an interaction to be possible,

4 Protein Hypernetworks

Input: a modal logic formula $\left(\bigwedge_{q \in P_{\downarrow} \cup I_{\downarrow}} \neg q\right) \wedge \mathcal{M}_{(P,I,C)}(i)$

Output: a set K of satisfying Kripke models

- 1: $K := \emptyset$
- 2: put \emptyset on the stack
- 3: **while** stack is not empty **do**
- 4: take set of blocked formulae B from stack
- 5: calculate tableau (strictly left expanding) assuming each labelled formula (s, ψ) with $\psi \in B$ to be *blocked*
- 6: let $k := (\{s\}, R, \Vdash)$ be the Kripke model derived from the tableau
- 7: **if** k is satisfying **then**
- 8: $K := K \cup \{k\}$
- 9: **for all** active disjuncts $s \Vdash \psi_j$ with $s \Vdash \bigvee_{k \in \{1, \dots, n\}} \psi_k$ and $j < n$ **do**
- 10: $B' := B \cup \{\psi_i\}$
- 11: put B' on the stack
- 12: **end for**
- 13: **end if**
- 14: **end while**

Algorithm 4.1: Algorithm to find all reasonable satisfying Kripke models for a minimal network state formula.

it may be necessary that proteins or other interactions are possible simultaneously. Further, other proteins or interactions may become impossible. This information will be captured in minimal network states. Later, minimal network states can be used to derive whether two individual interactions are possible simultaneously, or even to generate subnetworks of a protein network that are possible simultaneously (Chapter 6).

Using above recipe again, minimal network states will be derived from Kripke models satisfying a *minimal network state formula* in conjunction with a perturbation term. The perturbation term is necessary because a perturbation may affect the minimal network state. When no perturbations are applied, the term will be empty.

4.10 Definition (Minimal network state formula). Let (P, I, C) be a protein hypernetwork. For an interaction $i \in I$ the logic formula

$$\mathcal{M}_{(P,I,C)}(i) := i \wedge \bigwedge_{c \in C} c$$

is called minimal network state formula.

The conjunction of $\mathcal{M}_{(P,I,C)}(i)$ with a perturbation term of all perturbed proteins $P_{\downarrow} \subseteq P$ and interactions $I_{\downarrow} \subseteq I$

$$\left(\bigwedge_{q \in P_{\downarrow} \cup I_{\downarrow}} \neg q \right) \wedge \mathcal{M}_{(P,I,C)}(i)$$

enables us to calculate minimal network states. The formula shows the following behaviour when applying the tableau algorithm: For constraints of the form $i \Rightarrow \phi$,

the formula $\phi \in \mathfrak{Prop}(P \cup I)$ is expanded. This is obviously correct, because all these constraints contain information about the state of the network when i is existing. It can lead to additionally satisfied propositions which enforces the expansion of further constraints. However, constraints with disjunctions

$$\bigvee_{k \in \{1, \dots, n\}} \psi_k$$

as subformulae can produce situations in which a formula $\mathcal{M}_{(P,I,C)}(i)$ has several reasonable satisfying Kripke models. For example, a Kripke model with $s \Vdash \psi_1$ can be reasonable as well as one with $s \Vdash \psi_2$. Therefore, we guide the tableau to each of these Kripke models by subsequent blocking of these disjuncts (Algorithm 4.1). Further we need to use the strictly left expanding variant of the tableau here (Definition 3.9). Now a minimal network state for an interaction can be defined. We assume that an interaction for which we investigate a minimal network state is not perturbed in any way (neither directly nor indirectly due to other perturbations).

4.11 Definition (Minimal Network State). Consider a protein hypernetwork (P, I, C) with an interaction $i \in I$. Let $(\{s\}, R, \Vdash)$ be a satisfying Kripke model derived from the application of Algorithm 4.1 onto

$$\left(\bigwedge_{q \in P_{\downarrow} \cup I_{\downarrow}} \neg q \right) \wedge \mathcal{M}_{(P,I,C)}(i)$$

with perturbed proteins $P_{\downarrow} \subseteq P$ and interactions $I_{\downarrow} \subseteq I \setminus \{i\}$. Then the set of necessarily possible proteins or interactions is denoted as

$$Nec := \{i\} \cup \{q \mid q \in P \cup I, s \Vdash q, q \text{ subformula of an active } c \in C\},$$

whereas the set of impossible proteins or interactions is denoted as

$$Imp := \{q \mid q \in P \cup I, s \Vdash \neg q, \neg q \text{ subformula of an active } c \in C\}.$$

The pair (Nec, Imp) is called minimal network state for interaction i .

Minimal network states are a condensed version of the valuable information for the possibility of an interaction contained in the Kripke model. Since Algorithm 4.1 may lead to several Kripke models, there may be several minimal network states for one interaction. An example illustrates why it is reasonable to include only propositions or negated propositions that are subformulae of the right part of active constraints into Nec and Imp :

4.12 Example. Assume a protein hypernetwork with proteins $P = \{A, B, C, D\}$, interactions $I = \{\{A, B\}, \{A, C\}, \{C, B\}, \{C, D\}\}$ and constraints $C = \{\{A, B\} \Rightarrow \{C, B\}, \{A, C\} \Rightarrow \neg\{C, D\}, \{C, D\} \Rightarrow \neg\{A, C\}\}$. In other words, the interaction between A and B is dependent on the possibility of the interaction between C and B . Further the interactions $\{A, C\}$ and $\{C, D\}$ are mutually exclusive, for example

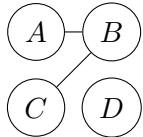
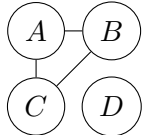
| $\{A, B\}$ | $\{C, B\}$ | $\{A, C\}$ | $\{C, D\}$ | $\mathcal{M}_{(P,I,C)}(\{A, B\})$ | Network state |
|------------|------------|------------|------------|-----------------------------------|---|
| 1 | 1 | 0 | 0 | 1 |  |
| 1 | 1 | 1 | 0 | 1 |  |

Table 4.3: A reduced truth table for the minimal network state formula of Example 4.12. The first row represents the state of a Kripke model for this formula. The second row is a satisfying Kripke state too, but the interaction $\{A, C\}$ is unnecessarily satisfied.

by competition on the same binding domain. The minimal network state formula for the interaction $\{A, B\}$ is given by

$$\begin{aligned} \mathcal{M}_{(P,I,C)}(\{A, B\}) = & \{A, B\} \wedge \\ & (\{A, B\} \Rightarrow \{C, B\}) \wedge \\ & (\{A, C\} \Rightarrow \neg\{C, D\}) \wedge (\{C, D\} \Rightarrow \neg\{A, C\}). \end{aligned}$$

A suitable satisfying Kripke model $(\{s\}, R, \Vdash)$ for this formula is shown in the first row of Table 4.3. As the interaction between protein A and C is not necessary for the interaction between A and B , it is not satisfied by the Kripke model. The minimal network state for this example is

$$(\{\{A, B\}, \{C, B\}, A, B, C\}, \emptyset).$$

That is, the interaction $\{C, B\}$ is necessary for interaction $\{A, B\}$, and there are no impossible interactions when A is interacting with B . Further, the proteins A , B and C have to be necessarily possible together with $\{A, B\}$. Neither $\{A, C\}$ nor $\{C, D\}$ are assumed to be impossible, because they are not negated due to an active constraint. That is obviously correct since the interaction between A and B is not dependent on those at all.

Now we define a relation *clashing*, which describes a situation where two minimal network states cannot be combined with each other without producing a conflict.

4.13 Definition (Clashing Minimal Network States). Two minimal network states (Nec, Imp) and (Nec', Imp') are clashing iff $Nec \cap Imp' \neq \emptyset$ or $Imp \cap Nec' \neq \emptyset$.

4.14 Theorem. Let (P, I, C) be a protein hypernetwork with perturbations $P_{\downarrow} \subseteq P$ and $I_{\downarrow} \subseteq I \setminus \{i, j\}$ and two arbitrary interactions $i, j \in I$, $i \neq j$. Consider the two minimal network states $m_i := (Nec_i, Imp_i)$ and $m_j := (Nec_j, Imp_j)$ for these interactions. If m_i and m_j are not clashing, then i and j are possible simultaneously.

Proof. The simultaneous possibility of i and j under all constraints and perturbations can be described by the conjunction

$$\xi := \left(\bigwedge_{q \in P_{\downarrow} \cup I_{\downarrow}} \neg q \right) \wedge \left(\bigwedge_{c \in C} c \right) \wedge i \wedge j.$$

We will show that ξ is satisfiable if m_i and m_j are not clashing. Therefore we define a Kripke model $(\{s\}, R, \Vdash)$ and retrieve the satisfiability relation \Vdash out of the two minimal network states. If the Kripke model then satisfies ξ , the proof is finished.

Using a set $Pos := Nec_i \cup Nec_j$ of positive propositions, we assume $s \Vdash q$ for each $q \in Pos$. Further the set $Neg := (P \cup I) \setminus Pos$ determines the rest of the satisfiability relation by setting $s \Vdash \neg q'$ for $q' \in Neg$.

From the definition of minimal network states we know that $(P_{\downarrow} \cup I_{\downarrow} \cup Imp_i \cup Imp_j) \subseteq Neg$. The Kripke model is well defined – thus there is no proposition q with $s \Vdash q$ as well as $s \Vdash \neg q$ – because we know that $Pos \cap Neg = \emptyset$.

Now we show that all conjuncts of ξ are satisfied by the Kripke state s . The conjuncts i and j are satisfied since $i \in Nec_i$ and $j \in Nec_j$. The conjunct

$$\bigwedge_{q \in P_{\downarrow} \cup I_{\downarrow}} \neg q$$

is satisfied because $(P_{\downarrow} \cup I_{\downarrow}) \subseteq Neg$. We investigate the last conjunct of ξ

$$\bigwedge_{c \in C} c.$$

For each $c = (r \Rightarrow \phi)$ there may appear two cases:

In the first case, s satisfies r ($s \Vdash r$). This means that $r \in Pos$. Without loss of generality we assume $r \in Nec_i \subseteq Pos$ and see that c has to be satisfied and active in the Kripke model (S_i, R_i, \Vdash_i) that lead to the minimal network state m_i . Thus the propositions q_1, \dots, q_n that are needed to satisfy ϕ are either contained in Nec_i or Imp_i and thus also either in Pos or Neg .

In the second case s does not satisfy r . Then c is satisfied by s regardless of the satisfaction of ϕ because of the definition (Definition 3.3) of the operator \Rightarrow .

Since the Kripke model satisfies all conjuncts of ξ , we know that ξ is satisfiable so that i and j are simultaneously possible. \blacksquare

Theorem 4.14 shows that not clashing minimal network states can be combined to receive proteins and interactions that are possible simultaneously. This approach will be used later for the prediction of protein complexes (Chapter 5.3). The proof contains an alternative test for the simultaneous possibility of two interactions: The tableau algorithm could be used to check the satisfiability of the formula ξ . However this would only reveal a part of the information inherent in minimal network states.

5 Prediction of Protein Complexes

Protein hypernetworks allow for improvements in the analysis of protein networks. The prediction of protein complexes serves as both an example for this and as a platform for two new methods of analysis which are proposed later (Chapter 6).

In literature, protein complexes (Chapter 1) are assumed to correspond to dense regions in protein networks (Li et al., 2005). Evidence for this is given by Spirin and Mirny (2003), showing that generally all proteins in a dense region are responsible for the execution of the same biological function. This observation matches the interpretation of protein complexes being molecular machines (Chapter 1).

A dense region in a protein network (P, I) is a connected (Definition 2.6) subgraph induced by a subset of proteins $P' \subseteq P$ (Definition 2.7) with a significantly higher *connectivity* than its environment. A measurement for the connectivity of a subgraph is the *density* or *clustering coefficient* (Li et al., 2005):

5.1 Definition (Density). Let (V, E) be an undirected, loop-free graph. The *density* or *clustering coefficient* of this graph is defined as

$$cc((V, E)) := \frac{|E|}{|V|(|V| - 1)/2} = \frac{2|E|}{|V|(|V| - 1)}.$$

The denominator $|V|(|V| - 1)/2$ describes the maximum number of edges in the graph. Accordingly a fully connected graph (that is, a clique) will have a density of 1, whereas a graph without any edge will have a density of 0. We denote a protein complex in the following way:

5.2 Definition (Protein Complex). Let (P, I) be a protein network. A protein complex is a subset $c \subseteq P$ of proteins with a connected induced subgraph $S_{(P, I)}(c)$.

Since complexes are considered to be sets of nodes, which can be represented as a subgraph, we use the *overlapping score* (Jung et al., 2010) to measure their similarity. In the literature, this score is also referred as *neighbour affinity* (Li et al., 2005).

5.3 Definition (Overlapping Score). For an arbitrary common superset S , given two sets $A \subseteq S$ and $B \subseteq S$, their overlapping score is defined as

$$OS(A, B) := \frac{|A \cap B|^2}{|A| \cdot |B|}.$$

If two sets A and B are equal, then $OS(A, B)$ equals 1. If they do not overlap at all, $OS(A, B)$ equals 0.

Two example complex prediction algorithms that both use density to predict likely complexes are *MCODE* (Bader and Hogue, 2003) and *LCMA* (Li et al., 2005). Since

these procedures only rely on protein networks, they are called *network based* in the following as a distinction to *hypernetwork based* complex prediction.

The MCODE (Molecular Complex Detection) algorithm (Bader and Hogue, 2003) consists of three steps. First it calculates a *score* for each protein, roughly based on the density of its neighbourhood (see chapter 2.1). Afterwards it predicts complexes by starting from a *seed protein* and moving recursively outwards adding proteins as long as their score is not more than a given percentage below the score of the seed protein. During this step, each protein will be visited only once and the seed proteins are selected descendingly according to their score. Hence the predicted complexes do not overlap. The third step provides a mechanism to allow overlapping complexes and cutting of loosely connected proteins. The time complexity of MCODE for a protein network (P, I) is $O(|P| |I| h^3)$ with h being the average neighbourhood size in (P, I) .

According to Li et al. (2005), LCMA offers better results and a more efficient runtime. Hence it was decided to concentrate on this algorithm in this thesis (Chapter 5.2). However, concerns remain about the comparison of results by Li et al. (2005) which we will investigate later.

5.1 Measuring Prediction Quality

In order to measure the quality of complex prediction, we use the *MIPS Comprehensive Yeast Genome Database* (<ftp://ftpmips.gsf.de/yeast/>). Firstly, it contains the definition of the *yeast protein network*. Secondly it contains a database of known yeast protein complexes – in the following referred as *MIPS complexes*. The latter consists of 1142 complexes from which 267 are annotated with their biological function. In the literature (Li et al., 2005; Jung et al., 2010), the functionally annotated complexes are considered to be more reliable, thus only annotated complexes are used as a benchmark for complex prediction on the yeast protein network.

Following Jung et al. (2010), if P is the set of predicted protein complexes and D is the set of known complexes from the database, then the set of true positively predicted complexes is denoted as

$$TP_P := \{p \in P \mid \exists m \in D : OS(p, m) > 0.2\}, \quad (5.1)$$

whereas the set of true positively matched known complexes from the database is

$$TP_D := \{m \in D \mid \exists p \in P : OS(p, m) > 0.2\}. \quad (5.2)$$

It can be seen that the overlapping score (Definition 5.3) is used to measure the similarity between a predicted complex and a database complex. Further the set of false positive predictions is denoted as

$$FP := P \setminus TP_P \quad (5.3)$$

and the set of false negatives, in other words the set of not matched complexes from the database, is denoted as

$$FN := D \setminus TP_D. \quad (5.4)$$

The quality of prediction can now be described by the three measurements *recall*, *precision* and *F-measure*.

5.4 Definition (Recall, Precision and F-Measure). Let D be a set of known protein complexes and P be a set of predicted complexes. The recall is defined as

$$r := \frac{TP_D}{|D|}$$

and the precision is defined as

$$p := \frac{TP_P}{|P|}.$$

The F-measure is defined as the combination of recall and precision

$$\frac{2rp}{r+p}.$$

The recall of a prediction describes the ability to recognize the complexes of a given database. The precision characterizes the ability to avoid erroneous predictions. The maximum value of all three measures is 1. If a prediction P leads to a higher recall than a prediction P' , it better recognizes the complexes in D . If the prediction P leads to a higher precision than P' , it contains less false predictions.

5.2 The LCM Algorithm

LCMA (Local Clique Merging Algorithm, Li et al., 2005) follows a bottom-up strategy to find dense regions in a loop-free undirected graph (V, E) (Definition 2.2). First, it detects local cliques by investigating the neighbourhood of each protein. In a second step, the algorithm merges local cliques with a significant overlap as long as the average density of all detected dense regions is high enough.

5.2.1 Detecting local cliques

In order to find a local clique, the density of a neighbourhood $n_{(V,E)}(v)$ with $v \in V$ is investigated. As long as it raises the density, the neighbour with the lowest degree is removed from the neighbourhood. Algorithm 5.1 shows the details of this procedure. For each node $v \in V$ at first the subgraph induced by its neighbourhood including itself is generated (Line 3). The density of the subgraph and the node with minimal degree are recorded (Lines 14, 5). In a loop, the node with minimal degree is removed from the neighbourhood if that raises the subgraph's density (Line 6). The subgraph is updated accordingly including edges and degrees of nodes. As the maximum value of density is 1, this loop stops once the subgraph is fully interconnected, in other words if it is a clique (Li et al., 2005). Of course this can also be the case due to the subgraph being a trivial clique, containing only one or two nodes. Only non-trivial cliques are considered to be a valid result (Line 12).

5 Prediction of Protein Complexes

Input: Graph (V, E)

Output: A set of local cliques C

```

1:  $C := \emptyset$ 
2: for  $v \in V$  do
3:    $(V', E') := S_{(V,E)}(n_{(V,E)}(v) \cup \{v\})$ 
4:    $\lambda := cc((V', E'))$ 
5:    $v' := \arg \min_{v \in V'} deg_{(V', E')}(v)$ 
6:   while  $cc(S_{(V', E')}(V' \setminus v')) > \lambda$  do
7:      $(V', E') := S_{(V', E')}(V' \setminus v')$ 
8:      $v' := \arg \min_{v \in V'} deg_{(V', E')}(v)$ 
9:      $\lambda := cc((V', E'))$ 
10:  end while
11:  if  $|V'| > 2$  then
12:     $C := C \cup \{V'\}$ 
13:  end if
14: end for
15: return  $C$ 

```

Algorithm 5.1: LCM algorithm (detect local cliques).

Input: Graph (V, E) , a set of local cliques C , an overlapping threshold ω

Output: A set of dense regions D

```

1:  $D := C$ 
2:  $\lambda := 1$ 
3: loop
4:    $D' := \emptyset$ 
5:   for  $d_1 \in D$  do
6:      $d := d_1$ 
7:     for  $d_2 \in D$  do
8:       if  $d_1 \neq d_2$  and  $OS(d_1, d_2) > \omega$  then
9:          $d := d \cup d_2$ 
10:      end if
11:    end for
12:     $D' := D' \cup \{d\}$ 
13:  end for
14:   $\lambda' := \frac{1}{|D'|} \sum_{d \in D'} cc(S_{(V,E)}(d))$ 
15:  if  $\lambda' \leq 0.95\lambda$  or no merge performed then
16:    return  $D'$ 
17:  else
18:     $D := D'$ 
19:     $\lambda := \lambda'$ 
20:  end if
21: end loop

```

Algorithm 5.2: LCM algorithm (merge dense regions).

5.2.2 Merging dense regions

After the detection of local cliques, a second step (Algorithm 5.2) merges them to bigger but still dense regions: they are iteratively merged until the average density is below 95% of the previous average density. The algorithm starts with the detected local cliques as dense regions D (Line 1). First the average density λ is set to 1, since the algorithm starts with entirely local cliques as dense regions (Line 2). In an iterative process the next set of dense regions (Line 4) is computed. Each region $d \in D$ is merged together with all dense regions $d' \in D$, for which the overlapping score (Definition 5.3) is higher than a given threshold ω . The parameter ω can be used to ensure that a merge is occurring only for significantly overlapping regions. The combination with the termination criterion (Line 15) shall ensure that the results after merging can still be considered dense. If the termination criterion is not met, another iteration is performed, using the new average density λ' and the new dense regions D' (Lines 18,19).

5.2.3 Implementation

Naive calculation of the overlapping score for two dense regions $OS(d, d')$ – needed for the check in line 8 of the merge step (Algorithm 5.2) – involves the intersection of the two. If using a hash set implementation for dense regions, the intersection always takes $O(|d|)$ operations provided $|d| \leq |d'|$. Algorithm 5.3 performs the check $OS(d, d') > \omega$ in the best case in one iteration, and $|d|$ iterations in the worst case.

Recalling the density formula (Definition 5.1), we see the necessity of tracking the number of edges in dense regions in order to be able to calculate the average density in line 14 of Algorithm 5.2. In other words, for each dense region d the subgraph $S_{(V,E)}(d)$ has to be retrieved, which takes $O(|E|)$ operations in the worst case. Accepting a loss in precision of density calculation, this can be sped up significantly. Therefore, we record the number of edges $|E|_d$ for each dense region d . When Algorithm 5.2 is started, each dense region $d \in D$ is a clique (Line 1), thus its number

Input: Two dense regions d, d' , an overlapping threshold ω

Output: True if $OS(d, d') > \omega$

```

1:  $c = 0$ 
2:  $u = \frac{1}{|d| \cdot |d'|}$ 
3: for all  $v \in d$  do
4:   if  $v \in d'$  then
5:      $c = c + 1$ 
6:     if  $c^2 \cdot u > \omega$  then
7:       return True
8:     end if
9:   end if
10: end for
11: return False

```

Algorithm 5.3: LCM algorithm (fast check for overlapping dense regions).

5 Prediction of Protein Complexes

of edges is $|E|_d = \frac{1}{2}|d|(|d| - 1)$. During the merge of two dense regions d and d' (Line 9), the new edge number can be approximated by

$$|E|_{d \cup d'} := |E|_d + |E|_{d'} - \frac{1}{2}|d \cap d'|(|d \cap d'| - 1). \quad (5.5)$$

Using this, the edge number for the merged dense region can be calculated in $O(|d|)$ operations, assuming $|d| \leq |d'|$ without loss of generality. Here, $\frac{1}{2}|d \cap d'|(|d \cap d'| - 1)$ describes the number of edges in a clique induced by the intersection of d and d' . Hence, if d and d' are cliques, this approximation calculates the size of the union between the edges of the two cliques, while it misses all edges between nodes of d and d' which are not contained in any of the cliques. We will now estimate the worst case approximation performance for d and d' being cliques. The maximum number of missed edges occurs if d and d' do not have any edge in common ($|d \cap d'| = 1$), and the subgraph induced by the merged region $S_{(V,E)}(d \cup d')$ is a clique, again. The optimal solution – in other words, the real number of edges for the merged region – can be written as

$$\begin{aligned} OPT(d \cup d') &= \frac{1}{2}|d \cup d'|(|d \cup d'| - 1) \\ &= \frac{1}{2}(|d| + |d'| - |d \cap d'|)(|d| + |d'| - |d \cap d'| - 1) \\ &= \frac{1}{2}(|d| + |d'| - 1)(|d| + |d'| - 2) \\ &= \frac{1}{2}(|d|^2 + |d'|^2 + 2|d||d'| - 3|d| - 3|d'| + 2). \end{aligned}$$

According to formula (5.5), the approximated number of edges is

$$\begin{aligned} |E|_{d \cup d'} &= \frac{1}{2}(|d|(|d| - 1) + |d'|(|d'| - 1)) - \frac{1}{2}|d \cap d'|(|d \cap d'| - 1) \\ &= \frac{1}{2}(|d|^2 - |d| + |d'|^2 - |d'|). \end{aligned}$$

Without loss of generality we assume that $|d| \geq |d'|$. Now the approximation performance can be estimated as

$$\begin{aligned} \frac{OPT(d \cup d')}{|E|_{d \cup d'}} &= \frac{\frac{1}{2}(|d|^2 + |d'|^2 + 2|d||d'| - 3|d| - 3|d'| + 2)}{\frac{1}{2}(|d|^2 - |d| + |d'|^2 - |d'|)} \\ &= \frac{|d|^2 - |d| + |d'|^2 - |d'| + 2|d||d'| - 2|d| - 2|d'| + 2}{|d|^2 - |d| + |d'|^2 - |d'|} \\ &= 1 + \frac{2|d||d'| - 2|d| - 2|d'| + 2}{|d|^2 - |d| + |d'|^2 - |d'|} \\ &< 1 + 2 \frac{|d'|^2 - 2|d'| + 1}{2|d'|^2 - 2|d'|} \\ &= 1 + \frac{|d'|^2 - |d'| - |d'| + 1}{|d'|^2 - |d'|} \\ &= 2 - \frac{|d'| - 1}{|d'|^2 - |d'|} = 2 - \frac{1}{|d'|}. \end{aligned}$$

Since this result resembles the worst case, we can guarantee the method to be a 2-approximation for the merging of two cliques.

| Algorithm | Recall | Precision | F-Measure |
|-----------|--------|-----------|-----------|
| MCODE | 0.213 | 0.314 | 0.254 |
| LCMA | 0.401 | 0.098 | 0.158 |

Table 5.1: Comparison of MCODE and LCMA prediction quality. Tests were performed with the protein network and complex data from the MIPS Comprehensive Yeast Genome Database (Jung et al., 2010).

The approximation becomes worse when merging two dense regions which are already merged cliques themselves. However, complex prediction on the yeast protein network showed no difference at all between the approximation and accurately calculating the subgraphs.

5.2.4 Discussion

Li et al. (2005) argue that LCMA outperforms MCODE regarding performance and prediction results. The prediction results are measured in the same way as by Bader and Hogue (2003). This means – differing from this thesis and Jung et al. (2010) – that the recall (which is called *sensitivity* by Bader and Hogue (2003)) is defined as $\frac{|TP_P|}{|TP_P|+|FN|}$. In the case of Bader and Hogue (2003) this definition equals our definition of the recall: this is because MCODE does not predict fully overlapping complexes, so that $|TP_P| = |TP_D|$ and $|TP_D| + |FN| = |D|$. For LCMA this is not the case. The fact that a predicted complex can be fully contained in another predicted complex, causes $|TP_P|$ to be always greater or equal than $|TP_D|$. Thus the prediction quality presented by Li et al. (2005) is most likely overestimated. A fair comparison was performed by Jung et al. (2010). It can be seen that MCODE has a better precision but LCMA has a greater recall value (Table 5.1).

The algorithm does not define what happens if several nodes of minimal degree exist in a neighbourhood. This gives rise to the problem that for the same input two distinct implementations of the algorithm can produce different local cliques. Figure 5.1 shows an example for such a situation.

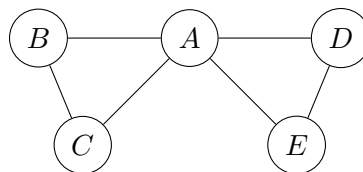


Figure 5.1: Example for LCMA predicting different cliques depending on selection of node with minimal degree. The shown graph is assumed to be the neighbourhood graph of node A. The nodes B, C, D and E all have a degree of 2, so one of them is selected to be removed in step one (Algorithm 5.1) of the LCM algorithm in order to detect a local clique from node A. If one of the nodes B or C is removed, the clique $\{A, D, E\}$ will be detected. Analogously the removal of D or E causes the detection of the clique $\{A, B, C\}$.

5.3 Complex Prediction with Protein Hypernetworks

Complex prediction out of a protein hypernetwork should take advantage of the included constraints. Jung et al. (2010) provide an approach which retrieves networks of simultaneously possible protein interactions out of information about mutually exclusive ones, and uses these to refine the network based complex prediction. Our approach is more general since the possibility of proteins and interactions can be – in the scope of propositional logic – arbitrarily constrained.

We use minimal network states to know about necessary and impossible interactions or proteins for each interaction. This information is used to predict complexes which contain only simultaneously possible proteins and interactions and do not lack any necessary interaction. For a protein hypernetwork (P, I, C) the prediction of likely complexes can be achieved as follows:

1. Perform perturbations and reveal not perturbed interactions and proteins.
2. Predict complexes on the protein network of all possible interactions and proteins with a network based complex prediction.
3. Compute minimal network states for all possible interactions.
4. Refine predicted complexes using minimal network states.

Similar to Jung et al. (2010), we do not modify a network based complex prediction algorithm but rather use its predicted complexes to obtain refined ones that respect the provided additional information.

5.3.1 Perform Perturbations

The effects of perturbations $P_{\downarrow} \subseteq P$ and $I_{\downarrow} \subseteq I$ can be predicted by using the perturbation formula (Definition 4.8) as described in Theorem 4.9. Doing this, we retrieve a set

$$Q \subseteq P \cup I \tag{5.6}$$

of not perturbed proteins and interactions. If P_{\downarrow} and I_{\downarrow} are empty, thus no perturbation was applied, then Q equals $P \cup I$.

5.3.2 Network Based Complex Prediction

This step is taken to divide the network into smaller subnetworks. First, the set of not perturbed proteins and interactions (5.6) is translated into a protein network (Definition 4.1).

5.5 Definition (Not Perturbed Protein Network). Let (P, I, C) be a protein hypernetwork and $Q \subseteq P \cup I$ be a set of not perturbed interactions and proteins. Then (P_Q, I_Q) with $P_Q := P \cap Q$ and $I_Q := \{\{p_1, p_2\} \mid \{(p_1, d_1), (p_2, d_2)\} \in I \cap Q\}$ defines the *not perturbed protein network*.

As can be seen, we abandon the domain information of the interactions from the protein hypernetwork, because network based complex prediction algorithms are restricted to undirected graphs.

Secondly, a network based complex prediction algorithm (here for example the LCM algorithm) is used to predict complexes $\mathcal{C}' \subseteq 2^{P_Q}$ on the not perturbed protein network (P_Q, I_Q) . If the algorithm for network based complex prediction is not able to handle loops (which is the case for LCMA), they have to be removed from (P_Q, I_Q) .

5.3.3 Compute Minimal Network States

The retrieval of all simultaneously necessary and impossible interactions or proteins for each interaction is done by the computation of all minimal network states for each not perturbed interaction $i \in Q \cap I$, as shown in chapter 4.2.2. By this we gain a set of minimal network states

$$M_i = \{(Nec_1, Imp_1), \dots, (Nec_n, Imp_n)\} \quad (5.7)$$

for each not perturbed interaction i .

5.3.4 Refine predicted complexes

On the one hand, a network based predicted complex $c \subseteq P_Q$ can contain interactions or proteins which may not be possible at the same time due to some constraints. On the other hand, refinement has to take care that a predicted complex does not miss any necessary interaction or protein. Thus the refinement process consists of two distinct steps:

1. For each network based predicted complex, generate subnetworks of simultaneously possible proteins and interactions, and again perform a network based prediction on those.
2. For each of these newly predicted complexes, add all interactions and proteins that are necessary for any contained interaction (according to its minimal network states).

In the first step, for each complex $c \subseteq P_Q$ we collect all minimal network states for contained interactions $i \in I_c$ with $(P_c, I_c) = S_{(P_Q, I_Q)}(c)$:

$$M_c := \{(Nec, Imp) \mid (Nec, Imp) \in M_i, i \in I_c\} \quad (5.8)$$

Algorithm 5.4 provides a map

$$clash : M_c \rightarrow 2^{M_c} \quad (5.9)$$

which maps each minimal network state $m \in M_c$ onto all clashing minimal network states in M_c , in other words onto states that may not be combined with m (Definition 4.13).

Input: set of minimal network states $M_c = \{m_1, \dots, m_n\}$

Output: map $clash : M_c \rightarrow 2^{M_c}$, which maps each minimal network state onto all clashing minimal network states.

```

1: for  $k \in \{1, \dots, n\}$  do
2:    $(Nec_k, Imp_k) := m_k$ 
3:   for  $l \in \{k + 1, \dots, n\}$  do
4:      $(Nec_l, Imp_l) := m_l$ 
5:     if  $Nec_k \cap Imp_l \neq \emptyset$  or  $Imp_k \cap Nec_l \neq \emptyset$  then
6:        $clash(m_k) := clash(m_k) \cup \{m_l\}$ 
7:        $clash(m_l) := clash(m_l) \cup \{m_k\}$ 
8:     end if
9:   end for
10: end for

```

Algorithm 5.4: Find clashes between minimal network states.

Algorithm 5.5 takes the set of minimal network states M_c (Set (5.8)) and the map $clash : M_c \rightarrow 2^{M_c}$ (Map (5.9)) to build a tree (R, E) of removal instructions. Each path from the root to a leaf represents a sequence of minimal network states the removal of which turns M_c to be free of clashing minimal network states. The maintained tree is defined as follows:

5.6 Definition (Minimal Network State Tree). Let M_c be a set of minimal network states. A tree (R, E) is called minimal network state tree if there exists a map $state : R \rightarrow M_c$, so that each node $v \in R \setminus \{Root(R, E)\}$ is mapped onto a minimal network state $state(v) = m \in M_c$. We say that v is referencing m . A path p in the tree is called a path referencing $M \subseteq M_c$ iff for all minimal network states $m \in M$ there exists a node $v \in p$ with $state(v) = m$.

Consequently, Algorithm 5.5 manages a map $state : R \rightarrow M_c$, which maps each node $v \in R \setminus \{Root(R, E)\}$ onto a minimal network state $m \in M_c$. For each minimal network state $m \in M_c$, the algorithm tries to extend all paths from root to leafs to reflect the removal of either m or $clash(m) = \{m_1, \dots, m_n\}$. This is achieved by traversing the tree recursively (Procedure *Append* of algorithm 5.5).

Upon the encounter of a leaf v (Lines 1 to 8), the procedure appends a node v' with $state(v) = m$ to the leaf. Further it appends a path of elements referencing elements of $clash(m)$ as a sibling of v' to v . However not all elements of $clash(m)$ are referenced in the path. The elements that were encountered along the path from the root to v are left out. This is reasonable, since the path from the root to v already instructs their removal, so that there is no need to remove them again. In order to manage this information, the procedure uses bit vectors b . For an element $m_k \in clash(m)$ a value of $b[k] = 1$ means that it was not yet encountered on that path and shall therefore be referenced by the path appended to a leaf.

Upon the encounter of an inner node v (Lines 10 to 20), the procedure investigates all children v' of v . If $state(v') = m$, recursion is not performed on that child because the instruction of the removal of m by v' turns the removal of elements of $clash(m)$ superfluous. Else, a new bit vector $b' = b$ is instantiated, and eventually the bit at position r with $state(v') = m_r \in clash(m)$ is set to zero in order to reflect that the removal of m_r is already instructed on this path. If there are still elements of

5.3 Complex Prediction with Protein Hypernetworks

Input: A set of minimal network states M_c , a map $clash : M_c \rightarrow 2^{M_c}$.

Output: Tree (R, E) of removing instructions and a map $state : R \rightarrow M_c$, which maps each node of the tree onto a minimal network state.

```

1:  $R := \{root\}$ ,  $E := \emptyset$ 
2: for all  $m \in M_c$  do
3:   if  $clash(m) \neq \emptyset$  then
4:     Append( $root, m, clash(m), b \in \{1\}^{|clash(m)|}$ )
5:   end if
6: end for

```

Procedure Append

Input: Node $v \in R$, a minimal network state $m \in M_c$, a set of clashing minimal network states $clash(m) = \{m_1, \dots, m_{|clash(m)|}\}$ and a bit vector $b \in \{0, 1\}^{|clash(m)|}$

```

1: if  $v$  is a leaf then
2:   append a new node  $v'$  with  $state(v') = m$  as new child to  $v$ 
3:   for all  $j \in \{1, \dots, |clash(m)|\}$  do
4:     if  $b[j] = 1$  then
5:       append a new node  $v'$  with  $state(v') = m_j$  as new child to  $v$ 
6:        $v := v'$ 
7:     end if
8:   end for
9: else
10:  for all  $v'$  child of  $v$  do
11:    if  $state(v') \neq m$  then
12:       $b' := b$ 
13:      if  $state(v') = m_r \in clash(m)$  then
14:         $b'[r] := 0$ 
15:      end if
16:      if  $\exists k \in \{1, \dots, |b|\} : b[k] = 1$  then
17:        Append( $v', m, clash(m), b'$ )
18:      end if
19:    end if
20:  end for
21: end if

```

Algorithm 5.5: Algorithm to build a tree of removal instructions.

$clash(m)$ that are not yet removed – in other words, if the bit vector b' still contains a bit of value 1 (Line 16) – the procedure is invoked recursively on the child v' with the updated bit vector b' .

The tree (R, E) produced by Algorithm 5.5 can now be used to generate sets of not clashing minimal network states by subtracting all minimal network states that are referenced by a path from the root to a leaf:

$$M_c \setminus \{state(v) \mid v \in (root(R, E), \dots, l) \text{ with a leaf } l\}. \quad (5.10)$$

However there may exist paths that lead to an unneeded removal of minimal network states. Such paths are called *redundant*.

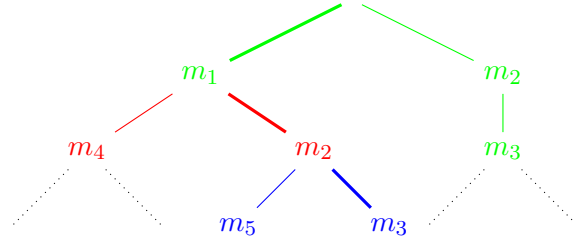


Figure 5.2: Example for a redundant path in a tree of removal instructions (R, E) produced by Algorithm 5.5. This shows a part of an exemplary tree of removal instructions for a set of minimal network states $M_c = \{m_1, m_2, \dots\}$. Nodes are annotated with the referenced minimal network state. Different colors separate different steps of appending a minimal network state m_k with its clashes $\text{clash}(m_k)$ – for example m_1 with $\text{clash}(m_1) = \{m_2, m_3\}$ (green). The path $(\text{root}(R, E), m_1, m_2, m_3)$ (thick) is redundant because it suggests the removal of m_1 although $\text{clash}(m_1) = \{m_2, m_3\}$ is removed later on.

5.7 Definition (Redundant path). Let (R, E) be a tree of removal instructions. A path p is called redundant if it is referencing both $m \in M_c$ and $\text{clash}(m)$. The set of all not redundant paths from the root to a leaf in (R, E) is denoted as $P_{(R, E)}$.

An example for a redundant path is shown in figure 5.2. We are now ready to provide the set $\mathcal{M}_c^{\text{all}} \subseteq 2^{M_c}$ of all *maximal combinations of minimal network states*

$$\mathcal{M}_c^{\text{all}} := \{M_c \setminus \{\text{state}(v) \mid v \in p\} \mid p \in P_{(R, E)}\}, \quad (5.11)$$

while a maximal combination of minimal network states is defined as follows:

5.8 Definition (Maximal combination of minimal network states). The set $M \subseteq M_c$ is called a maximal combination of minimal network states iff there exist no $m_1, m_2 \in M$ with m_1 clashing with m_2 , and the inclusion of a minimal network state m_3 leads to a clash.

In the following, we will prove that the set (5.11) contains indeed all maximal combinations of minimal network states. The following lemma provides a useful insight for paths in the tree of removal instructions:

5.9 Lemma. *Let (R, E) be a tree of removal instructions produced by algorithm 5.5 for a given set of minimal network states M_c and a map $\text{clash} : M_c \rightarrow 2^{M_c}$. For each node $m \in M_c$ with $\text{clash}(m) \neq \emptyset$, a not redundant path $(\text{root}(R, E), \dots, l)$ from the root to a leaf of (R, E) either references m or $\text{clash}(m)$.*

Proof. The path $p = (\text{root}(R, E), \dots, l)$ has to contain a node v that was a leaf when m and $\text{clash}(m)$ were appended to the tree. We investigate the successors v in p . In the first case, the successor v' of v in p is referencing m . Then the path referencing elements of $\text{clash}(m)$ was appended as a sibling node of v' . Further we know that p may not be referencing $\text{clash}(m)$ due to another appending procedure because p is not redundant.

In the second case, p is of the form $(\text{root}(R, E), \dots, v, v_1, v_2, \dots, v_n, \dots, l)$ so that it holds that $C' = \{\text{state}(v_k) \mid k \in \{1, \dots, n\}\} \subseteq \text{clash}(m)$. We know then that a node referencing m was appended as a sibling of v_1 . For each $m' \in \text{clash}(m) \setminus C'$ there

has to be a node v' on the path $(\text{root}(R, E), \dots, v)$ with $\text{state}(v') = m'$.¹ Hence we know that p is referencing $\text{clash}(m)$. Further p may not contain a node referencing m because it is not redundant. ■

5.10 Theorem. *Let (R, E) be a tree of removal instructions produced by algorithm 5.5 for a given set of minimal network states M_c and a map $\text{clash} : M_c \rightarrow 2^{M_c}$. Then a set*

$$M'_c := M_c \setminus \{\text{state}(v) \mid v \in p\},$$

with $p = (\text{root}(R, E), \dots, l)$ being a not redundant path between the root of (R, E) and a leaf l , does not contain any pair of clashing minimal network states.

Proof. Assuming that there exists a pair of clashing minimal network states inside M'_c we will prove the theorem by producing a contradiction. Let $m_1, m_2 \in M'_c$ be two clashing minimal network states. Then it holds that $m_2 \in \text{clash}(m_1)$ and $m_1 \in \text{clash}(m_2)$. Since $m_1, m_2 \in M'_c$ it follows that both m_1 and m_2 are not referenced by p .

However, $\text{clash}(m_1) \neq \emptyset$ and p is a not redundant path from the root to a leaf. Hence we know that either m_1 or $\text{clash}(m_1)$ has to be referenced by p and analogously for m_2 and $\text{clash}(m_2)$. Path p referencing m_1 contradicts our assumption, as does p referencing $\text{clash}(m_1)$ because $m_2 \in \text{clash}(m_1)$. Analog considerations lead to contradictions for p referencing m_2 or $\text{clash}(m_2)$. Thus there must not exist such a pair m_1, m_2 . ■

5.11 Theorem. *Let (R, E) be a tree of removal instructions produced by algorithm 5.5 for a given set of minimal network states M_c and a map $\text{clash} : M_c \rightarrow 2^{M_c}$. Then for a set*

$$M'_c := M_c \setminus \{\text{state}(n) \mid n \in p\},$$

with $p = (\text{root}(R, E), \dots, l)$ being a not redundant path between the root of (R, E) and a leaf l , each minimal network state $m \in M_c \setminus M'_c$ is clashing with one of the elements of M'_c .

Proof. Let $m_1 \in M_c \setminus M'_c$ be a minimal network state not included in M'_c . Assuming that there exists no $m_2 \in M'_c$ such that m_1 is clashing with m_2 we try to produce a contradiction.

If $\text{clash}(m_1) = \emptyset$ – that is, if m_1 does not clash at all – then algorithm 5.5 does not invoke the procedure *Append* for m_1 and $\text{clash}(m_1)$. Likewise there is no $m_3 \in M_c$ with $m_1 \in \text{clash}(m_3)$. This means that the tree (R, E) may not contain any node v with $\text{state}(v) = m_1$. Hence m_1 may also not be removed from M_c by the path p and $m_1 \in M'_c$. That is a contradiction to the assumption $m_1 \in M_c \setminus M'_c$.

If $\text{clash}(m_1) \neq \emptyset$, we know that either m_1 or $\text{clash}(m_1)$ is referenced by p because p is not redundant. If m_1 is referenced, then p may not be referencing $\text{clash}(m_1)$. Thus there exists a minimal network state $m_2 \in M'_c$ with $m_2 \in \text{clash}(m_1)$, leading to a contradiction. If $\text{clash}(m_1)$ is referenced by p , then m_1 may not be referenced either, so that $m_1 \in M'_c$, leading again to a contradiction. ■

The set $\mathcal{M}_c^{\text{all}}$ with all maximal combinations of minimal network states (5.11) is now used to predict protein complexes again. For each maximal combination $M'_c \in \mathcal{M}_c^{\text{all}}$ we generate the corresponding subnetwork of (P_Q, I_Q) (Definition 5.5).

¹Otherwise, m' would not have been removed by setting its index to 0 in the bit vector.

5.12 Definition (Simultaneous Protein Subnetwork). Let (P_Q, I_Q) be a possible protein network and $M'_c \in \mathcal{M}_c^{all}$ be a maximal combination of minimal network states. Then

$$(P_{M'_c}, I_{M'_c}) = S_{(P_Q, I_Q)}(P_{M'_c})$$

with $P_{M'_c} = \{p \mid p \in Nec \text{ with } (Nec, Imp) \in M'_c\}$ is called simultaneous protein subnetwork.

All proteins and interactions in $(P_{M'_c}, I_{M'_c})$ may exist simultaneously in the context of protein hypernetwork (P, I, C) because the minimal network states in M'_c do not clash with each other. In comparison to the subnetwork for the network based predicted complex (P_c, I_c) , the subnetwork $(P_{M'_c}, I_{M'_c})$ may have lost as well as gained several interactions or proteins.

We now compute the simultaneous protein subnetworks $(P_{M'_c}, I_{M'_c})$ for all $M'_c \in \mathcal{M}_c^{all}$. On each $(P_{M'_c}, I_{M'_c})$ we perform a network based complex prediction again², with the same algorithm as during the initial step (Chapter 5.3.2). From the simultaneous protein subnetwork they were predicted on, the new complexes inherit the property of being simultaneously possible.

Although each simultaneous protein subnetwork contains all simultaneously necessary interactions and proteins for the contained ones, the network based complex prediction algorithm may miss some of them. Therefore the last step adds all necessary proteins and interactions:

5.13 Definition (Refined Protein Complex). Let $(P_{M'_c}, I_{M'_c})$ be a simultaneous protein subnetwork. Let $c' \subseteq P_Q$ with the corresponding subgraph $(P_{c'}, I_{c'}) = S_{(P_{M'_c}, I_{M'_c})}(c')$ be a protein complex predicted on $(P_{M'_c}, I_{M'_c})$ by the network based complex prediction algorithm. Let

$$M_{\{p_1, p_2\}} := \{(Nec, Imp) \in M_i \mid i = \{(p_1, d_1), (p_2, d_2)\} \in I\}$$

be the set of minimal network states for an interaction $\{p_1, p_2\}$, and let

$$M_{I_{c'}} := \bigcup_{\{p_1, p_2\} \in I_{c'}} (M_{\{p_1, p_2\}} \cap M'_c)$$

be the set of minimal network states for all interactions in the complex c' .

Then the refined protein complex is

$$c_{\text{refined}} := c' \cup \{p \mid p \in Nec \cap P \text{ with } (Nec, Imp) \in M_{I_{c'}}\} \\ \cup \{p_1, p_2 \mid \{(p_1, d_1)(p_2, d_2)\} \in Nec \cap I \text{ with } (Nec, Imp) \in M_{I_{c'}}\}.$$

²Here, it should be noted that the used network based complex prediction may produce different results when it runs solely on the subnetwork. In our case, the number of iterations in the second step (Algorithm 5.2) of the LCM algorithm depends on the average density of all dense regions. Consequently, the number of iterations may differ when only investigating a subnetwork. Therefore we forced LCMA to perform the same number of iterations as in the initial network based complex prediction.

5.3.5 Implementation

The refinement of each network based predicted complex is independent from the others' refinement. Our implementation uses this for parallelization: After the minimal network states are computed, each complex refinement takes place in an independent processor thread. An internal scheduler ensures that not more than one thread for each processor core runs at the same time. This reduces the amount of occupied memory. The refinement step is the most expensive part of hypernetwork based complex prediction, so that its parallelization provides scalability to multi-core environments.

5.3.6 Discussion

The described steps transform all network based predicted complexes to refined ones. Since refined complexes do not contain simultaneously impossible interactions or proteins and further include necessary interactions and proteins, it is reasonable to assume that the prediction quality is superior to network based complex prediction.

Without the application of constraints, the protein hypernetwork based complex prediction provides the same results as the network based approach using the same complex prediction algorithm: On a not perturbed instance complexes are predicted network based, already providing the same results as the purely network based approach. For each interaction $i = \{(p_1, d_1), (p_2, d_2)\} \in I$ exactly one minimal network state

$$(Nec, Imp) = (\{i, p_1, p_2\}, \emptyset)$$

appears, since there is no constraint that can lead to additional negative or positive literals in the minimal network state formula. Thus the refinement step behaves like an identity map: For each complex exactly one simultaneous protein subnetwork is created, containing all proteins and interactions of the complex. The complex is predicted again on this network and the second refinement step does not add any protein or interaction of Nec since they are contained already.

To evaluate our approach we use the yeast protein network together with the MIPS complexes provided by the MIPS Comprehensive Yeast Genome Database (Chapter 5.1). As constraints we apply information about mutually exclusive interactions published by Jung et al. (2010). Those are translated to logic formulae as shown in Example 4.6, so that we gain the *yeast protein hypernetwork*

$$(P_{\text{yeast}}, I_{\text{yeast}}, C_{\text{yeast}}).$$

As network based complex prediction algorithm, LCMA with a threshold of $\omega = 0$ (as suggested by Li et al. (2005)) is used. The results of complex prediction on this hypernetwork are compared to the MIPS complexes with the methods described in chapter 5.1. Figure 5.3 shows the development of precision and recall values in dependency of the percentage of used constraints. For each percentage step from 1, 2, 3, up to 99 we took 50 random samples of the 458 constraints derived from Jung et al. (2010).

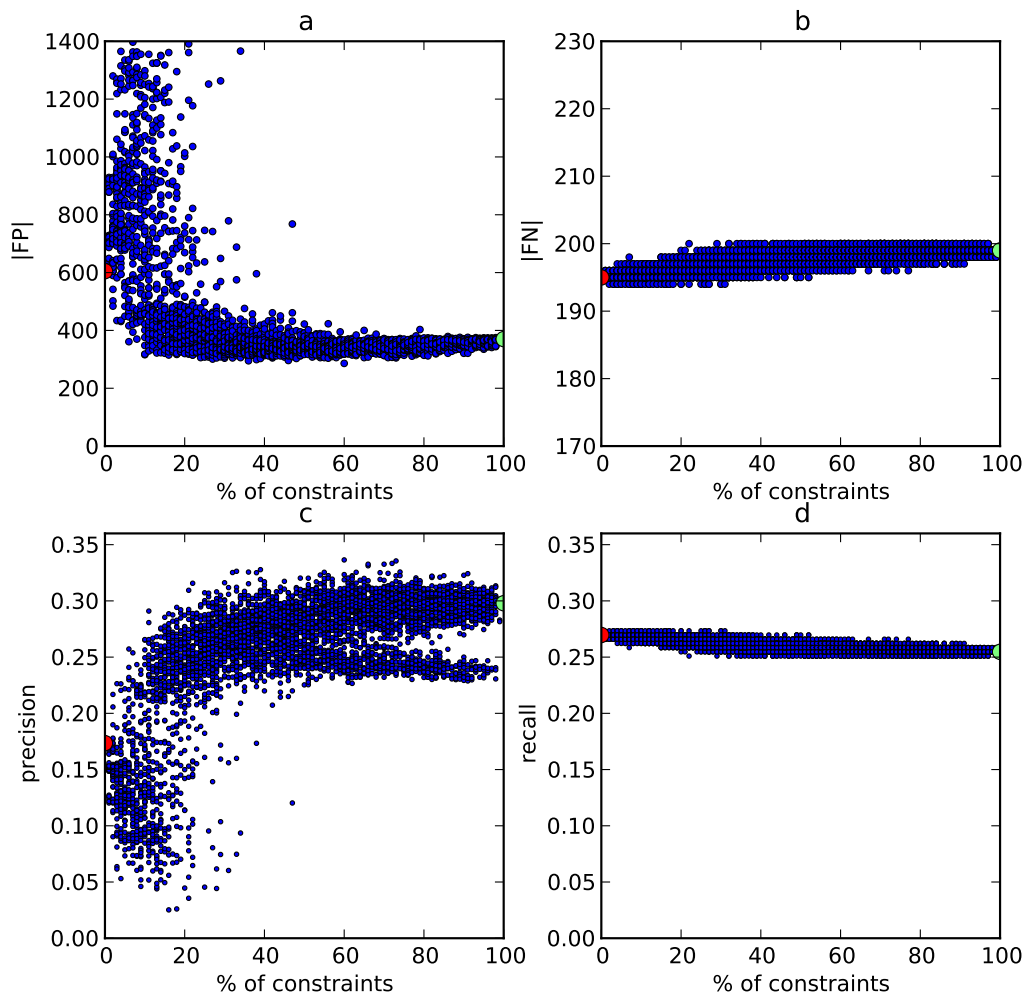


Figure 5.3: Complex prediction quality in dependence of the percentage of applied constraints. The red point indicates the case of not constrained prediction, coloured green is the case of 100% of applied constraints. (a) Number of false positive predictions plotted against percentage of applied constraints. Values above 1400 false positives were cut off. (b) Development of the amount of false negatives plotted against the percentage of applied constraints. (c) Precision of the prediction plotted against the percentage of applied constraints. (d) Recall of the prediction plotted against the percentage of applied constraints.

It can be seen that high numbers of constraints ($> 50\%$) always provide an improvement in the precision over the unconstrained instance with precision 0.17 (Figure 5.3 (c)). However the precision may initially decrease if only a few constraints are applied. This may be due to the following situation: A false positive predicted protein complex may contain two interactions that are mutually exclusive. Thus the refinement step leads to two simultaneous protein networks on which again nearly the whole original complex is predicted, but each without one of the mutual exclusive interactions (Figure 5.5). Since it contains one mutual exclusive interaction pair less, the complex is now closer to the reality, but still the constraints may not have been complete enough to turn it into a true positive prediction. In contrast to having the one false positive, refinement leads to two false positives in this case. One can easily imagine that this effect can have huge impact on prediction quality

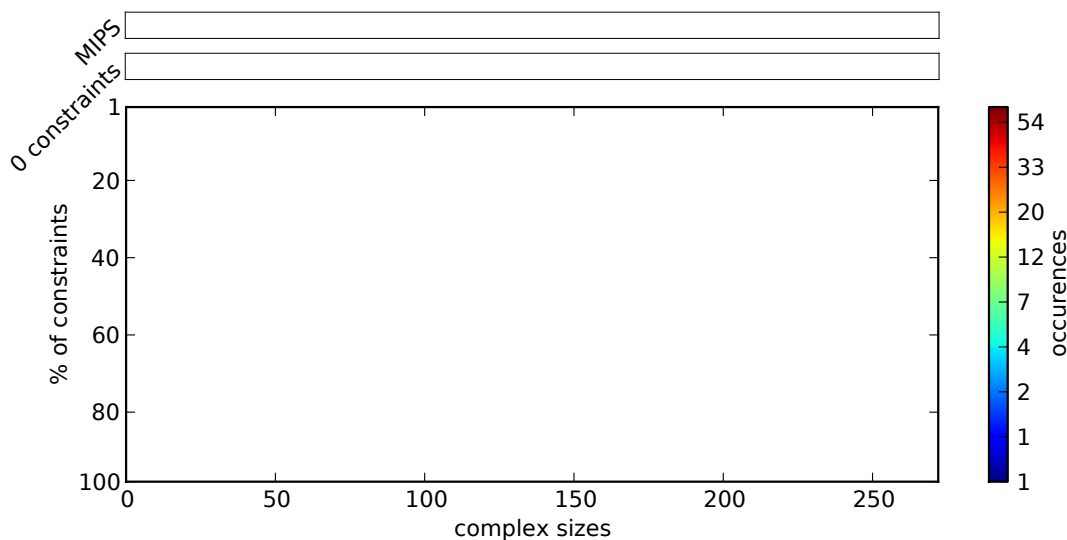


Figure 5.4: Size distribution of predicted complexes in dependence of the percentage of applied constraints. For each percentage step, the histograms’ complex sizes of the 50 samples were arithmetically averaged and plotted in one row. Blueish colors mean fewer, reddish mean more occurrences. As a control, MIPS complexes and prediction without any constraint are separately plotted at the top.

if the number of constraints is too small. Figure 5.3 (a) shows that indeed the false positive predictions are responsible for the dropping precision: Between 0 and 40% one can see an increase of false positive predictions compared to the unconstrained case, whereas all predictions above 50% exhibit a decrease. Figure 5.4 substantiates these observations by providing a view on the size distribution of predicted complexes. Compared to the MIPS complexes, the unconstrained prediction contains a number of big complexes consisting of more than 100 proteins. Applying less than 20% of the constraints results in an increased number of big complexes. This effect gradually disappears when applying more constraints until finally no complex contains more than 100 proteins.

The recall is not improved upon increasing the number of used constraints (Figure 5.3 (d)). In fact it even decreases slightly. This is due to a few MIPS complexes being no longer predicted. For example the complex

$$\{\text{YDL014W}, \text{YLR197W}, \text{YOR310C}\}$$

is contained in the purely network based prediction and the unconstrained instance as well as in the set of MIPS complexes. In the latter it is annotated as ribosomal RNA processing complex 440.12.30 (as well as “Nop56p/Nop1p complex”). However Jung et al. (2010) suggest YLR197W and YOR310C to be competing on the same binding site of YDL014W and thus the constraints

$$\{\text{YDL014W}, \text{YLR197W}\} \Rightarrow \neg\{\text{YDL014W}, \text{YOR310C}\}$$

and

$$\{\text{YDL014W}, \text{YOR310C}\} \Rightarrow \neg\{\text{YDL014W}, \text{YLR197W}\}.$$

This implies – under the assumption that protein complexes are dense – that the complex is not likely in reality since only a chain of YLR197W, YOR310C and

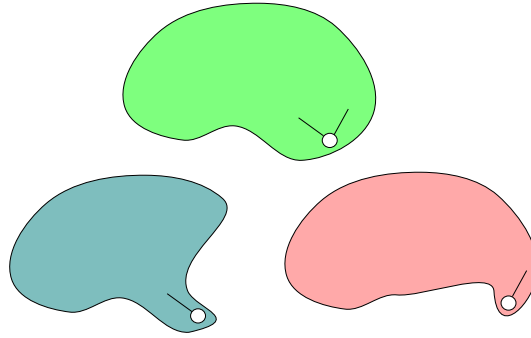


Figure 5.5: Duplication of false positive protein complexes upon refinement with only few constraints. Shown are three predicted complexes (green, blue, red). The two lower ones (blue, red) are resulting from the upper (green), original complex after the refinement step. The original complex contains two mutual exclusive interaction (in the lower right). The refined complexes each contain one of the two interactions. Assuming that the original complex is a false positive prediction, the situation can occur that the refined complexes are still false positive predictions if there are not enough constraints. Application of further constraints eliminates these false positives.

YDL014W remains. This situation either shows a shortcoming of the density assumption or an error in the used data.

Above analysis shows that the prediction of protein complexes can be enhanced significantly by using protein hypernetworks. For now it is limited by the incompleteness of available data, both regarding benchmarks (MIPS complexes) and constraints (information on mutual exclusive interactions from Jung et al. (2010)). Further, a fair amount of the MIPS complexes contains only one (54) or two (64) proteins, which cannot at all be predicted by the LCM algorithm (excluding these already raises the recall to 0.35 compared to 0.29 in case of a prediction with all constraints applied). Since logic constraints allow the modelling of scaffold dependency, the prediction quality might be further enhanced once the data is available. Moreover, some of the current false positives might be real complexes that are neither experimentally determined nor included in the MIPS database, as Li et al. (2005) state.

6 Protein Hypernetwork Analysis

Based on the prediction of protein complexes, we propose two new methods of analysis. First, the prediction of master switches shows a new way to rate proteins or interactions, estimating their expected functional importance. Secondly, the simulation of perturbation effects on predicted protein complexes is used to predict functional similarities between proteins or interactions.

6.1 Prediction of Master Switches

It has been shown above that we can predict likely protein complexes upon the perturbation of a protein or interaction. A big difference between the perturbed and the *native instance*¹ may be a hint for the functional importance of a protein or interaction. Therefore, in the following we aim to find a measure for this. We call a protein or interaction with a big perturbation effect a *master switch*. The desired measure that estimates this importance will be called *master switch score*.

As the above approach for the prediction of protein complexes is already capable of handling perturbations, the most straightforward way is to compare the predicted complexes of the native instance to those of the perturbation. For a protein hypernetwork (P, I, C) let $\mathcal{C} \subseteq 2^P$ be the set of predicted complexes for the native instance and $\mathcal{C}_{q\downarrow} \subseteq 2^P$ be the set of predicted complexes upon the perturbation of $q \in P \cup I$ (thus either $P_{\downarrow} = \{q\}$ or $I_{\downarrow} = \{q\}$). A suitable measure for the difference between the two sets is a map $\mu : 2^P \times 2^P \rightarrow [0, 1]$ with a result of $\mu(\mathcal{C}, \mathcal{C}_{q\downarrow}) = 0$ for $\mathcal{C} = \mathcal{C}_{q\downarrow}$ and a result $\mu(\mathcal{C}, \mathcal{C}_{q\downarrow}) = 1$ for $\mathcal{C} \cap \mathcal{C}_{q\downarrow} = \emptyset$. Using the overlapping score (Definition 5.3), we obtain these properties by setting

$$\mu_{OS}(\mathcal{C}, \mathcal{C}_{q\downarrow}) := 1 - OS(\mathcal{C}, \mathcal{C}_{q\downarrow}). \quad (6.1)$$

The well established *Jaccard index*, defined as $Jacc(A, B) := \frac{|A \cap B|}{|A \cup B|}$, produces a similar behaviour with

$$\mu_{Jacc}(\mathcal{C}, \mathcal{C}_{q\downarrow}) := 1 - Jacc(\mathcal{C}, \mathcal{C}_{q\downarrow}). \quad (6.2)$$

Lastly the third possibility is constituted by

$$\mu_{OS'}(\mathcal{C}, \mathcal{C}_{q\downarrow}) := 1 - OS'(\mathcal{C}, \mathcal{C}_{q\downarrow}) \quad (6.3)$$

with $OS'(A, B) := \frac{2|A \cap B|}{|A| + |B|}$.

Figure 6.1 shows the different behaviours of the measures as a function of the size of the intersection between \mathcal{C} and $\mathcal{C}_{q\downarrow}$. μ_{Jacc} rewards big differences – or small intersections – stronger than μ_{OS} , whereas $\mu_{OS'}$ provides a linear behaviour. Having the different possibilities in mind, the master switch score can now be defined.

¹In the native instance of predicted complexes, no perturbation has been applied ($P_{\downarrow} = I_{\downarrow} = \emptyset$).

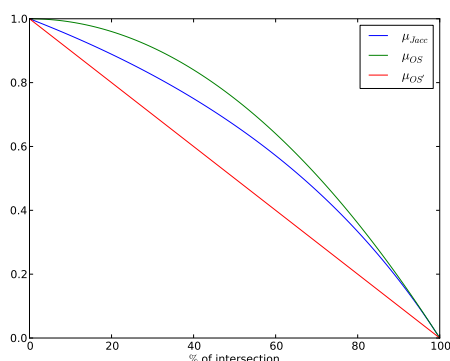


Figure 6.1: Measures for the difference between sets of predicted protein complexes. The horizontal axis shows the percentage of intersection $\mathcal{C} \cap \mathcal{C}_{q\downarrow}$, the vertical axis shows the corresponding output of each measure $\mu_{Jacc}(\mathcal{C}, \mathcal{C}_{q\downarrow})$, $\mu_{OS}(\mathcal{C}, \mathcal{C}_{q\downarrow})$ and $\mu_{OS'}(\mathcal{C}, \mathcal{C}_{q\downarrow})$.

6.1 Definition (Master Switch Score). Let (P, I, C) be a protein hypernetwork. The master switch score $MSS_{(P,I,C)} : P \cup I \rightarrow [0, 1]$ is defined as

$$MSS_{(P,I,C)}(q) := \mu(\mathcal{C}, \mathcal{C}_{q\downarrow})$$

with the predicted protein complexes $\mathcal{C} \subseteq 2^P$ and $\mathcal{C}_{q\downarrow} \subseteq 2^P$ for the native instance and the perturbation of $q \in P \cup I$.

Figure 6.2 shows the relationship between the master switch scores for the yeast protein hypernetwork $(P_{\text{yeast}}, I_{\text{yeast}}, C_{\text{yeast}})$. It can be seen that μ_{Jacc} and μ_{OS} are resulting in mostly the same values whereas $\mu_{OS'}$ is roughly producing a shift to lower values. Thus it was decided to use μ_{Jacc} as it is based on the most established measure.

For a protein hypernetwork (P, I, C) , master switch scores for each protein and interaction can now be computed in the following way:

1. Prediction of protein complexes \mathcal{C} for the native instance.
2. Perturbation of $q \in P \cup I$.
3. Prediction of protein complexes $\mathcal{C}_{q\downarrow}$ for the perturbation.
4. Calculate the master switch score $MSS_{(P,I,C)}(q)$.
5. Continue at step 2 with the next $q \in P \cup I$.

Implementation

The prediction of protein complexes is an expensive operation. Therefore it is important to avoid redundancy while iterating over all proteins and interactions in order to calculate their master switch scores. Recalling the steps of complex prediction on protein hypernetworks (Chapter 5.3) it can be seen that the refinement of predicted complexes (Step 4) may be partially redundant: For the perturbation of $q \in P \cup I$, as long as a network based predicted complex c is also predicted for the

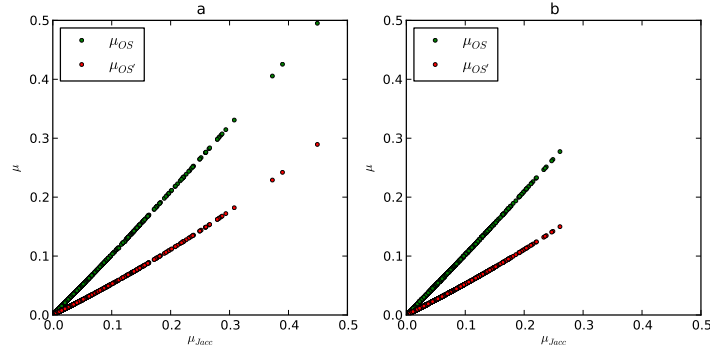


Figure 6.2: Influence of different measures on master switch scores for the yeast protein hypernetwork $(P_{\text{yeast}}, I_{\text{yeast}}, C_{\text{yeast}})$. (a) $MSS_{(P_{\text{yeast}}, I_{\text{yeast}}, C_{\text{yeast}})}$ values using μ_{Jacc} plotted against those using μ_{OS} (green) and $\mu_{OS'}$ (red) for all proteins $p \in P_{\text{yeast}}$. (b) The same for all interactions $i \in I_{\text{yeast}}$.

native instance and the minimal network states for contained interactions did not change upon perturbation of q , it is sure that the refined complex c_{refined} is the same for both the native instance and the perturbation. Thus only complexes for which this is not the case are refined. For others the results from the native instance are taken.

Since computation of the master switch score for a protein or interaction is not at all dependent on the computation for any other protein or interaction, the process can be parallelized easily. The implementation provides two mechanisms for parallelization: On the one hand, master switch prediction can be limited to a subset of proteins or interactions, so that several instances of the software can independently predict master switches on different computers (for example in a cluster). On the other hand, each master switch prediction has its own processor thread, so that all cores of a computer can be used in parallel. Again, the internal scheduler ensures that each processor core is only occupied by one thread to reduce memory consumption. The complex prediction which is part of master switch prediction is limited to one thread in this case.

Discussion

For a subset of the yeast protein network, Jeong et al. (2001) show that the connectivity of a protein is already a hint for its functional importance: The lethality of a perturbation was compared to the degree of the protein in the network. In the investigated yeast network, 93% of the proteins had a degree of less than 6, but only 21% showed a lethal effect upon perturbation. In contrast, among the proteins with a degree of more than 15 – only 0.7% of the proteins – 63% had a lethal effect upon perturbation. The authors infer from this that highly connected proteins are three times more likely to be lethal – and thus functional important – than others. In the following we will investigate how these results compare to master switch scores. Therefore we predicted master switches on the yeast protein hypernetwork $(P_{\text{yeast}}, I_{\text{yeast}}, C_{\text{yeast}})$ and related the master switch for each protein or interaction to its connectivity.

6 Protein Hypernetwork Analysis

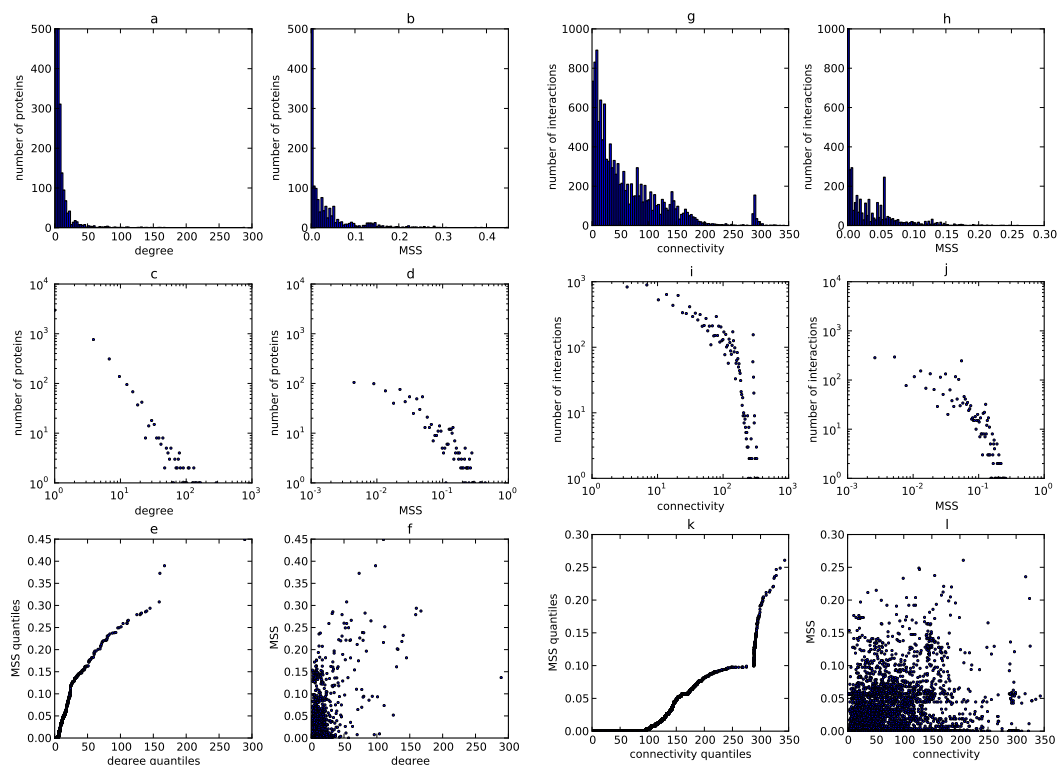


Figure 6.3: Master switch prediction for the yeast protein hypernetwork ($P_{\text{yeast}}, I_{\text{yeast}}, C_{\text{yeast}}$). (a) Distribution of protein connectivity (node degree). (b) Distribution of master switch scores. Distribution of protein connectivity (c) and of master switch scores (d), both in logarithmic scaled plots. (e) shows a quantile quantile plot of protein connectivity against master switch scores. (f) plots the connectivity of each protein against its master switch score. Analogously, (g) to (l) show the same for interactions. In (a) and (b), values above 500, in (h) values above 1000 were cut off.

Investigating the degree of proteins in the yeast protein network one can observe a power law distribution as can be seen by the linear distribution of points in Figure 6.3 (c). A quantile-quantile plot (Figure 6.3 (e)) shows the difference between the distribution of master switch scores and protein connectivity. We measure the connectivity of interactions by summing the degrees of its interacting proteins in the network. For an interaction $\{p_1, p_2\}$ the connectivity is defined as

$$\text{deg}(p_1) + \text{deg}(p_2) - 2.$$

Here, the subtraction of 2 ensures that the interaction itself is not counted. The comparison of the distribution of interaction connectivity to the distribution of master switches again suggests differences. Although the master switch score of many interactions corresponds to their connectivity, Figure 6.3 (l) shows interactions with high connectivity but low master switch score and vice versa. Again, the quantile-quantile plot (Figure 6.3 (k)) gives evidence for different distributions of master switch scores and connectivity.

Altogether the discussion shows that the master switch score provides additional information on the functional importance of proteins and interactions compared to

only looking at their connectivity. Since Protein complexes execute cellular functions (Chapter 1), the measurement of differences in the predicted protein complexes after perturbation should give a good hint on the impact of a perturbation and thus on the functional importance of the perturbed protein or interaction. Both strength and weakness of the master switch approach is the fact that it is dependent of the protein complex prediction quality: On the one hand, complex prediction still is imperfect and most of the predicted protein complexes cannot be verified due to incompleteness of databases. On the other hand, future developments in complex prediction algorithms can directly improve the measure.

6.2 Prediction of Functional Similarity

Proteins that are involved in the same cellular function are likely to be tightly interconnected or even cooperate in a complex for the execution of the function (Chapter 1). Therefore they are expected to have similar effects on the protein complexes upon perturbation. With this rationale, we evaluate if we can predict functional classes of proteins by comparing the perturbation effects in a protein hypernetwork (P, I, C) :

1. Calculate master switch score for all proteins $p \in P$.
2. Investigate the set $P_{MS} \subseteq P$ of all proteins p that exhibit a master switch score greater than zero ($MSS_{(P,I,C)}(p) > 0$).
3. For each pair $p_1, p_2 \in P_{MS}$ calculate a similarity based on the predicted protein complexes upon its perturbation.
4. Cluster the proteins in P_{MS} according to their similarity.

Two proteins that appear in the same cluster are then assumed to have similar effects upon perturbation, and thus execute the same function in the cell metabolism. We focus here on proteins with a master switch score greater than zero, because others do not have any perturbation effect, and thus no differences in functionality would be observable. First we define the *similarity of perturbation effects* for proteins.

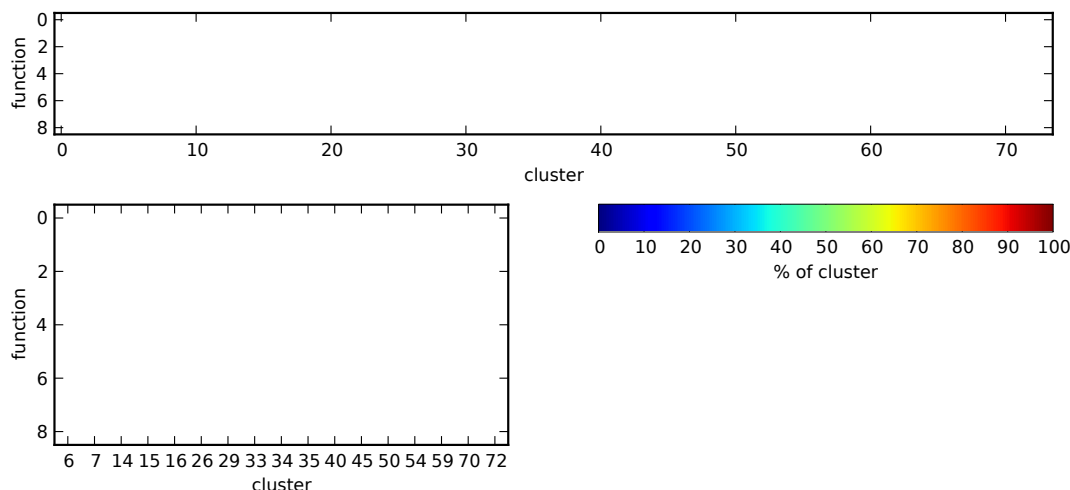
6.2 Definition (Similarity of Perturbation Effects). Let (P, I, C) be a protein hypernetwork. For two proteins $p_1, p_2 \in P$ with a master switch score greater than zero ($MSS_{(P,I,C)}(p_k) > 0$), the similarity of perturbation effects is defined as

$$-\log_{10}(\mu_{Jacc}(\mathcal{C}_{p_1\downarrow}, \mathcal{C}_{p_2\downarrow}))$$

with $\mathcal{C}_{p_k\downarrow} \subseteq 2^P$ being the set of predicted protein complexes upon perturbation of p_k for $k \in \{1, 2\}$.

We use transitivity clustering (Wittkop et al., 2010) to obtain a set of disjoint protein clusters $\Gamma \subseteq 2^{P_{MS}}$. Above arguments about functional similarity due to tight interconnection can also be applied for interactions. Since – in our system – both proteins and interactions can be treated equally regarding perturbation and master switch scores, the same procedure can be performed for interactions.

6 Protein Hypernetwork Analysis



| metabolism functions | | clusters | |
|----------------------|---|----------|------|
| index | function | index | size |
| 0 | Cell cycle | 0 – 54 | 2 |
| 1 | Cell polarity and structure | 55 – 63 | 3 |
| 2 | Intermediate and energy metabolism | 64 – 68 | 4 |
| 3 | Membrane biogenesis and traffic | 69 – 71 | 5 |
| 4 | Protein / RNA transport | 72 | 7 |
| 5 | Protein synthesis and turnover | 73 | 274 |
| 6 | RNA metabolism | | |
| 7 | Signalling | | |
| 8 | Transcription / DNA maintenance / chromatin structure | | |

Figure 6.4: Matching of metabolism functions to protein clusters. The considered cellular functions are listed in the left table (based on Gavin et al. (2002)). For each function, the set of proteins known to take part in its execution was collected from the data provided by Gavin et al. (2002). Proteins were clustered by the similarity of predicted complexes upon their perturbation. For each cluster the percentages of proteins executing the provided functions were calculated. A dark red square in one of the two images means that 100% of the clusters proteins is known to execute the function, whereas dark blue corresponds to 0%. The second image excludes all clusters in which not all proteins are annotated. Clusters are ordered ascending in their size, as shown in the right table.

Discussion

When performing above prediction method for the proteins of the yeast protein hypernetwork $(P_{\text{yeast}}, I_{\text{yeast}}, C_{\text{yeast}})$, a threshold of 1.8 for transitivity clustering turns out to be the best compromise between cluster sizes and number of clusters². In order to judge over the significance of the obtained clusters with respect to the cellular functions of contained proteins, we compare them to functional annotations of yeast proteins published by Gavin et al. (2002). From these data we extract for each function $f \in \{0, \dots, 8\}$ (Figure 6.4, left table) a set of proteins $P_f \subseteq P_{MS}$ executing it. The resulting function sets P_f are not disjoint, there are proteins which obviously can execute several metabolism functions (Figure 6.5). The comparison

²In the appendix, results for a different threshold show that the findings provided here are not unique to this threshold.

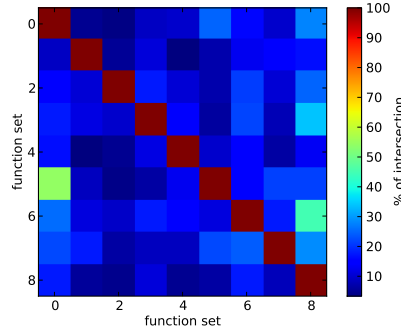


Figure 6.5: Overlap between function sets. In field (f, f') with $f \in \{0, \dots, 8\}$ the percentage $\frac{|P_f \cap P_{f'}|}{|P_{f'}|} \cdot 100$ of P_f contained in $P_{f'}$ is shown. The mean excluding the diagonal is 20.38%.

between predicted clusters and metabolism functions is shown in Figure 6.4. Only clusters containing more than 1 protein are considered.

The first image of Figure 6.4 shows the percentage of proteins of each cluster $\gamma \in \Gamma$ contained in each of the functional sets P_f :

$$\frac{|\gamma \cap P_f|}{|\gamma|} \cdot 100.$$

Because Gavin et al. (2002) do not provide an annotation for each protein in P_{MS} , we now only investigate the set of clusters $\Gamma' = \{\gamma \mid \gamma \in \Gamma : \gamma \subseteq \bigcup_{f=0}^8 P_f\}$ for which all proteins are annotated (Figure 6.4, second image). As can be seen, among the 17 clusters of Γ' , 14 are entirely dedicated to at least one function. We count these as true positive predictions $\Gamma'_{TP} \subseteq \Gamma'$, resulting in a precision (Definition 5.4) of

$$\frac{|\Gamma'_{TP}|}{|\Gamma'|} = \frac{14}{17} = 0.82. \quad (6.4)$$

The probability for a completely annotated cluster $\gamma \in \Gamma'$ to appear entirely at least in one of the function sets P_f by chance can be estimated by

$$\mathbb{P}(\gamma \subseteq P_f) := \left(\frac{|P_f|}{|\bigcup_{f=0}^8 P_f|} \right)^{|\gamma|}$$

when considering clusters of size $|\gamma| \leq 7$. We can assume that the function sets are disjoint and estimate the probability for cluster γ to appear entirely in one of the function sets P_f as

$$\mathbb{P}(\exists f \in \{0, \dots, 8\} : \gamma \subseteq P_f) \leq \sum_{f=0}^8 \mathbb{P}(\gamma \subseteq P_f).$$

This is reasonable because the overlapping of the function sets is mostly around 20% (Figure 6.5). Figure 6.6 shows that the probabilities for a cluster to be entirely dedicated one of the functions by coincidence are at 0.3 for clusters with two proteins

6 Protein Hypernetwork Analysis

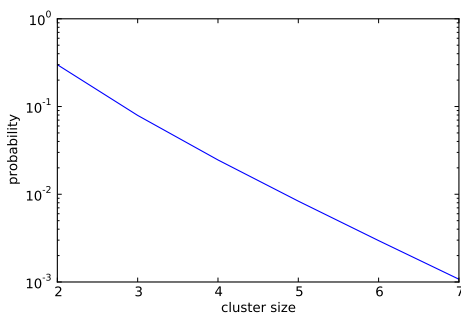


Figure 6.6: Probability for clusters of sizes from 2 to 7 entirely execute one metabolism function. The vertical axis is logarithmic.

and decreasing rapidly with the size of the cluster. Now, the estimation of the probability to obtain an at least as good precision as above (6.4) by chance is

$$\prod_{\gamma \in \Gamma'_{TP}} \mathbb{P}(\exists f \in \{0, \dots, 8\} : \gamma \in P_f) \leq 4.479 \cdot 10^{-11},$$

so that it can be assumed that the observed prediction quality is no coincidence.

Since the prediction is based on the comparison of perturbation effects on predicted complexes, the clusters might resemble protein complexes. Spirin and Mirny (2003) show protein complexes to be entirely dedicated to one function. It can be assumed that this effect fades with the application of constraints. This is because a perturbation can start a cascade of effects which are propagated to other parts of the network through the constraints. In our case, among the 74 clusters, only 8 have an overlapping score (Definition 5.3) of more than 0.5 to one of the underlying complexes. Accordingly application of constraints already decoupled functional clusters from protein complexes.

One can conclude that perturbations simulated with protein hypernetworks provide significant evidence for functional similarities: two proteins which have a similar perturbation effect are likely to execute the same function in cell metabolism. Regarding interactions, similar results are likely, however functional annotations are not easily obtainable at the moment.

7 Conclusion and Outlook

The goal of this thesis was the propagation of interaction logic toward protein hypernetworks – a model for protein networks that provides information of higher dimensionality in order to improve the results of conventional computational analysis as well as giving rise to new approaches. This was grounded by the idea, that dependencies between interactions contain information valuable for the inference of cellular functions in protein networks. It was shown that constraints using propositional logic are suitable to model the interaction logic in terms of scaffold dependency and competition on the same binding domain. Nevertheless, it was decided to use modal logic as a superset of propositional logic. This way it was possible to combine the propositional logic constraints in a single modal logic formula for the prediction of perturbation effects in the whole network. Further, the decision was taken with the flexibility regarding future extensions to probabilistic approaches in mind. Prediction of perturbation effects also illustrates the overall idea used in this thesis: A certain behaviour or situation is modeled with a (modal) logic formula in a way that allows a solution – in other words, a Kripke model – to directly contain the desired information. Using this approach, it was possible to compute the mentioned perturbation effects, as well as minimal network states for an interaction. The central tool for the work was the modal logic tableau algorithm. Apart from its common purpose to prove the satisfiability of a modal logic formula, it was used to retrieve the desired Kripke model, which raised the need for a modification (Chapter 3.2.3).

Protein hypernetworks turned out to be useful for the computational analysis of protein networks. Regarding the prediction of protein complexes, the usage of interaction logic in the form of constraints raised the quality by increasing the precision. Despite the general improvements, the application of only a few constraints for mutual exclusive interactions turned out to duplicate false positive predictions and therefore decreased prediction quality. Thus hypernetwork based complex prediction needs a sufficiently high number of constraints in order to work correctly. The method utilizes an arbitrary conventional network based complex prediction algorithm, here the LCM algorithm was selected. Since the approach is based on multiple invocations of the network based complex prediction for simultaneous protein subnetworks, speed was focused when implementing the LCM algorithm. Therefore the clique merging was implemented using an approximation for the number of edges in the merged region. The possibility to replace the network based complex prediction algorithm makes protein hypernetwork based complex prediction open to future improvements. Furthermore, since it is possible to simulate perturbation effects with protein hypernetworks those can be propagated to predicted complexes, allowing new methods of analysis like the master switch score and the prediction of functional similarity.

As an estimation of functional importance of proteins and interactions, the master switch score was proposed. Ideally, a protein or interaction changing all complexes

7 Conclusion and Outlook

in a network upon its perturbation is a master switch of maximum functional importance, whereas a protein or interaction the perturbation of which has no influence on protein complexes at all, is rather unimportant for the function of the network. Relying on protein complexes for this estimation seemed reasonable because these execute important functions in a network (Alberts et al., 2007). Master switch prediction makes use of the already existing implementation of protein complex prediction, so that it will directly benefit from improvements in this area.

Lastly the thesis provides a method to predict functional similarity of proteins and interactions. It relies on the clustering of proteins or interactions based on the similarity of their perturbation effects on predicted protein complexes. An evaluation showed that this method is already reliable, and it will again directly benefit from improvements in protein complex prediction.

The analyses implemented so far can be subjects of improvements without changing the basic ideas of protein hypernetworks: Network based protein complex prediction can be replaced by new developments in this field. Candidates are the SPICi algorithm (Jiang and Singh, 2010) which promises to be of comparable quality but faster than MCODE, and DECAFF (Li, Foo, and Ng, 2007) which combines the LCM approach with rating the reliability of interactions. Furthermore, Brohee and Helden (2006) show that using the Markov Cluster Algorithm might be a superior alternative to those specialized complex prediction algorithms. This motivates to generally review the usage of clustering algorithms for protein complex prediction and evaluate newer developments like transitivity clustering (Wittkop et al., 2010). Moreover, the prediction of functional similarities could be automated by the incorporation of transitivity clustering into the implementation.

Apart from those presented here, further analysis methods based on protein hypernetworks are thinkable:

In the literature, it is assumed, that protein networks consist of several independent functional modules that communicate with each other (Hartwell et al., 1999). Thereby each module works like a blackbox that, upon a certain input, generates an output that is received and processed from an associated module. The representation of modules in a protein network is assumed to be dense and only sparsely connected to its environment. This definition already shows a similarity to protein complexes. Spirin and Mirny (2003) state that distinguishing between complexes and modules is important because of their different biological meanings:

A protein complex is a molecular machine that consists of several proteins (nucleic acids and other molecules) that bind each other at the same place and time [...]. On the contrary, a functional module consists of a few proteins (and other molecules) that control or perform a particular cellular function through interactions between themselves. These proteins do not necessarily interact at the same time and place, or form a macromolecular complex [...]. (Spirin and Mirny, 2003)

Spirin and Mirny (2003) mention that making the distinction is hard in many cases because temporal and spatial information is not available. Now protein hypernetworks at least give the temporal information in the form of the possibility of simultaneous interaction or protein occurrences. Consequently, dense regions the

interactions of which do not appear simultaneously could give hints where to search for functional modules. Such regions could be found by performing a network based complex prediction to find all dense regions, and afterwards filtering out all regions that do not contain any pair of clashing minimal network states. A fast test for the latter is provided in the proof of Theorem 4.14: the satisfiability of the defined formula ξ can be checked with the tableau algorithm again.

For protein networks it can be observed that small subgraphs that represent certain patterns of interacting proteins – network motifs – occur far more often than in a random network. Some motifs like the *feed forward loop* are common to even non biological networks and suspected to be their basic building blocks (Milo et al., 2002). However Ingram, Stumpf, and Stark (2006) warn that the functional properties of motifs are not clearly determined by their pattern, thus limiting the insight provided by motif search. With further information about logic dependencies between interactions available, it might be useful to search for *logic motifs*: For a protein hypernetwork (P, I, C) a logic motif could be an arbitrary propositional logic formula $m \in \mathfrak{Prop}(P \cup I)$. The containment of such a motif in the protein hypernetwork could be determined by proving that

$$\left(\bigwedge_{c \in C} c \right) \Rightarrow m$$

holds. Here, again the tableau algorithm could be used to prove that $\bigwedge_{c \in C} c \wedge \neg m$ is unsatisfiable. Using wildcard propositions, logic motifs could describe logic relationships between arbitrary instead of specific proteins and interactions. In order to measure the significance of a motif, its occurrences in the hypernetwork would have to be compared to its occurrences in a suitable null model. Network generators for creating random instances to simulate scale-free or modular behaviour, which most biological networks exhibit (Guimera and Sales-Pardo, 2010; Jeong et al., 2001), are already available – like an approach based on stochastic block models by Guimera and Sales-Pardo (2010) – and used for conventional motif search. In order to gain a null model for logic constraints, common rules have to be found that underly their distribution.

When further biological information about interaction dependencies becomes available, it might be useful to assign probabilities to proteins and interactions. A straight forward interpretation of probabilities would be the expected concentration of each protein and interaction in a hypothetic volume. Here the flexibility of modal logic becomes obvious: The two modal operators \diamond and \square can be replaced by operators Δ_ρ annotated with a probability $\rho \in [0, 1]$ (Doberkat, 2009). Semantic is now captured by a *stochastic Kripke model* (S, K, \Vdash) with $K : S \rightarrow (2^S \rightarrow [0, 1])$ being a stochastic relation. Concerning the new modal operators Δ_ρ ,

$$s \Vdash \Delta_\rho \phi$$

7 Conclusion and Outlook

holds iff $K(s)(\llbracket \phi \rrbracket_{(S,K,\Vdash)}) \geq \rho$. In this context, $\llbracket \phi \rrbracket_{(S,K,\Vdash)}$ is the extension of formula ϕ – the set of satisfying states – analogously to Definition 3.3.¹ Moreover, we can define operators $\Delta_{=\rho}$ with $\Delta_{=\rho}\phi = \Delta_{\rho}\phi \wedge \Delta_{1-\rho}\neg\phi$ as a shortcut, so that

$$s \Vdash \Delta_{=\rho}\phi$$

holds iff $K(s)(\llbracket \phi \rrbracket_{(S,K,\Vdash)}) = \rho$. A *stochastic protein hypernetwork* (P, I, C, \mathbb{P}) can then be provided by the addition of a map $\mathbb{P} : P \cup I \rightarrow [0, 1]$ associating an a priori probability to each protein and interaction. These probabilities can be integrated into the logic formulae by using $\Delta_{=\mathbb{P}(q)}q$ as a statement about the a priori probability of $q \in P \cup I$ instead of only the proposition q as a statement about the possibility of q . Constraints could make use of further probability assumptions: For example $\neg\Delta_1\neg i_1 \Rightarrow \Delta_{0.25}i_2$ would mean that for interaction $i_1 \in I$ to have a probability greater than zero, interaction $i_2 \in I$ is required to have a probability of at least 0.25. In a conjunction

$$\Delta_{=0.5}i_1 \Delta_{=0.1}i_2 \wedge (\neg\Delta_1\neg i_1 \Rightarrow \Delta_{0.25}i_2)$$

with an a priori probability of 0.5 for i_1 and 0.1 for i_2 , $\Delta_{0.25}i_2$ would lead to a clash in a state $s \in S$ because of

$$K(s)(\llbracket i_2 \rrbracket_{(S,K,\Vdash)}) = 0.1 < 0.25.$$

Thus the formula would be not satisfiable. In contrast, using an a priori probability of for example 0.5 for i_2 would lead to satisfiability of the formula. This gives rise to an adapted perturbation formula (Definition 4.8)

$$\bigwedge_{q \in P \cup I} \left(\left(\Delta_{=\mathbb{P}(q)}q \wedge \bigwedge_{c \in C(q)} c \right) \vee \Delta_1\neg q \right),$$

provided that all constraints are formulated in the above way. The interpretation is straight forward: each protein or interaction $q \in P \cup I$ has its a priori probability $\mathbb{P}(q)$ if all of its constraints are satisfiable. If not, it has a probability of 0 instead. A minimal network state formula (Definition 4.10) would analogously translate to

$$\Delta_{=\mathbb{P}(i)}i \wedge \bigwedge_{c \in C} c,$$

and a minimal network state for an interaction i could consist of a set containing all proteins and interactions with a probability necessarily greater than zero, and a set of proteins and interactions that have a probability of zero in the solution of the formula – in other words, that are impossible to appear simultaneously with i .

In due course when data about interaction logic becomes widely available, protein hypernetworks provide a way to effectively represent this information. In turn they provide a platform for further analysis, including an improved prediction of protein complexes as well as new approaches like the prediction of master switches and functional similarity. Lastly, the extension to a probabilistic approach might even allow for the incorporation of more detailed biological knowledge in the future.

¹It is possible to translate conventional modal operators to this logic: A necessity statement $\Box\phi$ translates to $\Delta_1\phi$. A possibility statement $\Diamond\phi$ means that it is not impossible to reach a state which satisfies ϕ , and the translation $\neg\Delta_1\neg\phi$ is obviously consistent with Lemma 3.4.

Further Informations

The informations provided here are complemented by an appended DVD which contains additional data. In the folder `tests`, the DVD contains the raw data for all performed tests, as well as the python scripts used for analysis and plotting. Furthermore, it contains a reference implementation of the methods developed in this thesis, which is described in the following.

Implementation

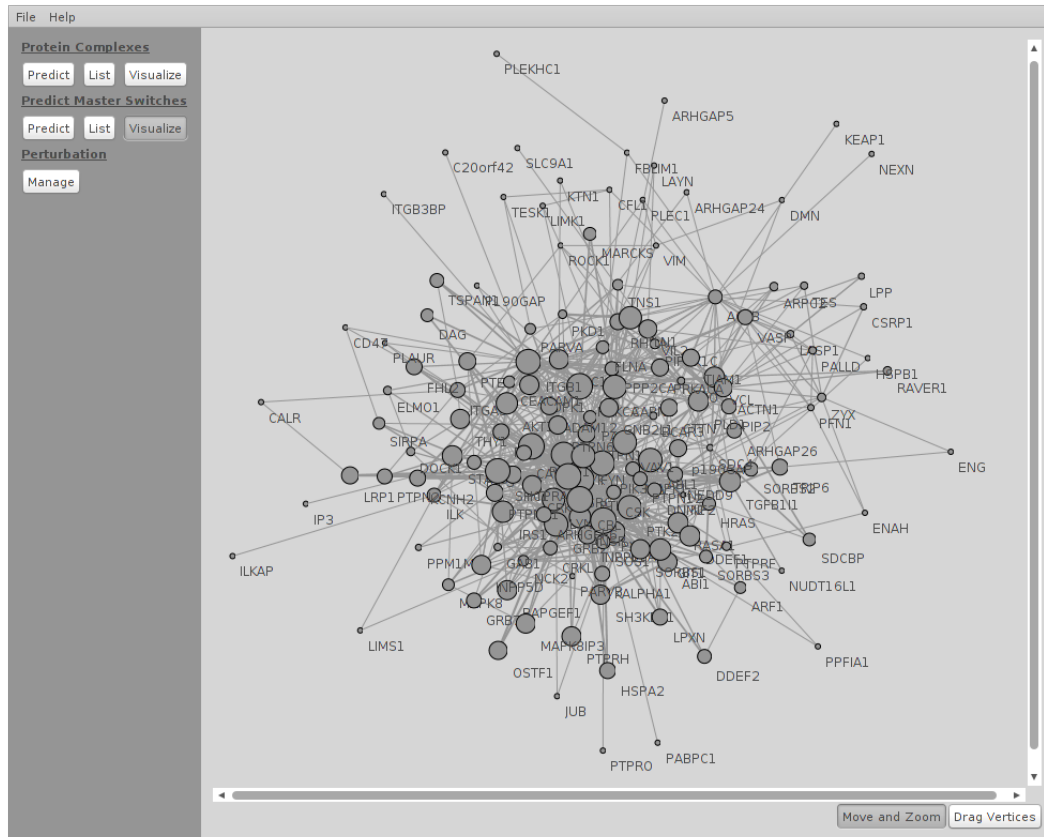
As part of the thesis, the presented protein hypernetwork approach was implemented in a collection of cross-platform JavaTM libraries and corresponding frontends, which are published under the *BSD License*. For implementation, apart from the JavaTM Class Libraries, the *JUNG Java Universal Network/Graph Framework* from the University of California (jung.sourceforge.net) and the *Commons Collections Generic* from Larva Labs (larvalabs.com/collections) were used. The sources of this implementation are found on the DVD in the folder `tool`.

For storage of protein hypernetwork definitions, *PHXML* an *XML* based format for proteins, interactions and logic constraints was developed. The document type definition is provided on the DVD under `tool/PHXML.dtd`.

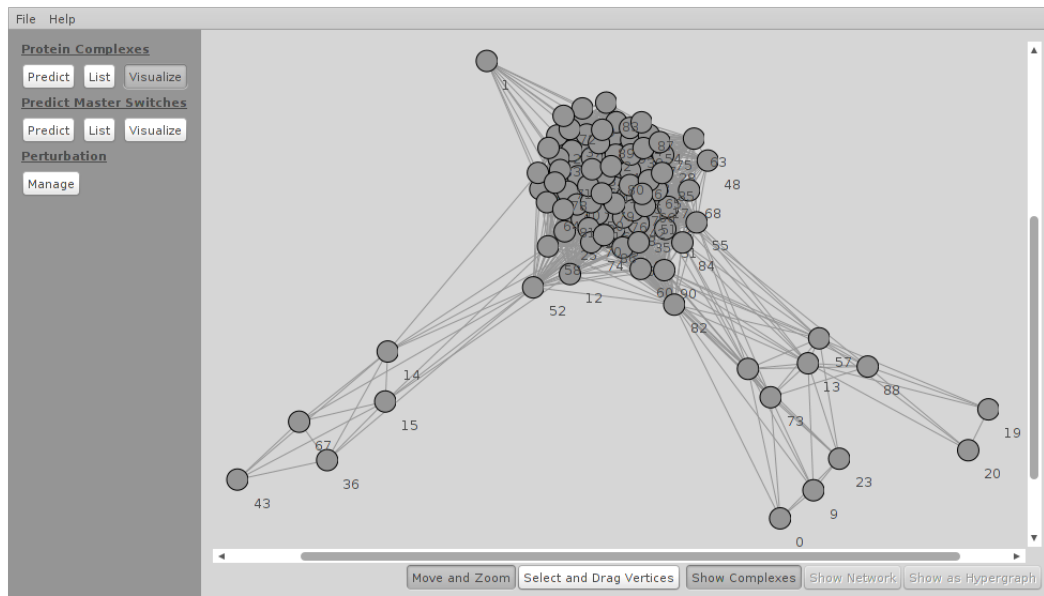
Modal logic together with the tableau algorithm is provided by the library *LibModalLogic2*. The library *LibProteinHypernetwork2* provides an implementation of protein hypernetworks, as well as facilities to save and load from the PHXML format, and the possibility to import protein networks defined as *simple interaction format* (.sif) files from *cytoscape* (cytoscape.org) or *PSIMAP* (psimap.com) files. An implementation of the methods presented in Chapter 4.2, Chapter 5 and 6.1 is provided by *LibModalLogicProteinHypernetwork*. *ProteinHypernetworkEditor* is a graphical user interface frontend to *LibProteinHypernetwork2* that allows the editing and definition of protein hypernetworks, loading from and saving to the PHXML format, as well as importing from the above specified formats. *ProteinHypernetworks* is a frontend to *LibModalLogicProteinHypernetwork* that allows the execution of the above mentioned methods with a graphical user interface or a command line interface. Further, the results can be visualized with the JUNG framework; examples are shown on the next pages. The two frontends are provided as *.jar* executables in the folder `tool` of the DVD (`tool/ProteinHypernetworkEditor.jar` and `tool/ProteinHypernetwork.jar`).

Further Informations

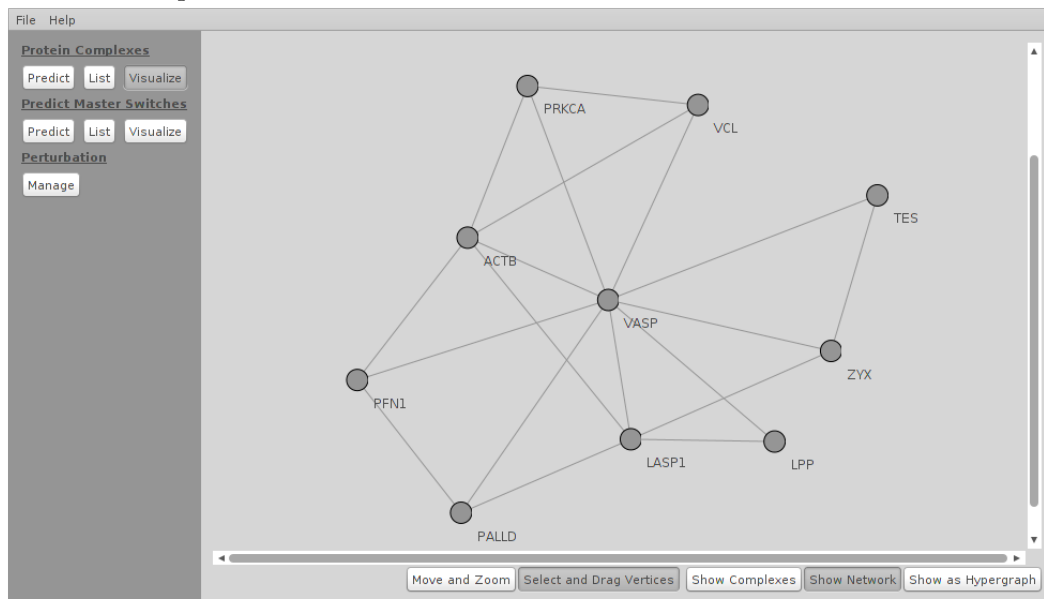
The prediction of master switches can be visualized by displaying the protein network as an undirected graph and changing the size of nodes and the thickness of edges according to their master switch score:



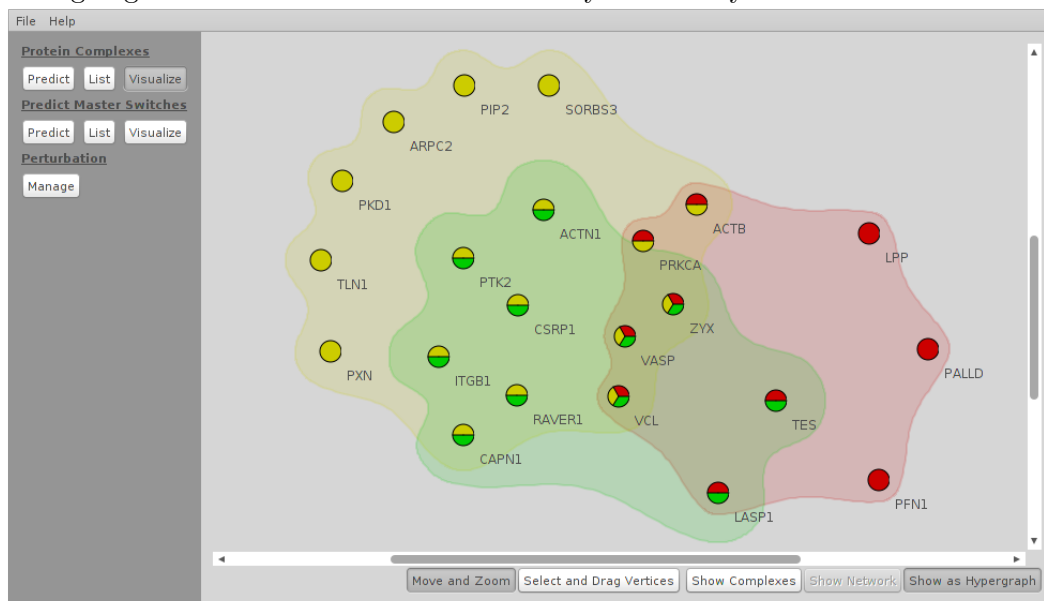
The visualization of protein complexes consists of three steps. First, each complex is displayed as a node in an undirected graph. An edge between two complexes symbolizes a significant overlap between them (A jaccard index of more than 0.2):



Secondly, complexes that were selected in this graph can be separately displayed as simultaneous protein subnetworks:

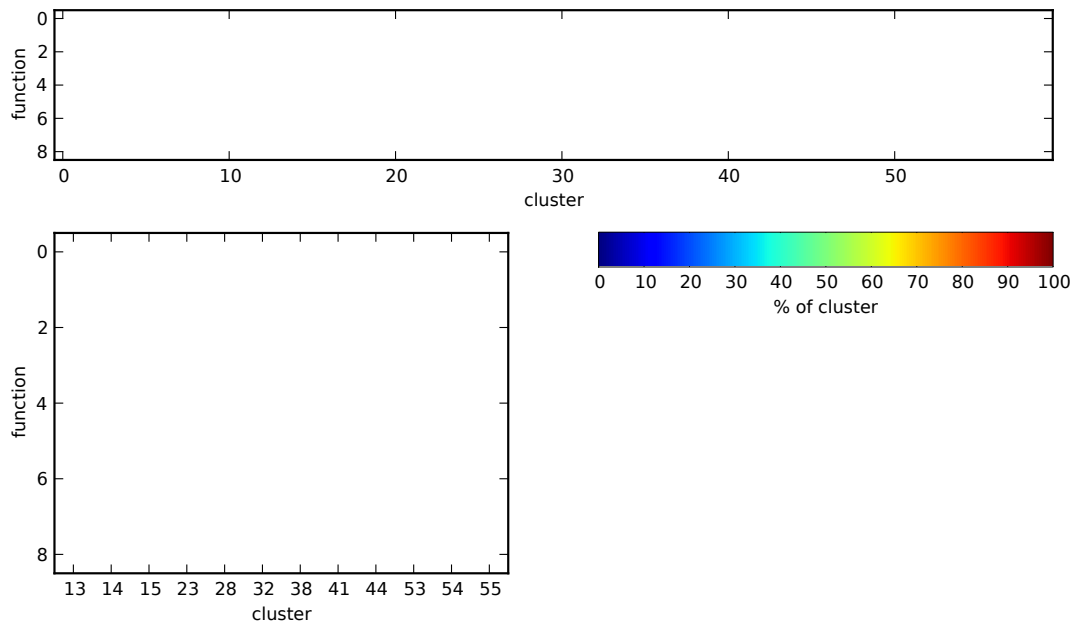


Lastly the selected complexes can be displayed as hyperedges in a hypergraph of proteins. A hypergraph is a generalization of an undirected graph, provided by allowing edges to be sets of nodes with arbitrary cardinality.²



²For this, the code of the project *JUNG Framework Enhancement - Hypergraph Visualization* from the Swiss Federal Institute of Technology Zürich (ETH Zürich) was fixed and adapted.

Prediction of Functional Similarity



| metabolism functions | | clusters | |
|----------------------|---|----------|------|
| index | function | index | size |
| 0 | Cell cycle | 0 – 45 | 2 |
| 1 | Cell polarity and structure | 46 – 54 | 3 |
| 2 | Intermediate and energy metabolism | 55, 56 | 7 |
| 3 | Membrane biogenesis and traffic | 57 | 8 |
| 4 | Protein / RNA transport | 58 | 9 |
| 5 | Protein synthesis and turnover | 59 | 658 |
| 6 | RNA metabolism | | |
| 7 | Signalling | | |
| 8 | Transcription / DNA maintenance / chromatin structure | | |

Matching of metabolism functions to protein clusters when using transitivity clustering with an alternative threshold of 1.4 (Chapter 6.2). The used functions are listed in the left table. For each function, the set of proteins known to take part in its execution was collected from the data provided by (Gavin et al., 2002). Proteins were clustered by the similarity of predicted complexes upon their perturbation. For each cluster the percentages of proteins executing the provided functions were calculated. A dark red square in one of the two images means that 100% of the clusters proteins is known to execute the function, whereas dark blue corresponds to 0%. The second image excludes all clusters in which not all proteins are annotated. Clusters are ordered ascending in their size, as shown in the right table.

Here, a distribution of cluster sizes that is worse than with the originally selected threshold of 1.8 still allows a precision of $\frac{8}{12} = 0.67$.

Figures

| | | |
|-----|---|----|
| 1.1 | Binding of a protein to another molecule | 1 |
| 1.2 | Scaffold dependent interaction | 2 |
| 1.3 | Competition of two proteins on the same binding domain | 2 |
| 3.1 | Example formula tree for $A \wedge B$ | 12 |
| 3.2 | Example tableau for $(\Box\neg A) \wedge (\Diamond A \vee \Diamond B)$ | 13 |
| 3.3 | Backjumping example | 15 |
| 3.4 | Dynamic backtracking example | 16 |
| 5.1 | Example for LCMA predicting different cliques | 41 |
| 5.2 | Example for a redundant path in a tree of removal instructions | 46 |
| 5.3 | Complex prediction quality in dependence of the percentage of applied constraints | 50 |
| 5.4 | Predicted complex size distribution dependent on the number of constraints | 51 |
| 5.5 | Duplication of false positive complexes upon refinement with only few constraints | 52 |
| 6.1 | Measures for the difference between sets of predicted complexes | 54 |
| 6.2 | Influence of different measures on master switch scores | 55 |
| 6.3 | Master switch prediction for the yeast proteins | 56 |
| 6.4 | Matching of metabolism functions to protein clusters | 58 |
| 6.5 | Overlap between function sets | 59 |
| 6.6 | Probability for predicted protein clusters of different sizes to entirely execute one metabolism function | 60 |

Tables

| | | |
|-----|--|----|
| 3.1 | Deterministic expansion rules for modal logic tableau | 14 |
| 3.2 | Non-deterministic expansion rule for modal logic tableau | 14 |
| 4.1 | Truth table for a constraint modelling scaffold dependency | 25 |
| 4.2 | Truth table for constraints modelling competition on the same binding domain | 26 |
| 4.3 | Truth table illustrating Kripke model for minimal network state formula | 32 |
| 5.1 | Comparison of MCODE and LCMA prediction quality | 41 |

Algorithms

| | | |
|-----|--|----|
| 3.1 | The modal logic tableau | 18 |
| 3.2 | The modal logic tableau (formula expansion) | 19 |
| 3.3 | The modal logic tableau (dynamic backtracking) | 20 |
| 3.4 | The modal logic tableau (modification of dynamic backtracking) | 21 |
| 4.1 | Find all Kripke models for a minimal network state formula | 30 |
| 5.1 | LCM algorithm (detect local cliques) | 38 |
| 5.2 | LCM algorithm (merge dense regions) | 38 |
| 5.3 | LCM algorithm (fast check for overlapping dense regions) | 39 |
| 5.4 | Find clashes between minimal network states | 44 |
| 5.5 | Algorithm to build a tree of removal instructions | 45 |

Bibliography

- Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Walter, P. (2007). *Molecular Biology of the Cell*. 5th ed. Garland Science.
- Bader, G. and Hogue, C. (2003). An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics* 4.1, 2–29.
- Brohee, S. and Helden, J. van (2006). Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics* 7.1, 488–507.
- Doberkat, E.-E. (2009). *Stochastic Coalgebraic Logic*. Tech. rep. Chair 10 of Computer Sciences, TU Dortmund.
- Gavin, A.-C. C., Bösche, M., Krause, R., Grandi, P., Marzioch, M., Bauer, A., Schultz, J., Rick, J. M., Michon, A.-M. M., Cruciat, C.-M. M., Remor, M., Höfert, C., Schelder, M., Brajenovic, M., Ruffner, H., Merino, A., Klein, K., Hudak, M., Dickson, D., Rudi, T., Gnau, V., Bauch, A., Bastuck, S., Huhse, B., Leutwein, C., Heurtier, M.-A. A., Copley, R. R., Edlmann, A., Querfurth, E., Rybin, V., Drewes, G., Raida, M., Bouwmeester, T., Bork, P., Seraphin, B., Kuster, B., Neubauer, G., and Superti-Furga, G. (2002). Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature* 415.6868, 141–147.
- Guimera, R. and Sales-Pardo, M. (2010). Missing and spurious interactions and the reconstruction of complex networks. *Proceedings of the National Academy of Sciences* 106.52, 22073–22078.
- Hartwell, L. H., Hopfield, J. J., Leibler, S., and Murray, A. W. (1999). From molecular to modular cell biology. *Nature* 402.6761 Suppl, C47–C52.
- Ingram, P. J., Stumpf, M. P., and Stark, J. (2006). Network motifs: structure does not determine function. *BMC genomics* 7.1, 108–120.
- Jeong, H., Mason, S. P., Barabasi, A. L., and Oltvai, Z. N. (2001). Lethality and centrality in protein networks. *Nature* 411.6833, 41–42.
- Jiang, P. and Singh, M. (2010). SPICi: a fast clustering algorithm for large biological networks. *Bioinformatics (Oxford, England)* 26.8, 1105–1111.
- Jung, S. H., Hyun, B., Jang, W.-H., Hur, H.-Y., and Han, D.-S. (2010). Protein complex prediction based on simultaneous protein interaction network. *Bioinformatics* 26.3, 385–391.
- Kreuzer, M. and Kühling, S. (2006). *Logik für Informatiker*. Pearson Studium.
- Li, X.-L. L., Foo, C.-S. S., and Ng, S.-K. K. (2007). Discovering protein complexes in dense reliable neighborhoods of protein interaction networks. *Computational systems bioinformatics / Life Sciences Society. Computational Systems Bioinformatics Conference* 6, 157–168.
- Li, X.-L. L., Tan, S.-H. H., Foo, C.-S. S., and Ng, S.-K. K. (2005). Interaction graph mining for protein complexes using local clique merging. *Genome informatics. International Conference on Genome Informatics* 16.2, 260–269.
- Li, Z. (2008). “Efficient and Generic Reasoning for Modal Logics”. PhD thesis. School of Computer Science, University of Manchester, UK.

Bibliography

- Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. (2002). Network Motifs: Simple Building Blocks of Complex Networks. *Science* 298.5594, 824–827.
- Monod, J., Wyman, J., and Changeux, J.-P. (1965). On the nature of allosteric transitions: A plausible model. *Journal of Molecular Biology* 12.1, 88–118.
- Spirin, V. and Mirny, L. A. (2003). Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Sciences of the United States of America* 100.21, 12123–12128.
- Wittkop, T., Emig, D., Lange, S., Rahmann, S., Albrecht, M., Morris, J. H., Böcker, S., Stoye, J., and Baumbach, J. (2010). Partitioning biological data with transitivity clustering. *Nature methods* 7.6, 419–420.

Acknowledgements

I want to thank my supervisors Prof. Dr. Sven Rahmann and Dr. Eli Zamir, who always offered me advice and support. Further, thank goes to the group of Dr. Eli Zamir at the Max Planck Institute of Molecular Physiology Dortmund, who were so kind to let me take part in their meetings and daily work. Lastly I want to thank everybody who supported me in writing this thesis.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 27. August 2010

Johannes Köster

