Computational Omics*

Prof. Dr. Sven Rahmann Genome Informatics, Institute of Human Genetics, Faculty of Medicine, University of Duisburg-Essen

Dipl.-Inf. Dominik Kopczynski Bioinformatics, Computer Science XI, TU Dortmund

Dipl.-Inf. Johannes Köster Genome Informatics, Institute of Human Genetics Faculty of Medicine, University of Duisburg-Essen

> Winter 2011/12 Draft, February 8, 2012

*Preparation of these lecture notes was partially supported by DFG SFB 876 "Providing Information by Data Analysis with Restricted Resources" (http://sfb-876.cs.tu-dortmund.de)

Contents

| 1 | The | Polymerase Chain Reaction (PCR) | 1 | |
|-------------------|-----|--|---|--|
| | 1.1 | Overview of the PCR Cycle | 1 | |
| | 1.2 | Applications of PCR | 2 | |
| | 1.3 | Mathematical PCR Model | 4 | |
| | 1.4 | Branching Processes | 5 | |
| | 1.5 | Estimation | 9 | |
| | 1.6 | Projects | 9 | |
| | | 1.6.1 Simulation | 9 | |
| | | 1.6.2 Computation | 9 | |
| | | 1.6.3 Mathematics | 0 | |
| | | 1.6.4 PCR process | 0 | |
| 2 | Nex | Generation Sequencing 1 | 1 | |
| | 2.1 | Overview | 1 | |
| | | 2.1.1 Roche 454 | 2 | |
| | 2.2 | Projects | 3 | |
| | | 2.2.1 Simulating Flowgrams | 3 | |
| | | 2.2.2 Bias in Transcriptome Data? | 3 | |
| 3 DNA Microarrays | | | | |
| | 3.1 | Overview | 5 | |
| | 3.2 | Low-Level Analysis | 8 | |
| | | 3.2.1 Microarray Image Analysis | 8 | |
| | | 3.2.2 Noise and Bias Correction | 8 | |
| | | 3.2.3 Normalization | 0 | |
| | | 3.2.4 Aggregation | 1 | |
| | 3.3 | High-Level Analysis | 1 | |
| | | 3.3.1 Identification of Differentially Expressed Genes | 1 | |
| | | 3.3.2 Clustering | 1 | |

| | | 3.3.3 | Bi-Clustering | 22 | | | | | | | | | |
|----|------------|--------------------------|--------------------------------------|----------|--|--|--|--|--|--|--|--|--|
| | | 3.3.4 | Classification and Prediction | 22 | | | | | | | | | |
| | 3.4 | ${ m ts}$ | 22 | | | | | | | | | | |
| | | 3.4.1 | Reading .cel files | 22 | | | | | | | | | |
| | | 3.4.2 | Normalization | 22 | | | | | | | | | |
| 4 | Met | etabolomics 23 | | | | | | | | | | | |
| - | / 1 | Overv | iew | 23 | | | | | | | | | |
| | ч.1 Д 2 | MCC | /ms | 20 | | | | | | | | | |
| | 7.2 | 4 2 1 | Massurement | 24 | | | | | | | | | |
| | | 4.2.1 | Data filo | 24 25 | | | | | | | | | |
| | 12 | H.Z.Z | | 20 | | | | | | | | | |
| | 4.0 | 1 9 1 | Perceling | 20 | | | | | | | | | |
| | 4 4 | 4.3.1 M. J.1 | | 20 | | | | | | | | | |
| | 4.4 | TIM | ing peaks | 21 | | | | | | | | | |
| | 4.5 | EM al | | 28 | | | | | | | | | |
| | | 4.5.1 | Set up a maximum-likelihood-function | 28 | | | | | | | | | |
| | | 4.5.2 | Expectation step | 29 | | | | | | | | | |
| | | 4.5.3 | Estimate the hidden values | 30 | | | | | | | | | |
| | | 4.5.4 | Maximization step | 30 | | | | | | | | | |
| | | 4.5.5 | Maximum-Likelihood-Estimators (MLE) | 31 | | | | | | | | | |
| | | 4.5.6 | Algorithm | 33 | | | | | | | | | |
| | 4.6 | Recou | rse-constrained detection | 36 | | | | | | | | | |
| | 4.7 | Projec | ${ m ts}$ | 36 | | | | | | | | | |
| | | 4.7.1 | Gaussian Bell | 36 | | | | | | | | | |
| | | 4.7.2 | IMS Measurement | 37 | | | | | | | | | |
| 5 | Prot | teomics | | 39 | | | | | | | | | |
| _ | | | | | | | | | | | | | |
| 6 | Inte | ractom | | 41 | | | | | | | | | |
| | 6.1 | Protei | n-Protein Interaction Networks | 41 | | | | | | | | | |
| | 6.2 | Protei | n Network Graphs | 43 | | | | | | | | | |
| | 6.3 | Protei | n Complex Prediction | 43 | | | | | | | | | |
| | | 6.3.1 | Local Clique Merging Algorithm | 44 | | | | | | | | | |
| | | 6.3.2 | Merging dense regions | 45 | | | | | | | | | |
| | 6.4 | .4 Protein Hypernetworks | | 46 | | | | | | | | | |
| | | 6.4.1 | Minimal Network States | 47 | | | | | | | | | |
| | | 6.4.2 | Prediction of Protein Complexes | 50 | | | | | | | | | |
| | | 6.4.3 | The Tableau Calculus | 51 | | | | | | | | | |
| Bi | bliog | raphy | | 55 | | | | | | | | | |

Preface

This report summarizes the lecture "Computational *omics", given for the first time at TU Dortmund in the winter term 2011/12.

The main idea of this course is to have not the classical lecture plus exercises organization, but to have several small team-projects where real is analyzed using methods from the lecture and open-source software.

The course is a specialization at the Master level for 6 ECTS credit points, with time requirements of 2V+2P.

Essen and Dortmund, February 8, 2012

Sven Rahmann

The Polymerase Chain Reaction (PCR)

1.1 Overview of the PCR Cycle

Instead of repeating everything that can (and should) be read on the web¹, we restrict ourselves to the key points of the PCR process.

PCR was developed 1983 by Kary Mullis (Nobel prize 1993); probably 2013 will celebrate "30 years of PCR".

Today, PCR is the main workhorse of molecular biology. It is able to answer two types of questions: Given a particular DNA sequence s of moderate length (up to 10 kbp) with (at least) known ends x and y (such that s = xty, where t can be unknown) and a biological DNA sample,

- 1. is a fragment with sequence s contained in the sample?
- 2. which fraction of the sample consists of s?

Classical PCR answers the first question, whereas later extensions also allow to answer the second question.

The key points to know are the following ones.

• PCR exponentially amplifies (i.e., copies) a double-stranded DNA molecule (of maximum size 10 to 40 kbp) called the *template* in n cycles into up to 2ⁿ copies, using natural (but engineered) DNA polymerase enzymes, DNA primers, and free nucleotides. Each cycle (approximately) doubles the existing number of molecules; it is common to do 20–40 cycles. Thus PCR is very sensitive (unfortunately, also for contaminating DNA), so even very small amounts of DNA can be reliably detected.

 $^{{}^{1}}e.g., {\tt http://en.wikipedia.org/wiki/Polymerase_chain_reaction}$

1 The Polymerase Chain Reaction (PCR)

- DNA polymerase (often: heat-resistant Taq polymerase, molecule of the year 1989, isolated from *Thermus aquaticus*, a thermophilic bacterium that survives in hot springs at 50–80 °C) is an enzyme that synthesizes a complementary DNA strand to a given template, consuming free nucleotides.
- Primers are short DNA oligonucleotides (20–30 nt) that bind to both 3' ends of the double-stranded template; they are necessary to initiate a cycle.
- A PCR cycle (30 seconds to 2 minutes) consists of:
 - 1. denaturation at 94–98 °C for 20–30 sec: double-stranded template melts into two single strands
 - 2. annealing of primers at 50–65 °C for 30–40 sec: complementary primers bind at the 3' end of each template, and the DNA polymerase binds to the template/primer hybrids. The primers form the 5' ends of new DNA molecules that are to be synthesized.
 - 3. synthesis of complementary strands at 72 °C: the primers are elongated in 5' to 3' direction at a speed of approximately 1000 nt/min, consuming free nucleotides (dNTPs desoxy nucleotide triphosphates).
- PCR Finalization: after the last cycle, the mixture is held at 70–74 °Cfor 5–15 min to allow for final elongation of incomplete strands. The mixture may be stored at 4–15 °Cfor a short amount of time.
- While the initial template may be longer, exponential amplification occurs only between the primers (inclusive). A sketch (Figure 1.1) will show this.
- Phases: When primers, polymerase and free nucleotides are in abundance, the number of molecules (approximately) doubles in each cycle ("exponential amplification"). After several cycles, these resources become scarcer and constitute the bottleneck, the process "levels off", and then reaches a "plateau" (no more new products).
- Checking for success: with agarose gel electrophoresis. The PCR products are stained and applied onto an agarose gel and subjected to an electric current. The polar DNA molecules move through the gel; the distance covered depends inversely on the molecular mass (and hence length). Thus, short DNA molecules move farther than long molecules. By comparing the position of the PCR products in the gel to a set of reference molecules of known lengths (so-called "DNA ladder"), their approximete length can be determined. If the observed length corresponds to the expected length |S| = |xty|, we can assume that the intended fragment was amplified. However, we really only know that *some* fragment of that length that ends with x and y was amplified. If we do not observe the fragment at the expected length or several fragments of different length or none at all, we must assume that either the reaction did not perform well, or that the expected molecule was not present in the sample, but others were.

1.2 Applications of PCR

The manifold applications of PCR include the following ones.



Exponential growth of short product

Figure 1.1: Overview of PCR from http://en.wikipedia.org/wiki/File:PCR.svg. Blue: original DNA template. Red arrows: primers Green: PCR products, then used as templates.

1 The Polymerase Chain Reaction (PCR)

- Early identification of infections (viruses, bacteria): A blood sample is examined for characteristic genomic sequences of viruses or bacteria. Since PCR is very sensitive, an infection can be found even before symptoms materialize.
- Identification of individuals by DNA profiling (aka DNA testing, DNA typing, genetic fingerprinting): Individuals (except monozygotic twins) vary in repeat regions of the genome called VNTRs (Variable Number Tandem Repeats), especially STRs (Short Tandem Repeats). Such a genomic region looks as follows: A specific sequence x is followed by a short (often 3-5 nt) sequence s, repeated n times, which is again followed by y, thus the sequence is $xs^n y$. An example is the TH01 locus on chromosome 11p15.5, which repeats s = AATG between 3 and 14 times. PCR is used to specifically amplify the region between and including x and y, and the product is length-analyzed (e.g., with agarose gel electrophoresis), from which the number n of repeats can be determined. As each human has two parents, the result is a (multi)set $\{\{n, n'\}\}$ of numbers. Such a test is performed for several (say, k) independent loci (the SGM plus test by Applied Biosystems uses 10 loci² plus a gender test; the result is a k-tuple of unordered pairs of numbers, which is called a fingerprint $f = (\{\{n_1, n'_1\}\}, \dots, \{\{n_k, n'_k\}\})$. DNA fingerprints, just like normal fingerprints, are stored in national databases (eg, for convicted criminals), which is subject to privacy concerns. The probability that two individuals share the same fingerprint is generally assumed to be very small, but sometimes subject to debate in court.
- Paternity testing: Fingerprinting is also employed for paternity testing. If two persons with fingerprints $f_{\rm P} = (\{\{p_1, p'_1\}\}, \dots, \{\{p_k, p'_k\}\})$ and $f_{\rm M} = (\{\{m_1, m'_1\}\}, \dots, \{\{m_k, m'_k\}\})$ have a child with fingerprint $f_{\rm O} = ((\{\{o_1, o'_1\}\}, \{\{o_k, o'_k\}\}))$, then we must have that $o_i \in \{\{p_i, p'_i\}\}$ and $o'_i \in \{\{m_i, m'_i\}\}$, or vice versa. Thus a non-compatible fingerprint can rule out fathership, while a compatible one provides more or less strong evidence for it. The strength of such evidence (p-value) is often a subject of debate in court.
- Specific selection of DNA loci for sequencing: Sometimes it is not sufficient to determine the number of repeat units, but we would like to known the whole sequence of a gene (or part of a gene).
- Gene expression analysis (determination of a gene's transcription level):

1.3 Mathematical PCR Model

Let N_t be the number of molecules produced after t PCR rounds. We specify $N_0 := n_0 \in \mathbb{N}$, an arbitrary initial number. For simplification, we presently assume $n_0 = 1$ and start with a single molecule.

In a typical PCR reaction, not every single molecule is amplified. Instead, there is a certain probability $0 of copying a particular molecule, typically around <math>p \approx 0.8$ under good conditions. This parameter is called the *PCR efficiency* (parameter).

Therefore, assuming efficiency p_t in round t, we have

$$N_t = N_{t-1} + B(N_{t-1}, p_t), (1.1)$$

²http://en.wikipedia.org/wiki/SGM+

where B(n, p) is a random variable with a binomial distribution with parameters n, p, that is

$$\mathbb{P}[B(n,p)=k] = \binom{n}{k} p^k (1-p)^{n-k}.$$

Carefully note that the parameter N_{t-1} in (1.1) is a random variable (not a constant), namely the result from the previous PCR round.

The fate of a single initial molecule can be envisioned as a *branching process* and visualized as a tree. In each round, a molecule has one or two children. The first child always exists and represents the original molecule. The second child exists with probability p an represents the copy. This process continues recursively for the given number T of levels.

The distribution of N_t . We compute expectation, variance, and the probability distribution function (pdf) of N_t . Clearly $\mathbb{E}[N_0] = n_0$ and

$$\mathbb{E}[N_t] = \mathbb{E}[N_{t-1}] + \mathbb{E}[B(N_{t-1}, p_t)].$$

Assuming that the efficiency $p_t \equiv p$ is constant over time, using $\mathbb{E}[B(n,p)] = np$ and rules for computing expected values from conditional expected values, this becomes

$$\mu_t := \mathbb{E}[N_t] = \mathbb{E}[N_{t-1}] + p \cdot \mathbb{E}[N_{t-1}] = (1+p) \cdot \mathbb{E}[N_{t-1}] = n_0 \cdot (1+p)^t.$$
(1.2)

Assuming that we already know the distribution of N_{t-1} , we compute the distribution of N_t . Note that by assumption, the distribution of N_0 is Dirac in n_0 , that is $\mathbb{P}[N_0 = k] = [k = n_0]$, which is 1 if $k = n_0$, and 0 otherwise. Now, what must happen to obtain exactly k copies after round t? There must have been between $j \in \{k/2, \ldots, k\}$ molecules after round t-1, and then for a given choice of j, exactly k - j of them must have been amplified (and the remaining j - (k - j) = 2j - k must not). Thus, summing over all possibilities for j, we obtain the recurrence

$$\mathbb{P}[N_t = k] = \sum_j {j \choose k-j} p^{k-j} (1-p)^{2j-k} \cdot \mathbb{P}[N_{t-1} = j].$$
(1.3)

The summation range for j starts at $j = \lceil k/2 \rceil$ and ends at $j = \min\{k, n_0 2^{t-1}\}$. If p_t is not constantly p, we simply use p_t instead of p in (1.3).

From the distribution, we can also derive the variance

$$\operatorname{Var}[N_t] = n_0 \cdot (1+p)^t \cdot \left[(1+p)^t - 1 \right] \cdot \frac{1-p}{1+p}.$$
(1.4)

1.4 Branching Processes

Let us look deeper into the process that generates (N_t) . As described earlier, it can be looked at as a branching process on a binary tree, where in each node the left child exists with probability 1 and the right child exists with probability p. N_t describes the number of nodes at level t.

1 The Polymerase Chain Reaction (PCR)

Here, we look at such processes in general, where the general model is as follows. We consider the development of a population (number of individuals Z_t) over several generations, assuming $Z_0 = 1$. Individuals live for a single generation and then die. Before dying, each individual *i* produces a random number ξ_i of offspring. The distribution of ξ_i is given by a probability vector $(p_0, p_1, p_2, ...)$ such that $\mathbb{P}[\xi_i = j] = p_j$ for all *j* and each *i*, and all ξ_i are independent.

Thus the dynamics are described by

$$Z_{t+1} = \sum_{i=1}^{Z_t} \xi_i.$$

Note that the upper summation limit is the random value from the previous generation.

For the PCR process, we use the probability vector (0, 1 - p, p, 0, 0, ...), i.e., there is probability 1 - p for one child, probability p for two children, and probability zero for all other numbers of children.

Let us compute mean and variance of the process (Z_t) . Therefore, let $M_t := \mathbb{E}[Z_t]$ and $V_t := \operatorname{Var}[Z_t]$ with $M_0 = 1$ and $V_0 = 0$.

1.1 Lemma (Mean and Variance for Branching Processes). Let $\mu := \mathbb{E}[\xi]$ and $\sigma^2 := \operatorname{Var}[\xi]$. Then

$$M_t = \mu^t \quad and \quad V_t = \begin{cases} \sigma^2 t & \text{if } \mu = 1, \\ \sigma^2 \mu^{t-1} (1 - \mu^t) / (1 - \mu) & \text{if } \mu \neq 1. \end{cases}$$

Proof. The result for M_t is proved by conditioning on $Z_{t-1} = k$. Thus

$$M_{t} = \mathbb{E}[Z_{t}]$$

$$= \sum_{k} \mathbb{P}[Z_{t-1} = k] \cdot \mathbb{E}[Z_{t} \mid Z_{t-1} = k]$$

$$= \sum_{k} \mathbb{P}[Z_{t-1} = k] \sum_{i=1}^{k} \mathbb{E}[\xi_{i}]$$

$$= \sum_{k} \mathbb{P}[Z_{t-1} = k] k\mu$$

$$= \mathbb{E}[Z_{t-1}] \mu$$

$$= M_{t-1} \mu.$$

From $M_0 = 1$, it follows inductively that $M_t = \mu^t$.

For V_t , we use the definition $V_t = \mathbb{E}[(Z_t - \mu^t)^2]$ and again condition the event that $Z_{t-1} = k$.

Thus,

$$\begin{split} V_t &= \mathbb{E}[(Z_t - \mu^t)^2] \\ &= \sum_k \mathbb{P}[Z_{t-1} = k] \cdot \mathbb{E}[(Z_t - \mu^t)^2 \mid Z_{t-1} = k] \\ &= \sum_k \mathbb{P}[Z_{t-1} = k] \cdot \mathbb{E}[(\sum_{i=1}^k \xi_i - k\mu + k\mu - \mu^t)^2] \\ &= \sum_k \mathbb{P}[Z_{t-1} = k] \cdot \left[\mathbb{E}[(\sum_{i=1}^k \xi_i - k\mu)^2] + 2 \mathbb{E}[(\sum_{i=1}^k \xi_i - k\mu)(k\mu - \mu^t)] + (k\mu - \mu^t)^2 \right] \\ &= \sum_k \mathbb{P}[Z_{t-1} = k] \cdot \left[k\sigma^2 + 0 + (k\mu - \mu^t)^2 \right] \\ &= \sum_k \mathbb{P}[Z_{t-1} = k] \cdot \left[k\sigma^2 + \mu^2(k - \mu^{t-1})^2 \right] \\ &= \sigma^2 M_{t-1} + \mu^2 V_{t-1} \\ &= \sigma^2 \mu^{t-1} + \mu^2 V_{t-1}. \end{split}$$

Since $V_0 = 0$, we have $V_1 = \sigma^2$. If $\mu = 1$, the recursion is easy to solve as claimed. If $\mu \neq 1$, the recursion is solved by induction.

In the next step, we derive an easy-to-remember expression for computing the distribution of Z_t . First, we introduce the concept of a *probability generating function* (pgf). If we have a random variable ξ distributed according to a probability vector (p_0, p_1, \ldots) , such that $\mathbb{P}(\xi = j) = p_j$, we can represent this distribution by a formal power series f (or polynomial if the vector is finite) called the pgf by defining

$$f(z) := \sum_{j \ge 0} p_j \, z^j.$$
(1.5)

We can imagine z to be any real or complex variable, and at the moment we do not care about the convergence of f.

For the PCR process with efficiency p, we have $p_1 = 1 - p$ and $p_2 = p$, while all other $p_j = 0$; thus

$$f_{\rm PCR} = (1-p)z + pz^2.$$
 (1.6)

How can we work with generating functions? We study how certain operations on random variables translate into operations between their generating functions.

The following lemma tells us that the pgs of the sum X + Y of two *independent* random variables X and Y with pgfs g and h, respectively, is given by the *product* g * h of the pgfs.

1.2 Lemma. Let X and Y be independent r.v.s on the natural numbers with pgfs $g(z) := \sum_j g_j z^j$ and $h(z) := \sum_j h_j z^j$, respectively, i.e., $\mathbb{P}[X = j] = g_j$ and $\mathbb{P}[Y = j] = h_j$. Let Z := X + Y, $f_j := \mathbb{P}[Z = j]$ and let $f(z) := \sum_j f_j z^j$ be the pgf of Z. Then

$$f(z) = g(z) \cdot h(z).$$

Proof. We have Z = j if and only if X = k and Y = j - k for some k. Since events for X and Y are independent, the respective probabilities are multiplied. Thus $f_j = sum_k g(k)h(j-k)$; we also say that the vector (f_j) is the *convolution* of the vectors (g_j) and (h_j) . In the product of the power series f(z) = g(z)h(z), the coefficient of z^j arises in the same way.

A typical application of the lemma is as follows. Consider the sum $S = \sum_{i=1}^{k} X_i$, where the X_i are independent and identically distributed (i.i.d) r.v.s. Let f be the pgf of each X_i . The probability $\mathbb{P}[S = j]$ is then given as the coefficient of z^j of the k-th power $f^k(z)$. We write this as $[z^j]f^k(z)$; this is also called *coefficient extraction notation*.

The following lemma shows how to compute the pgf of a sum of r.v.s when the number of terms is also random. This is a key lemma for branching processes.

1.3 Lemma. Let X_i (i = 1, 2, ...) be independent r.v.s with common distribution given by pgf f(z). Let Y be an integer-valued r.v. with pgf g(z) independent of the X_i , and let $Z := \sum_{i=1}^{Y} X_i$. Then the pgf h(z) of Z is

$$h(z) = g(f(z)).$$

It follows that, if g(z) is the pgf of the number of individuals after t-1 generations in a branching process, and f(z) is the pgf of the offspring variables ξ_i , then the pgf of the number of individuals after t generations is h(z) = g(f(z)).

Proof. Again, the trick is to condition on Y = k: We have

$$h_{j} = \mathbb{P}(Z = j)$$

$$= \sum_{k} \mathbb{P}(Z = j, Y = k)$$

$$= \sum_{k} \mathbb{P}(Y = k) \cdot \mathbb{P}(Z = j \mid Y = k)$$

$$= \sum_{k} g_{k}[z^{j}]f^{k}(z),$$

since the number of summands in Z is now fixed at k in each term. Thus

$$\begin{split} h(z) &= \sum_{j} h_{j} z^{j} \\ &= \sum_{j} \sum_{k} g_{k} [z^{j}] f^{k}(z) z^{j} \\ &= \sum_{k} g_{k} f^{k}(z) \\ &= g(f(z)), \end{split}$$

as claimed.

1.4 Theorem. The pgf of the number of individuals after t generations is $f^{[t]}(z)$, where $f^{[t]}(z)$ denotes the t-fold iteration of f, i.e., $f(f(\cdots(t)\cdots))$.

Proof. Since $Z_0 = 1$, $f^{[0]}(z) = z$ and $f^{[1]} = f(z)$. Now the theorem follows by induction from Lemma 1.3.

Let us finally consider the convergence properties of Z_t for $\mu > 1$ as in the PCR case with PCR efficiency p > 0, where $\mu = 1 + p > 1$. Of course, Z_t grows without bounds in this case. Therefore, we normalize the quantity and consider $Z_t^* := Z_t/\mu^t$, which has constant expectation 1, and its variance converges to $\sigma^2/(\mu(\mu - 1))$, which is finite. Moreover, Z_{t_t} forms a martingale, which means that $\mathbb{E}[Z_t^* | Z_{t-1}^*] = Z_{t-1}^*$. Therefore we can apply Doob's martingale convergence theorem and obtain the following result.

1.5 Theorem (Convergence of branching processes). Let Z_t be a branching process with offspring variables (ξ_i) in each generation. Let $\mu := \mathbb{E}[\xi] > 1$ and $\sigma^2 := \operatorname{Var}[\xi]$. Then the normalized process $Z_t^* := Z_t/\mu^t$ converges in distribution as $t \to \infty$.

1.5 Estimation

1.6 Projects

1.6.1 Simulation

Simulate the PCR reaction for a given number n_0 of initial molecules, a given maximum number T of rounds with a given constant efficiency p (or a given efficiency vector (p_1, \ldots, p_T)). To do this, you will need to simulate a B(n, p) variable. This can be done easily and robustly (even though perhaps slowly) by drawing n random numbers uniformly in the interval [0, 1] and counting those which are < p.

Repeat the simulation (e.g., for $n_0 = 1$, T = 20, p = 0.8) many times and record the resulting numbers for N_T . Empirically compute their expectation, variance, and draw a histogram. Compare with the theoretical values given above.

For all plotting, especially in the next subproject, consider using logarithmic scales on one or both axes. How does the graph change when you do that? Which type of axis is more informative?

1.6.2 Computation

Compute the exact distribution of N_t for given parameters. To do this, implement the formula (1.3).

Plot the resulting distributions for all combinations of $n_0 \in \{1, 5\}$, $T \in \{15, 20, 25\}$, $p \in \{0.7, 0.8, 0.9\}$ and describe your observations.

For this task, you need the binomial coefficients $\binom{n}{k}$. If your standard library provides them, that's fine. However, note that you must be careful to compute them not naively. The best way for both small and large numbers of n and k is to compute their logarithm

$$\log \binom{n}{k} = \log(n!) - \log(k!) - \log((n-k)!)$$

by using the Gamma function, for which $\Gamma(x) = (x-1)!$ for integer x, and use an implementation of the logarithm of the Gamma function (often called lgamma or lngamma; for example in Python3.2 available as math.lgamma). A Java implementation of these routines is available at http://code.google.com/p/jprobdist/source/browse/trunk/src/edu/udo/cs/ bioinfo/jprobdist/MathFunctions.java

Note that it is always advisable to compute the logarithms of the desired quantities (instead of the quantities directly) when working with very small or large numbers (small probabilities, binomial coefficients). We call this "working in log-space". Advantage: All multiplications (such as those in one term of (1.3)) become additions in log-space. Disadvantage: Addition becomes slightly more complicated. Assume you want to compute c = a + b in log-space, i.e., you only have $a' = \log a$ and $b' = \log b$ and desire $c' = \log c$. In principle, this is $c' = \log(\exp(a') + \exp(b'))$, but the exponentiation may take you out of the supported value range. Instead, assume $a \ge b > 0$. Then c = a+b = a(1+b/a) and hence $c' = a' + \log(1+b/a)$. The ratio b/a is conveniently written as $r = \exp(b' - a')$, so only the difference between b' and a' (which can be small even if a and b are large) needs to be exponentiated. To compute $\log(1+r)$ accurately for small r, you should not proceed by explicitly adding 1 and taking the logarithm because of loss of precision. Instead, use an available library function $\log 1p(r)$ that does the right thing for both small and large r. (Test this for $r = 10^{-20}$ by trying both ways.) Thus $c' = a' + \log \log(e^{(b'-a')})$ is the robust addition operation in log-space. Implement it, you will need it often. To add more than two values, proceed similarly by determining their maximum first.

Friendly warning: Don't start too late with implementing; both simulation and computation will take some time.

1.6.3 Mathematics

- 1. Make sure you understand the proof of the variance formula for V_t . Under the assumption that $\mu \neq 1$, complete the proof by induction.
- 2. Compute the pgf of the PCR process after 3 generations, starting with $Z_0 = 1$ molecule. What happens if you start with *n* molecules?

1.6.4 PCR process

Visualize the convergence of the normalized PCR process as $t \to \infty$ for different efficiency parameters p. Use moderately large values of t and plot the resulting approximate densities over each other.

chapter 2

Next Generation Sequencing

2.1 Overview

Review 10.02.2011, Vol 470. Nature.

TC Glenn. Field Guide to NGS. Mol Ecol Res.

DNA sequencing means determining the nucleotide base sequence of a piece of DNA. Current technologies can only determine the sequence of short DNA molecules (up to 1000 bp), not of entire chromosomes. NGS ("next generation sequencing") collectively refers to technologies invented from 2004, starting with the 454 technology (in other words, everything after Sanger sequencing). Another name is HTS ("high-throughput sequencing").

Existing NGS technologies:

- \bullet Roche: 454
- Illumina: Illumina
- Applied Biosystems: SOLiD
- Life Technologies: Ion Torrent
- Pacific Biosciences: SMRT
- Helicos HeliScope

The characteristics below are only meant to give a rough overview. The information was collected in fall 2011. For up-tp-date characteristics, please refer to the companies' official homepages.



Figure 2.1: Flowgram

2.1.1 Roche 454

Long reads, but few. Throughput $< 1~{\rm Gbp}$ / day. GS FLX with Titanium reagents:

- 1 M reads
- 400 bp read length
- run needs 10 h
- 16 regions, 16 barcodes: 256 samples

GS FLX+ with Titanitum XL+ reagents:

- 1 M reads
- 700 bp read length, up to 1000
- run needs 20 h (?)

The 454 sequencer outputs flowgrams, not sequences per se. repeatedly attempt T,A,C,G,...; measure light intensity. One base \approx intensity 100, proportional for homopolymers Output for sequence TCAG...could be (105,3,99,7,0,112,9,103,...).

2.2 Projects

2.2.1 Simulating Flowgrams

We analyze the reason why DNA sequence reads are often limited to a length of several 100 basepairs, taking the 454 and IonTorrent systems as examples. Recall that these use a fixed flow order $(T,A,C,G)^n$ to interrogate the sequences and record the flow intensity (or H⁺ ion concentration) such that 100 corresponds to approximately one nucleotide.

- Generate many random DNA sequences of length 2000 (longer than we will be able to sequence)
- For each sequence, compute the perfect theoretical flowgram (or ionogram)
- Add saturation and noise: Cut off each intensity at 999 (for example), then add Gaussian noise with mean 0 and standard deviation 30 + i/10 to each intensity, where *i* is the intensity. (You can also play with smaller noise levels.)

Now you already have a very cheap (and unrealistic) simulator for flowgrams. You could add other things now, such as sequence-specific errors, but we want to explore the desynchronisation of DNA reads.

Assume that for computational purposes, we have 1 million copies of the same DNA sequence. There is always a small fraction of DNA molecules that do not incorporate a nucleotide when they are supposed to (say, 0.1% in each stesp, but you should experiment with different error rates). Also, washing away the unbound nucleotides between each flow is never perfect, so a small amount of nucleotides of the wrong type remains. You could assume that in each step, you get 99.9% of the desired (new) nucleotide but 0.01% of the previous composition.

- Keep track of the nucleotide composition in each step. For example, after one step it is just T, after two steps it is 99.9% T and 0.01% A, and so on.
- Keep track of the state of each molecule, allowing for the possibility that nucleotide incorporation fails and for the possibility that the additional nucleotides lead to further reactions
- Generate the flowgrams accordingly, then add the same noise as above.
- Report the noisy flowgrams.
- Reconstruct the sequence from the flowgrams; note the error rate for each position by comparing the reconstructed sequence to the true sequence

2.2.2 Bias in Transcriptome Data?

Take a look at the file solid-5108-demo.csfasta.gz (see the website, about 600 MB). It is essentially a fasta file, but the sequences are in color space, except the initial T.

• Translate the sequences to DNA, assuming no errors. You can throw away the header lines starting with > in the process and the initial T of each sequence.

2 Next Generation Sequencing

• Look at the 6-mer composition at each position in the sequence (note that all sequences have the same length). Compare the composition at each position to the overall composition. Do some positions exhibit a strong bias? To compute the "distance", you can use a chi-square statistic for example.

Note that you can do the translation and counting of 6-mers on-the-fly by piping the output of **zcat** into your own tool, so you don't need to create another big file!

Chapter 3

DNA Microarrays

3.1 Overview

In contrast to sequencing, DNA microarrays measure transcription levels (of either single exons, whole transcripts or genes) not by identifying the molecules by their sequence, but by short complementary DNA probes (oligonucleotides, often of length 25). In the following, we will be discussing high density oligonucleotide microarrays, with the Affymetrix GeneChip being a typical example.

Figure 3.1 shows that a typical transcript is covered by several probes (their location being biased towards the 3' end of the transcript). The probes for all transcripts are synthesized on a chip (the DNA microarray). There can be several million different spots, where each spot contains several thousand copies of the same probe sequence. The size of such a chip is typically around 1.28 cm \times 1.28 cm.

The main idea is that mRNA (or other RNA) from a sample is first reverse transcribed into fluorescently labelled cDNA and then brought in contact with the microarray probes. The cDNA that is complementary to a specific probe will bind in a stable way at that location and give rise to a fluorescent signal whose strength is (approximately) proportional to the original amount of RNA (expression level).

The array's signals are captured with a CCD camera, giving rise to a large image file (extension .dat). On this image, the different spots are recognized and summarized as three values:

- 1. the overall (average) intensity of the spot,
- 2. the standard deviation of the computed average,
- 3. the number of pixels assigned to the spot.

???

Figure 3.1: mRNA with probe locations (biased towards the 3' end), reference sequence, example of a perfect match (PM) and a mismatch (MM) probe.

These values are stored in a file, which we consider to be the "raw" data (although it is already a processed image file). For the Affymetrix GeneChip arrays, it has the extension .cel. We are mostly interested in the intensity values only, but it is noteworthy that the other values (standard deviation and number of pixels) provide good hints about the reliability of the intensity values.

Usually, a gene is represented by 10–20 different probes (or rather PM/MM probe pairs, see below), and an exon is represented by a few such pairs¹. The intensity values of the different probes of a gene must be combined (in a reasonable way) to yield an estimation of the gene expression value. The probes of a gene are spatially "randomly" distributed across the chip, so that local defects of the chip are unlikely to affect all probes of any gene. While one would expect that also the location of the probes within a gene is uniformly distributed, this is not the case. Instead, the probes are generally found close to the 3' end of the gene. This has two reasons: First, mature mRNA ends with a poly-A sequences, so all mRNA molecules can be captured with the same poly-T primer sequence. Since mRNA is not a very stable molecule, it is improbable that the whole mRNA molecule is captured; it is more likely that some fragment of the 3' end is captured instead. Second and relatedly, all mRNA molecules in the cell are continuously degraded; this process starts from the 5' end.

Note the disadvantages of microarrays in comparison to direct sequencing (RNA-seq):

- The gene sequence must be known beforehand in order to design the probes
- The probes must be designed in a way such that they are specific (i.e., the 25-mer sequence chosen for the probes must be unique within the transcriptome, even considering a small error threshold).
- There is a high level of background noise (e.g., due to non-specific binding, NSB).
- Binding strength at a given temperature depends on the probe sequence (especially its GC content) and is not the same for all probes.

Genes, Transcripts, Exons, etc. Carefully note that the signal of a single probe provides only very "local" information: A strong signal at a spot means that there is *some* transcript that contains a 25-mer (almost) perfectly complementary to the probe sequence. Therefore, the 25-mer should be ideally unique within the whole genome in order to uniquely identify the origin of the signal and quantify the expression level of every single gene. However, even this could be problematic if one wants look look deeper than the gene level and quantify the expression of every distinct transcript (e.g., all alternative splice forms of a gene; they usually have many exons, and hence 25-mers, in common. A good compromise is to look at each exon separately and quantify exon expression values separately. These can be, under certain assumptions, deconvoluted into the different transcript expression values using linear algebra, if the many-to-many mapping between exons and transcripts is known (e.g., as a

¹In the following, we will use "gene", but the same statements apply to other transcripts, exons, etc.

bipartite graph). In practice this is very difficult due to the inherent noise in the data, and sequencing approaches will generally be much more accurate.

Data Analysis. There is a long way from these raw values to an interpretable result of the experiment, consisting of several steps, which we divide into low-level analysis steps and high-level analysis steps. In short,

- the low-level analysis pipeline starts with the raw image file and has the goal to produce an $m \times n$ expression matrix X with expression values of m genes in n experiments, whereas
- the high-level analysis pipeline then runs clustering and classification algorithms on this matrix in order to identify differentially expressed genes between two classes (e.g., cancer vs. normal tissue, or survivors vs. non-survivors in a long-term clinical trial), or to cluster genes according to their behavior across all experiments, etc. A small feature set of a good classifier between two clinically or biolgically distinct groups is also called a *gene signature* or a set of *biomarkers*.

Let us summarize the low-level analysis steps:

- 1. image analysis: spot recognition, summarizing per-pixel intensity values into three per-spot values (intensity, standard deviation, number of contributing pixels);
- 2. probe-level (i.e, per-spot) bias correction and background correction of the intensity values; this often involves also a transformation of the values (e.g., to log scale);
- 3. probe-level normalization between several arrays to correct for systematic differences between several experiments and make expression values comparable;
- 4. aggregation of probe-level values into gene-level values;
- 5. optionally, gene-level normalization between several arrays

Sometimes, either only probe-level or gene-level normalization is done; or the aggregation is combined with one kind or both kinds of normalization.

The high-level analysis steps are usually some of the following ones, where we assume that we have a set of (labelled) training experiments, where the class labels (e.g., "cancer" and "control") are known, and perhaps another set of test experiments where the labels are not known:

- 1. identification of differentially expressed genes between the two classes. This can be done by statistical tests (e.g., the t-test) or simple univariate classifiers to find out if there are some genes that perfectly separate the classes; this can also be done by clustering.
- 2. clustering of genes: it is usually observed that several genes behave similarly across all experiments. Of course, we want to know which groups of genes do this (and later why). Interpreting the rows of the expression matrix as vectors in *n*-dimensional space and running clustering algorithms on them provides an answer to this question.

(TODO: figure with a pixelated spot image) (

Figure 3.2: Microarray image analysis.

- 3. clustering of experiments: If there is no class annotation, one can nevertheless ask if the experiments cluster into two (or more) distinct groups that are internally homogeneous. Interpreting the columns of the expression matrix as vectors in *m*-dimensional space and running clustering algorithms on them provides an answer to this question.
- 4. bi-clustering: If a subset of genes behaves similarly in a subset of experiments (but incoherently in others), it may be of interest to cluster both genes and experiments simultaneously. This is called bi-clustering. The problem formally consists of finding a submatrix (a selection of rows and columns) such that all rows of the submatrix look alike and all columns of the submatrix look alike as well. Of course, the number of both rows and columns must be large enough for this to make sense, and there is always a trade-off between the size of the bi-cluster and its homogeneity.
- 5. identification of "gene signatures" that separate the classes; the difference to the previous task is that there, we look at each gene separately, and here we allow larger sets and complex rules to separate the classes. In terms of machine learning, we wish to select features and train a classifier on those features.
- 6. prediction: when we obtain data from a new experiment (and it has been properly normalized to be comparable with the data we trained the classifier on), we can predict its class label and evaluate the precision of the classifier

3.2 Low-Level Analysis

3.2.1 Microarray Image Analysis

This step is automatically performed by the microarray imaging device. While the algorithm could be replaced by another one, few researchers have found a reason to do so, and we will not touch this issue in detail either in this course.

From now on, we assume that the image has been analyzed and the per-pixel intensity values have been converted into probe-level values that are stored in a .cel file (see Projects). For each spot, this contains a triple of values: (average intensity, standard deviation of intensities, number of considered pixels). The file also contains a (usually empty) list of (manually) masked spots and a list of spots considered as outliers by the image analysis software. Usually, one should discard the value of each spot contained in any of those two lists.

3.2.2 Noise and Bias Correction

As far as sources of variation are concerned, we distinguish between

- **noise** (stochastic effects that are too random to model systematically) like DNA quality in the sample, unspecific hybridization, random defects on the chip, probe density on a particular spot, etc., and
- **bias** (systematic effects that are similar on several measurements or probes and can hence be estimated from the data) like probe binding affinity, DNA concentration in the sample, photodetection, etc.

At least on older Affymetrix chip designs, spots come in pairs of a so-called PM (perfect match) and MM (mismatch) spot. The PM spot contains the actual probes (oligonucleotides of length 25) of interest. The MM spot contains (almost) copies of those probes, with the exception that the 13th (middle) nucleotide has been replaced by its complement. The idea behind having PM and MM spots was that the MM spot should not measure the signal (since the binding of the target molecule to the MM probes should be unstable because of the mismatch), but the non-specific binding properties should be the same as the PM probe. Therefore, the MM probe should measure the "background" that can be subtracted from the PM probe intensity to get the actual signal. Thus, the signal S_i of the *i*-th probe (pair) should in fact be $S_i = P_i - M_i$, where P_i denotes the intensity of the *i*-th PM probe and M_i denotes the intensity of the *i*-th MM probe.

Unfortunately, when the paired design was introduced, nobody checked this assumption. Experiments later showed that in up to 30% of all probe pairs, the MM signal was actually higher than the PM signal, which would lead to nonsensical negative S_i values.

One possible explanation is that in fact, there exist (highly-expressed) transcripts (genes) that have a region that is almost perfectly complementary to the MM probe, thus the MM probe in fact acts as a PM probe for an entirely different gene; it just is not annotated as such. Therefore, several researchers have proposed to simply ignore the MM values and focus on the PM values alone.

In principle, if all transcripts that hybridize to each PM and each MM probe are known along with the hybridization strength, the whole transcript expression quantification problem can be solved (theoretically) at the probe level. We now discuss the corresponding model, but then will not pursue it further, because it is not used in practice.

For a given probe (spot) p, let T_p be the set of targets (transcripts) that hybridize significantly to p. Note that the PM/MM idea assumes that $|T_p| = 1$ for all PM probes and $|T_p| = 0$ for all MM probes. Here we do *not* make this assumption. Let (y_p) be the intensity measurement at probe p. Let (x_t) be the true relative expression level of transcript t; the scale (units) is arbitrary, so we may assume a that $\sum_t x_t = 1$ or a similar condition. We also assume (unrealistically) that we know the binding affinity $a_{p,t}$ between p and t and collect them in a matrix $A = (a_{p,t})$. Now obviously in vector-matrix form,

$$y \approx A \cdot x,$$

up to noise and bias. We see that this can only be solved if the number of probes is at least as large as the number of transcripts and if the matrix A has full rank.

What are the characteristics of the noise for probe p? We assume that there is both *additive* and *multiplicative* noise for each probe. Thus

$$y_p = (Ax)_p \cdot \mathbf{e}_p^\eta + \nu_p,$$

where η_p and ν_p are independent normal random variables for each probe, with variances that do not depend on the probe, i.e., $\eta_p \sim \mathcal{N}(0, \sigma_{\text{mult}}^2)$ and $\nu_p \sim \mathcal{N}(\sigma_{\text{add}}^2)$ for all p.

To account for several experiments (see normalization below), we may introduce different offsets α_j and scale factors β_j for different experiments j, i.e.,

$$y_{pj} = (Ax_j)_p \cdot \mathbf{e}_{pj}^{\eta} + \nu_{pj}$$

We assume that there is some base signal level (background, offset) α for the whole chip (independently of the probe), and an additive noise term ν with mean zero, independently of the probe. Furthermore, there is additional noise that depends on the magnitude of the probe signal. Thus,

•••

where we assume that $\nu \sim \mathcal{N}(0, \sigma^2)$ and

(TODO: Transformation)

3.2.3 Normalization

The purpose of normalization is to make measured values from different experiments comparable, so they can be analyzed together. Normalization is not usually necessary if only a single experiment is considered and never compared to another experiment. However, one should note that the absolute expression values from a single experiment are meaningless then. Only their magnitude relative to each other is meaningful.

3.1 Example (Normalization). Consider the following intensities from four genes (1,2,3,4) and two experiments (A,B).

| | А | В |
|---|----|-----|
| 1 | 11 | 22 |
| 2 | 9 | 18 |
| 3 | 55 | 110 |
| 4 | 2 | 4 |

Essentially, the experiments show the same result; however, in experiment B, all values are twice as high as in experiment A. This is more likely explained by systematic external differences (twice the amount of sample DNA, or twice the concentration of the fluorescent dye, or a gain factor of 2 in the imaging device) rather than by the fact that every gene in the cells in experiment B is twice as active as in experiment A. Therefore, there are no (biologically interesting) differences between A and B. A naive analysis without normalization, however, would highlight every gene as differentially expressed by a factor of 2. \heartsuit

The basic assumption for any normalization method is that the global properties of the distribution of intensity values does not change. Normalization methods differ in the exact assumption about the invariant properties and in the method that is employed to ensure that the invariants hold after normalization.

All normalization that we discuss below can be applied either on the probe level (before summarizing individual probe intensities into gene expression values) or on the gene level (after the aggregation step), or even at both times.

- Simple scaling according to mean or median
- Quantile normalization
- lowess regression
- Variance stabilization

3.2.4 Aggregation

Each gene (or other object of interest, like exons) is represented by more than one probe on the chip. Therefore, to estimate gene (or exon) expression values, the intensities of the different probes belonging to the gene (or exon) must be aggregated or summarized into a single consensus value representing the best possible estimate.

We assume that at this step, probe biases (such as different binding affinities) have been corrected. Therefore, all probe intensities belonging to the same gene should be approximately equal. In reality of course, this is never the case, and some method of aggregation remains necessary.

- average
- robust average: median
- robust average: trimmed mean

3.3 High-Level Analysis

As mentioned previously, high-level analysis refers to the identification of "interesting" features (genes) that separate the different experiments into different classes.

3.3.1 Identification of Differentially Expressed Genes

3.3.2 Clustering

We may either cluster experiments or genes. The methods remain the same; in both cases the features are real-valued vectors from some high-dimensional space \mathbb{R}^d .

3.3.3 Bi-Clustering

3.3.4 Classification and Prediction

3.4 Projects

3.4.1 Reading .cel files

Read and understand the binary .cel file format, for example at (version 4 only!). Write a module or library that reads a .cel file into a data structure, such that you can easily access the intensity value at each spot given by its coordinates (i, j), where i is the row number (called y-coordinate in the file description) and j is the column number (called x-coordinate in the file description). The following is some example Python 3 code showing how your module should be used.

```
cel = CELFile('patient123.cel')
cel.read() # reads the whole file
M = cel.intensities # numpy matrix of intensities
print(M[3,2]) # value in row 3, column 2 (i.e., x=2, y=3)
```

Test your module with the files 161.cel, 162.cel, 163.cel from the website. This is data from human exon arrays with approximately 4 million spots each; each file has a size of approximately 60 MB.

Optional extensions:

- Plot an 2560 x 2560 image from the matrix M and visualize the intensities.
- Extend the read method such that spots marked as "masked" or "outlier" have their intensity values set to NaN, e.g. by cel.read(nanmasked=True, nanoutliers=True).

3.4.2 Normalization

Consider the three datasets 161.cel, 162.cel, 163.cel from the website.

- Plot histograms or box plots or violin plots (histograms as a sort-of-boxplot) of their intensity distribution. Use both a linear scale and a log scale.
- Now assume that 161 is a reference dataset. Use quantile normalization to adjust 162 and 163 to match the intensity distribution of 161. Do the plots again and observe that they are now all identical.
- Now use the original datasets again, and generate MA-plots of 162 vs 161 and 163 vs 161, using a log-scale.
- Do a lo(w)ess (locally weighted scatterplot smoothing) normalization of 162 vs 161 and 163 vs 161, on the log-transformed data, using linear polynomials locally. You may want to exclude low-intensity values for reasons discussed in the lecture. Show the lowess curve in the scatterplot. Re-plot the intensity histograms of 162 and 163 after normalization.

CHAPTER 4

Metabolomics

4.1 Overview

Metabolism contains the transport, interaction and processing of chemical compounds in a biological organism and is the main function in living cells. Intermediates and products of metabolism are called metabolites. Metabolomics is the research of methods for identification and quantification of metabolites. Several methods of analyzing are already used to identify metabolites in tissues, liquids or gases:

- Gas chromatography mass spectrometry (GC/MS)
- Liquid chromatography mass spectrometry (LC/MS)
- Nuclear magnetic resonance (NMR) spectroscopy
- High performance liquid chromatography (HPLC)
- Ion mobility spectrometry (IMS)

In this chapter we will focus on the IMS technology with respect to the MCC/IMS device developed by the Leibniz-Institute ISAS¹ and the methods of analyzing MCC/IMS measurements.

¹Institute for Analytical Sciences, www.isas.de

4.2 MCC/IMS

Ion mobility spectrometry (IMS) is a technology to analyze the presence and concentration of compounds in the air. It is already used to detect drugs, explosives or other dangerous chemicals in sensitive places like airports. In the last ten years IMS technology became more and more important by its property to measure compounds with ambient pressure. By coupling an IMS with a multi-capillary column (MCC) we obtain a much higher level of precisely data. This fact makes IMS more attractive for scientific and economical fields like exhalation analysis in medicine. It is possible to measure the conentration of metabolits in the breath which denunciate specific diseases. Several diseases like chronic obstructive pulmonary disease (COPD), sarcoidosis or lung cancer as well as a rejection reaction of an engrafted lung can be diagnosted early. By coupling the IMS with MCC the precision of the measurement increases significant and thus the complexity of analysis.

4.2.1 Measurement

For the measurements an IMS device with a ${}^{63}NI \beta$ -ionization source will be used. The device itself is devided into two parts, the ionization chamber and the drift tube. In the first phase the gas reaches the ionization chamber. In this step the neutral molecules will be formed by reaction ions into product ion in chemical reactions. These product ions will be used to identify the specific molecules. In the second phase the ion shutter, which sepatates the ionization chamber from the drift tube, opens. Driftrings, which are placed in a uniform distance along the tube, generate an electric field E in the tube. The product ions will be pulled by the electric field through the tube. The ion velocity v is related to the electric field. Hence the ion mobility is inverse proportional to the drift time d. Molecule properties like weight, structure, temperature and polazibility lead to a characteristic drift time for every single compound. At the end of the drift tube the electric charge of the impacting ions are transfered on a Faraday plate. We obtain a time / signal measurement. The whole process takes approximate 50 milliseconds.



Figure 4.1: Lateral cut throuth an IMS device

different compounds with identical mean drift time. For this reason the IMS device is coupled with a MCC device which separates the compounds before they will pushed into the IMS device. By the occuring delay only a little part of the compounds in the breath gas will be measured. After a specific retention time r a compound reaches the IMS device. Thus an IMS measurement will be executed several times, approximate two times a second. We obtain a two dimensional spectrum S with r, d as the two-dimensions and the electric charge as the measured signal $S_{r,d}$. Measurements can be visualized as a two-dimensional heatmap, shown in Figure 4.2. A whole MCC/IMS measurement depending on the adjusted resolution consists of 3 - 75 million datapoints. Regions with a high signal intensity are called peaks. This peaks consists of several hundret datapoints.



Figure 4.2: Heatmap of an MCC/IMS measurement. X-axis: drift-time d im ms; Y-axis: retention-time r in seconds; intensity increase: white - blue - purple - red - yellow

A feature that appears in all MCC/IMS measurements is the reaction-ion peak (RIP), visible as a continious run at drift time d = 17.5ms in Figure 4.2. The nitrogen that is pushed through the drift tube against the electric field to clean the tube is also ionized. Because the cleaning process happens permanently the RIP appears in every single IMS spectrum and thus as a solid line in the whole MCC/IMS measurement.

4.2.2 Data file

The measurement files generated by the MCC/IMS device are IUPAC standard conform. It is an plain text file and structured as a comma separated values (CSV) file. The file is divided into two parts. The first 130 lines contain meta informations like MCC temparature, carrier/drift gas, carrier/drift flow etc. In the remaining lines the data is stored with the following structure:

4 Metabolomics

- Line 131: lists the retention time t_R in seconds as the timepoints when a 1D IMS spectrum measurement began
- Line 132: lists the spectra no. $0 \dots$)
- Line 133 ...:
 - 1. column: inverse mobility $1/K_0$ in s/cm^2
 - -2. column: corrected drift time $t_{D,corr}/ms$ with respect to the grid opening time
 - 3. column: 1st IMS spectrum
 - 4. column: 2nd IMS spectrum
 - ...

More detailed informations and a complete specification are written by Vautz et al.².

4.3 Preprocessing

As we can see in the visualization the data are strong noised and at this point unuseable, because of the RIP we still got in the measurement. Before we analyze the data we first have to invest computation to preparate the data for better results in further analysis.

4.3.1 Baseline correction

The main aim is to punish columns in retention time with a high mean/median value. Normal columns like at d = 35ms in Figure 4.2 have just a few high signal values but many low values. On the other hand RIP has a high mean/median value. The idea is to substract the median from all signal values in a column. Let $m_d = median(S_{1,d}, \ldots, S_{|R|,d})$ the median over all signal values at drift time d. The Baseline corretion is defined as follow

$$\forall d \in D : \forall r \in R : S_{r,d} = \begin{cases} S_{r,d} - m_d & \text{for } S_{r,d} - m_d > 0, \\ 0 & \text{else} \end{cases}$$
(4.1)

²Vautz W, Bödeker B, Bader S, Baumbach JI: Recommendation of a standard format for data sets from GC/IMS with sensor-controlled sampling. Springer Berlin / Heidelberg: 71-76

4.4 Modeling peaks

The challenge is to detect the peaks to draw conclusions from the position and volume of the peak about the compound. A good approach for detecting is to define a uniform structure of the peaks and try to find it in the measurement. For this issue mathematical functions are suited well for a peak description. Statistical distribution functions like normal distribution often are used to model/approximate real data. Thus the data (the peaks) consisting of several hundret datapoints, can be described with just a few parameters depending on the choice of distributions.

Exactly like in the measurement the model can be described as the function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, with retention time r and drift time d as input and a concentration as output. To choose adequate functions for the model we have to look at the horizontal and vertical cross section of a peak:



Figure 4.3: Horizontal and vertical cross section of a few peaks; x-axis in {ms, s}, y-axis in signal unit

Although the curves of the horizontal sections seem to be symmetric they possess a little skewness. At the curves of the vertical sections the skewness is clearly visible. In drift time the light skewness can be explained by the construction of the IMS device. While the ionized molecules are pulled through the drift tube the drift gas is pushed in opposite direction

through the tube to blow the deionized molecules of the tube. The resulting interactions lead to characteristic curves for every metabolite.

The two dimensional model must have a skewness in retention time as well as in drift time. Hence the inverse gaussian distribution IG was chosen. While the original distribution just has two parameters with fixed origin the modified distribution additional has a slide parameter which enables to set the origin and is defined as follows:

$$IG_{\mu,\lambda,o}(x) = \left(\frac{\lambda}{2\pi(x-o)^3}\right)^{\frac{1}{2}} \cdot \exp\left(-\frac{\lambda((x-o)-\mu)^2}{2\mu^2(x-o)}\right)$$
(4.2)

The 2D peak model function is a product of IG in retention time r, IG in drift time d and is defined as follows:

$$P_{\mu_r,\lambda_r,o_r,\mu_d,\lambda_d,o_d}(r,d) = \begin{cases} IG_{\mu_r,\lambda_r,o_r}(r) \cdot IG_{\mu_d,\lambda_d,o_d}(d) & \text{for } r > o_r \land d > o_d, \\ 0 & \text{else} \end{cases}$$
(4.3)

It is an advantage that the model is still normalized. To describe a weight/volume we multiply the model with the weight/volume parameter v, so $M_{\mu_r,\lambda_r,o_r,\mu_d,\lambda_d,o_d,v}(r,d) = v \cdot P_{\mu_r,\lambda_r,o_r,\mu_d,\lambda_d,o_d}(r,d)$. Hence a measurement can be modeled as a mixture of weighted models. The easy model makes the use of the EM algorithm attractive for further computation.

A problem that occures in the measurement is the so called "ion theft". Because in one IMS measurement the amount of reaction ions is restricted not every molecule in the ionization chamber will be ionized. When compounds with a high concentration are measured the probability that two ions bond together to a new artificial molecule called dimer increases. This dimer will also be measured but has has lower mobility, thus it appears later in drift time. We can see this effect in Figure 4.2 at retention time r = 93s. The monomer at d = 24ms seems to be narrowed because of its high concentration a dimer at d = 34ms has formed. When we add the dimer values with the values of the narrowed monomer we obtain the original monomer.

4.5 EM algorithm

The EM algorithm is a statistical method to estimate parameters in mixture models. Its an iterative process with two alternating phases: In the expectation "E" phase hidden values are estimated. This values discribe the weight of every datapoint to all models. The sum of all weights for one datapoint equals 1. The weight is estimated by the probability density function of the specific model. In the maximization "M" phase all parameters for all models will be optimized with maximum-likelihood-estimators (MLE).

4.5.1 Set up a maximum-likelihood-function

To use the EM algorithm we have to set up a maximum-likelihood-function first. Let's have following initial situation: Let R be set of all retention times and D the set of all drift times, a measurement with $n = |R| \cdot |D|$ datapoints with $x_i \in R \times D$ in range $1 \le i \le n$ and c models/peaks in range $1 \le j \le c$. Every model is weighted in the measurement with weight ω_j . The sum of all weights of the models equals 1, so $\sum_{j=1}^{c} \omega_j = 1$.

The hidden values describe the membership $W_{i,j}$ of a datapoint x_i to a model j, when x_i definitly belongs to the model j. Let $\sum_{j=1}^{c} W_{i,j} = 1$ and $\mathbb{P}_{\Theta,\omega}(W_{i,j} = 1) = \omega_j$.

Furthermore every model has a specific distribution function $P_{\Theta_j}(x_i)$, with Θ_j as the parameter vertor. The distribution function in this case is equation 4.3. The probability to describe a datapoint x_i with a single model j is

$$\mathbb{P}_{\Theta,\omega}(x_i, W_{i,j} = 1) = \omega_j \cdot P_{\Theta_j}(x_i).$$
(4.4)

To describe the mixture ratio we set up the probability function for one datapoint with a c-dimensional vector W_i

$$\mathbb{P}_{\Theta,\omega}(x_i, W_i) = \prod_{j=1}^{c} \left[\omega_j \cdot P_{\Theta_j}(x_i) \right]^{W_{i,j}}.$$
(4.5)

For all datapoints we obtain following likelihood function

$$L_{x,W}(\Theta,\omega) = \prod_{i=1}^{n} \prod_{j=1}^{c} \left[\omega_j \cdot P_{\Theta_j}(x_i) \right]^{W_{i,j}}.$$
(4.6)

Because the logarithm of the likelihood has its maximum at the same position as the normal function, thus we work with the so called *log-likelihood* to obtain an easier equation.

$$\mathcal{L}_{x,W}(\Theta,\omega) = \sum_{i=1}^{n} \sum_{j=1}^{c} W_{i,j} \cdot \left[\log \omega_j + \log P_{\Theta_j}(x_i)\right].$$
(4.7)

4.5.2 Expectation step

To estimate the hidden values we have to set up a target function with respect to the estimated parameters Θ^0 and the weights of the models ω^0 from the last iteration. It optimizes the parameters Θ^* and ω^* with respect to the new estimated hidden values. The target function is the expected value of the log-likelihood $\mathcal{L}_{x,W}(\Theta,\omega)$ over $W_{i,j}$. $W_{i,j}$ is the conditional probability under the condition that Θ^0 and ω^0 are known.

$$f_{\Theta^{0},\omega^{0},x}(\Theta,\omega) = \mathbb{E}_{(W|(x;\Theta^{0},\omega^{0}))}\mathcal{L}_{x,W}(\Theta,\omega)$$

$$= \mathbb{E}_{(W|(x;\Theta^{0},\omega^{0}))}\left[\sum_{i=1}^{n}\sum_{j=1}^{c}W_{i,j}\cdot\left[\log\omega_{j}+\log P_{\Theta_{j}}(x_{i})\right]\right]$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{c}\mathbb{E}_{\Theta^{0},\omega^{0}}\left[W_{i,j}|x_{i}\right]\cdot\left[\log\omega_{j}+\log P_{\Theta_{j}}(x_{i})\right]$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{c}W_{i,j}^{0}\cdot\left[\log\omega_{j}+\log P_{\Theta_{j}}(x_{i})\right].$$

$$(4.8)$$

4.5.3 Estimate the hidden values

To estimate $W_{i,j}^0$ for the target function we consider the estimated parameters Θ^0 and ω^0 from the last iteration

$$W_{i,j}^{0} = \mathbb{E}_{\Theta^{0},\omega^{0}} \left(W_{i,j} | x_{i} \right).$$
(4.9)

The expactation value is the probability that $W_{i,j} = 1$ under the condition that x_i is provided and the parameters Θ^0 and ω^0 are set up

$$W_{i,j}^{0} = \mathbb{P}_{\Theta^{0},\omega^{0}} \left(W_{i,j} = 1 | x_{i} \right).$$
(4.10)

With the Bayes' theorem for conditional probability we can build an equation with well known probability functions

$$W_{i,j}^{0} = \frac{\mathbb{P}_{\Theta^{0},\omega^{0}}\left(x_{i}|W_{i,j}=1\right) \cdot \mathbb{P}_{\Theta^{0},\omega^{0}}\left(W_{i,j}=1\right)}{\mathbb{P}_{\Theta^{0},\omega^{0}}\left(x_{i}\right)} = \frac{\mathbb{P}_{\Theta^{0},\omega^{0}}\left(x_{i},W_{i,j}=1\right)}{\mathbb{P}_{\Theta^{0},\omega^{0}}\left(x_{i}\right)}.$$
(4.11)

The probability for $\mathbb{P}_{\Theta^0,\omega^0}(x_i, W_{i,j} = 1)$ we already definied in equation (4.4). $\mathbb{P}_{\Theta^0,\omega^0}(x_i)$ is the probability for the datapoint x_i which is the sum of all weighted models at this point $\sum_{k=1}^{c} \omega_k^0 \cdot P_{\Theta_k^0}(x_i)$. Now $W_{i,j}^0$ is defined as follows

$$W_{i,j}^{0} = \frac{\omega_{j}^{0} \cdot P_{\Theta_{j}^{0}}(x_{i})}{\sum_{k=1}^{c} \omega_{k}^{0} \cdot P_{\Theta_{k}^{0}}(x_{i})}.$$
(4.12)

Equation (4.12) also ensures the condition $\sum_{j=1}^{c} W_{i,j} = 1$.

4.5.4 Maximization step

Now we use the heidden values to optimize Θ^* und ω^* which maximize the target function $f_{\Theta^0,\omega^0,x}(\Theta,\omega)$. Because ω^* is independent to the model we can set up a generic formula. The computation of the maximizer in Θ^* we describe further in section 4.5.5.

First we have to ensure that the constraint $\sum_{j=1}^{c} \omega_j = 1$ after the m step is still valid which can be solved with the help of the Lagrange multiplicator. Let

$$\mathbb{L}(\omega) = \beta \left(-1 + \sum_{j=1}^{c} \omega \right) + \sum_{i=1}^{n} \sum_{j=1}^{c} W_{i,j}^{0} \cdot \left[\log \omega_j + \log P_{\Theta_j}(x_i) \right].$$
(4.13)

be the new target function with Lagrange multiplicator β . Computing the first derivate

$$\frac{\partial \mathbb{L}(\omega)}{\partial \omega} = \beta + \sum_{i=1}^{n} \frac{1}{\omega_j} W_{i,j}^0, \qquad (4.14)$$

setting to 0 and solving for ω_i we obtain

$$\omega_j = \frac{\sum_{i=1}^n W_{i,j}^0}{-\beta}.$$
(4.15)

To solve β we use the constraint $\sum_{j=1}^{c} \omega_j = 1$ by extending the equation (4.15) with $\sum_{j=1}^{c}$, let

$$1 = \frac{\sum_{i=1}^{n} \sum_{j=1}^{c} W_{i,j}^{0}}{-\beta}.$$
(4.16)

 $\sum_{j=1}^{c} W_{i,j}^{0} = 1$ is already defined in the last section, thus $\beta = -n$. After inserting β in (4.15) we obtain the following maximizing formula

$$\omega_j^* = \frac{1}{n} \sum_{i=1}^n W_{i,j}^0, \tag{4.17}$$

which can be interpretated as the mean weight of all coherent hidden values to a model j. Indeed the specification of the parameters in Θ^* depend on the model but we can set up a generic approach for all parameters l in Θ_j , let

$$\frac{\partial f_{\Theta^0,\omega^0,x}(\Theta,\omega)}{\partial \Theta_{j,l}} = \sum_{i=1}^n W_{i,j}^0 \cdot \frac{\partial \left(\log P_{\Theta_j}(x_i)\right)}{\partial \Theta_{j,l}} = 0.$$
(4.18)

4.5.5 Maximum-Likelihood-Estimators (MLE)

To compute a function that maximizes the target function we have to derivate the generic approach (4.18) with respect to the parameter Θ_l and solve it for the parameter. Let

$$\log P_{\Theta_j}(r,d) = \frac{1}{2} \log \left(\frac{\lambda_d}{2\pi (d-o_d)^3} \right) + \left(-\frac{\lambda_d ((d-o_d) - \mu_d)^2}{2\mu_d^2 (d-o_d)} \right) + \frac{1}{2} \log \left(\frac{\lambda_r}{2\pi (r-o_r)^3} \right) + \left(-\frac{\lambda_r ((r-o_r) - \mu_r)^2}{2\mu_r^2 (r-o_r)} \right).$$
(4.19)

be the logarithm of the model distribution. As the last issue before computing we have to consider the signal values s_i . At this point the derivate function (4.18) would treat every datapoint as they would have the concentration 1. Hence s_i will be inserted into the function

$$\frac{\partial f_{\Theta^0,\omega^0,x}(\Theta,\omega)}{\partial \Theta_{j,l}} = \sum_{i=1}^n W_{i,j}^0 \cdot s_i \cdot \frac{\partial \left(\log P_{\Theta_j}(x_i)\right)}{\partial \Theta_{j,l}} = 0.$$
(4.20)

Derivating μ_r^*

We compute the first parameter by derivating (4.20) with respect to μ_r and solve for μ_r . Because this derivation doesn't depend on the model index j, it will not be notated here, execpt $W_{i,j}^0$. The maximizer for μ_d will be analogious computed.

$$\frac{\partial f_{\Theta^0,\omega^0,x}(\Theta,\omega)}{\partial \mu_r} = \sum_{i=1}^n W_{i,j}^0 \cdot s_i \cdot \left(-\frac{\lambda_r}{2(r-o_r)} \cdot \frac{-2\left((r_i-o_r)-\mu_r\right)\mu_r^2 - 2\mu_r\left((r_i-o_r)-\mu_r\right)^2\right)}{\mu_r^4} \right)$$
$$\mu_r^* = \frac{\sum_{i=1}^n W_{i,j}^0 \cdot s_i \cdot r_i}{\sum_{i=1}^n W_{i,j}^0 \cdot s_i} - o_r \tag{4.21}$$

The problem occures that for copmuting μ_r^* the parameter o_r^* has to be already known. We solve this problem by copmuting μ_r^* by using o_r^0 first, solve o_r^0 using μ_r^* and finally μ_r^* by using o_r^0 .

Derivating λ_r^*

For an easier notation μ_r^* will be replaced by μ_r . The maximizer for λ_d will be analogious computed.

$$\frac{\partial f_{\Theta^{0},\omega^{0},x}(\Theta,\omega)}{\partial \lambda_{r}} = \sum_{i=1}^{n} W_{i,j}^{0} \cdot s_{i} \cdot \left(\frac{1}{2\lambda_{r}} - \frac{((r_{i} - o_{r}) - \mu_{r})^{2}}{2\mu_{r}^{2}(r_{i} - o_{r})}\right)$$
$$\lambda_{r}^{*} = \frac{\sum_{i=1}^{n} W_{i,j}^{0} \cdot s_{i}}{\sum_{i=1}^{n} W_{i,j}^{0} \cdot s_{i} \cdot \left(\frac{1}{r_{i} - o_{r}} - \frac{1}{\mu_{r}}\right)}$$
(4.22)

Derivating o_r^*

$$\frac{\partial f_{\Theta^0,\omega^0,x}(\Theta,\omega)}{\partial o_r} = \sum_{i=1}^n W_{i,j}^0 \cdot s_i \cdot \left(\frac{3}{2(r_i - o_r)} - \frac{-4\mu_r^2\lambda_r(r_i - o_r - o_r)(r_i - o_r) + 2\mu_r^2\lambda_r(r_i - o_r - \mu_r)^2}{4\mu_r^2(r_i - o_r)}\right)$$
$$0 = \sum_{i=1}^n W_{i,j}^0 \cdot s_i \cdot \left(\frac{3}{\lambda_r(r_i - o_r)} + \frac{1}{\mu_r^2} - \frac{1}{(r_i - o_r)^2}\right)$$
(4.23)

At this point its not possible to solve (4.23) for o_r . Thus the Newton method will be used, to compute o_r . Its an approximization method to find roots of given functions. Let $x^{new} = x^{old} - \frac{f(x^{old})}{f'(x^{old})}$ be the generic approximization function and α the iteration step, thus we obtain the following function

$$o_r^{\alpha+1} = o_r^{\alpha} - \frac{\sum_{i=1}^n W_{i,j}^0 \cdot s_i \cdot \left(\frac{3}{\lambda_r(r_i - o_r^{\alpha})} + \frac{1}{\mu_r^2} - \frac{1}{(r_i - o_r^{\alpha})^2}\right)}{\sum_{i=1}^n W_{i,j}^0 \cdot s_i \cdot \left(\frac{3}{\lambda_r(r_i - o_r^{\alpha})^2} - \frac{1}{(r_i - o_r^{\alpha})^3}\right)}.$$
(4.24)

The maximizer for o_d will be analogious computed, too.

Derivating v

Because the volume under the distribution density equals 1 we inserted v into the model. Since v does not appear in the likelihood function we can't derivate a maximizer like we did in the recent derivations. The volume of the peak is related to the weight ω of the model in the measurement. To cumpute the volume we just have to multiply the sum of all signals in the measurement with the weight of the model

$$v = \sum_{i=1}^{n} s_i * \omega_j. \tag{4.25}$$

4.5.6 Algorithm

In the first step the datapoints have to be captured/measured or loaded. It is necessary that the amount of peaks and their approximate positions are already known. For all values in W and in Θ we use two matrices. Additionally we insert into the matrix Θ the background model with the probability function $P(x_i) = \frac{1}{n}$. Since the data is biased through the IMS device it is important to preprocess the data with several filter like low-pass, ion reconstruction or baseline correction to restore the data.

As a default value we assign every hidden parameter $W_{i,j}$ with the uniform distributed probability $\frac{1}{c}$. In Θ we assign all $\omega = \frac{1}{c}$, too. The origin parameter should have a value before the 1% quantile of the distribution. This is important, because every retention time $r \leq o_r$ will be assigned to 0. The other parameters can be initialized with any little positive value. Of course the parameters can be estimated better with a heuristic in the preprocessing phase.

After the preprocessing and initialization we store the actual model parameters for the further convergence test. In the E step the hidden values are estimated. We have to watch out that the probability for background model shall not be computed with the IG probability density function. The probability function for the background model doesn't change.

Finally in the M step all model parameters are estimated by their specific maximizing function shown in 4.5.5. The only exception is the background model which has no model parameters. Only the weight ω must be computed for the BGM. After every EM step we have to check the convergence of the parameters. There are two possibilities: We can compute the *log-likelihood* (4.7) and control the difference between the actual and the last likelihood value. The other way is to control every model parameter which is a better option since the situaltion could be possible that after an iteration the log-likelihood would not change significantly but the model values. Let ϵ be a threshold. For a successful convergence check following equation must be valid

$$\sum_{j=1}^{c} \sum_{l=1}^{L} \left(\frac{\Theta_{jl}^{0}}{\Theta_{jl}^{*}} - 1 \right)^{2} \le \epsilon.$$

$$(4.26)$$

When the condition is satisfied the EM phase interrupts. The lower the threshold the better the models are fitted the more EM iterations are executed. Tests have shown that $\epsilon = 10^{-4}$ yields the best solutions. The last step is to calculate the volumes of the models except the BGM. Figure 4.4 illustrates the whole process pipeline.



Figure 4.4: Workflow of EM-algorithm

Uniform Parameters

Depending on the start parameters several independent EM executions provide different values in fact of the same parameters. The reason for this feature is, because we inserted the offset o parameter in IG distribution. This parameter allows to form similar curves with totally different parameters. Let

- $P_1: \mu = 10.93, \quad \lambda = 106.42, \quad o = 9.63$
- $P_2: \mu = 20.41, \quad \lambda = 786.08, \quad o = 0$

be two different parameterized IG models. Although the parameters are complete different, the curves are very similar, visible in Figure 4.5. For further computation like peak alignment/comparisn we need uniform parameters. Because the curves are similar, their modes are close together. Let

$$x_{mode} = \mu \left[\left(1 + \frac{9\mu^2}{4\lambda^2} \right)^{\frac{1}{2}} - \frac{3\mu}{2\lambda} \right] + o \qquad (4.27)$$

be the formula for the computation of mode in IG distributions. This example shows that the difference between P_1 and P_2 in μ is about 9.48, in λ about 679.66 and in o about 9.63. The two curves also have the same mean and variance/standard deviation. The uncorrected mean is $e = \mu_r + \lambda_r$ and the standard deviation is $\sigma = \sqrt{\mu^2/\lambda}$.



Figure 4.5: Different parameters but similar curves

Example for Uniform Parameter

All in all three parameter als suitable as uniform parameter, namely (x_{Mod}, e, σ) . As an example Figure 4.6 shows a section of a measurement. We can see three models labeled with M1, M2, M3 from bottom up. The results are shown in the Table 4.1. We also can see that the model parameters differ significant. On the right hand side of the table the uniform parameter are notated which derivation is very low.



Figure 4.6: Section of IMS measurement with three peaks, enumerated from down to up

The uniform parameters have the advantage that the original model parameters can be back-calculated. The formulas are follows

$$p = \frac{((x_{Mod} \cdot (-2 \cdot e - x_{Mod}) + 3 \cdot (e^2 - \sigma^2)))}{(2 \cdot (x_{Mod} - e))}$$

$$q = -\frac{(x_{Mod} \cdot (-3 \cdot \sigma^2 - e \cdot x_{Mod}) + e^3)}{(2 \cdot (x_{Mod} - e))}$$

$$o = -\frac{p}{2} - \sqrt{\left(\left(\frac{p}{2}\right)^2 - q\right)}$$
(4.28)

$$\mu = e - o \tag{4.29}$$

$$\lambda = \frac{\mu^3}{\sigma^2}.\tag{4.30}$$

4 Metabolomics

| | Mo | del param | Uniform parameter | | | |
|------|---------|---------------------|-------------------|-----------|-------|----------|
| | μ_r | μ_r λ_r | | r_{Mod} | e | σ |
| | 4.67 | 56.25 | 26.53 | 30.65 | 31.20 | 1.34 |
| М | 7.48 | 259.84 | 23.68 | 30.85 | 31.16 | 1.26 |
| 1111 | 16.74 | 3501.24 | 14.38 | 31.00 | 31.12 | 1.16 |
| | 6.36 | 161.72 | 24.81 | 30.82 | 31.18 | 1.26 |
| | 7.08 | 120.53 | 35.77 | 42.26 | 42.85 | 1.71 |
| M | 11.71 | 588.70 | 31.11 | 42.27 | 42.82 | 1.65 |
| 112 | 6.36 | 93.77 | 36.43 | 42.19 | 42.81 | 1.65 |
| | 4.69 | 28.81 | 38.17 | 41.85 | 42.86 | 1.89 |
| | 8.97 | 150.50 | 45.54 | 53.75 | 54.51 | 2.18 |
| 14 | 7.23 | 75.08 | 47.28 | 53.54 | 54.51 | 2.24 |
| 1113 | 8.46 | 162.14 | 45.90 | 53.72 | 54.36 | 1.93 |
| | 5.62 | 32.19 | 48.91 | 53.25 | 54.54 | 2.34 |

| Table 4.1: | $\operatorname{Comparism}$ | between | model | parameters | and | uniform | parameters | in | ${\rm multiple}$ | mea- |
|------------|----------------------------|------------|------------------------|------------|------|---------|------------|----|------------------|------|
| | surements v | with diffe | erent st | art parame | ters | | | | | |

4.6 Recourse-constrained detection

The main aim in near future is to produce a mobile MCC/IMS device in size of a cell phone. Because in such devices the resources like storage or battery power are constrained it is necessary to build software that handles gentle with the hardware.

In this case there is no question to save the whole measurement. Unfortunatly the described method is not applicable any more, because it needs the whole spectrum. The idea is to store just a few IMS spectra while the measurement and operate only in this window and finally discard the raw data after processing.

4.7 Projects

4.7.1 Gaussian Bell

Estimate the parameters of gaussian bells in a multi model measurement using the points written in the file gaussdata.txt. Assume there are four models in the measurement. Plot the data using gnuplot or any other plotting tool. Let $GB_{\mu_x,\mu_y,\sigma}(x,y) = G_{\mu_x,\sigma}(x) \cdot G_{\mu_y,\sigma}(y)$ with $G_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$ be the model of a single variate gaussian bell with the parameters $\{\mu_x, \mu_y, \sigma\}$. First derivate the maximum likelihood estimators for μ and σ . Use the EM algorithm to estimate the parameters of the models. You can use random values as start parameter or think about how you could compute good start parameters. The better the start parameters are chosen the better and faster the EM algorithm works.

4.7.2 IMS Measurement

Estimate the parameters of an IMS measurement stored in the file imsmeasurement.csv. Do the following steps:

- Write a parser that handles IMS measurements in CSV files as described in the previous section. You can ignore the meta informations in the first 130 lines.
- Use the baseline correction and Savitzky-Golay filter to clean the data. Optional you can try to implement a DFT and cut all frequencies after a value of 150. Assume that the "ion theft" effect does not appear in this measurement.
- Plot the data using gnuplot or any other plotting tool. For gnuplot:
 - Write the values as a simple matrix into a text file.
 - open gnuplot
 - set an palette color, type: set palette defined (0 1 1 1, 1 0 0 1, 2 1 0 1, 4 1 0 0, 12 1 1 0, 20 1 1 0)
 - set the intensity range, type: set cbrange [0:100]
 - plot the matrix, type: plot "matrix.txt" matrix w image
- You can cheat by setting all start parameters of all models by your own. Assume there are six important peaks with modes at:

| _ | r = 121.5s | d = 23.91ms | $s (K_0^{-1} = 0.6773 \frac{Vs}{cm^2})$ |
|---|------------|--------------|--|
| _ | r = 18.2s, | d = 24.41 ms | $(K_0^{-1} = 0.6914 \frac{Vs}{cm^2})$ |
| _ | r = 34.4s, | d = 20.31 ms | $(K_0^{-1} = 0.5753 \frac{Vs}{cm^2})$ |
| _ | r = 9.1s, | d=19.55ms | $(K_0^{-1} = 0.5538 \frac{Vs}{cm^2})$ |
| _ | r = 6.0s, | d = 15.89 ms | $(K_0^{-1} = 0.4501 \frac{Vs}{cm^2})$ |
| _ | r = 4.0s, | d = 17.81 ms | $(K_0^{-1} = 0.5045 \frac{Vs}{cm^2})$ |

• Estimate the model parameters with the EM algorithm and compute the uniform parameters.

Chapter 5

Proteomics

In this chapter, we discuss mass-spectrometry based proteomics. First, what is proteomics? **5.1 Definition** (Proteomics).

Next, we discuss mass spectrometry as a technology.

CHAPTER 6

Interactomics

Interactomics is concerned with the interactions between biomolecules and how they cooperate to achieve the necessary functions to sustain life.

Here, we shall primiarily discuss protein-protein interactions, but of course, proteins also interact with DNA (for example, the binding of transcription factors to certain DNA motifs) and metabolites or other small molecules.

6.1 Protein-Protein Interaction Networks

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.

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).

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 6.1). 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. Most interactions

6 Interactomics



Figure 6.1: 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.



Figure 6.2: An example allosteric effect. 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 are generated 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*. E.g. an interaction between two proteins A and B may be dependent on an interaction between B and C that changes the conformation of B such that the interaction between A and B is possible. Apart from allosteric effects, two proteins may *compete* on the same domain of a third, thus mutually inhibiting their binding (Figure 6.3). Lastly, the binding of small molecules may affect protein conformation and thus induce or inhibit other bindings. A prominent example is phosphorylation and dephosphorylation which is conducted by kinases and phosphatases. All these mechanisms allow the cell to rapidly react on changing environmental conditions via massively parallel state changes of its proteins.

By interacting in the described way, proteins form large networks with emergent systemlevel 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, graph based abstractions are popular now.

6.2 Protein Network Graphs

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. On this level, occuring protein interactions can be measured by high-throughput technologies like mass spectometry.

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

Protein network graphs can be generated for many organisms by obtaining interaction data from online databases. However, there exists a tradeoff between comprehensiveness of the database and the accuracy of information. The Biogrid database¹ for example provides a comprehensive list of interactions gained by diverse types of experiments. However, since most experimental methods for determining protein interactions fail to distiguish direct physical interactions from small complexes containing intermediate interactions, Biogrid contains a considerable amount of false positives. In contrast, the DIP² (Database of Interacting Proteins) provides a core set of manually curated protein interactions that have been verified to be direct by careful literature mining. The downside of this high accuracy is that only a subset of the real proteins and interactions in an organism are captured.

6.3 Protein Complex Prediction

Protein networks can be divided into modules that execute distinct cellular functionalities by the cooperation of many interacting proteins. It has been verified that such modules correspond to clusters in protein network graphs (Spirin and Mirny, 2003). The state of the interactions in protein network modules in general can vary across time and space in the cell. A special class of modules are protein complexes. These are small molecular machines containing often less than 20 proteins that are densely interconnected and stable across time. Protein complexes execute major functions like DNA translation or the synthesis

```
<sup>1</sup>http://www.biogrid.org
```

²http://http://dip.doe-mbi.ucla.edu





of new proteins (the ribosomes). Using clustering approaches, protein complexes can be predicted from protein network graphs.

6.2 Definition (Connectivity). Let (V, E) be a graph. (V, E) is called *connected* if there exists a path (v, \ldots, v') for all nodes $v, v' \in V$. (V, E) is a *clique* if for all nodes $v, v' \in V$ there exist edges $(v, v'), (v', v) \in E$.

A dense region in a protein network graph (P, I) is a connected (Definition 6.2) subgraph induced by a subset of proteins $P' \subseteq P$ with a significantly higher *connectivity* than its environment. A measurement for the connectivity of a subgraph is the *density* or *clustering coefficient*:

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

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

The denominator |V|(|V| - 1)/2 describes the maximum number of edges in the graph. Accordingly a fully connected graph (i.e. a clique) will have a density of 1, whereas a graph without any edge will have a density of 0.

For any graph (V, E) we call (V', E') with $V' \subseteq V$ and $E' \subseteq E$ a subgraph. Since such a subgraph is a graph itself all properties discussed above can be similarly applied to subgraphs.

6.3.1 Local Clique Merging 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 G = (V, E). 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.

We describe the algorithm in Python³ and assume reasonable implementations for the given graph G and all undefined functions.

```
1 def lcma(G, w = 0.4):
2 C = local_cliques(G)
3 D = merge_dense_regions(G, C, w) if C else C
4 return D
```

Detecting local cliques

In order to find a local clique, for each node, the density of its neighbourhood is investigated. As long as it raises the density, the neighbour with the lowest degree is removed from the neighbourhood.

³http://www.python.org

```
def local_cliques(G):
1
       C = set()
2
3
       for u in vertices(G):
           S = subgraph(G, neighborhood(G, u) | \{u\})
4
           d = density(S)
\mathbf{5}
6
           while True:
7
                # remove vertices from neighborhood until density is 1
8
                V = vertices(S)
9
                v = min(V, key=lambda v: degree(S, v))
10
                S_{-} = subgraph(S, V - \{v\})
11
                d_{-} = density(S_{-})
12
                if d_ <= d:
13
                     # stop if density was not increased
14
15
                     break
                S, d = S_{, d_{}}
16
17
           if len(V) > 2:
18
                # consider only non-trivial cliques
19
                C.add(V)
20
21
       return C
```

For each node $v \in V$ at first the subgraph induced by its neighbourhood including itself is generated. The density of the subgraph and the node with minimal degree are recorded. In a loop, the node with minimal degree is removed from the neighbourhood if that raises the subgraph's density. 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. 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.

6.3.2 Merging dense regions

The second step merges the detected local cliques to bigger but still dense regions: they are iteratively merged until the average density is below 95% of the previous mean density.

```
def merge_dense_regions(G, C, w):
1
2
      D, d = C, 1
      while True:
3
           D_{-} = set()
4
           for V in D:
\mathbf{5}
               V_{-} = set(V)
6
               for U in D:
7
                    if U != V and overlap(U, V) > w:
8
                         # merge two regions if they overlap more than w
9
                         V_.update(U)
10
               D_.add(frozenset(V_))
11
           d_ = sum(density(subgraph(G, V_)) for V_ in D_) / len(D_)
12
13
           if d_ <= 0.95 * d or D == D_:
14
               # stop when mean density falls below 95%
15
               return D_
16
```

17 else: 18 D, d = D_, d_

The algorithm starts with the detected local cliques (given as sets of vertices) as dense regions D. First the mean density d is set to 1, since the algorithm starts with the local cliques as dense regions. In an iterative process the next set of dense regions is computed. Each region $V \in D$ is merged together with all dense regions $U \in D$, for which the overlapping score $\frac{|U \cap V|^2}{|U| \cdot |V|}$ is higher than a given threshold w. The parameter w can be used to ensure that a merge is occuring only for significantly overlapping regions. The combination with the termination criterion shall ensure that the results after merging can still be considered dense. If the termination criterion is not met, another iteration is performed.

6.4 Protein Hypernetworks

While protein network graphs provide a high scalability toward very large protein networks, they do not take interaction dependencies into account and thus predictions that are made with these models suffer from inacuracies. In contrast, quantitative approaches based on the law of mass action are only feasible for very small subnetworks in two ways. First, they are based on solving differential equations and thus computationally expensive. Second, they need extensive experimental work to determine the parameters of the chemical reactions in the focused context.

As an intermediate approach, we want to discuss protein hypernetworks. They allow to increase the detail of protein network graphs by including interaction dependencies while maintaining their scalability. We first develop an approach for incorporating interaction dependencies using propositional logic formulas. The propositional logic $\mathfrak{Prop}(Q)$ is the set of all propositional logic formulas over the propositions Q (the atomic units of the logic). This is the smallest set of formulas such that q itself is a formula for all $q \in Q$ and that is closed under the following operations: For $\phi, \phi' \in \mathfrak{Prop}(Q)$, all of $\neg \phi, \phi \land \phi', \phi \lor \phi'$, and $\phi \Rightarrow \phi'$ are in $\mathfrak{Prop}(Q)$ as well. The operators $\neg, \land, \lor, \Rightarrow$ have the usual semantics "not", "and", "or", and "implies", respectively. Note that the implication $\phi \Rightarrow \phi'$ is equivalent to $(\neg \phi \lor \phi')$. As propositions Q, we use both proteins P and interactions I, so $Q := P \cup I$. A constraint is a formula with a particular structure over these propositions.

6.4 Definition (Constraint). A constraint is a propositional 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 constraints.

A constraint $q \Rightarrow \psi$ restricts the satisfiability of q by the satisfiability of ψ . In other words: if q is satisfied, then the same has to hold for ψ . A constraint $q \Rightarrow \psi$ is equivalent to the disjunction $\neg q \lor \psi$. We call the disjunct $\neg q$ the *default* or *inactive* case for the obvious reason that if q is not true, then ψ does not need to be satisfied. For example (see Figure 1a), the dependency of an interaction i on an allosteric effect due to a scaffold interaction j can be formulated by the constraint $i \Rightarrow j$. Mutual exclusiveness of two interactions $i, j \in I$ can be modeled by the two constraints $i \Rightarrow \neg j$ and $j \Rightarrow \neg i$. The usage of propositional logic allows also to define constraints of higher order: An interaction i could be either dependent on two scaffold interactions j_1 and j_2 or compete with an interaction j_3 , modeled by the constraint $i \Rightarrow ((j_1 \land j_2) \lor \neg j_3).$

Now, we can define protein hypernetworks as a set of proteins (nodes) connected by interactions (edges) extended by a set of constraints (dependencies between nodes or edges):

6.5 Definition (Protein Hypernetwork). Let P and I be sets of proteins and interactions. Let $C \subseteq \mathfrak{C}(P \cup I)$ be a set of constraints that contains the *default constraints* $i \Rightarrow p \land p'$ for each interaction $i = \{p, p'\} \in I$. Then the triple (P, I, C) is called a *protein hypernetwork*.

Figure 1a shows an example protein hypernetwork.

6.4.1 Minimal Network States

Following the incorporation of constraints in a protein hypernetwork, we now explain how to sum and propagate their effects in the system. The key idea is that it is sufficient to examine the implications for each protein or interaction $q \in P \cup I$ separately first, and then combine the information in a systematic way. We formalize this idea by defining sets of *minimal network states*. Intuitively, a minimal network state of q tells us which other proteins or interactions are *necessary* or *impossible* to occur simultaneously with q. It is minimal in the sense that it lists only these and no more entities.

For each $q \in P \cup I$, we define a *minimal network state formula*, for which we then find certain satisfying models, which in turn define minimal network states.

6.6 Definition (Minimal network state formula). Let (P, I, C) be a protein hypernetwork. For $q \in P \cup I$, the *minimal network state formula* of q is

$$MNS_{(P,I,C)}(q) := MNS(q) := q \wedge \bigwedge_{c \in C} c.$$

A solution for a propositional logic formula is captured by a satisfying model or interpretation given by a map $\alpha : P \cup I \to \{0, 1\}$ that assigns a truth value to each proposition. A formula is satisfiable if any satisfying model exists. For a propositional logic formula ϕ , we denote $\alpha \Vdash \phi$ if α satisfies ϕ . Further, for any model α , we denote with $\alpha_{q \mapsto x}$ the model with $\alpha_{q \mapsto x}(q) = x$ and $\alpha_{q \mapsto x}(q') = \alpha(q')$ for all $q' \neq q$ and $x \in \{0, 1\}$.

We assume that MNS(q) is satisfiable for all $q \in P \cup I$, i.e., each single protein or interaction by itself is compatible with all constraints.

For example, consider propositions $Q = \{q_1, q_2\}$ and a formula $\phi = \neg q_1 \land (q_1 \lor q_2)$. The only satisfying model is $\alpha : q_1 \mapsto 0, q_2 \mapsto 1$. In the protein hypernetworks framework, we interpret a model α as follows: A protein or interaction q is said to be *possible* in α iff $\alpha(q) = 1$. All possible proteins and interactions may (but need not) exist simultaneously (spatially and temporally) in the cell.

There can be many satisfying models for MNS(q). Among these, we wish to enumerate all *minimal satisfying models* (MSMs). An MSM is minimal in the sense that no constraint is artificially activated. In other words, no proposition is satisfied unless claimed by an active constraint.

6 Interactomics

6.7 Definition (Minimal satisfying model). Let (P, I, C) be a protein hypernetwork and MNS(q) be a minimal network state formula. A satisfying model α is minimal iff for each $q' \in P \cup I$ with $q' \neq q$ and $\alpha(q') = 1$, there exists a constraint $(q'' \to \psi) \in C$ with $\alpha(q'') = 1$ and $\alpha_{q' \to 0} \nvDash \psi$.

A suitable method for finding MSMs is the tableau calculus for propositional logic (Smullyan, 1995). In a nutshell, the tableau algorithm decomposes a formula into its parts. It accumulates conjuncts, branches on disjuncts, and backtracks when a contradiction is encountered.

The general problem of finding a satisfying model for any given propositional logic formula ϕ is NP-complete. However, MNS(q) has a special structure: it is a conjunction of a proposition and of (many) constraints. If all constraints are of a particularly simple structure, we can prove linear running time for the tableau calculus.

Each MSM α defines a minimal network state, consisting of both necessary and impossible entities. The intuition is that the necessary entities k are simply the "true" ones ($\alpha(k) = 1$), and that the impossible entities are those that are *explicitly* forbidden by an active constraint.

6.8 Definition (Minimal Network State). Let (P, I, C) be a protein hypernetwork and $q \in P \cup I$. Let α be a MSM of MNS(q). We define sets of *necessary* and *impossible* proteins or interactions, respectively, as

$$Nec_{\alpha} := \{q' \in P \cup I \mid \alpha(q') = 1\},$$

$$Imp_{\alpha} := \{q' \in P \cup I \mid \exists \text{ constraint } (q'' \Rightarrow \psi) \in C$$

with $\alpha(q'') = 1 \text{ and } \alpha_{q' \mapsto 1} \nvDash \psi.\}.$

The pair $(Nec_{\alpha}, Imp_{\alpha})$ is called a *minimal network state* for q (belonging to the MSM α).

For each proposition q, there can be several minimal network states. We write M_q for the set of all minimal network states for q. We call $M := M_{(P,I,C)} := \bigcup_q M_q$ the set of all minimal network states for all proteins and interactions.

Now we define a relation *clashing*, describing that two minimal network states cannot be combined without producing a conflict.

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

As we prove in Theorem 6.10, in order to know if two proteins or interactions are simultaneously possible, it is sufficient to determine whether any pair of non-clashing minimal network states exists for them.

6.10 Theorem. Let (P, I, C) be a protein hypernetwork. Let $q, q' \in P \cup I$ be two proteins or interactions, $q \neq q'$. Assume that there exists a non-clashing pair of minimal network states $(m, m') \in M_q \times M_{q'}$. Then q and q' are possible simultaneously, i.e., the following formula is satisfiable.

$$\xi := \left(\bigwedge_{c \in C} c\right) \land q \land q'.$$

Proof. Let $m = (Nec, Imp) \in M_q$ and $m' = (Nec', Imp') \in M_{q'}$ be non-clashing; we show that ξ is satisfiable by defining a satisfying model α . Define $True := Nec \cup Nec'$ and $False := Imp \cup Imp'$. Since m and m' are not clashing, $True \cap False = \emptyset$. Let $\alpha(r) := 1$ for $r \in True$, and $\alpha(r) := 0$ otherwise. We show that α satisfies all parts of ξ .

The propositions q and q' in ξ are satisfied since $q \in Nec$ and $q' \in Nec'$, so $\alpha(q) = \alpha(q') = 1$.

For each $c^* = (r \Rightarrow \psi)$ in the conjunction $\bigwedge_{c \in C} c$, there may appear two cases: $\alpha(r) = 0$ or $\alpha(r) = 1$. If $\alpha(r) = 0$, then c is satisfied regardless of the satisfaction of ψ because of the implication semantics. If $\alpha(r) = 1$, or equivalently $r \in True$, then $r \in Nec$ or $r \in Nec'$ (or both). First, consider the case that $r \in Nec$. By assumption, $c^* \in C$ is then satisfied in $\bigwedge_{c \in C} c \wedge q$. Additionally, it is not clashing with q' because $True \cap False = \emptyset$. Therefore, it is also satisfied in ξ . The case $r \in Nec'$ is analogous.

Inclusion of Perturbation Effects

The protein hypernetwork framework allows to systematically compute consequences of perturbations. We distinguish between *perturbed* and *affected* proteins or interactions: A perturbed one is the direct target of an experimental intervention which causes its complete removal from the system (e.g. by gene knock-down for proteins or point mutations for interactions), whereas an affected one is altered due to the propagation of the perturbation in the hypernetwork. Assume that proteins $P_{\downarrow} \subseteq P$ and interactions $I_{\downarrow} \subseteq I$ are perturbed, and thus removed from the system. The problem at hand is to compute all affected proteins and interactions. This is done by recursively removing minimal network states m = (Nec, Imp)that necessitate a perturbed or affected entity (protein or interaction) $q \in Nec$, while counting a protein or interaction as affected once it has no minimal network state left. Formally, we proceed as follows.

6.11 Definition. Let M_q be the set of all minimal network states for entity q (Definition 6.8), and let M be any subset of all minimal network states $M_{(P,I,C)}$. For a set of entities $A \subset P \cup I$, let

$$\bar{M}_A := \{ (Nec, Imp) \in M \mid A \cap Nec \neq \emptyset \}$$

be the set of minimal network states from M that becomes invalid when any entity in A is perturbed. Let $R(A, M) := M \setminus \overline{M}_A$ be the remaining set of minimal network states. Let $Q(A, M) := \{q \in P \cup I \mid M_q \cap R(A, M) = \emptyset\}$ be the set of entities for which no minimal network state is left.

We recursively define a map ρ that maps a set of perturbed entities and a set of minimal network states to the set of affected entities. Let $\rho : 2^{P \cup I} \times 2^{M_{(P,I,C)}} \to 2^{P \cup I}$ be defined by

$$\rho(A, M) := \begin{cases} \emptyset & \text{if } A = \emptyset, \\ A \cup \rho(Q(A, M), R(A, M)) & \text{otherwise.} \end{cases}$$

Let (P, I, C) be a protein hypernetwork with perturbations $P_{\downarrow} \subseteq P$ and $I_{\downarrow} \subseteq I$ and minimal network states $M_{(P,I,C)}$. Then

$$Q_{\downarrow} := \rho(P_{\downarrow} \cup I_{\downarrow}, M_{(P,I,C)})$$

is the set of all affected proteins and interactions.



Figure 6.4: (a) Example protein hypernetwork with two constraints. (b) Prediction of protein complexes in four steps.

This provides a module that enables any algorithm that makes predictions based on protein networks to be applied also on a perturbed network, considering the dependencies between interactions.

6.4.2 Prediction of Protein Complexes

The prediction of protein complexes in hypernetworks consists of four steps, illustrated in Figure 1b. First, for each protein and interaction $q \in P \cup I$, the set of minimal network states M_q is obtained. Then, with a network based complex prediction algorithm, an initial set of protein complexes is predicted. Each complex c is given as a subnetwork (P_c, I_c) .

The third step is more complicated. Let $M_c := \bigcup_{q \in P_c \cup I_c} M_q$ be the set of minimal network states of the complex's entities. We want to combine the individual states without introducing clashes, as formalized by the following definition.

6.12 Definition (Maximal combination of minimal network states). For a complex c, a set $M \subseteq M_c$ is called a *maximal combination of minimal network states* iff (1) there exists no clashing pair of minimal network states in M, and (2) the inclusion of any further minimal network state from M_c would result in a clashing pair.

All maximal combinations of minimal network states for a given complex c can be obtained by recursively building a tree of minimal network states to be removed from M_c . The root of the tree is annotated with M_c ; each other node is annotated with a remaining set M. If M does not contain any pair of clashing states, the node is a leaf, and M is added to the result set of maximal combinations. Otherwise, we take any m with clashing m', m'', \ldots and branch off two children which remove m on the one hand, and remove m', m'', \ldots on the other hand. The tree is explored in a depth-first manner, checking for redundancies in each node. Let $\mathcal{M}_c \subseteq 2^{M_c}$ be the set of all found maximal combinations of minimal network states. Its cardinality equals the number of non-redundant leaves in the removal tree. For each maximal combination $M \in \mathcal{M}_c$, we generate the corresponding subnetwork of (P, I). In theory, the number of maximal combinations may grow exponentially with the number of constraints inside a complex if all of them lead to clashing minimal network states. However, in practice most predicted complexes are small (between 3 and 20 proteins) and clashes are rare.

6.13 Definition (Simultaneous Protein Subnetwork). Let $M \in \mathcal{M}_c$ be a maximal combination of minimal network states for complex c. Let P_M be the set of all necessary proteins and I_M the set of all necessary interactions in M, i.e., $P_M := P \cap \bigcup_{(Nec,Imp)\in M} Nec$ and $I_M := I \cap \bigcup_{(Nec,Imp)\in M} Nec$. Then (P_M, I_M) is called a simultaneous protein subnetwork.

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

Finally, in the fourth step, we perform a network based complex prediction on each simultaneous protein subnetwork (P_M, I_M) again with the same algorithm as during the initial step (thereby it has to be ensured that the network based complex prediction does not produce biased results when performed only on subnetworks). The proteins and interactions in the new complexes are simultaneously possible. Since the plain-network based complex prediction algorithm is not aware of constraints, the prediction may miss necessary interactions (e.g. due to constraints modeling allosteric regulations) and proteins outside the initially predicted complex. Therefore we force these omitted entities to be contained in the predicted complex, thus obtaining complexes that do not violate any constraint.

6.4.3 The Tableau Calculus

A suitable method for finding satisfying models is the tableau calculus for propositional logic (Smullyan, 1995): For an input formula ϕ , it generates a deductive tree (the *tableau*) of assumptions about ϕ . Each assumption a in the tree can be made due to an assumption a' in an ancestral node. We say that a' results in a, and the generation of a out of a' is called *expansion* of a'. The propositional logic tableau algorithm generates satisfying models α for ϕ . We write $\alpha \Vdash \psi$ if α satisfies a subformula ψ . The tableau algorithm now generates assumptions of the type $\alpha \Vdash \psi$ with ψ being a subformula of the input formula. That is, a conjunction $\alpha \Vdash \psi_1 \wedge \psi_2$ is expanded into $\alpha \Vdash \psi_1$ and $\alpha \Vdash \psi_2$ on the same path, and a disjunction $\alpha \Vdash \psi_1 \vee \psi_2$ results in branching into $\alpha \Vdash \psi_1$ and $\alpha \Vdash \psi_2$ (see Fig. 6.5).

Each path from the root to a leaf represents a model α . If a path does not contain any contradictory assumptions, the model satisfies the input formula. Implementations of the tableau algorithm explore the tree in a depth-first way, and use backtracking once a contradiction occurs.

Different variations of the tableau algorithm exist. For example, one may be interested only in the decision "does a satisfying model exist?", or the task could be to output an (arbitrary) satisfying model (if one exists), or to list all satisfying models. The latter is the task we face when enumerating minimal network states. In theory, the tableau algorithm exhibits an exponential worst case complexity, as it operates by complete enumeration of all cases with backtracking.



Figure 6.5: Tableau for the propositional logic formula $\phi = \neg A \land (A \lor B)$. The path marked by \notin does not lead to a satisfying model α , because it contains a contradiction between the assumptions $\alpha \Vdash \neg A$ and $\alpha \Vdash A$. The path marked by \checkmark is free of contradictions, hence its generated model satisfies ϕ .

However, elaborate backtracking strategies can significantly reduce the running time in practice. Insights into such strategies and implementation details (even for modal logic) are provided in (Li, 2008). Also, faster heuristics exist, like GSAT (Selman et al., 1992), but they are not adequate for the problem, as they do not guarantee a correct and complete answer.

For our purpose, the implementation has to ensure that

- 1. for each constraint $q \Rightarrow \psi$ (i.e., disjunction $\neg q \lor \psi$), the default case $\neg q$ is explored first, and that ψ is expanded only if the constraint is necessarily active (this allows to find MSMs);
- 2. all satisfying models that comply with 1. are enumerated in the process.

In our application, the tableau algorithm can be expected to perform acceptably, since we solve only minimal network state formulas with a fixed structure (a conjunction of constraints) and expect most constraints to be of a simple form. In particular, we expect mostly mutually exclusive interactions, modeled by constraints of the form $i \Rightarrow \neg j$, and scaffold dependent interactions that can be represented by a constraint of the form $i \Rightarrow j$. To prove the performance of the tableau algorithm when all constraints are of this form, for a protein hypernetwork (P, I, C) we now show that it will need only $\mathcal{O}(|C|)$ expansions to find a satisfying model. Since expansions generate the deductive tree, that also limits all tableau operations like backtracking or contradiction tests to be polynomial in |C|.

6.14 Theorem. Let $MNS_{(P,I,C)}(q)$ with $q \in P \cup I$ be the minimal network state formula for a protein hypernetwork (P, I, C). Assume that each constraint in C is of the form $c = (q_1 \Rightarrow \ell)$ with a literal $\ell \in \{q_2, \neg q_2\}$ and $q_1, q_2 \in P \cup I$. Then the tableau algorithm needs at most $\mathcal{O}(|C|)$ expansions to find a satisfying model.

Proof. We show that a constraint that is active cannot be rendered inactive again when assuming that the formula is satisfiable. Assume that an active constraint $c = (q_1 \Rightarrow \ell)$ is the cause of a conflict, hence ℓ contradicts some literal ℓ' . Since we require above that the tableau explores the inactive case first, we know that $\neg q_1$ caused a contradiction, too. We now assume that ℓ is removed and we expand c to $\neg q_1$ again to resolve the contradiction.

Then, the formula is found to be not satisfiable, because $\neg q_1$ can either contradict q or another constraint, in which case the argument can be applied recursively.

Now we show that each constraint is expanded at most two times. There are three cases: (1) The constraint is never activated; then only the inactive case is expanded and the constraint is expanded only once. (2) The constraint is activated immediately because $\neg q_1$ leads to a conflict. This needs two expansions. (3) The constraint is first inactive and then activated because of a backtracking. This needs again two expansions. Hence, the tableau algorithm needs to perform $1 + 2|C| = \mathcal{O}(|C|)$ expansions.

Note that $MNS_{(P,I,C)}(q)$ with above simple constraints is essentially a Horn formula for which it is known that calculating a satisfying model has polynomial complexity with specialized algorithms. However, proving the complexity of the general tableau algorithm for this case remains useful: While we expect most of our constraints to have this simple form, we cannot be sure for all of them. Hence it is reasonable to provide a computational approach that can handle full propositional logic, but will have comparable complexity to specialized horn formula algorithms in the majority of cases.

Bibliography

- X.-L. L. Li, S.-H. H. Tan, C.-S. S. Foo, and S.-K. K. Ng. Interaction graph mining for protein complexes using local clique merging. *Genome informatics*, 16(2):260-269, 2005. ISSN 0919-9454. URL http://view.ncbi.nlm.nih.gov/pubmed/16901108.
- Z. Li. Efficient and Generic Reasoning for Modal Logics. PhD thesis, School of Computer Science, University of Manchester, UK, 2008. URL http://www.cs.man.ac.uk/ \~{}schmidt/mltp/Thesis.pdf.
- B. Selman, H. J. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In P. Rosenbloom and P. Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.4724.
- R. M. Smullyan. *First-Order Logic*. Dover Publications, Jan. 1995. ISBN 0486683702. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/ 0486683702.
- V. Spirin and L. A. Mirny. 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, 2003. ISSN 0027-8424. doi: 10.1073/pnas.2032324100. URL http: //dx.doi.org/10.1073/pnas.2032324100.