LUDWIG-MAXIMILIANS-UNIVERSITÄT
TECHNISCHE UNIVERSITÄT MÜNCHEN

Wissenschaftszentrum Weihenstephan:
Lehrstuhl für Genomorientierte Bioinformatik

Bachelorarbeit
in Bioinformatik

A Toolbox for Systematic Model Reduction

Manuel Kroiss

Aufgabensteller:	Prof. Fabian Theis
Betreuer:	Prof. Fabian Theis
	Dennis Rickert
Abgabedatum:	15.09.2011

Ich versichere, dass ich diese Bachelorarbeit
selbständig verfasst und nur die angegebenen
Quellen und Hilfsmittel verwendet habe.

15.09.2011 Manuel Kroiss .

Manuel Kroiss

Abstract

Based on previous work by D. Rickert at the Helmholtz-Zentrum München, we developed a toolbox for the automated reduction of biological models using Boolean logic. The method is very versatile and can be used to reduce any set of elements with constraints that can be expressed in Boolean algebra. We focused our work on the reduction of interaction networks, where usually a costly simulation using parameter optimization is required to validate or invalidate a reduced model. However, we designed a method that can automatically deduce constraints from the network topology to strongly limit the amount of possibilities that need testing. Further, as a simulation usually requires parameter optimization, certain models can be skipped when others are accepted or rejected. Finally, since the order in which we test the possible models greatly influences the computational costs, we developed strategies to improve the order of testing. Our toolbox is easily applicable with a simple Matlab interface and direct support for the SB model format. We put a special emphasis on providing various options to represent the different resulting models. The toolbox was exhaustively benchmarked and successfully used to help in the development of a fatty acid β-oxidation model and to give valuable input in the reduction of a zirconium ingestion compartment model.

Aufbauend auf bisheriger Forschung von D. Rickert am Helmholtz-Zentrum München haben wir eine Toolbox für die automatische Reduktion von biologischen Modellen mit Hilfe von boolescher Logik entwickelt. Die Methode ist sehr vielseitig gestaltet und kann für die Reduktion einer jeden Menge von Elementen, auf welcher sich Bedingungen in boolescher Algebra definieren lassen, verwendet werden. Der Schwerpunkt unserer Arbeit lag auf der Vereinfachung von Interaktionsnetzwerken, für welche meist eine aufwendige Simulation mit Parameteroptimierung benötigt wird, um ein reduziertes Modell auf seine Korrektheit zu prüfen. Jedoch war es uns möglich eine Methode zur automatischen Herleitung von Bedingungen anhand der Netzwerktopologie zu gestalten, welche die Anzahl an möglichen Modellen stark einschränkt. Zusätzlich können, da eine Simulation meist Parameteroptimierung benötigt, bestimmte Modelle übersprungen werden falls andere bereits geprüft wurden. Schlussendlich hat die Reihenfolge in der wir die Kombinationen testen einen starken Einfluss auf die Laufzeit, weswegen wir Strategien entwickelt haben um diese Reihenfolge zu verbessern. Allgemein ist unsere Toolbox, durch ein einfaches Matlab-Interface und eine eingebaute Unterstützung für das SB Format, sehr leicht anzuwenden. Besonderer Wert wurde auf die Darstellung der Ergebnisse eines Reduktionsdurchlaufs gelegt, wodurch eine Vielzahl von übersichtlichen Repräsentationen entstand. Die Toolbox wurde ausgiebigen Benchmark-Tests unterzogen und konnte bereits erfolgreich bei der Entwicklung eines Fatty-Acid-Oxidation Modells einen Beitrag leisten und bei der Reduktion eines Modells für den Abbau von Zirkonium verwendet werden.

Contents

1. Introduction

1.1 Modeling in biology

The modeling of biological systems has become a main topic in bioinformatics. High throughput methods such as micro array analysis or co-immunoprecipitation produce large amounts of data. Unfortunately building quantitative models describing the biological data has proven to be very difficult, as there are still many processes that cannot be measured, especially not inside a living organism.

Instead most measurements are performed on isolated compartments so called in vitro experiments or merely on chips. However, particularly in the case of chips, the interactions are highly theoretical as the compounds might never meet in reality and therefore the interaction would never take place. In a living organism, the interacting species may be separated in different compartments or involved in completely different pathways.

Still measuring the concentrations of different species and determining whether a pairwise interaction exists can be done fairly well compared to the rate of the reaction. When it comes to the kinetics, extensive in vitro experiments emulating the exact conditions of the interacting species would have to be done to measure how much substrate gets processed. Therefore only vague estimates or no kinetics at all are known for most reactions.

1.2 Parameter optimization of models

The rate or kinetic of a reaction is usually referred to as parameter. Since the parameters of a model are usually unknown due to experimental limits, we look for ways to circumvent this problem. What we usually did not use up to this step when building a model are the concentration measurements of the species which are performed over a series of time.

The measured concentrations can be compared with predictions we make using the model and certain parameters, usually by means of simulating the model. We define this comparison as an error-function f that returns a measure of distance for the prediction of our model with parameter values p_1 to p_n and the expected data. If defined like this, a high value generally stands for a bad prediction.

$$f(p_1, \ldots, p_n) = error$$

This comparison between predicted and expected data is called a fit. It is clear that only certain combinations of parameter ranges will produce good fits with a low error value. Finding these combinations can be transferred to the field of parameter optimization, a very developed field of numerical analysis in informatics.

A parameter optimizer will, in theory, try all possible combinations of parameters and attempt to minimize the result of the error function. As this is not feasible and not even finite in a continuous space, the optimizer will attempt to approximate areas of the space with minima (see Figure 1) and follow them to their highest point, in our case the best fit.

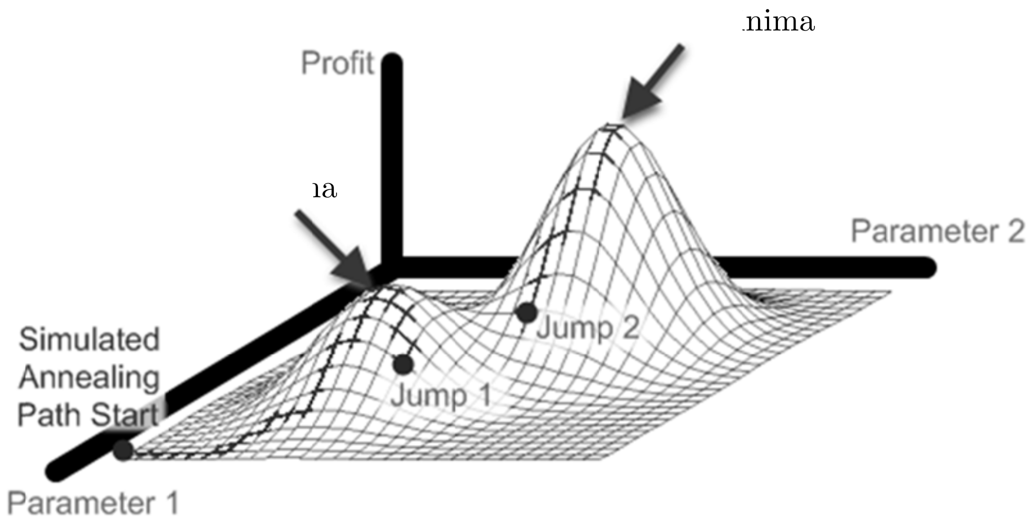

Figure 1: Simulated annealing (Dama, 2009)

Simulated annealing is a good example for a parameter optimization method that finds a good solution and does not scan the whole space. While a simple hill climbing algorithm would stop at the first local minima, simulated annealing has the great advantage that it can escape such local minima by performing chaotic jumps as shown in Figure 1.

The result of an optimization method is a set of parameters that should in theory produce the best fit possible. Unfortunately due to the nature of the approximations, there is no guarantee that the found solution is really the best one. Furthermore the results of a mere error minimization have no biological relevance. Therefore, without examining the complete parameter space or including other constraints into the error function, this method should preferably only be used to check whether a model could possibly describe measured data, but not claim that the parameters are the correct ones.

1.3 Model reduction

We already discussed that chips and in vitro experiments cannot replicate the conditions in a cell properly. This leads to unconfirmed reactions that might never occur. Models that are constructed from the data would then be overly complex with redundant interactions and even wrong if we only allow active reactions with positive kinetics.

Another case that we did not discuss and might occur is a compound which is modeled as two different states, when it is in fact a single compound.

In model reduction, the goal is now to clean up and remove or modify these unnecessary parts. The result will be a less complex model with fewer reactions or states. A common approach to do this is to follow a rule based system that detects certain motifs and transforms then into smaller ones. While this method is very simple and fast, it holds many disadvantages. An example is that certain motifs might only be applicable for specific network types or may perhaps not be found at all. Furthermore the order in which the motifs are transformed could result in different outcomes which are often not considered.

A more sophisticated approach would be to perform all possible reductions and check all smaller models individually. In order to do this we need to define a finite space of possible models. While merging states is possible, we first only focused on the operation of removing a reaction. For a model with 20 reactions, this will still lead to $2^{20} - 1$ different combinations of reactions that can be removed, which all require testing when no rules are defined.

To test whether a reduction is valid, we can simply perform a parameter optimization on the sub-model and compare it with the original one. If the result is not worse than a specific cutoff value, then it will be considered valid. Obviously, checking all possible reductions is not feasible in most cases, only perhaps on very small models or ones with many constraints.

However, since we perform a parameter optimization to analyze whether the model is valid, we can use the fact that the optimizer will try all possible values for a parameter, even the one nullifying the effect of the parameter. Therefore when we try to remove additional parameters, the result of the error function can only be equal or worse, given that the optimizer performs correctly.

Considering this, if a reduction now turns out to be invalid, it will include all sub-models where we remove additional parameters. In parallel, when the error function classifies a reduction as valid, the models where we remove less parameter will also be valid. This is the fundamental theorem of this method that makes a complete model reduction possible.

1.4 Toolbox for systematic model reduction

The focus of our work was to build a toolbox that allows the automated reduction of such models with the defined operations and our core theorem.

Dennis Rickert already established the core ideas of this method with set based reduction, binary decision diagrams and topology analysis in his bachelor thesis on empirical reduction of signaling networks (Rickert, 2009). Based on these ideas, we simplified and rebuilt the methods as a reference implementation for a well-structured base of the reduction framework.

Around this core we added algorithms and crucial strategies to allow for an easy to interface and fast end user application of the method. A run of the reduction method can be structured the flow chart depicted in Figure 2.

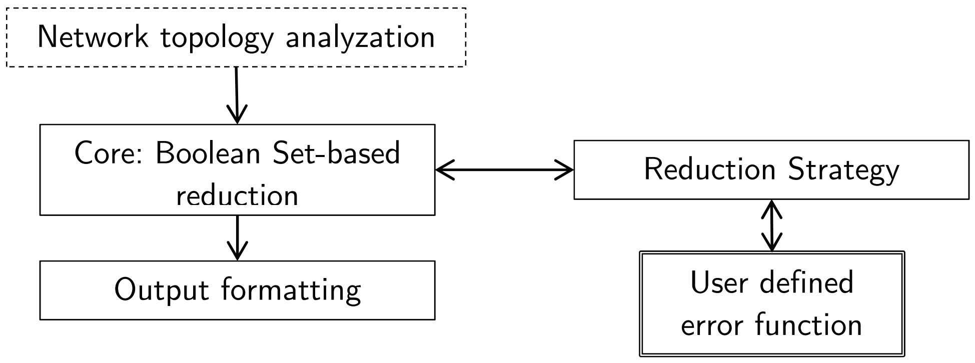
Network topology analyzation
Core: Boolean Set-based reduction
Reduction Strategy
Output formatting
User defined error function

Figure 2: Flowchart of the reduction method

This figure is just to convey a quick overview of the toolbox and its structure, all parts will be explained in detail. Basically, if a network exists, constraints will be deduced from its topology and given to the core of the method defines all the rules based on Boolean logic. The analysis is necessary to limit the amount of possible reductions in order for the toolbox to be usable on medium size networks. This core is so versatile that it could also be applied on a simple set of elements, while the constraints could be defined by the user himself.

The reduction strategy repeatedly picks one of the remaining untested reductions to test with the error function. A clever strategy does not pick the reductions randomly, but tries to maximize the information that is gained when the reduction is accepted or rejected. For each picked element, the result will be reported to the core reduction method. After all calculations are done, there might be many possible models. We put a special emphasis on designing algorithms that give a clear and quick representation of these results.

2. Set based reduction based on Boolean algebra

The finite space of possible reduced models is the list of all combinations of reactions that can be removed. The attempt of removing a combination can be generalized as a set based reduction. While reactions/parameters are simply different elements, for example A, B, C and D, an actual reduction is defined as a set of these elements e.g. {A,B,D} which will be called ABD as an abbreviation.

In search for a basic structure that allows for an efficient representation of these sets, as well as combining them with other rules and constraints that describe the network topology, we decided to use Boolean algebra. A direct way to implement Boolean formulas and their operations are Binary Decision Diagrams (BDD). For the toolbox we used the JavaBDD implementation (Whaley, 2007).

We defined three Boolean formulas which hold different information and when combined, allow us to efficiently identify the whole solution space of the reduction.

- $topology$: the constraints that can be made using the network topology of the model
- $accepted$: all tested valid sets and their subsets
- $rejected$: all tested invalid sets and their supersets

The topology variable is filled in the network analysis of our method and defines all combinations that could work and where the models make sense. For the reduction results we took too separate variables accepted and rejected are filled by the results when using the error function.

While it is obvious that all formulas could be combined into one variable to define our solution space, we needed to keep them separately as the information was treated differently in the reduction strategies.

2.1 Boolean operations for reductions

We will now try to convey a quick overview of how to combine our topology accepted and rejected variable to get the different information spaces needed in the course of the reduction method.

As we established, a reduction can be defined as a set of elements. To check whether a reduction has already been accepted or rejected, we take the reduction and transform it into its Boolean representation.

$$reduction = \bigwedge_{e \in ReductionElements} e \wedge \bigwedge_{e \in Elements \setminus ReductionElements} \neg e$$

To test whether this reduction is rejected, we can simply combine this variable with the topology and the rejected combinations:

$$IsRejected(reduction) = topology \wedge rejected \wedge reduction$$

Same can be done to check if it was accepted:

$$IsAccepted(reduction) = topology \wedge accepted \wedge reduction$$

In the reduction strategy we often need to enumerate all remaining reductions that still require testing. For this we do not need to check every reduction individually, but can simply transform combine the spaces as follows and get all satisfying combinations of the Boolean formula (All-SAT).

$$remaining = topology \wedge \neg accepted \wedge \neg rejected$$

Finally when all reductions have been tested we get our final space and perform an All-SAT operation again.

$result = topology \wedge accepted \wedge \neg rejected$

2.2 Accepting or rejecting a reduction

A reduction is a set of elements we want to reduce and can either result in a valid or invalid model. When the reduction is valid, then automatically all subsets of this reduction-set have to be valid as well, since the parameter optimizer also allows for removing elements.

Similarly if the reduction has turned out to be invalid, then we can assume that all supersets are also invalid for the same reason. By using the parameter optimizer, we guarantee that subsets always have to perform equal or better and supersets always have an equal or worse result.

We will try to illustrate this on a quick example for four elements A, B, C, D and their combinations that we want to test for reduction. In a reduction, for each element we can either decide to include an element or not, leading to $2^4 = 16$ different reductions. As a given situation we assume that we are somewhere in the middle of our reduction process and the reductions B, CD were valid and ABC, BCD, ABCD are invalid (see Figure 3).

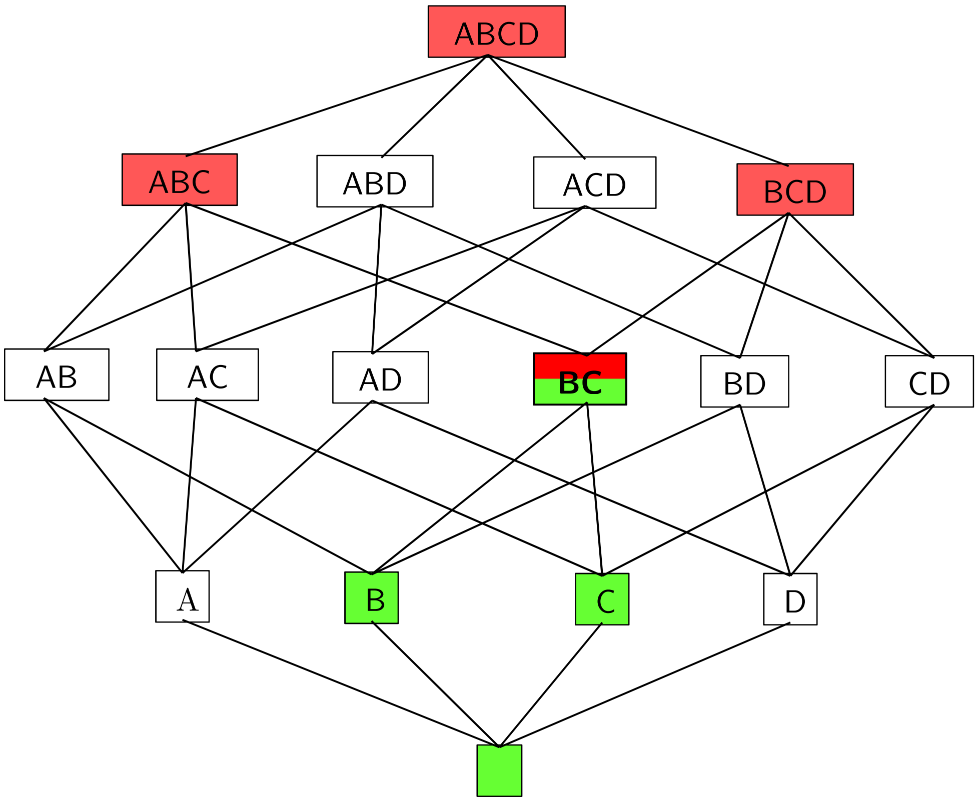

Figure 3: Example for a reduction graph when testing BC

When the test of the reduction AB identified a valid model, we can skip the reductions A, B, {} because they are subsets of AB. If reducing AB was invalid then we can skip ABC, ABD, ABCD since they are supersets of AB. Skipping a result reduction means that it will be added the list of accepted or rejected reductions. As keeping every reduction individually in a list is very inefficient, the lists are described by Boolean formulas.

For the valid reductions the representing formula is called $accepted$ and can be constructed by or-connecting the working reductions and their subsets. The subsets are represented by AND-linking all elements that are not in this reduction.

$$accepted = accepted_1 \vee accepted_2 \vee \ldots \vee accepted_n$$

$$accepted_n = \bigwedge_{e \in AllElements \setminus ReductionElements} e$$

The invalid reductions are described be the $rejected$ formula and is basically and OR-connection of all reductions-sets.

$$rejected = rejected_1 \vee rejected_2 \vee \ldots \vee rejected_n$$

$$rejected_n = \bigwedge_{e \in ReductionElements} e$$

3. Analyzing the network topology

In interaction networks, we try to decrease the amount of possible reductions we have to test by defining a series of constraints from the network. This is necessary, as a testing a model is very computationally intensive due to the parameter optimization and the problem would not be feasible for an unconstrained set of elements. Dennis Rickert used a concept called controllability and observability (Rickert, 2009), which we transferred to our work. However, we used different constraints which we feel are easier to understand and implement.

In a network the compounds are often called states. We declare a subset of states as input and another one as observed/output states. Input states are compounds with a beginning concentration before the simulation starts, while output states are the compounds that are measured and where we expect a resulting concentration. Note that subsets can overlap.

The three constraints that we defined are:

A) Every reaction must originate from a state that is activated.
B) Every output state must be at least activated by one input state.
C) Every reaction must originate from a useful state; one that is activates an output state or is involved in a reaction of such a state.

3.1 Reactions must be activated

The first topological basically states that for every reaction that is not reduced, there must be a possibility that it will be used at some point. In order for it to be used, one of the origin states must reach a concentration higher than zero, meaning it must be activated. This rule can be recursively defined as follows.

$$\bigwedge_{r \in UnreducedReactions} ActiveRec(r.origin, \{\,\}) = true$$

$$ActiveRec(state, visited) = \begin{cases} state \in visited: false \\ state \in InputStates: true \\ else: \bigvee_{r \in IncomingReactions(state)} ActiveRec(r.origin, visited \cup state) \end{cases}$$

The case where a reaction has several substrates/origins is not shown in the formulas for simplicity but is covered in the implementation as reactions can have different combinations of origin states to be activated.

3.2 Output states must be activated

Secondly every output state must be connected to at least one input state. There must be a directed way leading from an input state to an output state, as otherwise the output state could never get a concentration below zero.

$$\bigwedge_{o \in OutputStates} ActiveRec(o, \{\,\}) = true$$

3.3 Reactions must have an impact

The last rule states that every reaction must have some impact on an output state; otherwise it has to be removed. If the origin state of the reaction never activates an output state or is never involved in a reaction with a state that does, then its concentration does not make a difference for the observed states. Reductions which have reactions without impact are not valid and can be skipped.

$$\bigwedge_{r \in Reactions} \bigvee_{o \in r.origins} UsefulRec(o) = true$$

$$UsefulRec(state, visited) = \begin{cases} state \in visited: false \\ state \in OutputStates: true \\ else: \bigvee_{r \in OutgoingReactions(state)} \bigvee_{n \in r.origins \cup r.targets} UsefulRec(n, visited \cup state) \end{cases}$$

3.4 ODE-model analysis

These above defined rules for the network topology can be applied to most models governed by interactions. Ordinary Differential Equations are one possible way of describing the reactions. Generally, an ODE-model in the SB-Toolbox format contains states, reactions, variables and parameters (H. Schmidt, 2005).

While separately defining reactions and variables usually results in a clean and structured model, one can also solely use states and parameters. As we want to account for the most general case, we first resolve all reactions and variables used in the definitions of the states, resulting in a model where we only have states and parameters.

To derive the activating reactions, we have to investigate the influence of each parameter on a state. We set all parameters of the reactions except one to zero or its nullifying value. Assuming the parameters are independent, we get the direct impact of the parameter on the state and all states which are involved in the reaction.

As this part is conveniently implemented in Matlab, we can simply check if the formula can ever get above or below zero, indicating whether the influence of this formula on the states is activating, inhibiting or both.

4. Reduction Strategies

Clearly the order in which we try the reductions greatly influences the total runtime of our reduction method. A reduction strategy is a method that simply picks the next reduction attempt and determines the order in which we visit the reduction graph discussed in Figure 3. Two simple methods for visiting the graph are BFS or DFS. However, these have very poor performance and a strategy should maximize the information gain, which of course can only be estimated using heuristics.

4.1 Simple BFS and DFS approaches

Very simple approaches are the two common methods for graph traversal, namely breadth first search and depth first search. In the case of model reduction, BFS takes the sets of reduction in order of their size. When doing the usual BFS, we would first take combinations reducing only a few elements and later the ones with many elements.

This method is good when we only expect very few reductions to work or almost all of them. In case many reductions are possible, we simply reverse the BFS to start with long reductions first and taking short reductions at the end. Unfortunately in other cases especially when the result is unknown, both methods have a very bad performance, as the information gain is very little in the average case.

Depth first search is also a common method to visit the nodes in a graph. It adds additional elements to the set of reduction and then uses backtracking if there are no more elements to be added. In our test (see 4.5 Benchmarking) this strategy has not proven to be a good reduction method.

4.2 Maximum average information gain

A very intuitive approach is to maximize the average information gain by taking a reduction that performs well in either case, both when rejected and accepted. To retrieve this reduction, we iterate over all reductions and see how many remaining reductions we have when accepted or rejected.

To efficiently iterate over the remaining reductions we first perform the BDD-Allsat operation which returns a list of patterns describing the remaining combinations. Each pattern consists of n numbers strung together, where n is the number of elements that we initially wanted to perform the reduction method on. The numbers of a pattern are either 0 for false, 1 for true or -1 as a wildcard for either 0 or 1. Now replacing all wildcards recursively by either a zero or one gives us all possible combinations of remaining reductions.

Finding out the amount of reductions that can be skipped when the current reduction is accepted or rejected could be done by assuming that it was reported and using the Boolean formulas made in 2.2 Accepting or rejecting a reduction. Unfortunately it turns out that this operation takes very long for great amounts of reductions.

Therefore we preprocess the patterns from the BDD-Allsat of the remaining reductions into a directed acyclic word graph (DAWG). To first get the reductions that can be skipped when accepted, we build a binary representation of the reduction, which matches the BDD-Allsat format, while ones are replaced with wildcards. With this pattern, the DAWG will be recursively visited while comparing the node value with the respective value in the pattern. When following a branch we keep track of the possibilities when matching wildcards in the pattern by doubling the possibilities we found so far. We do the same process for the rejection case by simply replacing all zeros by wildcards.

As a result we now got the two values n_{acc} and n_{rej} for each reduction, which are the number of elements that can be skipped when accepted and when rejected. It is obvious that these values are usually indirect proportional. If we have a high n_{acc} value, we expect a low n_{rej} value (for illustration see reduction graph in Figure 3). Since we do not know the probability that a reduction is accepted or rejected, we want to maximize the information gain for both cases. This is done by the following formula:

$$\max(n_{acc} * n_{rej})$$

This strategy resulted in a great increase in performance and helped us solve the Zirconium model reduction (see 6.4 Compartment model for Ingestion of Zirconium in

Human Body). The calculation was not finished after two days with both normal and reverse BFS, but after an hour with this strategy.

4.3 Heuristic approach

To further improve the runtime, we wanted to calculate the probabilities that a reduction is accepted or rejected and combine this with the number of elements that can be skipped. Furthermore the cost if accepted or rejected will also be taken into account, as we expect different runtimes for the two cases. Often reductions are immediately accepted on the first optimization attempt, but we have to run it several times when rejected if not using a deterministic parameter optimizer.

The formula we want to maximize now looks as follows:

$$IG(RED) = P(A) * \frac{N(RED)}{C(RED)} + P(\neg RED) * \frac{N(\neg RED)}{C(\neg RED)}$$

Predicting the costs is pretty straight forward by averaging all the costs observed so far when accepted or rejected. In each case we not only report the result but also the runtime and create new cost averages that will be used in the next prediction.

Calculating the probabilities was much harder and slowly became the main focus of this bachelor thesis. It seemed very natural to apply the theory of conditional probability to this problem, but creating a feasible solution became very hard.

As an example we will again take our four element scenario, where the elements that we wanted to were A, B, C and D.

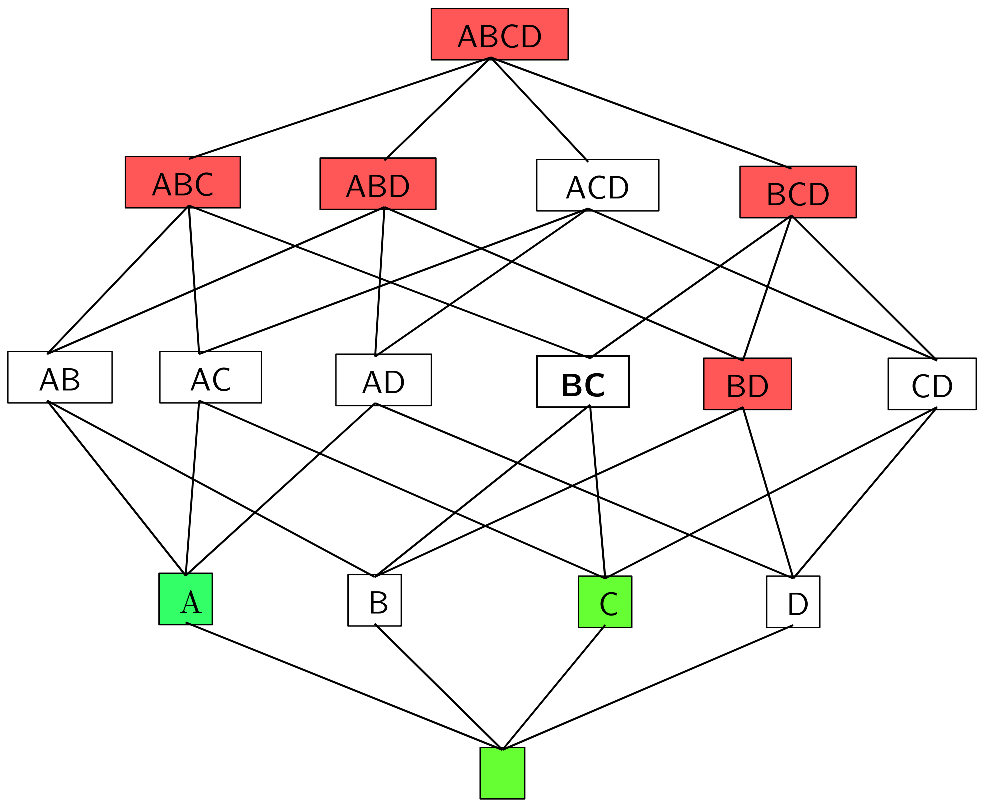

Figure 4: Reduction graph for BC with Bayes Theorem

We want to determine the probability that the reduction BC is valid, given the situation that A and C worked and BD and ABC did not work. This can be expressed with conditional probability as follows.

$$P[AB|A \wedge C \wedge \neg BD \wedge \neg ABD \wedge \neg BCD \wedge \neg ABC \wedge \neg ABCD]$$

The idea now was to solve this formula in a way so it could be easily computed for every probability. We tried to rearrange the equations for a long time, but unfortunately this did not lead to easily computable formulas. Nevertheless, one could solve this formula using Bayesian networks, but for each probability of a reduction we want to get, a new network needs to be created and this is not feasible for the amount of nodes we may get.

Simulating the probability graph

After reading several books (including Bishop, 2007), the idea came to mind to solve this network of probabilities by means of simulation. The underlying assumption is for each node we know the probability conditioned to the full situation. Here we again tried to use conditional probability to find a formula that would allow updating a node.

When rearranging the formulas, the aim was to update a node only using the probabilities of the neighboring nodes one layer below and above. It was possible to solve this formula using an approximation:

$$P(ABC|X) = P(ABC|AB \wedge BC \wedge AC \wedge X) \cdot P(AB \wedge BC \wedge AC|X)$$

$$= P(ABC|AB \wedge BC \wedge AC \wedge X) \cdot \frac{P(AB|X)P(BC|X)P(AC|X)}{P(A \wedge B \wedge C|X)}$$

$$= P(ABC|AB \wedge BC \wedge AC \wedge X) \cdot \frac{P(AB|X)P(BC|X)P(AC|X)}{P(A|X)P(B|X)P(C|X)}$$

$$= \big[P\big(ABC\big|AB \wedge BC \wedge AC \wedge X \wedge (\neg ABCY{:}\,\forall Y)\big)P(\neg ABCY{:}\,\forall Y|AB \wedge BC \wedge AC \wedge X) + P\big(ABC\big|AB \wedge BC \wedge AC \wedge X \wedge \neg(\neg ABCY{:}\,\forall Y)\big)\big(1 - P(\neg ABCY{:}\,\forall Y|AB \wedge BC \wedge AC \wedge X)\big)\big] \cdot \frac{P(AB|X)P(BC|X)P(AC|X)}{P(A|X)P(B|X)P(C|X)}$$

$$= [P\big(ABC\big|AB \wedge BC \wedge AC \wedge X \wedge (\neg ABCY{:}\,\forall Y)\big)P(\neg ABCY{:}\,\forall Y|AB \wedge BC \wedge AC \wedge X) + 1 \cdot \big(1 - P(\neg ABCY{:}\,\forall Y|AB \wedge BC \wedge AC \wedge X)\big)] \cdot \frac{P(AB|X)P(BC|X)P(AC|X)}{P(A|X)P(B|X)P(C|X)}$$

$$\approx \Big[P\big(ABC\big|AB \wedge BC \wedge AC \wedge X \wedge (\neg ABCY{:}\,\forall Y)\big)\prod\nolimits_{\forall Y} P(\neg ABCY|AB \wedge BC \wedge AC \wedge X) + 1 - \prod\nolimits_{\forall Y} P(\neg ABCY|AB \wedge BC \wedge AC \wedge X)\Big] \cdot \frac{P(AB|X)P(BC|X)P(AC|X)}{P(A|X)P(B|X)P(C|X)}$$

$$= \Big[P\big(ABC\big|AB \wedge BC \wedge AC \wedge X \wedge (\neg ABCY{:}\,\forall Y)\big)\prod\nolimits_{\forall Y}(1 - \frac{P(ABCY|X)}{P(AB \wedge BC \wedge AC|X)}) + 1 - \prod\nolimits_{\forall Y}(1 - \frac{P(ABCY|X)}{P(AB \wedge BC \wedge AC|X)})\Big] \cdot \frac{P(AB|X)P(BC|X)P(AC|X)}{P(A|X)P(B|X)P(C|X)}$$

The approximation is because we just did not see another way but to assume independence in this case. $P\big(ABC\big|AB \wedge BC \wedge AC \wedge X \wedge (\neg ABCY{:}\,\forall Y)\big)$ is the remaining unknown term which can be recursively solved as follows:

Let Y_i be the different possible Y-Elements where $i = 1, \dots, n$ and n is the amount of different possibilities for Y.

$$\mathrm{P}\big(\mathrm{ABC}\big|\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC} \wedge \mathrm{X} \wedge (\neg\mathrm{ABCY}{:}\,\forall\mathrm{Y})\big) = \mathrm{P}\big(\mathrm{ABC}\big|\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC} \wedge (\neg\mathrm{ABCY}{:}\,\forall\mathrm{Y})\big)$$
$$= P(ABC|AB \wedge BC \wedge AC \wedge (\neg ABCY_{\mathrm{i}}{:}\,\mathrm{i} = 1, \dots, \mathrm{k}))$$
$$= \frac{\mathrm{P}\big(\mathrm{ABC}\big|\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC} \wedge (\neg\mathrm{ABC}Y_{\mathrm{i}}{:}\,\mathrm{i} = 1, \dots, \mathrm{k}-1)\big) - \mathrm{P}\big(\mathrm{ABCY_k}\big|\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC} \wedge (\neg\mathrm{ABC}Y_{\mathrm{i}}{:}\,\mathrm{i} = 1, \dots, \mathrm{k}-1)\big)}{1 - \mathrm{P}\big(\mathrm{ABCY_k}\big|\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC} \wedge (\neg\mathrm{ABC}Y_{\mathrm{i}}{:}\,\mathrm{i} = 1, \dots, \mathrm{k}-1)\big)}$$
$$= \frac{\mathrm{P}\big(\mathrm{ABC}\big|\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC} \wedge (\neg\mathrm{ABC}Y_{\mathrm{i}}{:}\,\mathrm{i} = 1, \dots, \mathrm{k}-1)\big) - \mathrm{P}(\mathrm{ABCY_k}|\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC})}{1 - \mathrm{P}(\mathrm{ABCY_k}|\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC})}$$
$$= \frac{\mathrm{P}\big(\mathrm{ABC}\big|\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC} \wedge (\neg\mathrm{ABC}Y_{\mathrm{i}}{:}\,\mathrm{i} = 1, \dots, \mathrm{k}-1)\big) - \frac{\mathrm{P}(\mathrm{ABCY_k})}{\mathrm{P}(\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC})}}{1 - \frac{\mathrm{P}(\mathrm{ABCY_k})}{\mathrm{P}(\mathrm{AB} \wedge \mathrm{BC} \wedge \mathrm{AC})}}$$

The only thing that is not given now is the probability of a reduction when not knowing anything, for example $P(ABCY_k)$. This value will be learned during the process of the reduction and can be estimated with different criteria. A simple assumption would be that all reductions of a given length follow the same distribution.

As a starting distribution for the currently implemented learning table we used:

$$l_0 = 1 \qquad l_i = (l_{i-1})^i * 0.5$$

Where $i > 0$ and the index represents a reduction of that length. This table can be easily updated by averaging the values with the reported ones. Another idea we had was the distance to observed nodes as a learning table. We can basically put any criteria as an underlying distribution for the unknown nodes.

With this formula the implementation is very straightforward. The nodes are updated in each simulation cycle using the formula and when the probabilities converge or we reached a maximum of runs the simulation is aborted. A simply iteration of the remaining solutions combined with the formula to maximize the information gain with probabilities returns the best reduction when using this strategy.

Unfortunately, we realized, that due to the topology analysis we do prior to executing a reduction strategy, the formula cannot be applied to the problem so easily, as our theorem that smaller and bigger reductions are included does not hold. We can circumvent this by not using such an invalid node from the graph and instead define this node by going one layer further and so on. This should allow for a proper implementation, but we did not manage to do so in time.

4.5 Benchmarking

After developing all these different reduction strategies it was critical to build a benchmarking framework that allows a fast comparison of the strategies. As reduction runs can take several days or might never finish using certain unsuited strategies, we had to cache the results of the calculations and the costs for accepting or rejecting were averaged. This allowed easy emulation of the strategies and quite accurate comparison.

To do a thorough benchmarking of the reduction strategies, we implemented a random network generator and a random solution generator. The random networks were generated using a random amount of states and random reactions between the states following a given distribution. For each reaction a random target and several origins were chosen. However, as we were unsure about the special attributes of biological models, we opted to take three biological models we already worked with as our benchmark models.

Using these models we built random results with a probability value following a uniform distribution from 0.00 to 1.00, indicating how many solutions we get in relation to the total amount of possible models in the beginning. For example when generating solutions for a value of 0.75 case, we will randomly pick a remaining possibility from the network which will be set to valid with a 75 percent chance and rejected with a 25 percent chance. This step is repeated until there are no more remaining reductions.

Strategy	**Time to solve with high rejection cost**	**Time to solve with equal costs**
Greedy	102	5
BFS	115	26
BFS-Reverse	898	26
DFS	307	9
Heuristic	?	?

Figure 5: Benchmark of the reductions strategies

The benchmarking results (see Figure 5) were created with 3 different models and 20 different solutions for each model and then averaged over 10 runs. Each time value in the table indicates the amount of minutes the reduction strategy would have needed to find the solution and test all models.

While the BFS strategies have the same runtime with equal costs when accepted and rejected, the reverse BFS needed much more time to find the solution when the costs for a rejection are ten times as high, since it would run into more costly rejections on its way to calculate the solution.

5. Output of the reduction toolbox

After a run of the reduction toolbox and all possible reductions have been tested for validity, we can directly retrieve the result by intersecting the topology with all working reductions:

$result = topology \land accepted \land \neg rejected$

To get a list of patterns that represent all possible models that fulfill this variable, we use the BDD-Allsat operation like before (4.2 Maximum average information gain). This returns a matrix where each row is a solution and a cell indicates whether an element/parameter is included in this solution.

There are three possible values a cell can assume. It is zero when the element is excluded from the solution, one when it is included and minus one when it can be both. In our case a possible solution always includes all supersets, meaning that we can simply replace all -1 values by 0.

After we replaced all wildcards by zero, there are rows that are redundant and included by others. A solution s_1 is included by another one s_2 when the only differences are ones in s_1 where there are zeros in s_2. A solution s_1 can be removed when the following is true:

$$\exists s_1, s_2 \in solutions: s_1 \neq s_2 \wedge \forall i = 1, \dots, n: s_{1,i} = 1 \vee s_{1,i} = s_{2,i}$$

The cleaned up matrix with all solutions will be referred to as the solution matrix.

5.1 Decision Tree

As the solution matrix can get very long for many solutions, a clearer representation for the results was necessary. This can be achieved by transforming the solution matrix into a decision tree, which is a good visual representation of the solutions. Figure 8 shows an example of such a tree which really improves the readability of the output.

At each node in a decision tree, we decide either to take an element or to go another way and not collect the element at this step. A complete valid model is reached, when walking from the root to a leaf of the tree and including every visited node into the model. Alternatively we can also create the opposite tree which tells us what elements not to take for a model.

To create the decision tree, we used a divide and conquer approach that separates the rows into two subgroups at each step, building a node in the tree. The two subgroups stand for either taking an element or not taking it. The remaining rows of each subgroup are then again split recursively.

Which element we split, is determined by maximizing the information gain. There are different information gain criteria, but when creating a decision tree a well-established one is entropy. The general definition in this case is, that "the expected information gain is the change in information entropy from a prior state to a state that takes some information as given" (Wikipedia, 2011)

$$H(x) = x * \log_2 x$$
$$IG(e) = H(r) - \left(\frac{n_e}{r} * H(n_e) + \frac{r-n_e}{r} * H(r-n_e)\right)$$

Where r is the amount of rows in the current subgroup and n are the number of ones in the examined column. The information gain will be maximized by trying every $e \in Elements$.

In our case, we can also leave out the explicit option where we do not take an element, as reaching a leaf without visiting an element implies that it was not taken. Additionally, chains with no options can be merged into one node to minimize the height of the diagram.

5.2 Boolean formula

The decision tree can easily be transformed into a representation of the solutions as a Boolean formula. When visiting the nodes of the tree generated in 5.1 Decision Tree with a depth-first-search we simply attach a Boolean AND as well as a bracket opener if we go one layer deeper. We add an OR operation if we go up and immediately down again. Closing brackets are appended to the Boolean string every time we go up a layer.

6. Application of the Reduction Toolbox

6.1 Setting up the reduction method

Even though the core of the toolbox was implemented in Java, we mainly focused on an easy Matlab accessibility. This was done to support the popular SB format for ODE models (see 3.4 ODE-model analysis).

In the following we will discuss a quick example in Matlab on how to use the toolbox:

```
addpath ../redbox

% Model
sbm = SBmodel('model.txt');
SBPDmakeMEXmodel(sbm, 'model_mex');

% Setup reducer
inputStates         = { 'y9' };
outputStates        = { 'y1', 'y7' };
roots               = SBparameterZeros(sbm);
red                 = REDsbmodelReducer(sbm, inputStates, outputStates, roots);

% Constraints
red.addConstraint(red.getVar(4-1).not());

% Process
cutoff              = 154;
optimizer           = @(x)user_fitting(x, cutoff);
solutions           = REDcalculate(red, 'incl-bfs', optimizer, cutoff, 1);

% Output
display(solutions);
display(red.getSolutionsFormula(true));
red.drawSolutionsDiagram(true, true, 'result.jpg', 150, 100);
display(red.getSolutionsList()+1);
```

In the beginning of the code we basically add the reduction toolbox to our path and open a model file, which is compiled into a MEX-file to allow for faster simulation in the error function. In the next step we let the toolbox analyze the network topology by setting our input and output states (see 3. Analyzing the network topology). The zero points which nullify the parameters must also be provided, but can be automatically retrieved when the model uses reactions. In addition we can define constraints by getting a variable using the JavaBDD variables.

The main process is started using the REDcalculate function by providing a user defined error function and a cutoff. In most cases the error function has to be nested into parameter optimizer function which tries to minimize the error function. See the examples on how to setup a parameter optimizer and an error function. After all calculations are done, the different representations discussed in 5. Output of the reduction toolbox are used as an output.

In the example we chose 'incl-bfs' as a strategy, which will perform a BFS search while following the theorem of included sub and super-sets for reduction. We implemented the following different strategies:

Strategy	Abbreviation	Description
Explicit	explicit	Explicitly tests all possibilities without assuming the theorem.
BFS	incl-bfs	Uses theorem and performs breadth first search, reducing few elements first
BFS (reverse)	incl-bfsreverse	Same as BFS but reverse, reducing many elements first
DFS	incl-dfs	Uses theorem and performs depth first search
Greedy	incl-greedy	Uses theorem and tries to reduce as possibilities as possible on average
Heuristic	*not done*	Uses theorem and performs a probability estimation using a heuristic based on Bayesian networks

6.2 Determining a cutoff value

When determining a cutoff we have to ask ourselves what models do we want to filter. The best possible result should occur, when we remove no parameters at all. A common application of the toolbox is to simply get all models that are not more than 10% worse than the one with the lowest error. In this case we would multiply the best error for no removed parameters by 1.1 and this would be our cutoff.

Another way is when our reference model is not the one without any parameters removed, but a model that is only a sub model of the complete one. In this case we usually want to know what parts we need to add in order to improve the error value. When we want to improve the model by at least 10%, the cutoff would be 0.9 times the error of our reference model. It is clear that determining a cutoff is pretty straight forward.

6.3 Modeling the dynamics for fatty acid β-oxidation

Our first real application was to aid in the improvement of a model for the fatty acid β-oxidation, which was developed by Ferdinand Stückler and Alexandra Grigore. The core of the model is called the β-oxidation cascade and is structured as a chain (see Figure 6) where a small portion directly reaches the last output state.

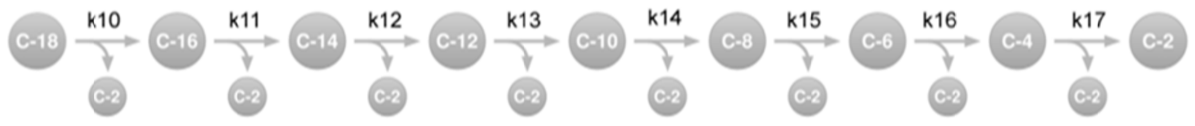

Figure 6: β-oxidation cascade

When comparing measured data from patients with a simulation of this model, it became clear that the chain cannot be everything that is needed. Ferdinand and Alexandra made the assumption that there might be some linear reactions leading in and out from every state in the chain and added them with unknown states to the model (see Figure 7).

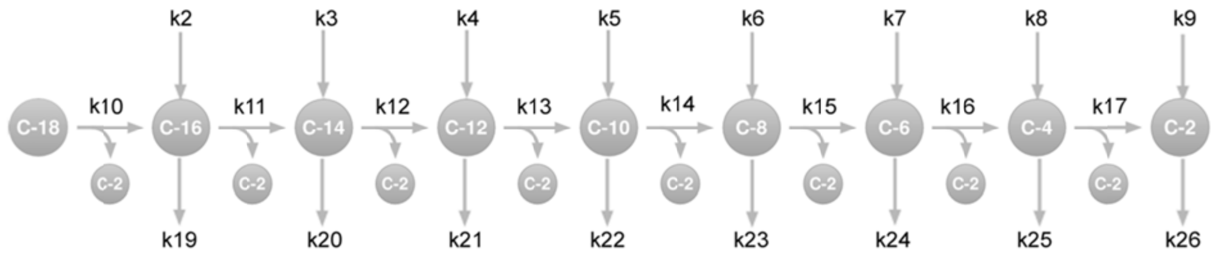

Figure 7: Extension of the β-oxidation chain

The reduction method was now used to reduce this blown up model and get the reactions that are really necessary to reach a significantly lower error value. Conveniently Alexandra managed to combine the patient data into 3 different clusters.

For each dataset we had to determine a proper cutoff which was done by simulating the model with all parameters and then when removing all but the core chain. The cutoff was chosen slightly above the best error of all parameters to account for optimization approximation errors.

Datasets	all parameters error	core chain error	cutoff <= error
all patients	0.4236	0.5964	0.45
cluster 1	0.9255	1.6441	0.95
cluster 2	1.3641	1.5647	1.40
cluster 3	0.4735	0.6602	0.55

Each of these runs returned a set of different models that has an error value below this cutoff.

All patients:

```
k_10 & k_11 & k_12 & k_13 & k_14 & k_15 & k_16 & k_17 & (k_4 & k_21 | k_5)
```

Cluster 1:

```
k_5 & k_4 & k_10 & k_11 & k_12 & k_13 & k_14 & k_15 & k_16 & k_17 & (k_2 &
k_20 & k_23 & (k_24 & (k_18 | k_19 | k_6) | k_3) | k_21 & (k_18 | k_19 |
k_2 | k_23 | k_24 | k_25 | k_3 | k_7) | k_22) | k_4 & k_6 & k_10 & k_11 &
k_12 & k_13 & k_14 & k_15 & k_16 & k_17 & (k_21 | k_22 & (k_2 | k_3))
```

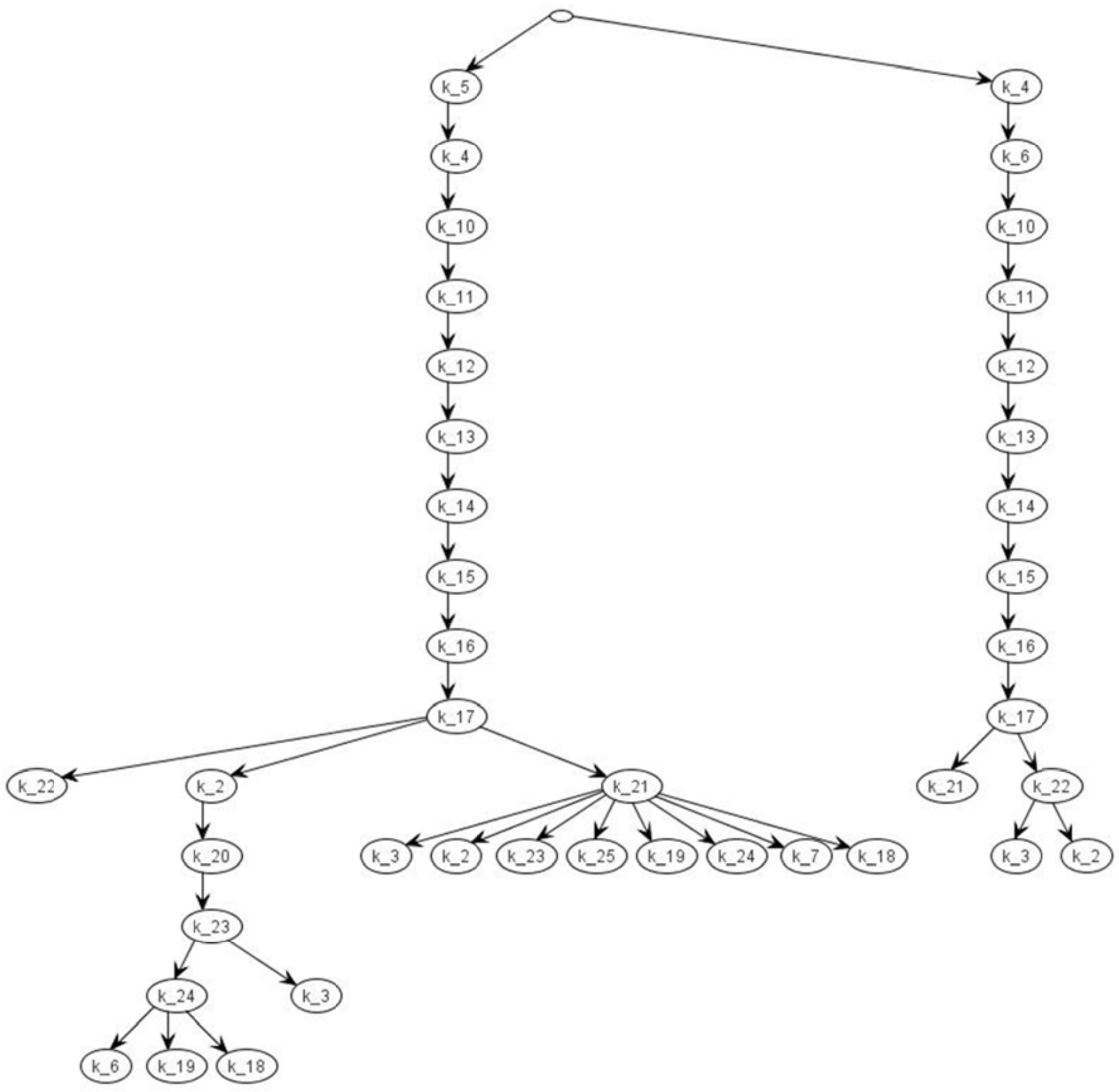

Figure 8: Decision tree for the solutions of Cluster 1

Cluster 2:

```
k_5 & k_8 & k_10 & k_11 & k_12 & k_13 & k_14 & k_15 & k_16 & k_17 & k_21 &
k_23 & k_25
```

Cluster 3:

```
k_10 & k_11 & k_12 & k_13 & k_14 & k_15 & k_16 & k_17 & (k_4 & (k_20 | k_21
| k_22 | k_6) | k_5 | k_8 | k_9)
```

We can see that these formulas can get pretty messy. Usually one would try to further lower the cutoff in order to decrease the number of possible models, but as the simulated annealing already had problems with the 24 parameters, we had to give it some room for error. For Cluster 1 the decision tree was attached as the formula is hardly readable anymore, but when looking at the decision tree the result becomes quite clear.

What all these results have in common is an incoming reaction at C-10 with k5 or at C-12 with k4, which works for most of them. In some cases it is even enough to only have input reactions at the end. It seems that starting from C-12, there is definitely some input missing, as we always need more concentrations when we further get down the chain.

However, we have to note that the error was only lowered by an average of 25% for the different data sets. This might not justify adding an additional parameter when we want to keep the model complexity to a minimum.

As our results were a bit unspecific and did not directly indicate which model to use, a comparison run was made using BCI, a method which basically does a similar calculation, but it punishes for every additional parameter we add so we can simply use one parameter optimizer. The results of BCI confirmed our results in almost every case, since the best results indicated an input reaction at C-12 and C-10 or C-2, or no input parameter at all as the punishment was very high.

6.4 Compartment model for Ingestion of Zirconium in Human Body

In an interesting talk by Sabine Hug we heard that she and Daniel Schmidl, two other group members of the CMB of the Helmholtz Zentrum, were working on a compartment model (called HMGU) for the biokinetics of zirconium in the human body after ingestion (Schmiedl, Hug, Bo Lie, Greiter, & Theis, 2011). This model was compared with another one that models the same process but is from the International Commission on Radiology Protect (IRCP). Both models contain only reactions have Mass action kinetics and each reaction only has one substrate and one product.

When we approached them they were interested to see in what kind of model a reduction of a merge between the IRCP and their HMGU model would result in. They claimed that the HMGU model made more biological sense hoped it would be the result of the reduction. Further they wanted to know whether the HMGU model is final or whether parts of it are unnecessary for the model to reach the same cutoff.

We used parameter estimations from prior experiments to define a 95% confidence interval as an upper bound for our parameter values. The lower value was not used as we have to allow the parameters to reach zero. Unfortunately both the IRCP and HMGU had a similar error when sum of squared residuals (SSR) over all patients as an error function. Thus a reduction for the merged models does not make sense as it would in the best case only return both models separately.

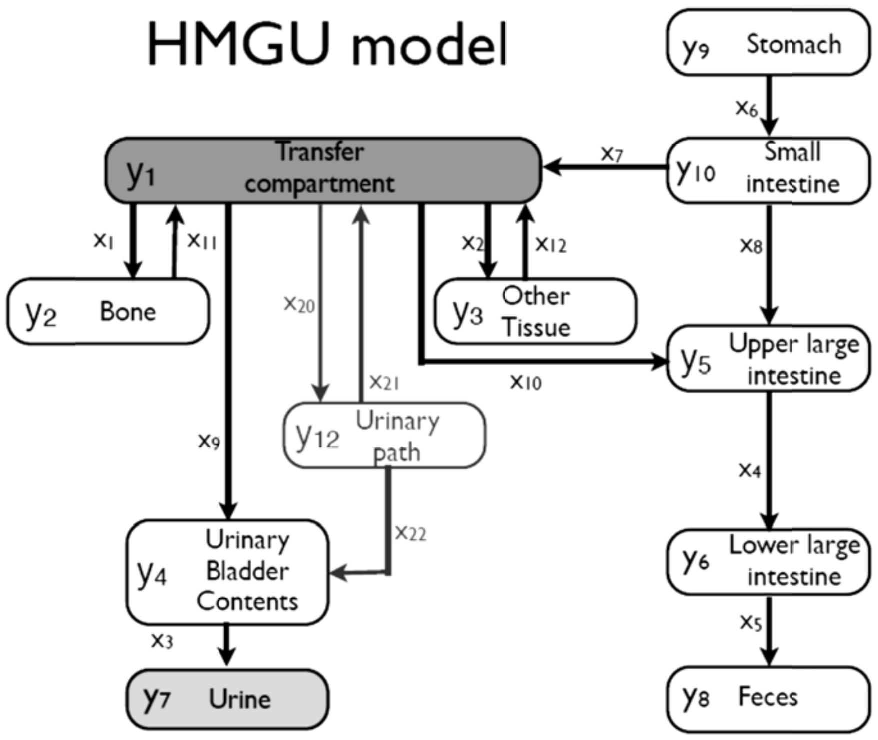

Figure 9: HMGU model for ingestion of zirconium

However, the reduction for the HMGU model (see Figure 9) looked more promising. We set constraints to consider only models that followed certain rules to ensure that the result made sense from a biological perspective.

x1 & x2 & x3 & x6 & x7 & x8 & x11 & x12 & x20 & x21 & (x22 | x9)

The interpretation of the result is that we do not need x10, which is the reaction from the Transfer compartment (basically blood) to the Upper large intestine. Secondly, we only need one way leading to the Urine, either x9 or x22, but definitely not both. As a direct connection from the blood to the bladder does not make sense, x22 which first goes through the urinary bladder should sufficient.

x4 and x5 were reduced from the model as they cannot be observed and have no impact on the blood or urine, but should not be removed from the model. It seems that after removing x9 and x10, the model looks even more biologically realistic.

7. Discussion and perspectives

The development of the reduction toolbox went very smooth until we hit a rock when exploring different reduction strategies. We tried many different strategies, but most are not shown in the benchmark due to poor performance. It was very hard to find a strategy that performed well for every model. Judging from the benchmark results, the greedy strategy is performing well on average, but is not better than the BFS if we have prior information about the results we expect.

Unfortunately we did not manage to implement the heuristic strategy in time. The approximation of the model probabilities in combination with the maximum information gain seemed to be the best way to go. We still hope that our heuristic will be able to drastically improve the runtime of the reduction toolbox and make the method applicable even for a great amount of initial possibilities.

Still, the current implementation the toolbox is very versatile and we feel it is quite easy to apply the reduction method to most network based problems. In the very end we even decided to split the main theorem of inclusion from the core reduction toolbox. This allows for a general application of the method, where we can do explicit model reduction given that we define enough constraints to keep the amount of possible models to a minimum.

So far we only allowed the removal of different reactions in the reduction method. The next step would be to include additional reduction operations, such as the merging of different states. To add this option, we would need to define the merging of two states with simple Boolean logic.

We could define a series of Boolean variables that indicate whether two states are merged in a pairwise manner. To account for all possible mergers, we need $\frac{s*(s-1)}{2}$ variables, where s is the amount of states. In case we merge three states together, we would only need two variables and not three to represent the merger. Therefore we perform a prior analysis of all possible combinations of values for the variables. For our example we would restrict the third variable to true. Additionally, we can prohibit self-loops of states and other motives that would not make sense in a biological way.

This structure would allow an easy representation of the possible mergers. However, the method would only be feasible if we could somehow apply the theorem of inclusion to the merging of states. This would depend highly on the type of model and at this point we have not done enough investigation into the issue.

To conclude this work, we think that the main objective of building a toolbox for the systematic reduction of models was achieved successfully. During the development, the toolbox was extensively tested and was able to provide valuable input for other groups as shown in the results of the examples. We feel the method might be a valuable tool for other research groups that work on developing models to describe biological systems.

References

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning.*

Dama, M. (2009). *Simulated Annealing in Trading Optimizartion.* Retrieved from http://maxdama.blogspot.com/2008/07/trading-optimization-simulated.html

H. Schmidt, M. J. (2005, November). Systems Biology Toolbox for MATLAB: A computational platform for research in Systems Biology. *Bioinformatics Advance Access.*

Rickert, D. (2009). *Empirical reduction of signaling networks.*

Schmiedl, D., Hug, S., Bo Lie, W., Greiter, M., & Theis, F. (2011, August 12). Bayesian Model Selection determines the Biokinetics of Zirconium in the Human Body after Ingestion.

Whaley, J. (2007). *JavaBDD.* Retrieved from http://javabdd.sourceforge.net

Wikipedia. (2011). *Information gain in decision trees.* Retrieved August 5, 2011, from http://en.wikipedia.org/wiki/Information_gain_in_decision_trees

Table of Figures