

Metabolische Stimulus-Response-Experimente: Werkzeuge zur Modellierung, Simulation und Auswertung

Vom Fachbereich Elektrotechnik und Informatik der
Universität Siegen

zur Erlangung des akademischen Grades

**Doktor der Ingenieurwissenschaften
(Dr.-Ing.)**

genehmigte Dissertation

von

Marc Daniel Haunschild

1. Gutachter: Prof. Dr. Wolfgang Wiechert, Universität Siegen

2. Gutachter: Prof. Dr. Bernd Freisleben, Universität Marburg

Vorsitzender: Prof. Dr. Rainer Brück, Universität Siegen

Tag der mündlichen Prüfung: 13.01.2006

urn:nbn:de:hbz:467-2077

Danksagungen

Die vorliegende Dissertationsschrift basiert auf der Forschungsarbeit, die gemeinsam an der Universität Siegen, der Forschungszentrum Jülich GmbH und der Philipps-Universität Marburg geleistet wurde. Ganz besonders danken möchte ich

- Prof. Dr. Wolfgang Wiechert, Universität Siegen, für die Betreuung dieser Arbeit und für seine umfassende Hilfe und gute Erreichbarkeit.
- Prof. Dr. Bernd Freisleben, Arbeitsgruppe für verteilte Systeme im Fachbereich Mathematik und Informatik der Universität Marburg, für Beratungen und Diskussionen bzgl. der Informatik-Themen dieser Arbeit.
- Dr. habil. Ralf Takors als Leiter der Fermentationsgruppe am Institut für Biotechnologie, Forschungszentrum Jülich, und sein Nachfolger Dr. Marco Oldiges, für die Kooperation im Rahmen des DFG-Projekts. Insbesondere für die experimentellen Grundlagenforschungen, die ganz essentiell für die Anwendung dieser Arbeit sind.
- Dr. Daniela Degenring und Dipl. Math. Cornelia Frömel, Forschungszentrum Jülich, die während der Umstellung von MMT1 auf MMT2 mit beiden Softwaretools parallel gearbeitet haben und mit viel Geduld die Einführung von MMT2 ermöglicht haben.
- Dipl.-Ing. S. Aljoscha Wahl und Dipl.-Ing. Jorgen Magnus, Forschungszentrum Jülich, als Nutzer von MMT2 für das Testen und Einbringen von Ideen. Auch ihnen möchte ich für die Geduld danken, die Beta-Tester haben müssen.
- Dipl.-Inf. Ermir Qeli, Universität Marburg als Kooperationspartner hinsichtlich der Visualisierungssoftware MetVis und MatVis, die auf den von MMT2 produzierten Simulationsdaten aufsetzen.
- Dipl.-Inf. Thomas Friese und Dipl.-Ing. Matthew Smith, Universität Marburg, für die Zusammenarbeit bei der Eingliederung von MMT2 in die von ihnen entwickelte Software für Ad-Hoc Grid-Computing.
- Dr. Jochen Hurlebaus für die Pionierarbeit mit MMT1, auf dessen Erfahrungen in dieser Arbeit zurückgegriffen werden konnte.
- Allen Mitarbeitern des Lehrstuhls Simulationstechnik an der Universität Siegen die mir in vieler Hinsicht geholfen haben diese Arbeit erfolgreich zu bearbeiten.
- Und auch ganz besonders meiner lieben Frau Susanna.

Zusammenfassung

Die Modellbildung metabolischer Systeme wurde in den letzten Jahren zu einem der Hauptarbeitsgebiete im Metabolic Engineering, um die komplexen Regelmechanismen einer lebenden Zelle zu verstehen. In dieser Arbeit wurde ein vielseitiges Werkzeug entwickelt, um Modellierer bei der Erstellung und Überarbeitung von Modellen zu unterstützen, die auf Messdaten aus einem Stimulus-Response-Experiment aufbauen.

Im Verlauf der Modellbildung ist der Modellierer meist nicht nur mit einem einzigen Modell beschäftigt, sondern mit Sequenzen, Alternativen und strukturellen Varianten von Modellen. Für die Unterstützung der Modellbildung dynamischer biochemischer Netzwerke, die auf *in-vivo* Daten basieren, ist daher mehr als nur Simulation erforderlich. In dieser Arbeit wurde ein neues Konzept für Modellfamilien spezifiziert und implementiert. Mit diesem Konzept können eine Vielzahl von ähnlichen Modellen in einer einzigen Beschreibung gespeichert werden, mit der Hilfe von Netzwerk- und kinetischen Varianten. Dadurch wird ein automatisches Navigieren im Raum der Modellvarianten möglich, in dem biologisch unsinnige Modelle auf der Basis einer Elementarmodenanalyse ausgeschlossen werden können.

Das Einbeziehen von Messdaten wird durch die Möglichkeit unterstützt, Splines an Stelle von Zustandsvariablen zu verwenden. Anschließend werden leistungsfähige automatische Methoden benötigt, die den Modellierer bei der Modellbildung, Organisation und Auswertung alternativer Modelle unterstützen.

Dieses Werkzeug wurde als Rechenkern entworfen, der in eine Kette von Werkzeugen eingebaut werden soll. Durch automatische Codegenerierung, automatische Differentiation zur Sensitivitätsanalyse und Grid-Computing Technologie wird eine Hochleistungsrechenumgebung erstellt. Sie unterstützt die Modellspezifikation in XML und bietet mehrere Softwareschnittstellen.

Die Leistungsfähigkeit und Nutzbarkeit des Softwarewerkzeugs dieser Arbeit wird an mehreren Beispielen von laufenden Forschungsprojekten gezeigt. Darüber hinaus wurde ein Optimierungsalgorithmus entwickelt, der dem Softwarewerkzeug ermöglicht die Aufgabe der Modellvariantensuche und Parameteranpassung zu übernehmen. Das Ergebnis der Berechnung ist ein Ranking der Modellvarianten, die die Messdaten am besten reproduziert haben.

Summary

Metabolic modeling has become a major activity in metabolic engineering in recent years, in order to understand the complex regulation phenomena in a living cell. In this thesis, a versatile tool has been developed to support a modeler with the setup and refining of a model, that simulates the data extracted from a rapidly sampled stimulus response experiment.

In the course of the modeling process, the modeler is typically not only concerned with a single model but with sequences, alternatives and structural variants of models. Supporting the modeling process of dynamic biochemical networks based on sampled in vivo data requires more than just simulation. For this purpose, the new concept of model families is specified and implemented in this thesis. With this concept, a multitude of similar models can be formulated in a single description by using network and kinetic variants. The concept allows to automatically navigate in the space of models and to exclude biologically meaningless models on the basis of elementary flux mode analysis.

An incremental usage of the measured data is supported by using splined data instead of state variables. Powerful automatic methods are then required to assist the modeler in the organization and evaluation of alternative models.

This tool has been developed as a computational engine, intended to be built into a tool chain. By the use of automatic code generation, automatic differentiation for sensitivity analysis, and Grid computing technology, a high performance computing environment is achieved. It supplies XML model specification and several software interfaces.

The performance and usability of the tool of this thesis is illustrated by several examples from ongoing research projects. An optimization algorithm was developed, enabling the software tool to automatically carry through the tasks of model variant switching and parameter fitting. The computation results into a ranking of model variants fitting best to the experimental data.

Inhaltsverzeichnis

Danksagungen	iii
Zusammenfassung Deutsch	v
Zusammenfassung Englisch	vii
Inhaltsverzeichnis	ix
Abkürzungsverzeichnis	xiii
I Grundlagen	1
1 Einleitung	3
1.1 Anwendungsgebiet dieser Arbeit	3
1.2 Problematik der Modellbildung in der Biologie	4
1.3 Systembiologie und Metabolic Engineering	6
1.4 Projektkontext und Zielsetzung	8
1.5 Struktur dieser Arbeit	8
1.6 Experimentelle Modellbildung	9
2 Experimentelle Grundlagen	13
2.1 In Vitro und In Vivo Experimente	13
2.2 <i>E. coli</i> und <i>C. glutamicum</i> als Modellorganismen	15
2.3 Stimulus-Response-Experimente und schnelle Probenahme	16
2.4 Zellaufschluss und Analytik	19
3 Modellierung metabolischer Netzwerke	21
3.1 Modellierung komplexer Systeme	21
3.2 Problematik der Modellbildung biologischer Systeme	23
3.3 Vereinfachende Modellannahmen	25
3.4 Struktur metabolischer Modelle	27
3.5 Stöchiometrische Matrix	29
3.6 Chemische Reaktionskinetiken	30

3.7	Beispiel	31
3.8	Wahl der Systemgrenzen	32
4	Verfügbare Software	35
4.1	Simulation ganzer Zellen	36
4.2	Simulation (bio-)chemischer Reaktionsnetzwerke	37
4.3	Experimentelle Modellbildung	37
4.4	Spezialanwendungen	38
5	Präzisierung der Zielsetzung dieser Arbeit	39
5.1	MMT1	39
5.2	Fortsetzung des Projekts	41
5.3	Rahmenbedingungen und Umsetzung	42
II	Methoden und Implementierung	47
6	Modellfamilien als Modellierungswerkzeug	49
6.1	Modellbasierte Datenauswertung	49
6.2	Kinetische Varianten	51
6.3	Netzwerkvarianten	53
6.4	Kombinatorische Modellgenerierung	55
6.5	Zusammenlegung kinetischer- und Netzwerk-Varianten	56
6.6	Ausschluss biologisch sinnloser Modelle	58
6.7	Elementare Flussmoden	58
7	Metabolic Modeling Markup Language - M3L	61
7.1	Überblick über existierende Modellierungssprachen	61
7.2	Modellvarianten	64
7.3	Messdaten	64
7.4	Splines	67
7.5	Verfahrensparameter	69
8	Source-Code Generierung	71
8.1	Architektur	72
8.2	Implementierung	73
8.3	Hilfsprogramme	75
9	Numerische Methoden	79

9.1	ODE-Löser	80
9.2	Modellbewertung	82
9.3	Parameteranpassung	86
9.4	Sensitivitätsanalyse	90
9.5	Automatische Differenzierung	92
 III Anwendung von MMT2		97
 10 Exemplarische Modellstudie mit MMT2		99
10.1	Erstellung der Modellstruktur	99
10.2	Vervollständigen der Modellstruktur	100
10.3	Simulatorerzeugung und Simulationsläufe	103
10.4	Messdaten hinzufügen	104
10.5	Parameteranpassung	106
10.6	Splines hinzufügen	107
10.7	Modellfamilie erstellen	110
 11 Komplexe Anwendungsbeispiele		113
11.1	Auswertungsprozess für Rapid-Sampling-Experimente	115
11.2	<i>E. coli</i> L-Phenylalanin Produktionsstamm	116
11.3	<i>C. glutamicum</i> L-Valin Produktionsstamm	119
 IV Modellsuche		123
 12 Formale Beschreibung von Modellfamilien		125
12.1	Grundlegende Annahmen	125
12.2	Allgemeine Struktur von Modellfamilien	129
12.3	Anwendungsbeispiel: Metabolische Modelle	134
 13 Optimierungsalgorithmus für die Modellselektion		139
13.1	Eigenschaften des Modellraums	139
13.2	Vorgehensweise	140
13.3	Struktur des Algorithmus	142
13.4	Beschreibung des Algorithmus	144
13.5	Implementierung	147
 14 Analyse des Modellselektionsalgorithmus		149

14.1	Referenzmodell	149
14.2	Testmodellfamilie	151
14.3	Verfahrensparameter	152
14.4	Analyse des Laufzeitverhaltens	153
14.5	Analyse des Ergebnisses	156
14.6	Verrauschte Datenbasis	158
14.7	Schlussfolgerungen	158
14.8	Zusammenfassung	159
15	High-Performance Computing	161
15.1	Compute-Cluster	161
15.2	Grid Computing	163
15.3	MMT2 als Grid-Service	164
V	Schluss	167
16	Zusammenfassung und Diskussion	169
16.1	Zielsetzung	169
16.2	Verlauf des Projekts	170
16.3	Modellfamilien und Modelldiskriminierung	170
16.4	Resultate und Ausblick	171

Abkürzungsverzeichnis

1. Chemische Substanzen

Abkürzung	Bedeutung
6PG	6-Phosphogluconat
AcCoA	Acetyl-Coenzym A
ADP	Adenosindiphosphat
AIC	Akaike Informations Kriterium
AMP	Adenosinmonophosphat
API	Application Programmable Interface
ATP	Adenosintriphosphat
BM	Biomasse
C5P	Summenkonzentration der gemessenen Zucker-5-Phosphate des PPP
DAHP	3-Deoxy-D-arabino-heptulosonat-7-phosphat
DHAP	Dihydroxyacetonphosphat (Glyceronphosphat)
DHQ	3-Dehydrochinat
DHS	3-Dehydroshikimat
E4P	Erythrose-4-phosphat
EPSP	5-Enolpyruvoyl-3-phosphat
F6P	Fructose-6-phosphat
FBP	Fructose- 1,6-bisphosphat
G6P	Glucose-6-phosphat
GAP	Glyceraldehyd-3-phosphat
LEU	L-Leucin
L-Phe	L-Phenylalanin
NAD/ NADH	Nicotinamid-Adenin-Dinucleotid (oxidierte/reduzierte Form)
NADP/ NADPH	Nikotinamid-Adenin-Dinucleotid-Phosphat (oxidierte/reduzierte Form)

Abkürzung	Bedeutung
PEP	Phosphoenolpyruvat
Pyr	Pyruvat
R5P	Ribose-5-Phosphat
RIBU5P	Ribose-5-Phosphat
S3P	Shikimat-3- Phosphat
S7P	Sedo-heptulose-7-Phosphat
SHIK	Shikimat
SRE	Stimulus-Response-Experiment
TCA	Zitronensäurezyklus (Cydric Acid Cycle)
VAL	L-Valin
X5P	Xylulose-5-Phosphat
XYL5P	Xylulose-5-Phosphat

2. Biotechnologie / Biologie

Abkürzung	Bedeutung
C. glutamicum	Corynebacterium Glutamicum (Bodenbakterium)
E. coli	Escherischa Coli (Darmbakterium)
In-Silico	(Lat: im Silizium) Am Rechner durchgeführte Experimente
In-Vitro	(Lat: im Reagenzglas) Experimente unter künstlichen Bedingungen
In-Vivo	(Lat: im Leben) Experimente im lebenden System
LC-MS	Liquid Chromatography Mass Spectrometry
MM	Michaelis-Menten Kinetik
S. cerevisiae	Saccharomyces Cerevisiae (Backhefe)

3. Stoffwechselwege

Abkürzung	Bedeutung
EMP	Embden Meyerhof Parnas Pathway
PPP	Pentose Phosphate Pathway
ANA	Anaplerotic Reactions
GS	Glyoxylate Shunt
TCC	Tricarboxylic Acid Cycle

4. Algorithmen / Softwarewerkzeuge

Abkürzung	Bedeutung
BDF	Backward Differentiation
BFGS	Broyden-Fletcher-Goldfarb-Shanno
CSV	Comma Separated Values
ES	Evolutionsstrategie
FD	Finite Differenz(en)
FQS	Fehlerquadratsumme
GUI	Graphical User Interface
JNI	Java Native Interface
LSODA	Livermore Solver for Ordinary Differential Equations
ODE	Ordinary Differential Equation (gewöhnliche Differentialgleichung)
PDE	Partial Differential Equation (partielle Differentialgleichung)
M3L	Metabolic Modeling Markup Language
MatVis	Matrix Visualizer
MetVis	Metabolic Visualizer
MMT	Metabolic Modeling Tool
NAG	Numerical Algorithms Group
NMS	Nelder-Mead-Simplex
RDBS	Relationales Datenbanksystem
RMI	Remote Method Invokation
RPM	Redhat Package Manager
SBML	Systems Biology Markup Language
SPP	Schwerpunktprogramm
SSQ	Sum of Squares (Fehlerquadratsumme)
UML	Unified Modeling Language
XML	Extensible Markup Language

5. Sonstige

Abkürzung	Bedeutung
DFG	Deutsche Forschungsgemeinschaft
IBT	Institut für Biotechnologie

Teil I

Grundlagen

Einleitung

Dieses Kapitel stellt das Anwendungsgebiet für das im Rahmen dieser Arbeit entwickelte Werkzeug vor. Dazu wird zunächst ein grober Rahmen gesteckt (Abschnitt 1.1) und dann die zentrale Problemstellung diskutiert (Abschnitt 1.2). Anschließend wird der allgemeine Forschungsbereich vorgestellt in dem sich diese Arbeit ansiedelt (Abschnitt 1.3) sowie der Projektkontext dieser Arbeit (Abschnitt 1.4). Schließlich wird der Aufbau dieser Arbeit vorgestellt (Abschnitt 1.5), bei der die Vorgehensweise der experimentellen Modellbildung als Leitfaden dient (Abschnitt 1.6).

1.1. Anwendungsgebiet dieser Arbeit

Die Biologie hat in den letzten Jahrzehnten durch die Entwicklung von immer präziseren analytischen Methoden zur Quantifizierung von chemischen Stoffen in zellulären Systemen einen Stand erreicht, in dem die Fülle der verfügbaren experimentellen Daten nicht mehr ohne die Unterstützung von speziell für das Anwendungsgebiet entwickelter Software zu bearbeiten ist. Die Synthese der Daten in formalen mathematischen Modellen, die das gemessene Zellverhalten in Bezug auf die zugrundeliegenden Fragestellungen beschreiben, wird damit immer mehr zur Notwendigkeit, weil die Komplexität der Interaktionen der Zellkomponenten und die Menge der dazu verfügbaren Daten ohne ein systematisches Framework nicht mehr zu bewältigen sind.

Die mathematische Modellierung ist in der Physik, Chemie und in den Ingenieurwissenschaften ein etabliertes Werkzeug zur Interpretation und Vorhersage natürlicher Phänomene und experimenteller Daten. Auch die Simulation dynamischer biochemischer Systeme wird immer mehr propagiert um ein ausreichendes interpretatives und vor allem ein prädiktives Verständnis zellulären Verhaltens zu entwickeln.

In der Mikrobiologie stellen Modelle üblicherweise empirische Zusammenstellungen aktuellen Wissens und Hypothesen über fehlende Informationen dar und bieten damit die Möglichkeit im Rahmen einer Modelldiskriminierung durch Falsifizierung oder Validierung über die Eingangs angenommene Verhaltensbeschreibung Aussagen

zu treffen. Nach der Erstellung eines für die Fragestellung und die verfügbaren Daten adäquaten Modells, gibt es verschiedene Ansätze um die Modelleigenschaften zu analysieren. Bei den Modellen, die in dieser Arbeit behandelt werden, handelt es sich um dynamische metabolische Modelle, auf der Grundlage von Differentialgleichungen, die überwiegend durch Simulation und Sensitivitätsanalysen untersucht werden müssen.

Es besteht kein Zweifel daran, dass Computerwerkzeuge absolut notwendig sind, um diese Aufgabenstellungen zu bearbeiten. Weil sich die Mikrobiologie noch am Anfang eines umfassenden Verständnisses zellulärer Systeme *in-vivo* befindet, ist eine große Vielfalt an Vorgehensweisen bei der Modellbildung entstanden und damit auch eine große Menge an Softwarewerkzeugen, die auf ganz bestimmte Vorgehensweisen, Fragestellungen und Datenquellen abgestimmt sein müssen. Trotzdem verbleiben Bereiche, in denen bisher keine adäquaten Computerwerkzeuge verfügbar sind. Im Rahmen dieser Arbeit wurde ein Computerwerkzeug entwickelt, das auf einen speziellen Typ von Experimenten zugeschnitten ist.

1.2. Problematik der Modellbildung in der Biologie

.....

Im Allgemeinen wird die mathematische Modellbildung komplexer Systeme mit verschiedenen Zielsetzungen betrieben. Sind die Interaktionen der Einheiten eines Systems gut bekannt und gut zu beschreiben, so sind die Voraussetzungen geradezu ideal für die Modellbildung. Dies ist beispielsweise in der Elektrotechnik der Fall, in der es Schaltkreissimulatoren gibt, die so hoch entwickelt und realitätstreu sind, dass man in vielen Fällen bereits gänzlich auf den Bau von Prototypen verzichten kann [Wiechert, 2002]. Die Entwicklung völlig neuer technischer Systeme und die Optimierung von bestehenden Systemen wird in der Anfangsphase nur am Computer durchgeführt, weil die verfügbaren Simulatoren in der Lage sind die Interaktionen zwischen den Bauteilen so genau berechnen, dass durch eine Änderung am realen System das veränderte Systemverhalten im Computer mit ausreichender Genauigkeit vorhergesagt werden kann.

Eine wesentliche Charakteristik von Schaltkreissimulatoren liegt darin, dass es relativ wenige Grundkomponenten gibt (Widerstand, Spule, Kondensator), die an vielen Stellen des Systems wieder verwendet werden können. Weiterhin sind das Verhalten der Einzelkomponenten sowie auch die Interaktionen zwischen den Komponenten

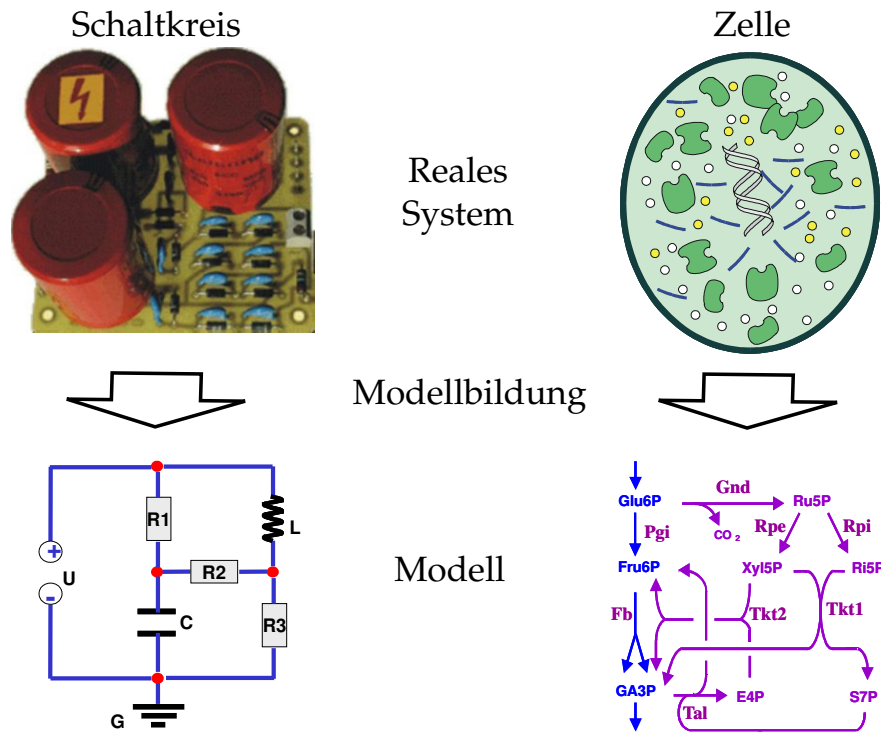


Abbildung 1.1: Gegenüberstellung der Modellbildung in der Elektrotechnik und Biologie. Links: Die Systemstruktur eines Schaltkreises ist genau bekannt und kann in das Modell übernommen werden. Rechts: In der Biologie sind die möglichen Wechselwirkungen zwischen den Komponenten nicht vollständig.

gut bekannt, da die Systemstruktur (durch die Leiterbahnen) vorgegeben ist (Abb. 1.1 links).

Die Situation in der Mikrobiologie ist dagegen gänzlich anders. Bei Zellen (Abb. 1.1 rechts) ist zum Einen die zugrunde liegende Systemstruktur nicht vollständig bekannt und zum Anderen die Anzahl der Grundkomponenten wesentlich höher. Weiterhin ist die einzelne Zelle ein hochkomplexes System, in dem alle Grundkomponenten in einer wässrigen Lösung, dem Cytoplasma, gelöst sind, in dem keine Systemstruktur erkennbar ist. Man kann also die Wechselwirkung zweier oder mehr chemischer Stoffe nicht von vorn herein ausschließen oder annehmen.

Die molekularen Bestandteile und viele Wechselwirkungen im Inneren der Zelle wurden zwar schon gut erforscht, aber „anders als in der Physik ist die Biologie bis heute nicht über das Stadium einer induktiven, beschreibenden Wissenschaft hinausgekommen ... Die Biologie befindet sich in einem zur Physik vor Newton vergleichbaren Entwicklungszustand“, was [Bleeken, 1990] vor mehr als zehn Jahren zu bedenken gab.

Das Wissen über den Metabolismus ist in der Mikrobiologie in den letzten Jahrzehnten erheblich angewachsen, was den Fortschritten in den biochemischen Messmetho-

den zu verdanken ist. Dadurch konnten die Funktionsweisen von verschiedenen Enzymen und chemischen Stoffen in der Zelle erforscht werden. Aber „der Molekularbiologie ist es nicht gelungen, den Weg von den Einzelkomponenten zurück zum lebenden Organismus zu finden ... In den Kreis ihrer Bewunderer mischen sich daher zunehmend kritische Stimmen, die beklagen, dass die Molekularbiologie ihren Ehrgeiz aufgegeben hat, das Funktionieren eines lebenden Organismus zu erklären, und ihre Forschungsstrategie wegen des Fehlens eines theoretischen Rahmens zur reinen Daten- und Faktensammlung verflacht.“ [Bleeken, 1990]

Seit den 90'er Jahren, in denen [Bleeken, 1990] diesen provokativen Anstoß gegeben hat, ist die Rechenleistung der Computer um ein zigfaches angewachsen, was das Aufkommen der Systembiologie ermöglichte.

1.3. Systembiologie und Metabolic Engineering

.....

Das Hauptziel der Systembiologie ist es, die Struktur und Dynamik biologischer Systeme zu verstehen, wozu [Kitano, 2002] feststellt: „Um die Biologie auf Systemebene zu verstehen, müssen wir die Struktur und Dynamik ... untersuchen, anstatt die Charakteristika isolierter Bestandteile einer Zelle oder eines Organismus.“

Dazu ist zum Einen Modellbildung und Simulation nötig [Tomita, 2001]: „Das komplexe Verhalten der Zelle kann nicht verstanden oder vorhergesagt werden wenn nicht ein Computermodell der Zelle erstellt und simuliert wird.“ Zum Anderen wird Datenmaterial über biologische Systeme benötigt [Kitano, 2002], um die Modellbildung mit der Zielsetzung der explorativen Datenanalyse zu betreiben.

Trotz der Fortschritte der analytischen Chemie, die zusammen mit der Entwicklung leistungsfähiger Computer die Modellbildung zur explorativen Datenanalyse erst ermöglicht, wird in [Kitano, 2002] angemerkt, dass selbst heute noch „viele Durchbrüche bei den experimentellen Apparaturen, fortgeschrittener Software und analytischen Methoden nötig sind.“

Das Metabolic Engineering kann aus heutiger Sicht als ein Zweig der Systembiologie verstanden werden und wird von Stephanopoulos wie folgt definiert: „Unter Metabolic Engineering versteht man die gezielte Verbesserung der Produktivität oder von Zelleigenschaften durch die Modifikation bestimmter biochemischer Reaktionen oder Einführung neuer Reaktionen unter Verwendung der rekombinanten DNA Technologie.“



Abbildung 1.2: Den Metabolismus des *C. glutamicum* kann man durch Genmanipulation so umprogrammieren, dass die Aminosäure Lysin produziert wird. Weltweit liegt die bakterielle L-Lysin-Produktion bei 600 000 t pro Jahr mit einem jährlichen Zuwachs von 7 - 10 %.

Das Metabolic Engineering hat also zum Ziel, eine Zelle, die eine winzige chemische Fabrik darstellt, genetisch so zu verändern, dass sie zur Synthese von chemischen Substanzen nutzbar ist. Grundvoraussetzung dafür ist ein Verständnis der Funktion der Zelle, welches von der Systembiologie erbracht wird.

Die Nutzung bakterieller Produktionsstämme in der Bioverfahrenstechnik bietet bei der Synthese mancher Substanzen erhebliche Vorteile gegenüber der chemischen Synthese. Das prominenteste Beispiel dafür ist die bakterielle L-Lysin-Produktion (Abb. 1.2) mit dem *Corynebacterium glutamicum*, die weltweit bei 600 000 Tonnen pro Jahr liegt mit einem jährlichen Zuwachs von 7 - 10 %. Gelingt es einem Unternehmen seinen Produktions-Stamm zu optimieren, so kann bei gleich bleibendem Rohstoffverbrauch mehr Lysin produziert werden. Bei dieser Produktionsmenge kann eine kleine Steigerung bereits ein Marktvorteil bedeuten.

Die Wildstämme von Zellen sind im Gegensatz dazu ausgerichtet zu wachsen und sich zu vermehren. Insbesondere steuert die Zelle ihren Stoffwechsel in einer Weise, dass die benötigten Stoffe zum richtigen Zeitpunkt in der richtigen Menge produziert werden. Mit den Vorgehensweisen des Metabolic Engineering versucht man nun dieses Gleichgewicht so zu stören, dass sie von einem Stoff mehr produziert als sie eigentlich benötigt. Damit hat man eine winzige Chemiefabrik, die einerseits einen bestimmten Stoff produziert, deren Effizienz bei der Zellteilung und dem Zellwachstum aber

erheblich eingeschränkt ist und somit ihre Überlebensfähigkeit „in freier Wildbahn“ gesunken ist.

1.4. Projektkontext und Zielsetzung

Gefördert wurde diese Arbeit von der DFG (Deutsche Forschungsgesellschaft) innerhalb des Projekts „Modellierung metabolischer Netzwerke auf der Grundlage von In-Vivo-, In-Vitro und In-Silicio-Daten“ im Rahmen des Schwerpunktprogramms SPP 1063 „Informatikmethoden zur Analyse und Interpretation großer genomischer Datenmengen“ .

Diese Arbeit wurde am Lehrstuhl für Simulationstechnik der Universität Siegen durchgeführt und ist in einen größeren Projektkontext eingebunden. Während des Projekts wurde ein intensiver Austausch mit einer großen experimentellen Arbeitsgruppe am IBT (Institut für Biotechnologie) des Forschungszentrum Jülich und der Distributed Systems Group, einer Informatikgruppe an der Universität Marburg, gepflegt.

Die Entwicklung des Softwarewerkzeugs MMT2 baut auf einem Projekt auf [Hurlebaus, 2001], das am IBT des Forschungszentrum Jülich durchgeführt wurde. Dort ging es um die Stoffflussmodellierung, basierend auf der Interaktion zwischen Modellbildung und Experimenten. Das im Rahmen dieses Projekts erstellte Softwarewerkzeug MMT1, wird in Kapitel 5.1 kurz vorgestellt. Aufgrund der guten Erfahrungen und Erfolge mit Hilfe dieser Software wurde ein weiteres Projekt initiiert, in dem dieses Werkzeug weiter entwickelt wird und die verwendeten Algorithmen und Vorgehensweisen auf den Stand der Technik gebracht werden sollten.

Im Rahmen dieses Projekts entstand das Softwarewerkzeug MMT2, das bereits in mehreren experimentellen Arbeiten unterstützend zum Einsatz kommen konnte. In Kapitel 11 werden diese Projekte in Bezug auf MMT2 vorgestellt.

1.5. Struktur dieser Arbeit

Abb. 1.3 soll in diesem Abschnitt dazu verwendet werden, die Themen der einzelnen Kapitel dieser Arbeit in einen Gesamtzusammenhang zu bringen. Dazu werden jedem Kapitel einer oder mehrere Teile der Abb. 1.3 zugeordnet.

Diese Arbeit ist in fünf große Teile eingeteilt, die jeweils mehrere Kapitel zusammenfassen. Im ersten Teil, Grundlagen, wird der Stand der Technik im Bezug auf den Projektkontext, die zugrunde liegenden Experimente (Kap. 2 und Abb. 1.3A) und die

Modellbildung (Kap. 3 und Abb. 1.3B) vorgestellt. Kapitel 4 gibt einen Überblick über die existierenden Softwarewerkzeuge, die ähnliche Bereiche abdecken. Abgeschlossen wird der Grundlagenteil mit Kapitel 5, das die Resultate des Vorgängerprojekts und die Ausgangssituation dieser Arbeit vorstellt.

Im zweiten Teil werden die zugrunde liegenden Methoden und die Implementierung des Softwarewerkzeugs MMT2 behandelt. Eine neue und effektive Vorgehensweise ist die softwareseitige Unterstützung von Modellfamilien, die in Kapitel 6 (Abb. 1.3E) an einem Beispiel vorgestellt wird. Kapitel 8 und 9 beschreiben die wichtigsten Details der Implementation von MMT2 (Abb. 1.3F) und Kapitel 7 den Aufbau des XML-basierten Speicherformats (Abb. 1.3D).

Im dritten Teil dieser Arbeit wird zunächst in Kapitel 10 anhand eines einfachen Beispiels die Benutzung von MMT2 veranschaulicht. Kapitel 11 stellt drei experimentelle Arbeiten vor, in denen das Softwarewerkzeug unterstützend zum Einsatz kam.

Der vierte Teil dieser Arbeit, Modellsuche, befasst sich im Detail mit dem Formalismus der Modellfamilien, der eine der wesentlichen Neuerungen darstellt und den Modellbildungsprozess in seiner Effektivität maßgeblich unterstützen soll. Dazu werden in Kapitel 12 die Modellfamilien, welche bereits in Kapitel 6 anhand eines Beispiels informal vorgestellt wurden, auf einem höheren Abstraktionsniveau formal definiert. Aufbauend auf diesen Formalismus stellt Kapitel 13 einen Optimierungsalgorithmus vor, der die Modellfamilien als Grundlage nutzt. In Kapitel 14 wird ein Anwendungsbeispiel für diesen Algorithmus vorgestellt und diskutiert. Aufgrund des hohen Rechenleistungsbedarfs wurde MMT2 nicht nur auf Einzelplatzrechnern eingesetzt sondern auch auf Workstation-Clustern, Hochleistungsrechnern und in einer Grid-Infrastruktur (Kapitel 15).

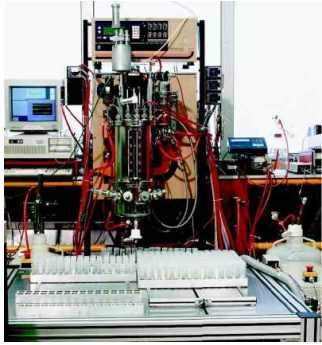
Im letzten Teil der Arbeit werden in Kapitel 16 die wichtigsten Ergebnisse noch einmal zusammengefasst, die Resultate diskutiert und ein Ausblick gegeben.

1.6. Experimentelle Modellbildung

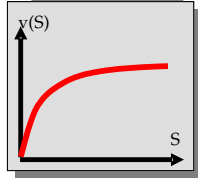
.....

Abb. 1.3 diente im letzten Abschnitt dazu, die Struktur dieser Arbeit daran zu illustrieren, dass jedes Kapitel einer oder mehrerer Teilabbildungen zugeordnet werden kann. In erster Linie stellt Abb. 1.3 aber ein Ablaufschema für eine Art der experimentellen Modellbildung dar, die dieser Arbeit zugrunde liegt. In ähnlicher Form wurde Abb. 1.3 als Poster auf der Tagung *Metabolic Engineering V* vorgestellt [Haunschild et al., 2004].

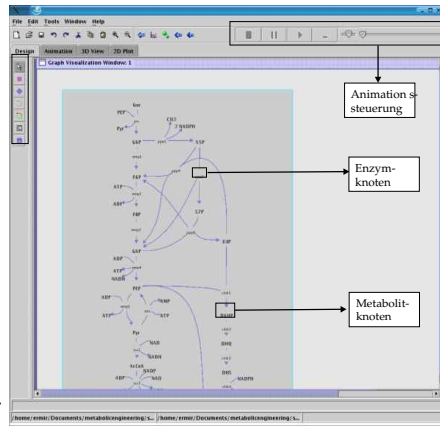
A) Rapid Sampling



B) Biochemisches Wissen



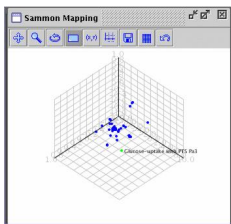
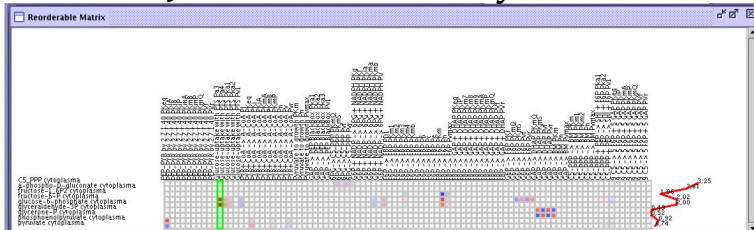
C) Modell-Setup



- Vollgraphischer Editor
- XML-Export des Netzwerks
- Visualisierung der simulierten Systemdynamik

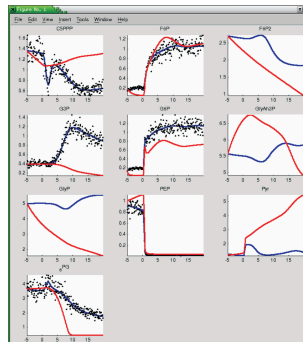


J) Statistische Analyse



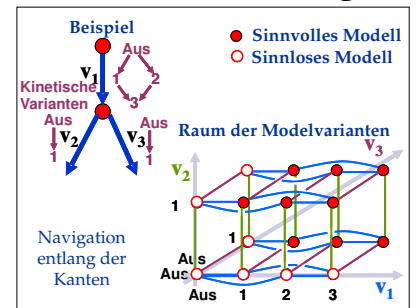
- Visualisierung der Sensitivitätsmatrizen
- Farbvisualisierung von zeitabhängigen Sensitivitäten und Korrelationen
- Multidimensionale Skalierung als alternative Visualisierungstechnik
- Synchronisierte Visualisierungsmethoden

I) Parameteranpassung

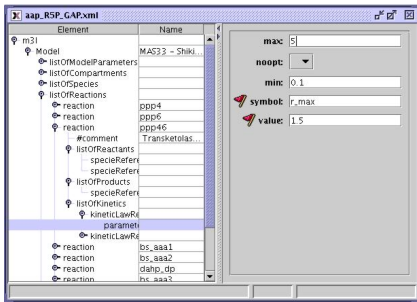


- Plotten der Ergebnisse der Parameteranpassung
- Untersuchung der besten Modelle der Modellfamilie

H) Modell-diskiminierung



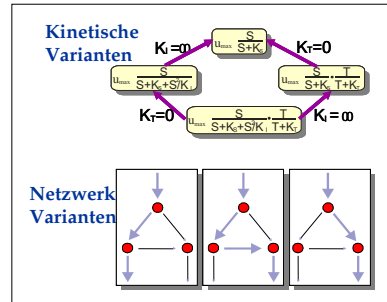
D) Modell im M3L-Format



- XML-basiertes an SBML angelehntes Format zum Speichern metabolischer Netzwerke
- Netzwerkstruktur erweitert durch kinetische Informationen
- Metabolite können durch geglättete Messdaten ersetzt werden
- Modellvarianten ermöglichen mehrere ähnliche Modelle in einer Beschreibung zu halten



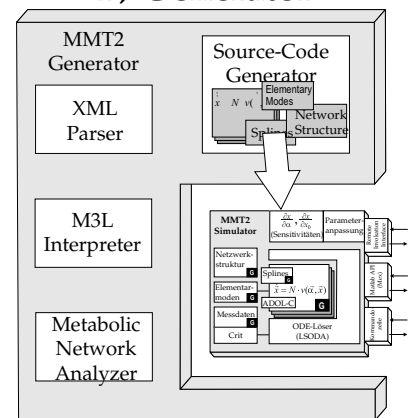
E) Modell Familie



- Modellvarianten können durch Angabe von kinetischen- und Netzwerkvarianten spezifiziert werden
- Spezialisierungs- und Verallgemeinerungsrelationen werden im XML-File gespeichert



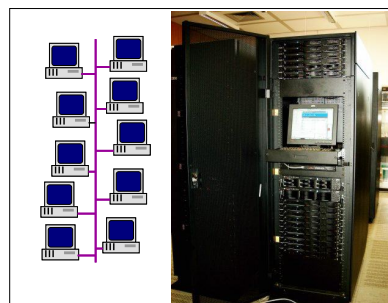
F) Generator



- C Quelltextgenerierung für high performance computing
- Generierter Simulator enthält alle Varianten des Modells und kann durch ein externes Interface (z. B. Matlab) parametrisiert werden
- Automatisches Differenzieren zur Sensitivitätsanalyse



G) Grid Computing



- Hochoptimierte Simulation und Optimierung
- Verteilte Parameteranpassung der einzelnen Modelle auf einem Compute-Cluster
- Einrichtung zum Grid-Computing an der Univ. Siegen mit 256 AMD 64 Opteron Prozessoren

- Modelle haben eine Nachbarschaftsbeziehung
- Automatisches Navigieren im Raum der sinnvollen Modelle
- Modelldiskriminierung ist Ein hochdimensionales diskret- kontinuierliches Optimierungsproblem



Abbildung 1.3: Grundlegende Vorgehensweise bei der experimentellen Modellbildung und auch Leitfaden für die Struktur dieser Arbeit.

Die experimentelle Modellbildung beginnt mit dem Experiment (Abb. 1.3A), dessen Messdaten für die weitere Arbeit der Modellbildung als Grundlage dienen. Zusammen mit Expertenwissen über das biologische System (Abb. 1.3B), dessen quantitative Formulierung meistens aus biologischen Datenbanken stammt, wird entweder ein einfaches Modell erstellt (Abb. 1.3C) oder ein bestehendes Modell verändert (Kreislaufstruktur der Abb. 1.3), mit dem Ziel, das gemessene Verhalten besser darzustellen.

Abb. 1.3C zeigt einen Screenshot des Werkzeugs MetVis (Metabolic Visualizer) [Qeli et al., 2003], das zum Einen ein graphischer Editor zur Erstellung von metabolischen Netzwerken ist, aber auch die von MMT2 berechnete Dynamik animieren kann. Abb. 1.3J zeigt das neue Analysewerkzeug MatVis (Matrix Visualizer) [Qeli et al., 2005], das die Daten, die durch Simulation (Abb. 1.3G) und statistische Analyse (Abb. 1.3I) erzeugt wurden, auswertet und graphisch animiert darstellen kann. Diese beiden Werkzeuge wurden in [Qeli, 2005] entwickelt und unterstützen den Austausch von Daten mit MMT2.

Experimentelle Grundlagen

Dieses Kapitel soll einen Einblick in die Experimente (Abb. 1.3A) geben, die zusammen mit dem biochemischen Wissen (Abb. 1.3B) eine der Grundlagen bilden, auf die die Modellbildung aufbaut.

Die Systembiologie profitierte in den letzten Jahren immer mehr von neuen und präziseren Messtechniken die das Verhalten der Zellen *in-vivo* (=lat. im Leben, an der lebenden Zelle) messen können. Diese Messdaten sind eine wertvolle Grundlage um das intrazelluläre Verhalten zu untersuchen und damit limitierende und steuernde Prozesse zu entdecken. Zu den modernen Forschungsmethoden gehören heutzutage Experimente, die automatisiert durchgeführt werden und dadurch auch große Mengen an hochinformativen Daten produzieren. Im Gegensatz zu früher ist man daher heute viel mehr auf hoch entwickelte Softwarewerkzeuge angewiesen die bei der Datenanalyse und Interpretation helfen.

2.1. In Vitro und In Vivo Experimente

.....

In Kapitel 1.2 wurde die experimentelle Modellbildung mit folgendem Zitat motiviert: „Um die Biologie auf Systemebene zu verstehen, müssen wir die Struktur und Dynamik ... untersuchen, anstatt die Charakteristika isolierter Bestandteile einer Zelle oder eines Organismus.“ [Kitano, 2002]. In Frage gestellt wird das alleinige Untersuchen der „Charakteristika isolierter Bestandteile einer Zelle“ . Generell haben diese *in-vitro* Studien einen großen Beitrag geleistet das Verhalten der Zelle zu verstehen.

Bei diesem Ansatz wird ein bestimmtes Enzym aus der Zelle isoliert und im Reagenzglas mit verschiedenen Stoffen der Zelle in Verbindung gebracht, wodurch die Wechselwirkungen der gereinigten Bestandteile genau charakterisiert werden können. Die Resultate solcher Studien kann man in einem chemischen Reaktionsnetzwerk zusammenfassen, wie in Abb. 2.1 gezeigt. Jede Verbindungslinie steht für eine enzymkatalysierte Reaktion, jeder Punkt steht für einen Metaboliten, für jede Reaktion wird ein anderes Enzym benötigt, von denen in Abb. 2.1 etwa 500 abgebildet sind. Dieses Netzwerk kann aber nur einen kleinen Einblick in die Komplexität des Stoffwechsels geben,

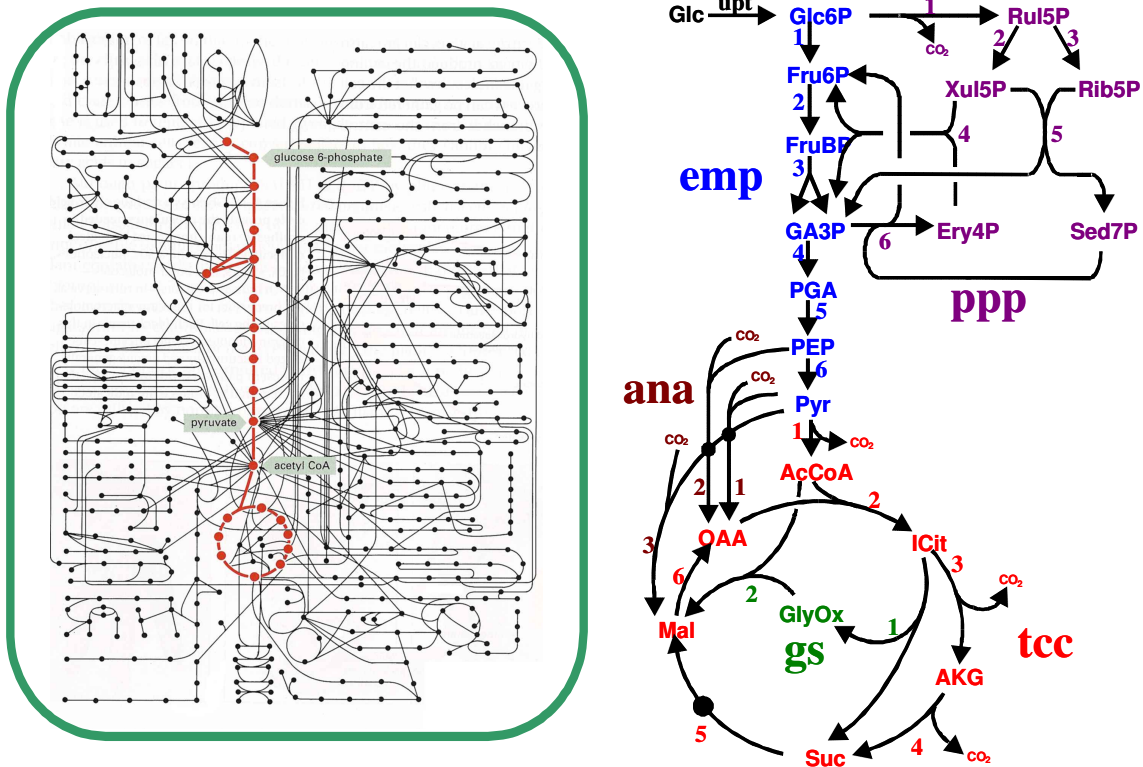


Abbildung 2.1: Links: Reaktionsnetzwerk eines Organismus. Der rote Teil markiert den Zentralstoffwechsel, in dem die höchsten Reaktionsraten und Stoffkonzentrationen auftreten [Alberts et al., 2004]. Rechts: Reaktionsnetzwerk des Zentralstoffwechsels.

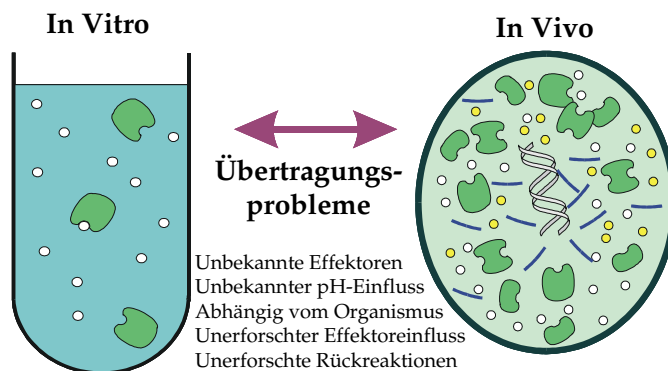


Abbildung 2.2: Durch In Vitro-Vermessung von Enzymen wurde viel über den Stoffwechsel der Zellen herausgefunden. Problematisch ist nur, diese Erkenntnisse auf die lebenden Zellen zu übertragen.

da hier die Co-Metabolite und Effektoren nicht eingezeichnet sind. Rot hervorgehoben sind zwei wichtige Reaktionswege, die Glykolyse und der Zitronensäurezyklus, die Teil des Zentralstoffwechsels sind. Andere Reaktionswege münden entweder in den Zentralstoffwechsel, um beispielsweise benötigte Metabolite zu liefern, oder entspringen von ihnen für die Biosynthese.

Obwohl durch diesen Ansatz viel über die Organisation des Stoffwechsels herausgefunden werden konnte, ist nach wie vor unklar, ob sich die einzelnen Bestandteile im Reagenzglas (*in-vitro*) - gereinigt und isoliert von allen anderen - anders als in der lebenden Zelle verhalten (*in-vivo*, Abb. 2.2). Die wichtigsten Gründe dafür sind:

- Viele Substanzen, die durch die Reinigung entfernt wurden, hatten vielleicht doch eine Einwirkung auf das Enzym.
- In Abb. 2.1 kann man sehen, dass vom Metabolit Pyruvat aus viele Linien abgehen. Das bedeutet, dass hier viele Enzyme um das Pyruvat als Substrat konkurrieren. *In-vitro* waren sie jedoch getrennt voneinander und man beobachtete das Verhalten ohne die Konkurrenz.
- Die Enzymkonzentrationen *in-vivo* sind wesentlich höher als bei *in-vitro* Experimenten.
- Selbst nach Jahrzehnten der Enzymforschung sind noch immer nicht alle Funktionsweisen der Enzyme im Zentralstoffwechsel entdeckt [Wiechert, 2002], was zur Folge hat, dass die Aufzeichnungen im Reaktionsnetzwerk (Abb. 2.1) unvollständig sind.

2.2. *E. coli* und *C. glutamicum* als Modellorganismen

.....

Wie das in 1.3 vorgestellte *C. glutamicum*, ist auch *E. coli* erfolgreich im Einsatz in industriellen Produktionsprozessen und wird beispielsweise zur Herstellung von Insulin, Penicillin Acylase, Tryptophan oder Ethanol verwendet. Wie *C. glutamicum* gehört es zu der Klasse der Prokaryonten, die im Gegensatz zu den Eukaryonten einfacher aufgebaut und kleiner als 1 μm sind.

Unter den *E. coli* Bakterien gibt es pathogene Stämme, die Infektionen in Darm, Blase, Lunge und Nervensystem hervorrufen können [Blattner et al., 1997]. Andere Stämme sind aber als Darmbakterium wichtige Helfer im Körper von Mensch und Tier. *E. coli* ist in der Lage außerhalb seines natürlichen Umfeldes zu überleben, mit einer Vielzahl von chemischen Umgebungsbedingungen fertig zu werden und sich trotzdem rasch zu vermehren. Im Labor ist das Kultivieren von *E.coli*-Stämmen nicht

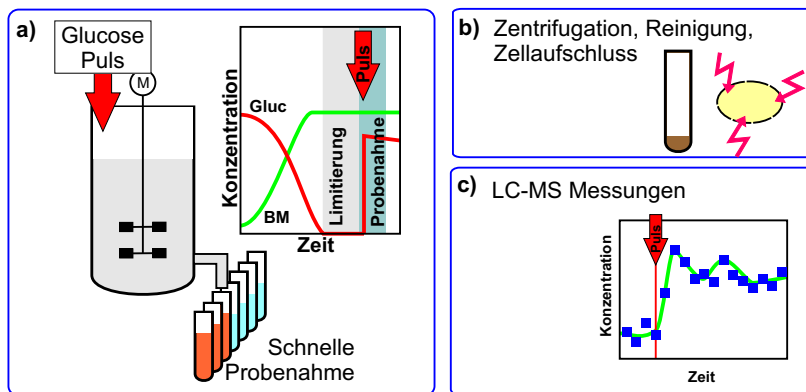


Abbildung 2.3: a) Mit einer Apparatur zur schnellen Probenahme werden während des Experiments Zellproben mit einer Frequenz von etwa 5/s aus dem Bioreaktor entnommen. Nach der Entnahme werden die Zellproben sofort tief gefroren, um die Aktivität des Stoffwechsels zu stoppen [Schaefer et al., 1999]. b) Zur Quantifizierung der Metabolite müssen die Zellproben zunächst aufbereitet werden bevor die c) Stoffkonzentrationen mit einem LC-MS Gerät gemessen werden können.

schwierig, da keine aufwendigen Sicherheitsstandards einzuhalten sind und man einfache Nährmedien zur Kultivierung verwenden kann. Es teilt sich alle 45 Minuten, unter Laborbedingungen bei 37 °C sogar alle 22 Minuten. Somit vermehrt sich eine einzelne Zelle in 20 Stunden zu einer Milliarde (10^9) Zellen.

2.3. Stimulus-Response-Experimente und schnelle Probenahme

.....

In vivo-Messungen von Zellmetaboliten [Gancedo and Gancedo, 1973] wurden erstmalig 1973 mit dem Ziel durchgeführt, die durch *in vitro*-Messungen gewonnenen Erkenntnisse über den Stoffwechsel der Zelle zu validieren. Seit Anfang der 90'er Jahre werden dazu metabolische Modelle auf der Basis von Stimulus-Response-Experimenten (SRE) erstellt und validiert. Den experimentellen Aufbau, bestehend aus dem Bioreaktor und der Apparatur für die schnelle Probenahme, zeigt Abb. 2.4. In Abb. 2.5 ist das LC-MS Massenspektroskop abgebildet, mit dessen Hilfe die Zellmetabolite quantisiert wurden. Experiment und Modell müssen aufeinander abgestimmt sein, um überhaupt Erkenntnisse über den Mikroorganismus erlangen zu können. Der Aufbau solcher Modelle und die Modellannahmen werden im nächsten Kapitel vorgestellt. Das Zeitfenster für das Experiment ist so kurz zu wählen, dass die genetischen

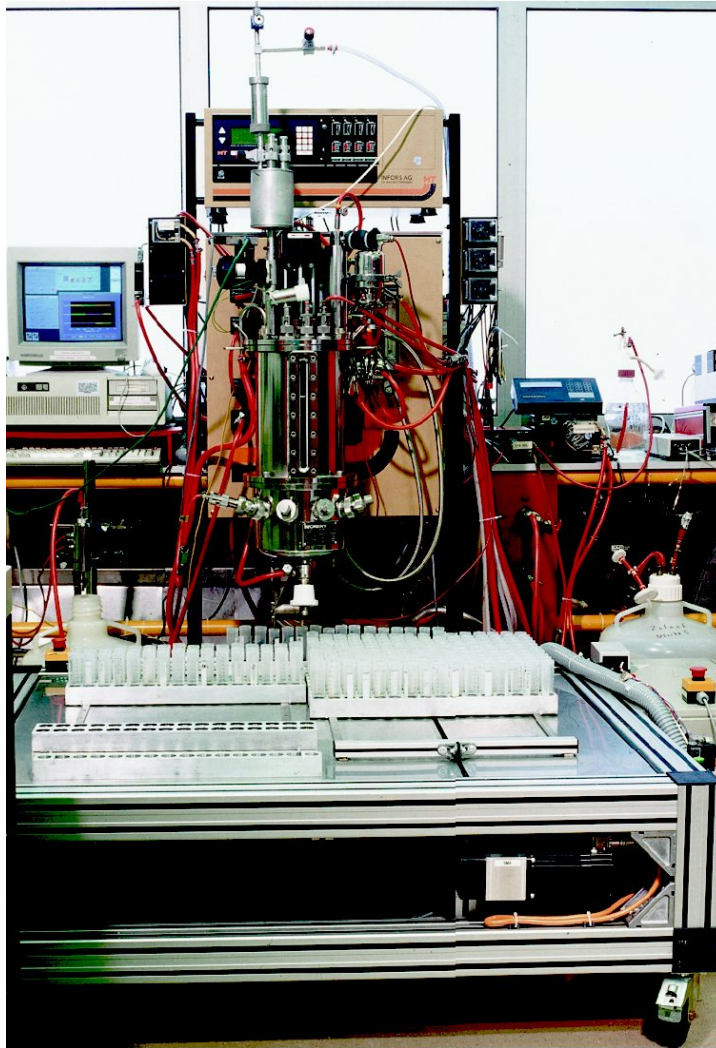


Abbildung 2.4: Der Aufbau des Bioreaktors (oben) am Forschungszentrum Jülich für das Stimulus-Response-Experiment mit der Apparatur zur schnellen Probenahme (unten).



Abbildung 2.5: LC-MS Massenspektroskop der Firma Biosystems, mit dem die Quantifizierung der Metabolite vorgenommen wurde.

Abläufe innerhalb der Zelle sowie die Zellteilung vernachlässigbar sind. Innerhalb dieses kurzen Zeitfensters befindet sich die Durchflussrate vieler Reaktionen in einem quasistationären Zustand. Misst man die Veränderungen der Metabolite innerhalb dieser Zeit, so verändert sich nicht viel und man kann daher auch nicht viele Aussagen darüber machen, wie einzelne Reaktionen ablaufen.

Genau dort setzt das Stimulus-Response-Experiment (SRE) an. Durch einen Substrat-Puls wird der Stoffwechsel der Zelle angeregt und man kann dadurch das dynamische Verhalten des Stoffwechsels überhaupt erst beobachten. Aufgrund der hohen Stoffwechselleistung der Zelle ist ein Beobachtungszeitraum von nur wenigen Sekunden ausreichend, um die Antwort auf den Puls nachvollziehen zu können. Innerhalb dieser Sekunden müssen entsprechend viele Proben genommen werden, um einen Zeitverlauf der Metabolitkonzentrationen zu erhalten.

Stimulus-Response-Experimente sind seit Anfang der 90'er Jahre eine häufig eingesetzte Methode zur Untersuchung des Zellstoffwechsels [de Koning and van Dam, 1992, Rizzi et al., 1997, Lange et al., 2001]. Am Forschungszentrum Jülich wurde eine solche Apparatur für SREs mit schneller Probenahme entwickelt, mit der verschiedene *E. coli* und *C. glutamicum* Stämme untersucht wurden [Schaefer et al., 1999]. Abb. 2.3a zeigt die Fermentationsstrategie in der die Bakterien zunächst im Batch-Betrieb angezogen werden und in der späteren Wachstumsphase ein Feed mit Glukose zugeschaltet wird, um eine höhere Dichte zu erreichen. Nach dem Erreichen der stationären Phase wird dieser Feed auf 10 % gedrosselt, wodurch eine Nährstofflimitierung entsteht. In dieser Phase stellt sich ein neuer Gleichgewichtszustand ein, bei dem die extrazelluläre Glukosekonzentration extrem niedrig ist.

Fünf Sekunden vor dem Substrat-Puls beginnt die Probenahme mit der automatischen Entnahme von Zellproben in Zeitabständen von 220 ms aus dem Bioreaktor. Die Zellproben haben eine Temperatur von 37 °C und werden nach der Entnahme in Reagenzgläser gefüllt, die mit 15 ml einer -50 °C kalten Methanol-Lösung gefüllt sind. Dadurch wird die Stoffwechselaktivität der Zellen für die spätere Analyse gestoppt. Die Reagenzgläser werden unter dem Entnahmeventil herbewegt und nach dem Experiment bei -28 °C gelagert. Insgesamt werden 160 Proben entnommen (16 vor dem Puls und 144 nach dem Puls).

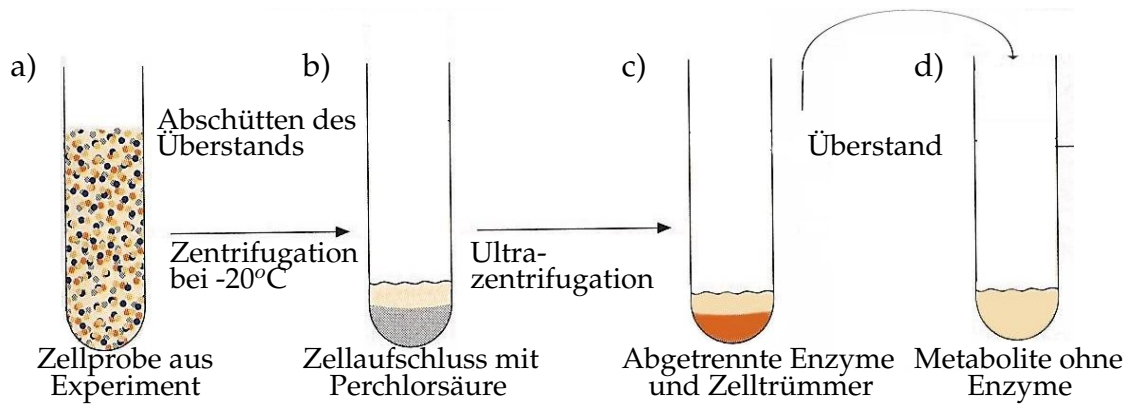


Abbildung 2.6: Um die Metabolitkonzentrationen der Zellen messen zu können, müssen zunächst die Zellen aufgeschlossen und ihre Bestandteile aufbereitet werden.

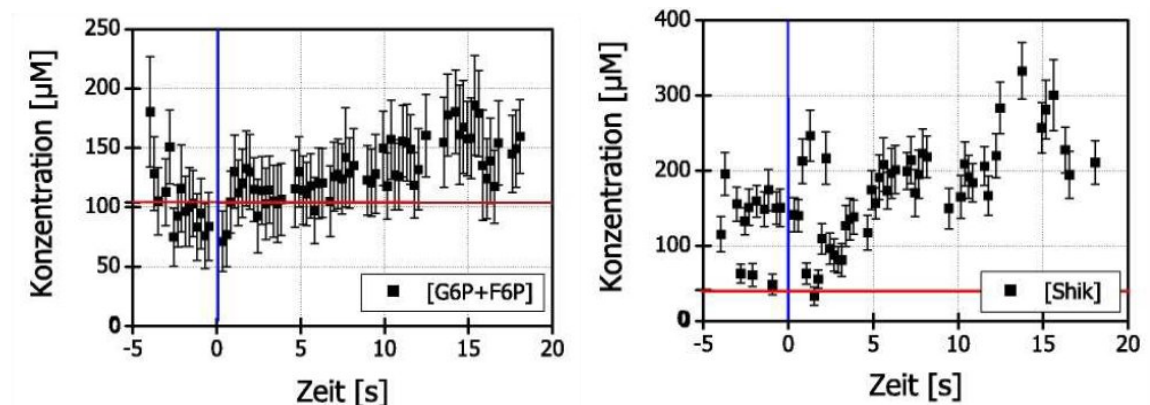


Abbildung 2.7: Messergebnisse für zwei ausgewählte Metabolite aus einem Stimulus-Response-Experiment. Die horizontalen Linien zeigen die Grenze der Messgenauigkeit an, die vertikalen den Zeitpunkt des Substrat-Pulses.

2.4. Zellaufschluss und Analytik

Vorbereitung und Aufbau des Experiments sind zeitintensive Prozeduren die aber noch von den analytischen Methoden zur Metabolit-Quantifizierung und der dafür nötigen Aufbereitung der Zellproben übertroffen werden. Jede der 160 Proben aus dem Experiment muss aufbereitet, die Zellen aufgeschlossen und die Metabolite quantifiziert werden. Die Dauer eines Experiments und Analyse beträgt derzeit ca. 2-3 Monate.

Nach dem Experiment werden die Zellproben im Gefrierschrank zwischen gelagert bis die Quantifizierung der intrazellulären Metabolite vorgenommen wird. Um die Metabolitkonzentrationen der Zellen messen zu können, müssen sie aufgeschlossen und ihre Bestandteile aufbereitet werden. Zunächst werden die Zellen von der Methanol-Lösung durch fünfminütige Zentrifugation bei -20°C und 10000 g ge-

trennt. Dann werden die verbleibenden Zellen durch Perchlorsäure aufgeschlossen (Abb. 2.6b) und ultrazentrifugiert, wodurch zunächst die Zelltrümmer und verbleibende ganze Zellen vom Cytoplasma getrennt werden. Durch die Ultrazentrifugation werden weiterhin die festen von den flüssigen Bestandteilen des Cytoplasmas getrennt, insbesondere Organellen und Enzyme. Durch das Abtrennen der Enzyme ist dann auch kein Stoffwechsel mehr möglich. Der klare Überstand in Abb. 2.6c enthält die Metabolite in der gereinigten Form, wie sie für die weitere Analytik benötigt werden.

Die Quantisierung der aufbereiteten Zellmetabolite wurde mit einem LC-MS Massenspektroskop (Abb. 2.5) vorgenommen. Aufgrund identischer Masse konnten die Metabolite G6P und F6P nicht voneinander getrennt werden, was später in der Simulation berücksichtigt werden muss. Fehlerabschätzungen wurden für alle Messungen durchgeführt [Degenring, 2004] und sind in Abb. 2.7 dargestellt. Abb. 2.7 links zeigt die Messergebnisse der gemeinsamen Konzentration von F6P+G6P, die eine relativ große Messunsicherheit aufweisen und in Abb. 2.7 rechts die Messergebnisse für Shikimat, das mit einer wesentlich geringeren Messunsicherheit quantifiziert werden konnte.

Modellierung metabolischer Netzwerke

In der Übersicht von Abb. 1.3 behandelt dieses Kapitel den Punkt B (bereits vorhandenes biologisches Wissen) und Punkt C (die darauf aufbauende Modellbildung). In diesem Kapitel werden nach einer kurzen Einführung zunächst prinzipielle Vorgehensweisen bei der Modellbildung vorgestellt (Abschnitt 3.1). Dann wird nochmals (vgl. Kapitel 1.2 und Abb. 1.1) die eigentliche Problematik bei der Modellbildung biologischer Systeme aufgegriffen, die durch Messdaten wieder handhabbar wird (Abschnitt 3.2). Abschnitt 3.3 führt die vereinfachenden Modellannahmen auf, auf denen die Bestandteile und Struktur der metabolischen Modelle (Abschnitten 3.4 bis 3.6) dieser Arbeit basieren. Darauf aufbauend werden in Abschnitt 3.7 die Modell-ODEs dargestellt. Schließlich führt Abschnitt 3.8 den Einsatz von Splines ein, durch den Systemgrenzen auf Basis von Messdaten frei gewählt werden können.

3.1. Modellierung komplexer Systeme

.....

Modelle sind vereinfachte Darstellungen eines Systems. Es gibt sie in materieller Form, wie z. B. ein Modellflugzeug, aber auch in abstrakter Form als mathematisches Modell. Letztere sind in allen möglichen Bereichen im Einsatz wie z. B. in Elektrotechnik, Maschinenbau, Wirtschaftswissenschaften oder in der Meteorologie. Mit einem Modell wird ein Ausschnitt eines realen Systems beschrieben, um damit eine bestimmte Fragestellung zu bearbeiten. Diese Ausschnitte bestehen aus Prozessen und ihren Interaktionen, die dann mit Hilfe eines Satzes von ODEs formuliert werden.

Das Modell ist das Herzstück der Simulation und damit ein entscheidender Faktor für die Aussagekraft der Ergebnisse. Genauso wie für andere Anwendungsgebiete gilt auch für die Modellbildung der Leitsatz: „So einfach wie möglich und so komplex wie nötig“ . Es sollte so wenig wie möglich aus dem realen System beschreiben, um nicht unnötigerweise eine hohe Rechenzeit zu benötigen, aber so viel wie nötig, um nicht entscheidende Faktoren der Fragestellung zu vernachlässigen.

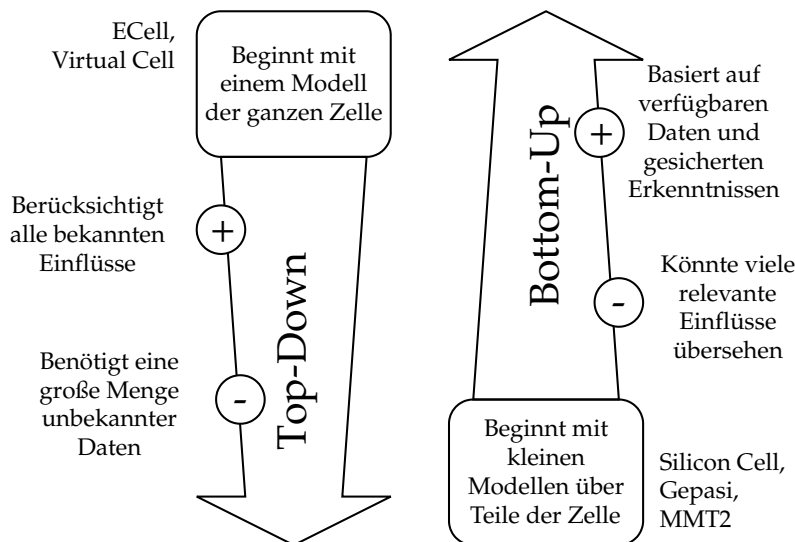


Abbildung 3.1: Zwei Ansätze um die Funktionsweise von biologischen Zellen zu untersuchen. Die Modelle dieser Arbeit bauen auf der Bottom-Up Philosophie auf, die sich streng entlang gesicherter Erkenntnisse orientiert.

Das zu untersuchende System ist in dieser Arbeit ein Stoffwechselabschnitt einer Bakterie. Abb. 2.1 soll einen Eindruck von der Komplexität des realen Systems geben, ist aber bereits eine sehr vereinfachte Darstellung. In [Csete and Doyle, 2002] wird die Komplexität von biologischen Systemen mit der einer Boeing 777 verglichen:

Erst seit kurzer Zeit haben die von Ingenieuren entwickelten Systeme einen Grad von Komplexität erreicht, wie sie auch in der Biologie zu finden ist. Eine startklare Boeing 777 hat 150000 verschiedene Subsysteme, die durch hoch entwickelte Protokolle in komplexe Steuersysteme und -Netzwerke organisiert sind mit ungefähr 1000 Computern, die alle Flugzeugfunktionen automatisiert ansteuern können. Aus Sicht der Kosten und der Komplexität ist die 777 im Wesentlichen ein riesiges Regelungssystem und Computernetzwerk, das in der Lage ist zu fliegen.

Mathematische Modelle können auf verschiedene Art und Weise formuliert werden. Deskriptive Modelle stellen einfach eine abstrakte Funktion dar, die auf die Messdaten passt. Ihre Parameter haben allerdings keinen Bezug mehr zum biologischen Hintergrund. Daher gibt ein deskriptives Modell, das gut auf die Daten passt, keine Erklärung der Daten. Anders ist dies bei mechanistischen Modellen. Sie beschreiben die einzelnen Prozesse und Strukturen des realen Systems und beinhalten direkt die Annahmen, die der Modellierer trifft und die verifiziert werden sollen. Die Parameter des Modells haben einen direkten Bezug zum biologische System und beschreiben Ei-

enschaften der Systemkomponenten. Der Nachteil bei den mechanistischen Modellen ist die Vielzahl der Parameter, die angepasst werden müssen und oftmals redundant sind [Wastney et al., 1999].

Generell findet man für die Erstellung eines Modells für die Zelle (mit der Komplexität einer Boeing 777) zwei Herangehensweisen, die hier Top-Down und Bottom-Up genannt werden (Abb. 3.1). Mit der Top-Down Methode versucht man ein Modell zu erstellen, das möglichst viele Eigenschaften der Zelle berücksichtigt, wodurch ein riesiges Netzwerk entsteht in dem versucht wird alle relevanten Prozesse abzubilden. Nachdem erstmalig ein solches Modell erstellt wurde, wird es durch Modellanalyse und Simulation untersucht. In einem iterativen Prozess wird dann das Reaktionsnetzwerk reorganisiert um sich dem gewünschten Modellverhalten zu nähern. Ein Problem dieses Ansatzes ist, dass nicht genug qualitative und quantitative Daten zur Validierung zur Verfügung stehen.

Die Bottom-Up Methode versucht hingegen sich an gesicherten Erkenntnissen zu orientieren. Basierend auf einem oder mehreren *in-vivo* Experimenten wird ein Modell erstellt, das das Experiment nachbilden soll. Angefangen wird mit einem einfachen Modell mit wenigen Metaboliten und Reaktionen. Auch bei dieser Methode wird in einem iterativen Prozess durch Modellanalyse und Simulation das Modell getestet. Darauf aufbauend werden weitere leicht verbesserte Modelle erstellt.

3.2. Problematik der Modellbildung biologischer Systeme

.....

In diesem Abschnitt soll an einer weiteren Problematik (wie bereits in Kap. 1.2) die Notwendigkeit von *in vivo* Messdaten zur Validierung von metabolischen Modellen gezeigt werden.

Einer der bekanntesten Reaktionsschritte im Zentralstoffwechsel, ist die sog. Phosphofruktokinase-Reaktion (Abb. 3.2). Dieses Enzym wandelt den Metaboliten Fru6P in Fru16P um, wird aber von vielen anderen Stoffen in der Zelle beeinflusst. In der Literatur findet man neben den Co-Faktoren ATP und ADP noch weit mehr als 30 Stoffe, die möglicherweise bei dieser Reaktion eine Rolle spielen. Dies macht die Problematik bei der Modellbildung deutlich. Damit ein Modell nicht zu kompliziert wird, und zu viele unbestimmte Parameter enthält, muss der Modellierer empirisch entscheiden, welche Metabolite einen relevanten und welche einen vernachlässigbaren Einfluss auf das Enzym ausüben.

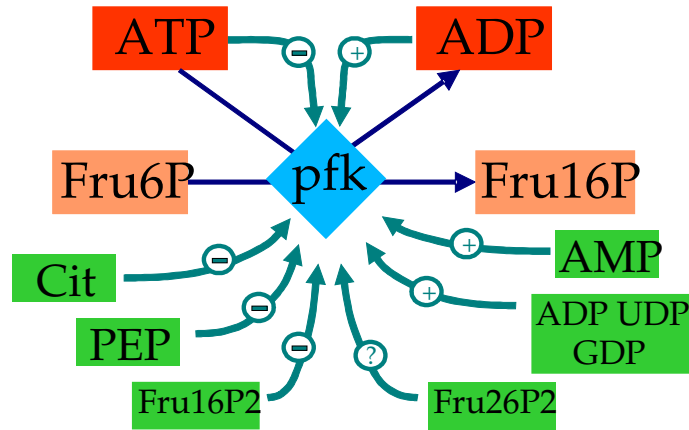


Abbildung 3.2: Das Enzym *pfk* (Phosphofruktokinase) katalysiert eine Reaktion, die Fru6P in Fru16P umwandelt. Bei diesem Katalysator gibt es eine Vielzahl von inhibierenden und aktivierenden Substanzen, aber auch Substanzen, deren Einfluss bisher noch unklar ist.

Abb. 3.2 zeigt eine Vielzahl von Metaboliten, die das Verhalten des Enzyms beeinflussen. In [Hofmann and Kopperschläger, 1982] findet man folgenden reaktionskinetischen Ausdruck, der die Reaktionsgeschwindigkeit der PFK beschreibt. Dabei wird der Einfluss der meisten Metabolite vernachlässigt und nur die Wirkung des Substrats Fru6P berücksichtigt, die inhibierende Wirkung des Co-Faktors ATP, sowie die aktivierende Wirkung von ADP und AMP.

$$v(F6P, ATP, ADP, AMP) = v_{max} \cdot \frac{ATP}{ATP + K_{ATPS} \left(1 + \frac{ADP}{K_{ADPC}}\right)} \cdot \frac{1}{1 + L_O / \left(1 + \frac{F6P}{K_{F6P}}\right)^8} \cdot \frac{F6P}{F6P + K_{F6PS} \cdot \left(1 + \frac{ATP}{K_{ATP1}} + \frac{ADP}{K_{ADP2}} + \frac{AMP}{K_{AMP2}}\right) / \left(1 + \frac{ADP}{K_{ADP1}} + \frac{AMP}{K_{AMP1}}\right)} \quad (3.1)$$

Bei einer relativ geringen Anzahl von Effektoren werden bei dieser mechanistischen Beschreibung also bereits 11 verschiedene kinetische Parameter benötigt (v_{max} , K_{ATPS} , K_{ADPC} , K_{F6P} , K_{F6PS} , K_{ATP1} , K_{ADP2} , K_{AMP2} , K_{ADP1} , K_{AMP1}). Diese Parameter können aus Publikationen oder Datenbanken entnommen werden. Vergleicht man die Parameter aus verschiedenen Datenbanken miteinander, so stellt man fest dass diese stark voneinander abweichen. In [Ewings and Doelle, 1980] wird demonstriert, wie dies zustande kommt. Je nach Wahl

- des genauen Organismus oder Stamm,
- der verwendeten experimentellen Methoden und
- der verwendeten Mess- und Analysemethoden,

wurden bei der Bestimmung der Parameter unterschiedliche Ergebnisse erzielt.

In [Ewings and Doelle, 1980] wurde die Substrataffinität K_{ATPS} untersucht. In Abhängigkeit von pH-Wert sowie An- oder Abwesenheit von anderen Substanzen

nahm dieser Parameter Werte von 0.012 mmol/l bis 0.110 mmol/l an. Bei Parameterwerten für Inhibitoren zeigt sich, dass der Einfluss der Umgebungsbedingungen noch größer ist, und daher noch stärker schwankt.

Ein Parameter, der prinzipiell nicht aus der Literatur entnommen werden kann ist v_{max} , der die Enzymaktivität beschreibt, die in erster Linie von der Menge an Enzymen in der Zelle abhängt. Sie ist bei jedem Organismus anders und ist zusätzlich noch von dem physiologischen Zustand abhängig, in der sich der Organismus befindet.

Zusammenfassend wird nochmals aufgeführt, welche Probleme bei der Erstellung metabolischer Modelle auftreten:

- Regulatorische Effektoren sind unbekannt,
- Datenbanken sind inkonsistent,
- Netzwerkstruktur ist nicht genau bekannt,
- Regulatorische Struktur ist nicht bekannt,
- Enzymkinetiken sind unbekannt,
- Kinetiken sind zu komplex,
- v_{max} fehlt.

Darüber hinaus ist noch weit weniger Wissen vorhanden über beispielsweise

- Membrangebundene Systeme,
- Metabolite, die in sehr geringen Konzentrationen vorliegen,
- Phosphorylierung und
- Instabile Enzyme.

Hieraus wird klar, dass bei der Erstellung metabolischer Modelle an vielen Stellen nur rein empirisch vorgegangen werden kann. Um überhaupt in der Lage zu sein, valide Modelle zu erstellen, sind *in vivo* Messdaten von grundlegender Wichtigkeit.

3.3. Vereinfachende Modellannahmen

.....

Es wird klar, dass es bei einem derart riesigen System, aus dem man eine große Menge an Messdaten erzeugt hat, „hoffnungslos ist, ohne die Hilfe von Software die Daten zu analysieren oder in irgendeiner Weise etwas Sinnvolles aus den Daten zu machen“ [Werner, 2002]. Es ist auch ersichtlich, dass es überhaupt nicht möglich ist, das komplette System in einem einzigen (mechanistischen, Abschnitt 3.1) Modell zu beschreiben und zu simulieren [Gombert and Nielsen, 2000]. Die genaue Untersuchung kann sich daher nur auf einen kleinen Teil des Gesamten konzentrieren. Die Modelle dieser Arbeit beschreiben eine einzelne Zelle, die als Durchschnittszelle im Bioreaktor angesehen wird. Es wird angenommen, dass der Reaktor nur aus solchen Durchschnittszellen besteht, die alle die gleichen Umgebungsbedingungen haben. Insbesondere sollen die Modelle die metabolische Regelung und den Stofffluss beschreiben. Vernachlässigt werden folgende Phänomene:

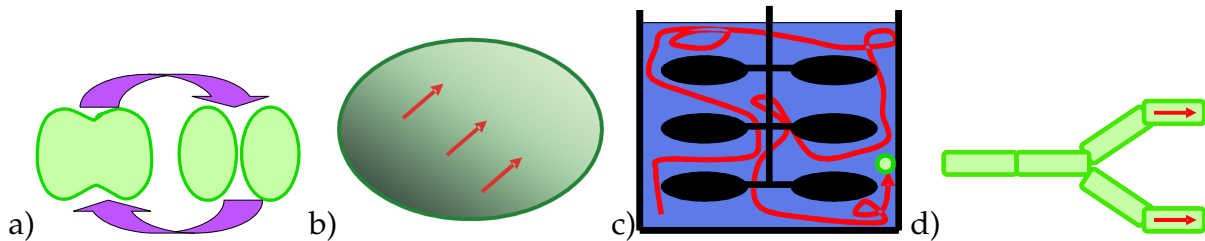


Abbildung 3.3: Vier der Modellvereinfachungen. Vernachlässigt werden a) Zellzyklus und -rhythmus, b) Intrazelluläre Stoffgradienten, c) Bioreaktor Inhomogenitäten, d) Zellmorphologie und -alter.

- **Durchschnittszellen.** Die Zellpopulation, die im Bioreaktor kultiviert wird, besteht aus Zellen, die sich durch Größe, Gewicht, Alter, Zellzyklus, u. a. unterscheiden und damit auch in ihrer Stoffwechselaktivität. Da die Experimente in kleineren Bioreaktoren durchgeführt werden, wird davon ausgegangen, dass diese Variationen nur gering sind.
- **Kein Zellzyklus** (Abb. 3.3a). E. Coli Zellen teilen sich etwa alle 20 Minuten. Während dieser Zeit durchläuft die Zelle verschiedene Phasen, in denen die Stoffwechselaktivität umgestellt wird. Es wird angenommen, dass dieser Effekt im Zentralstoffwechsel, der von den Modellen betrachtet wird, sich vernachlässigbar gering auswirkt.
- **Gleichmäßige Molekülverteilung in der Zelle** (Abb. 3.3b). Es ist bekannt, dass innerhalb der Zelle Stoffgradienten existieren. So findet z. B. die Nahrungsaufnahme durch die Zellwand über Diffusionsprozesse und Transporterreaktionen statt. Je größer die Zelle ist, desto stärker wirkt sich dieser Effekt aus. Die Zellen, die in dieser Arbeit betrachtet werden, sind hinreichend klein, so dass die Metabolite als räumlich gleichmäßig verteilt betrachtet werden können.
- **Keine Gradienten im Bioreaktor** (Abb. 3.3c). Der Stoffwechsel der Zellen ist abhängig von den Umgebungsbedingungen. Im Bioreaktor existieren räumliche Unterschiede in der Temperatur, Nährstoffkonzentrationen, dem Druck und pH-Wert. Wegen der Durchmischung durch den Rührer und des kleinen Volumens wird angenommen, dass auch dieser Effekt vernachlässigbar gering ist.
- **Vernachlässigung Zellmorphologie** (Abb. 3.3d). Während eine Zellpopulation wächst, kommt es vor, dass sich bestimmte Zellen spezialisieren, um besondere Funktionen im Organismus zu übernehmen. Bei den Myzelen der Pilze ist bekannt, dass die noch wachsenden Zellen an den Spitzen einen aktiveren Stoffwechsel haben als die älteren.
- **Keine Genetische Regulation.** Zellen passen sich ihrer Umgebung an, indem sie ihre intrazellulären Enzymkonzentrationen verändern und dadurch die Stoffwechselaktivität anpassen. Dies ist ein langsamer Prozess, der sich erst nach

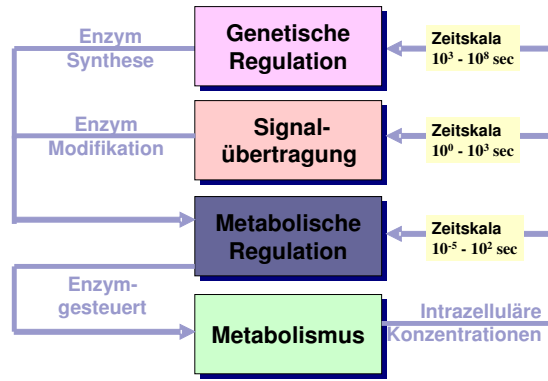


Abbildung 3.4: In der Zelle gibt es verschiedene Regulationsmechanismen, die auf verschiedenen Zeitskalen ablaufen.

ca. 30 Minuten (Abb. 3.4) auswirkt. Auch hier wird angenommen, dass durch die kurze Dauer des Experiments von nur 30 Sekunden, dieser Effekt vernachlässigbar ist.

- **Vernachlässigung von stochastischen Effekten.** Ist ein Metabolit nur in geringen Mengen in der Zelle vorhanden, so kann die Substrat-Enzymbindungszeit nicht mehr vernachlässigt werden, da es sich dann um einen stochastischen Prozess handelt, bei dem die Metabolite, die nur in kleiner Anzahl vorhanden sind, das Verhalten maßgeblich bestimmen. Durch die ausschließliche Betrachtung des Zentralstoffwechsels, in dem nur die höchsten Metabolitkonzentrationen und schnellsten Flussraten der Zelle auftreten, ist keine stochastische Modellierung erforderlich.

3.4. Struktur metabolischer Modelle

.....

Abb. 3.5 zeigt ein metabolisches Netzwerk in Form eines Graphen, der Metabolite (rechteckige Knoten), chemische Reaktionen (Rauten) und Regulationen (gestrichelte Kante) darstellt. Die allgemeine mathematische Beschreibung hat folgende Form [Heinrich and Schuster, 1996]:

$$\dot{\vec{x}} = N \cdot \vec{v}(\vec{\alpha}, \vec{S}, \vec{x}), \quad \vec{x}(0) := \vec{x}_0 \tag{3.2}$$

Hier ist \vec{x} der Vektor der simulierten Metabolitkonzentrationen und \vec{S} der Vektor der Metabolitkonzentrationen außerhalb der Systemgrenzen. N ist die stöchiometrische Matrix (Abschnitt 3.5), in der die Struktur der Reaktionen enthalten ist und \vec{v} ein Funktionsvektor von Reaktionskinetiken (Abschnitt 3.6), der abhängig ist von den Modellparametern $\vec{\alpha}$ und den Systemvariablen \vec{x} .

In Abb. 3.5 sind die Metabolitpools, die in \vec{x} enthalten sind orange hinterlegt. Im Vektor \vec{S} sind die weiß hinterlegten Metabolitpools (*GLC* und *ATP*) enthalten, die sich außerhalb der Systemgrenze befinden und somit nicht simuliert werden. Ihr Einfluss

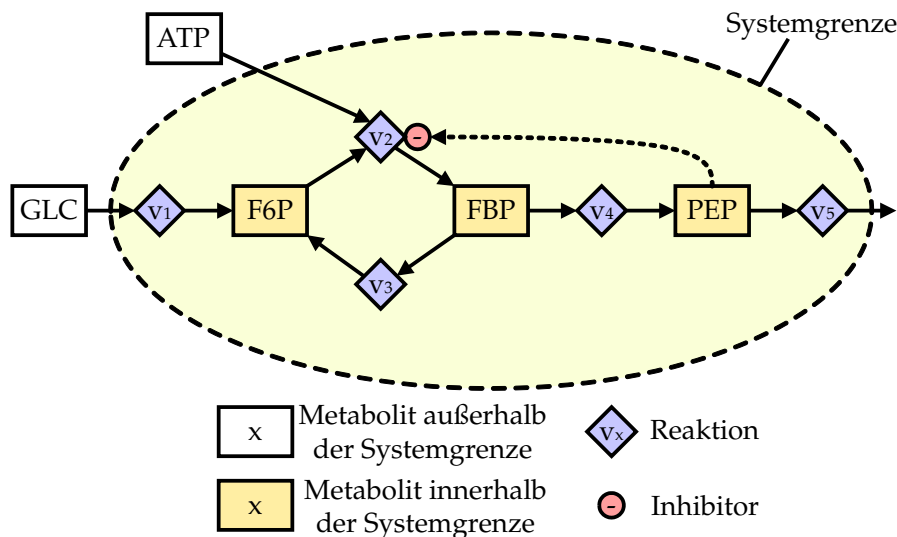


Abbildung 3.5: Strukturelle Darstellung eines metabolischen Netzwerkes, die die kinetischen Formeln, Parameterwerte und Startwerte des Modells nicht beinhaltet.

wird von den Messdaten genommen um die Komplexität des Modells zu reduzieren. Somit beschreibt das Modell nur die Interaktionen von $G6P$, FBP und PEP , ohne den Einfluss von GLC und ATP zu vernachlässigen.

Der regulatorische Einfluss von PEP auf v_2 wird durch eine gestrichelte Kante dargestellt. Je nachdem, ob es sich um einen hemmenden (Inhibitor) oder fördernden (Aktivator) Einfluss handelt, wird die Kante durch ein + oder - Zeichen gekennzeichnet. Die Reaktion v_4 unterscheidet sich von den restlichen dadurch, dass ein FBP in zwei PEP umgewandelt wird. In jedem Metabolit-Knoten herrscht das Gesetz der Massenerhaltung, d. h. die Änderung der Poolgröße ist gleich der Differenz aller Zuflüsse und Abflüsse.

Des Weiteren ist jedem Reaktionsknoten eine Reaktionskinetik zugeordnet. Dies ist eine Funktion, welche die Reaktionsgeschwindigkeit in Abhängigkeit von der Konzentration aller beteiligten Metabolitpools beschreibt. In dieser Reaktionskinetik wird auch der Einfluss von Aktivatoren, Inhibitoren und Kofaktoren festgelegt.

Der Graph bietet einen guten und strukturierten Überblick über das Reaktionsnetzwerk. Nimmt man noch die Reaktionskinetiken hinzu, so kann man automatisch die mathematischen Modellgleichungen aufstellen, was auch in dem Werkzeug realisiert wurde, welches für diese Dissertation erstellt wurde. Um den Stoffwechsel zu simulieren muss das Netzwerk in einer mathematischen Notation vorliegen. In den folgenden Abschnitten wird aus dem Graphen und den zugeordneten Reaktionskinetiken ein Differentialgleichungssystem aufgestellt, welches als Eingabe eines Integrators dient, der die Differentialgleichungen löst und den Verlauf der Metabolitpools in Abhängigkeit von der Zeit ermittelt.

3.5. Stöchiometrische Matrix

.....

Die Struktur des Reaktionsnetzwerks kann auch als Matrix dargestellt werden, die direkt aus dem erweiterten bipartiten Graphen herleitbar ist. Sie beinhaltet aber nur die stöchiometrischen Koeffizienten der chemischen Reaktionen ohne die Effektoren und somit nicht alle Informationen aus dem bipartiten Graphen.

Diese stöchiometrische Matrix N für den im letzten Abschnitt beschriebenen erweiterten bipartiten Graphen lautet:

$$\begin{array}{c}
 F6P \\
 FBP \\
 PEP
 \end{array}
 \begin{array}{ccccc}
 \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 & \mathbf{v}_5 \\
 \left(\begin{array}{ccccc}
 +1 & -1 & +1 & \cdot & \cdot \\
 \cdot & +1 & -1 & -1 & \cdot \\
 \cdot & \cdot & \cdot & +2 & -1
 \end{array} \right) = \mathbf{N}
 \end{array}
 \quad (3.3)$$

Die Zahlen beschreiben die stöchiometrischen Koeffizienten, die in der Einheit *Mol* angegeben werden. In N ist z. B. in der Spalte für \mathbf{v}_4 angegeben, welche Substrate verbraucht werden und welche entstehen. In dieser Reaktion wird 1 mol *FBP* in 2 mol *PEP* umgewandelt. Damit das gesamte metabolische Netzwerkmodell mathematisch formuliert werden kann, müssen Stoffbilanzen formuliert werden und die stöchiometrische Matrix noch mit einem Stoffflussvektor \mathbf{v} multipliziert werden.

Somit lautet das gesamte metabolische Netzwerkmodell in mathematischer Notation (Gleichung 3.2):

$$\begin{array}{c}
 \dot{F6P} \\
 \dot{FBP} \\
 \dot{PEP}
 \end{array}
 =
 \begin{array}{ccccc}
 \left(\begin{array}{ccccc}
 +1 & -1 & +1 & \cdot & \cdot \\
 \cdot & +1 & -1 & -1 & \cdot \\
 \cdot & \cdot & \cdot & +2 & -1
 \end{array} \right) \cdot
 \begin{array}{c}
 \mathbf{v}_1 \\
 \mathbf{v}_2 \\
 \mathbf{v}_3 \\
 \mathbf{v}_4 \\
 \mathbf{v}_5
 \end{array}
 \end{array}
 \quad (3.4)$$

$$\begin{array}{l}
 \dot{F6P} = \mathbf{v}_1 - \mathbf{v}_2 + \mathbf{v}_3 \\
 \dot{FBP} = \mathbf{v}_2 - \mathbf{v}_3 - \mathbf{v}_4 \\
 \dot{PEP} = 2 \cdot \mathbf{v}_4 - \mathbf{v}_5
 \end{array}
 \quad (3.5)$$

Der Stoffflussvektor \mathbf{v} ist allerdings nicht konstant, sondern wiederum abhängig von den reaktionskinetischen Parametern α . Dazu kommen die Konzentrationen der Effektoren die direkt von den Konzentrationen der intrazellulären Metabolitpools \mathbf{X} abhängen. Die extrazellulären Metabolitpools \mathbf{S} haben zwar auch Einfluss auf die Reaktionen, werden aber nicht bilanziert.

$$\mathbf{v} = \mathbf{v}(\alpha, \mathbf{S}, \mathbf{X}) \quad (3.6)$$

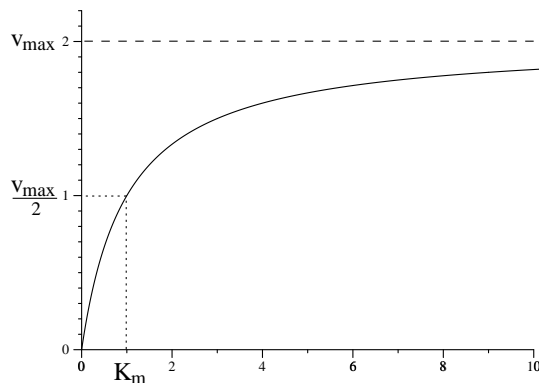


Abbildung 3.6: Reaktionskinetik nach Michaelis-Menten

3.6. Chemische Reaktionskinetiken

Um die Modellabhängigkeiten formal zu spezifizieren, müssen die Eigenschaften von reaktionskinetischen Ausdrücken im Detail verstanden sein. In [Hucka and et. al, 2003] sind über 30 solcher Reaktionskinetiken für verschiedene Anwendungsfälle zusammengestellt. Darüber hinaus gibt es in der Literatur eine große Menge von weiteren kinetischen Beschreibungen, denen verschiedene Ansätze zugrunde liegen, angefangen von Prinzipien über Phänomenologie oder Black-Box Ansätze:

1. In der Theorie der Enzymkinetiken [Cornish-Bowden, 1996, Segel, 1975] gibt es etablierte Ansätze zur Herleitung von reaktionskinetischen Termen, die sich auf einen genau bekannten Ablauf der Reaktion stützen. Die resultierenden Formeln sind immer rationale, multivariate Polynome in den Konzentrationsvariablen und können eine große Anzahl von Parametern haben. Eines der einfacheren Beispiele ist die Standard (nicht-reversible) Michaelis-Menten Kinetik (Abb. 3.6), die außer vom Substrat von keinem anderen Metaboliten beeinflusst wird:

$$v^{MM} = v_f \cdot \frac{S}{S + k_S} \quad (3.7)$$

Diese Reaktionsrate v^{MM} ist nur von der Größe des Metabolitpools S abhängig, der maximalen Reaktionsrate v_f und der Substrataffinität k_S . Mit dieser Gleichung wird beschrieben, dass in der enzymkinetischen Reaktion ein Substrat an ein Enzym gebunden wird, das keine weitere Bindungsstelle besitzt. v_f ist die maximal mögliche Reaktionsrate, da nur eine begrenzte Anzahl von Enzymen zur Verfügung steht.

Ist auch ein Fluss in die Gegenrichtung möglich, dann beschreibt die reversible Michaelis-Menten Kinetik diesen bidirektionalen Reaktionsschritt. Zusätzlich zum Substrat wird sie jetzt auch vom Produkt P beeinflusst.

$$v^{MMR} = \frac{v_f \frac{S}{k_S} - v_r \frac{P}{k_P}}{1 + \frac{S}{k_S} + \frac{P}{k_P}} \quad (3.8)$$

Gilt die Randbedingung $k_P = \infty$, so ist die Gleichung 3.8 eine Verallgemeinerung der Standard Michaelis-Menten Kinetik (Gleichung 3.7). Im Allgemeinen lassen sich reaktionskinetische Formeln dadurch vereinfachen, dass bestimmte Parameter auf Null oder Unendlich gesetzt werden.

2. Um die Theorie der Enzymkinetik rigoros anzuwenden, müssen alle Interaktionen zwischen Enzymen und beteiligten Metaboliten bekannt sein, was üblicherweise nicht der Fall ist. In dieser Situation bedienen sich Modellierer häufig eines phänomenologischen Ansatzes [Petersen et al., 2003], in dem Enzymkinetiken mit weiteren Termen erweitert werden. Das Resultat wird hier als Pseudokinetik bezeichnet. Als Beispiel könnte man die Standard Michaelis-Menten Kinetik (Gleichung 3.7) durch einen multiplikativen Term erweitern, der einen inhibierenden Einfluss eines Metaboliten I ausdrückt:

$$v^{MMI} = v_f \cdot \frac{S}{S + k_S} \cdot \frac{k_I}{k_I + I} \tag{3.9}$$

Hier wird implizit angenommen, dass der Metabolit I unabhängig von S auf dem Enzym agiert. Diese Vereinfachung ist streng genommen selten korrekt, aber es kann in einer Anwendung angemessen sein, die Anzahl der Parameter, die anhand von Messdaten identifiziert werden sollen, klein zu halten.

3. Verschiedene Black-Box Ansätze existieren in der Literatur, welche die Netzwerkstruktur beibehalten, aber die Reaktionskinetiken grober beschreiben. Ihre Struktur ist sehr ähnlich zu den verschiedenen Arten von Polynomen. Einige bekannte Formalismen sind
 - Allgemeine Potenzgesetzformalismen [Voit and Savageau, 1982]
 - Lin-Log Kinetiken [Visser and Heijnen, 2003]
 - Fuzzy-Regelsysteme [Yen and Lee, 1996]

Im Allgemeinen macht es wenig Sinn Enzymkinetiken mit diesen Black-Box Ansätzen zu vermischen. Trotzdem gibt es bei den Black-Box Ansätzen auch die Verallgemeinerungs- und Vereinfachungsstruktur durch Setzen von Parametern auf Null oder Unendlich.

3.7. Beispiel

.....

Als Beispiel für das in Abb. 3.5 gegebene Reaktionssystem nehmen wir als Eingabe einen externen Metabolitpool GLC an. Für v_1 , v_3 und v_5 gelte die Michaelis-Menten Kinetik. Für die Reaktion v_4 nehmen wir die umkehrbare Michaelis-Menten Kinetik an, welche mit v_f und v_r zwei „ v_{max} “ Werte besitzt, einen für die Vorwärtsrichtung und einen für die Rückwärtsrichtung.

Die Gleichung 3.2 stellt die allgemeine Form des Differentialgleichungssystems für das Systemverhalten dar. Setzt man noch die Werte $v_1 \dots v_5$ aus 3.10 in die Gleichung 3.4 ein, so erhält man das Differentialgleichungssystem 3.11 für das Systemverhalten.

- Reaktionskinetiken:

$$\begin{aligned}
 v_1 &= v_{max,v_1} \cdot \frac{GLC}{K_{m,v_1} + GLC} \\
 v_2 &= v_{max,v_2} \cdot \frac{F6P}{K_{m,v_2} + F6P} \\
 v_3 &= v_{max,v_3} \cdot \frac{FBP}{K_{m,v_3} + FBP} \\
 v_4 &= \frac{v_{f,v_4} \cdot \frac{FBP}{K_{ms,v_4}} - v_{r,v_4} \cdot \frac{PEP}{K_{mp,v_4}}}{1 + \frac{FBP}{K_{ms,v_4}} + \frac{PEP}{K_{mp,v_4}}} \\
 v_5 &= v_{max,v_5} \cdot \frac{PEP}{K_{m,v_5} + PEP}
 \end{aligned} \tag{3.10}$$

- Komplettes Differentialgleichungssystem:

$$\begin{aligned}
 \dot{F6P} &= v_{max,v_1} \cdot \frac{GLC}{K_{m,v_1} + GLC} \\
 &\quad - v_{max,v_2} \cdot \frac{F6P}{K_{m,v_2} + F6P} \\
 &\quad + v_{max,v_3} \cdot \frac{FBP}{K_{m,v_3} + FBP} \\
 \dot{FBP} &= v_{max,v_2} \cdot \frac{F6P}{K_{m,v_2} + F6P} \\
 &\quad - v_{max,v_3} \cdot \frac{FBP}{K_{m,v_3} + FBP} \\
 &\quad - \frac{v_{f,v_4} \cdot \frac{FBP}{K_{ms,v_4}} - v_{r,v_4} \cdot \frac{PEP}{K_{mp,v_4}}}{1 + \frac{FBP}{K_{ms,v_4}} + \frac{PEP}{K_{mp,v_4}}} \\
 \dot{PEP} &= 2 \cdot \frac{v_{f,v_4} \cdot \frac{FBP}{K_{ms,v_4}} - v_{r,v_4} \cdot \frac{PEP}{K_{mp,v_4}}}{1 + \frac{FBP}{K_{ms,v_4}} + \frac{PEP}{K_{mp,v_4}}} \\
 &\quad - v_{max,v_5} \cdot \frac{PEP}{K_{m,v_5} + PEP}
 \end{aligned} \tag{3.11}$$

3.8. Wahl der Systemgrenzen

.....

Wie bereits am Beispiel von Abb 3.5 gezeigt, kann eine Systemgrenze definiert werden, indem der Einfluss der Metabolite, die außerhalb der Systemgrenzen liegen durch einen geglätteten Spline von den Messdaten in das Modell eingebracht werden.

Modellbildung beginnt typischerweise mit einem einfachen Modell, welches dann in einem iterativen Prozess verfeinert und erweitert wird. Ist das Modell zu einfach, so ist es evtl. überhaupt nicht in der Lage die Dynamik der experimentellen Daten zu reproduzieren. Anstatt alle experimentellen Daten für die Parameteranpassung zu verwenden, können sie auch als Input für das Modell dienen, um die Komplexität zu reduzieren. Beispielsweise könnte der komplexe *ATP-ADP* Mechanismus von den Daten genommen werden und im Modell durch künstliche Metabolite modelliert werden,

deren Konzentration durch einen Spline festgelegt ist. Somit kann sich der Modellierer auf einfachere Mechanismen konzentrieren und das Verhalten und die Beeinflussung durch die Umgebung von den Daten nehmen.

Verfügbare Software

Modellbildung und Simulation von biochemischen Netzwerken sind heute wichtige Aspekte in den Disziplinen *Systembiologie* [Wolkenhauer, 2001] und *Metabolic Engineering* [Stephanopoulos et al., 1998]. Sie sind auf das ganzheitliche Verständnis zellulärer Systeme aus einer theoretischen und angewandten Sicht ausgerichtet, mit dem Hauptaugenmerk auf der quantitativen Analyse der regulatorischen Netzwerke. Durch die Untersuchung des dynamischen Verhaltens biochemischer Netzwerke erhält man zusätzliche Informationen zur Struktur der zellulären Systeme, die man allein aus dem Genom, Transkriptom und Proteom nicht gewinnen kann.

Modellbildung und Simulation helfen sowohl dabei die Struktur des hierarchisch organisierten zellulären Regulationsnetzwerkes zu erforschen, als auch die komplexen Wechselwirkungen zwischen den zellulären Komponenten. Das Design von entsprechenden Simulationswerkzeugen hängt jedoch stark von den Forschungszielen und den verfügbaren Daten ab. Daher sind bis heute eine wachsende Anzahl verschiedener Werkzeuge für die Modellierung und Simulation biochemischer Netzwerke in Entwicklung von denen die meisten aus dem akademischen Bereich stammen. Einige der wenigen kommerziellen Werkzeuge sind DigitalCell/VisualCell für die Modellierung von regulatorischen Netzwerken (Gene Network Sciences, <http://www.gnsbiotech.com>), PathwayPrism für zelluläre Signalgebung (Physiome, <http://www.physiome.com>) oder PhysioLab (Entelos, <http://www.entelos.com>).

Die Softwarewerkzeuge sind sehr stark durch die Forschungsarbeit geprägt aus denen sie ihren Ursprung haben. Es gibt einige wenige Werkzeuge, die einen hohen Bekanntheitsgrad besitzen und einen sehr hohen Anteil an Werkzeugen die einen sehr eingeschränkten und speziellen Verwendungszweck haben. Die Webseite <http://www.sbml.org> listet derzeit über 80 Softwarewerkzeuge, die die gemeinsame Basis SBML als Datenformat teilen. Einen allgemeinen Überblick über Simulationsmethoden im Rahmen des Metabolic Engineering gibt [Wiechert, 2002]; in [Pettinen et al., 2005] werden eine Reihe von Software-Werkzeugen vorgestellt mit einer Auflistung der unterstützten Simulationsmethoden. [Arkin, 2001] beschränkt sich auf Software-Werkzeuge, die die Modellierung experimenteller Daten unterstützen. Allgemein kann man diese Werkzeuge grob in drei Kategorien einteilen (s. a. [Wiechert, 2002]), die in den folgenden Abschnitten vorgestellt werden.

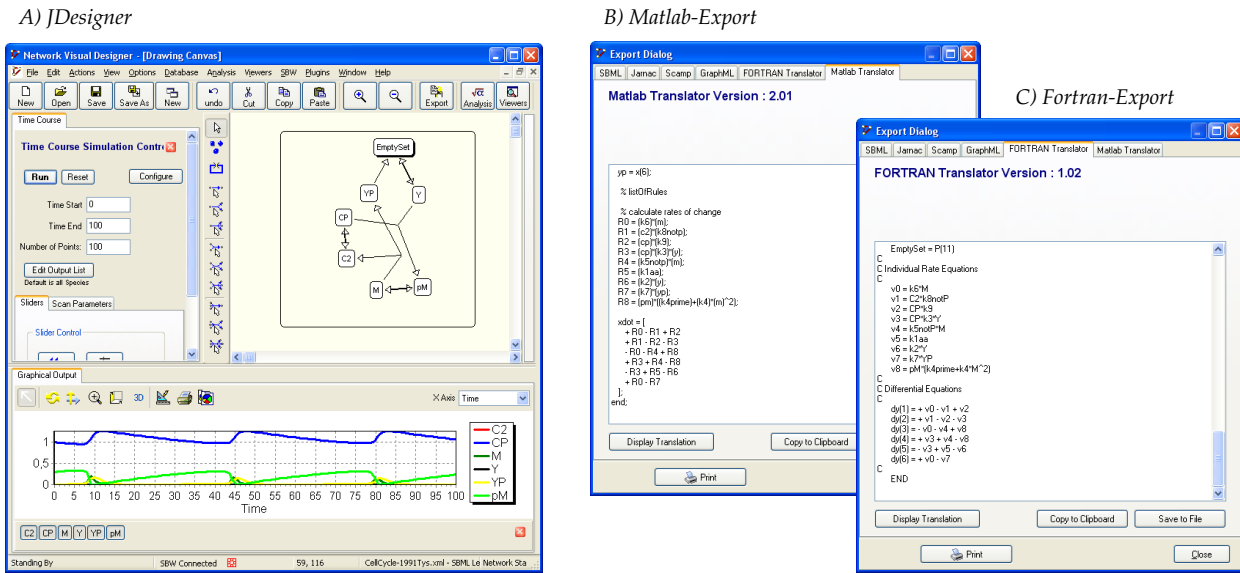


Abbildung 4.1: Das Softwarepaket Jarnac/JDesigner kombiniert einen vollgraphischen Editor (JDesigner) für metabolische Netzwerke mit einer skriptgesteuerten Simulationssoftware (Jarnac).

4.1. Simulation ganzer Zellen

In diesem Abschnitt werden ein paar Werkzeuge erwähnt, die auf das ganzheitliche Verständnis der zellulären Regulation ausgerichtet sind. Sie konzentrieren sich auf die Unterstützung der konzeptionellen Modellierung und den hierarchischen Zusammenbau von Netzwerken [Takahashi et al., 2003, Tränkle et al., 2000]. Typischerweise basieren sie auf bereits in Datenbanken vorhandenen reaktionskinetischen Daten und werden vorzugsweise dafür genutzt die globalen Eigenschaften zellulärer Netzwerke grob zu erkunden.

Das E-Cell Simulation Environment (E-Cell SE, <http://www.e-cell.org>, [Tomita et al., 2001]) und das Virtual Cell Project (V-Cell, <http://www.nrcam.uchc.edu>, [Loew and Schaff, 2001]) gehören in dieser Gruppe zu den Werkzeugen, die meisten genutzt werden. Sie unterstützen die Simulation einer einzelnen Zelle mit Kompartimenten, Chromosomen, Membranen und dem Cytoplasma bis hin zu den Genen.

Für die Modellerstellung ist eine GUI vorhanden mit der die Metabolite und Stoffflüsse in Form eines Graphen modelliert werden können und auch genetische Mechanismen wie Genaktivierung und Geninhibierung. Bei den Reaktionen wird die Eingabe kinetischer Daten unterstützt. Die Simulation basiert auf einem Differentialgleichungssystem.

4.2. Simulation (bio-)chemischer Reaktionsnetzwerke

.....

Einige Werkzeuge wurden entwickelt um die Anwendung von neu entwickelten Konzepten zu erleichtern, die aus der Metabolic-Control-Theorie [Heinrich and Schuster, 1996], der biochemischen Systemtheorie [Voit and Savageau, 1982] oder der Metabolischen Netzwerkanalyse [Mendes, 1997, Schilling et al., 1999, Sauro, 1993] stammen. Für diese Arten von Analysen muss die Netzwerkstruktur und ihre kinetischen Parameter bekannt sein. Einige dieser Werkzeuge sind nicht nur in der Lage den stationären Zustand zu simulieren, sondern auch das dynamische Verhalten.

Jarnac [Sauro, 2000] ist eine Skriptsprache in der man biochemische Netzwerke beschreiben und simulieren kann. Jarnac verwendet zum Lösen der Differentialgleichungen wahlweise die frei erhältlichen Softwarebibliotheken CVODE [Cohen and Hindmarsh, 1996] oder LSODA [Hindmarsh, 1983]. Während Jarnac relativ komplex aufgebaut ist und daher eine intensive Einarbeit erforderlich ist, gibt es ein graphisches Frontend namens JDesigner (Abb. 4.1), über das man auf eine Untermenge der Funktionalität von Jarnac zugreifen kann. Jarnac und JDesigner sind zwei separate Werkzeuge, die über einen Applikations-Broker namens Systems-Biology-Workbench (SBW) [Hucka et al., 2001] miteinander verbunden werden. JDesigner [Sauro, 2001] stellt eine vollgraphische Entwicklungsumgebung für biochemische Netzwerke zur Verfügung, bei der der Benutzer durch Drag-and-Drop Technik Metabolite und Reaktionen definieren kann und diese mit weiteren Attributen wie z.B. einer kinetischen Reaktionsbeschreibung erweitern kann.

4.3. Experimentelle Modellbildung

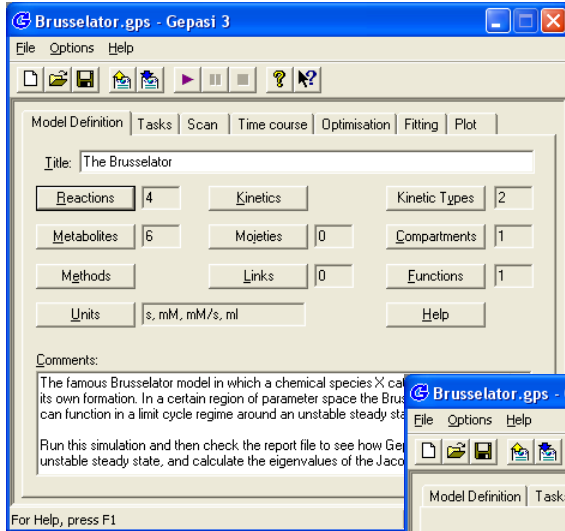
.....

Überraschenderweise gibt es nur wenige Werkzeuge, die eine auf experimentellen In-Vivo Daten basierende Bottom-Up Modellentwicklung unterstützen. Diese auf Messdaten basierenden Werkzeuge sind besonders wichtig für die Auswertung von schnellen Probenahmeexperimenten im Metabolic Engineering aber auch für transiente Beobachtungen in Gen-Regulations-Prozessen.

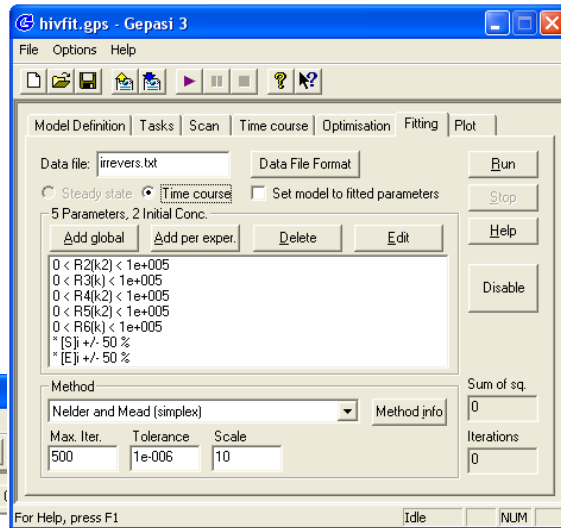
Solche Werkzeuge sind denen in Abschnitt 4.2 beschriebenen hinsichtlich ihrer Grundfunktionalität sehr ähnlich, jedoch bieten sie darüber hinaus weitere Funktionalität, was meistens ihre Bedienung erheblich komplizierter macht. Das Anpassen der Parameter des Modells an experimentelle Messdaten gehört zu den Mindestanforderungen solcher Werkzeuge.

Gepasi [Mendes, 1997] (Abb. 4.2) ist ein solches Werkzeug, bei dem man die Modellparameter an die Daten anpassen kann. Seit 2002 wird Gepasi nicht mehr weiter entwickelt, sondern in einem Nachfolgeprojekt Namens Copasi <http://www.copasi.org> repimentiert und erweitert, in dem auf den Erfahrungen, die im Einsatz mit Gepasi gemacht wurden, aufgesetzt wird.

A) Modelldefinition



C) Parameteranpassung



B) Simulation

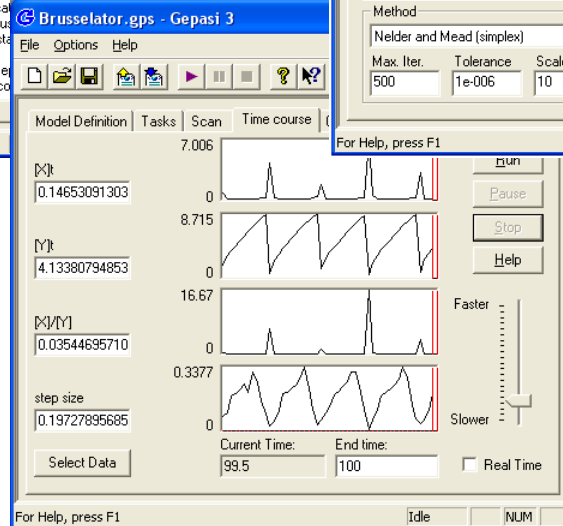


Abbildung 4.2: Gepasi ist eine Simulationssoftware für metabolische Modelle und unterstützt das Anpassen der Modellparameter an Daten unter Nebenbedingungen.

Ein weiteres Softwaretool Namens SiliconCell wurde von der Silicon Cell Initiative Amsterdam entwickelt <http://www.bio.vu.nl/hwconf/Silicon/>, die eine ihrer Schwerpunkte in der Krebsforschung hat.

4.4. Spezialanwendungen

.....

Übersteigen die Anforderungen die Funktionalität der Werkzeuge und sind keine angemessenen Möglichkeiten zur Erweiterung vorhanden, so gibt es den Ausweg über Mehrzweck-Werkzeuge wie Matlab oder Modelica die Funktionalität selber zu programmieren. Für diesen Weg gibt es Werkzeuge, die Teilaufgaben übernehmen, wie z. B. das Übersetzen eines biochemischen Netzwerkes in Differentialgleichungen (Abb. 4.1B+C). Eine solche Lösung hat den Nachteil, dass viel Arbeitszeit in die Programmierung investiert werden muss, die dann bei der eigentlichen Aufgabe, der Modellbildung, nicht zur Verfügung steht.

Präzisierung der Zielsetzung dieser Arbeit

In den Kapiteln 1 bis 3 wurde das Anwendungsgebiet für die hier entwickelten Methoden aus dem Bereich des Metabolic Engineering vorgestellt. In Kapitel 4 wurden verfügbare Softwaretools und ihr jeweiliger Anwendungshintergrund vorgestellt.

Ein Werkzeug, um nach dem Experiment die gemessenen Daten auszuwerten und damit auch zu verstehen, ist die experimentelle Modellbildung. Obwohl diese Vorgehensweise bei der Bottom-Up Modellierung standardmäßig angewendet wird, ist es erstaunlich, dass es wenig softwareseitige Unterstützung dafür gibt (Kap. 4). Mangels dieser Unterstützung wurde in [Hurlebaus, 2001] eine Software entwickelt, die diesen Modellbildungsprozess unterstützen soll.

Zunächst wird diese Forschungsarbeit kurz vorgestellt und erläutert wie in der vorliegenden Arbeit auf den Erfahrungen in [Hurlebaus, 2001] aufbaut wird. Weiterhin wird die Zielsetzung dieser Arbeit präzisiert und die softwaretechnische Vorgehensweise in dieser Arbeit vorgestellt.

5.1. MMT1

.....

Das Ziel der Arbeit von [Hurlebaus, 2001] war es, metabolische Modelle zu erstellen, die in der Lage sind, das Stoffwechselnetzwerk von Mikroorganismen qualitativ und quantitativ zu beschreiben. Durch die Modellierung sollen weiterhin die Kontrollmechanismen in Stoffwechselwegen identifiziert werden. Die Arbeit war Teil eines Gesamtprojektes, das aus Experimenten, Modellentwicklung und -anpassung, Modelldiskriminierung und experimental Design bestand. Grundlage des Projekts waren Daten aus Stimulus-Response-Experimenten mit *E. coli* und *C. glutamicum*.

Wie in den Kapiteln 2 und 3 beschrieben, wurden Modelle erstellt und an die Messdaten angepasst. Mangels softwareseitiger Unterstützung wurde im Rahmen einer Promotion [Hurlebaus, 2001] ein erstes Softwaretool namens MMT1 (Metabolic Modeling Tool) entwickelt um die Arbeiten bei der Modellerstellung, Simulation und Parameteranpassung zu unterstützen.

5.1.1 Aufbau des Softwaretools MMT1

Abb. 5.1 zeigt die Architektur von MMT1. Im Rahmen der vorliegenden Arbeit wurde diese Architektur grundlegend überarbeitet und erweitert. MMT1 enthält eine GUI, die in der Skriptsprache Tcl/Tk implementiert wurde. Alle Daten für die Simulation

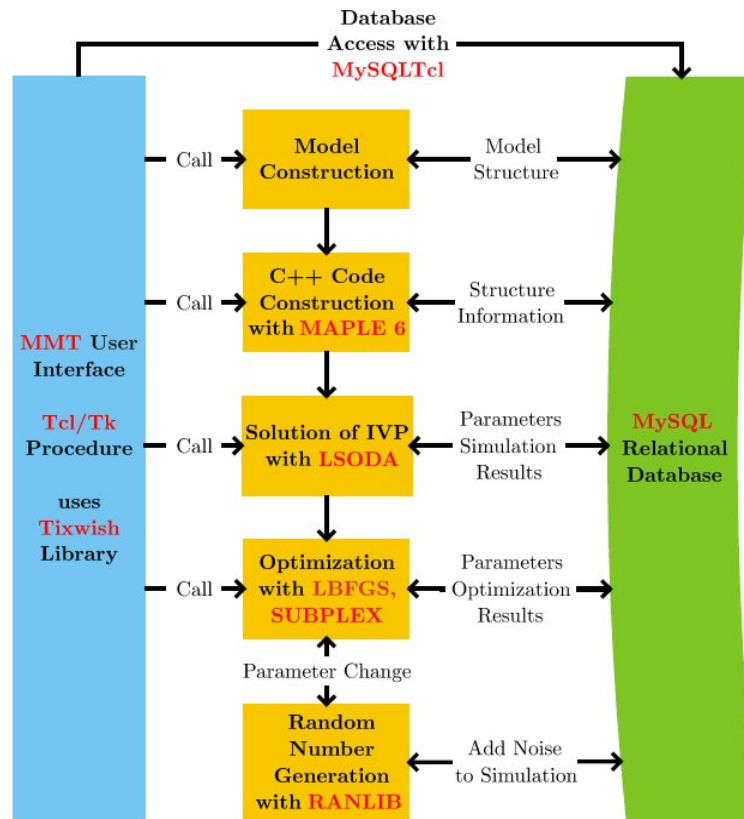


Abbildung 5.1: Architektur der MMT1 Software. Mit roter Schrift wird dargestellt an welchen Stellen Softwarebibliotheken oder externe Software eingesetzt wird. Die vertikalen Pfeile stellen den Informationsfluss, bzw. Funktionsaufrufe dar. In horizontaler Richtung findet der Ablauf der Modellerstellung, Source-Code Generierung, Simulation, Optimierung und Parametervariation statt.

und Optimierung werden in einem RDBS (MySQL) gespeichert. Experimentelle Daten können mittels einer Referenz, die auf eine Datei im Dateisystem zeigt, eingebunden werden.

Von der GUI aus kann man halbautomatisch den Prozess, der im mittleren Teil von Abb. 5.1 abgebildet ist, durchführen. Im ersten Block wird zunächst das Modell aus der Datenbank eingelesen. Im zweiten Block werden mit Hilfe der computeralgebraischen Software Maple die Modell-ODEs aufgestellt und in eine C++ Quelltextdatei geschrieben. Diese wird dann zusammen mit weiteren Routinen und Bibliotheken, die in C++, C und Fortran vorliegen, kompiliert, woraus der Simulator entsteht. Die Vorgehensweise von MMT2, die in Abb. 1.3F gezeigt wird, ist eine Abwandlung davon.

5.1.2 Erfahrungen

In der Arbeit [Hurlbaas, 2001] wurde demonstriert, dass die Software MMT1 im Projektkontext eine Lücke füllt, die andere Softwaretools nicht abdecken können. Theoretisch könnte durch den Einsatz von Mehrzwecktools wie Matlab oder Simulink die Modellbildung durchgeführt werden. Der überaus höhere Implementierungsaufwand

macht das Erstellen vieler Modelle aber praktisch unmöglich. MMT1 bietet im Gegensatz dazu eine Alternative für

- die Modellbildung chemischer Reaktionsnetzwerke, die Messdaten in das Modell mit einbeziehen;
- die Simulation dieser Modelle und
- das Anpassen der Modellparameter an die Messdaten.

Eine Erfahrung in der Softwareentwicklung ist, dass nach einer langen Implementierungsphase, in der sich die Anforderungen immer wieder geändert haben, eine Überarbeitung der Implementierung unumgänglich ist. Einige der Nachteile, die sich deutlich bei MMT1 gezeigt haben, sind die Folgenden.

- Durch das Auswählen und Testen verschiedener Softwarebibliotheken für die einzelnen Teile von MMT1 und durch das Entwickeln grundsätzlicher Vorgehensweisen, die bei der Arbeit von MMT1 eingehalten werden müssen, ist der Aufbau von MMT1 unübersichtlich geworden. Das wirkt sich auch negativ auf den Aufwand einer Neuinstallation von MMT1 aus, zu der nur sehr versierte Nutzer in der Lage sind.
- Durch den Einsatz proprietärer Software als Bestandteil des Kerns von MMT1 ist es problematisch die Software zu veröffentlichen. MMT1 ist nicht funktionsfähig, wenn keine Maple-Installation vorhanden ist.
- Wie auch die Implementierung, so war die Datenbankarchitektur gewachsen und bedurfte einer Überarbeitung. Weiterhin hat sich das Speichern der Modelle in einer Datenbank als unpraktisch erwiesen, wenn es darum geht Modelle zu archivieren. Der allgemeine Trend der Softwaretools im Bioinformatikbereich (Kap. 4) geht dahin, Modelle in einem XML-Format zu speichern.

5.2. Fortsetzung des Projekts

.....

Nach Ende der Promotion [Hurlebaus, 2001] und damit auch Ende der Entwicklung an MMT1 war klar, dass solch eine Software einen sehr hohen Wert für die experimentelle Modellbildung darstellt, welche wiederum für das Verständnis experimenteller Daten essentiell ist. Daher wurde bei der DFG (Deutsche Forschungs-Gesellschaft) im Rahmen des Schwerpunktprogramms SPP 1063 ein Projekt beantragt, um die Arbeit an MMT1 fortzusetzen. Im Rahmen dieses Projekts werden neben der vorliegenden Arbeit noch zwei weitere finanziert, die miteinander kooperieren und sich auf verschiedene Aspekte der Modellbildung konzentrieren.

Dies ist zum Einen die Arbeit [Qeli, 2005], die auf die Visualisierung von metabolischen Netzwerken und deren Simulationen einen Hauptschwerpunkt legt. Diese Arbeit ist mit der vorliegenden insofern verbunden, als die dort entwickelten Werkzeuge Bestandteil des Kreislaufs der experimentellen Modellbildung (Abb. 1.3) sind. Zum

Anderen ist dies die Arbeit [Wahl, 2005] (Kap. 11.2), in der, u. A. , das Softwaretool dieser Arbeit und die neu implementierten Features eingesetzt und getestet werden.

Die Anforderungen an die Software MMT2 ergeben sich aus den Erfahrungen, die mit MMT1 gemacht worden sind, sowie aus neuen Ideen für Vorgehensweisen. Die Aktivitäten während der Laufzeit dieses Projekts kann man in folgenden Punkten zusammenfassen:

- XML-Sprache zur Spezifizierung von Modellen und Varianten dieser Modelle (Kap. 7),
- Spline Unterstützung (mit Sprungstellen) zur Eingabe von geglätteten Messdaten (Kap. 7.4) für die Simulation,
- Unterstützung von Messdaten für simulierte Metabolite mit Angaben über ihre Genauigkeit (Kap. 7.3, Abb. 2.7),
- Automatische Code-Generierung ohne den Einsatz proprietärer Software (Kap 8),
- Automatisches Differenzieren der Modellgleichungen zur statistischen Analyse (Kap. 9.4),
- Modellvarianten (Kap. 6 und 12),
- Gridcomputing (Kap. 15) für Parameteranpassung (Kap. 9.3) und Modelldiskriminierung (Kap. 9.3.5),
- Optional nutzbare Schnittstelle zu Matlab (Kap. 10) zum Postprocessing des simulierten Modells.

Insbesondere sollte keine GUI mehr entwickelt werden, wie es bei MMT1 der Fall war, in der die Funktionalität der Software zugänglich gemacht wird. Vielmehr sollten verfügbare Werkzeuge genutzt werden, wie z. B. ein XML-Editor, mit deren Hilfe ohne zusätzlichen Implementierungsaufwand die Funktionalität dem Benutzer in einer angemessenen Form bereit gestellt wird. Hauptgrund dafür war, dass mit der Implementierung einer GUI eine erhebliche Menge an Ressourcen in Form von Arbeitskraft investiert werden müsste, die der eigentlichen Forschungsarbeit nicht zugute kommt.

5.3. Rahmenbedingungen und Umsetzung

.....

In diesem Abschnitt sollen die Rahmenbedingungen der Softwareentwicklung im experimentellen Umfeld aufgezeigt werden. Da die Entwicklungsarbeit an MMT2 im Rahmen dieser Promotion nicht unabhängig von den Anwendern geschehen sollte, wurde die Entwicklung nach einem evolutionären Entwicklungsmodell durchgeführt, dessen Grundprinzip in Abb. 5.2 dargestellt ist. Die Software stand den Anwendern bereits während der Laufzeit dieser Arbeit zur Verfügung und sie konnten ein Feedback zu den neu implementierten Teilen in MMT2, bzw. Anregungen zu neuen Features geben.

Im Rahmen einer Diplomarbeit [Haunschild, 2001] wurde zunächst der Aufbau und die Algorithmen von MMT1 untersucht, neue Algorithmen getestet und ein Prototyp

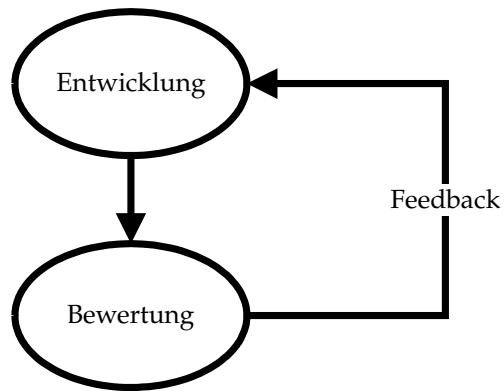


Abbildung 5.2: Das Grundprinzip der evolutionären Software-Entwicklung ist die Erfahrung, dass die Umsetzung der Anforderungen nicht auf einmal erfolgen kann und erst beim Entwicklungsergebnis zu erkennen ist, welche Anforderungen weitere Arbeiten benötigen.

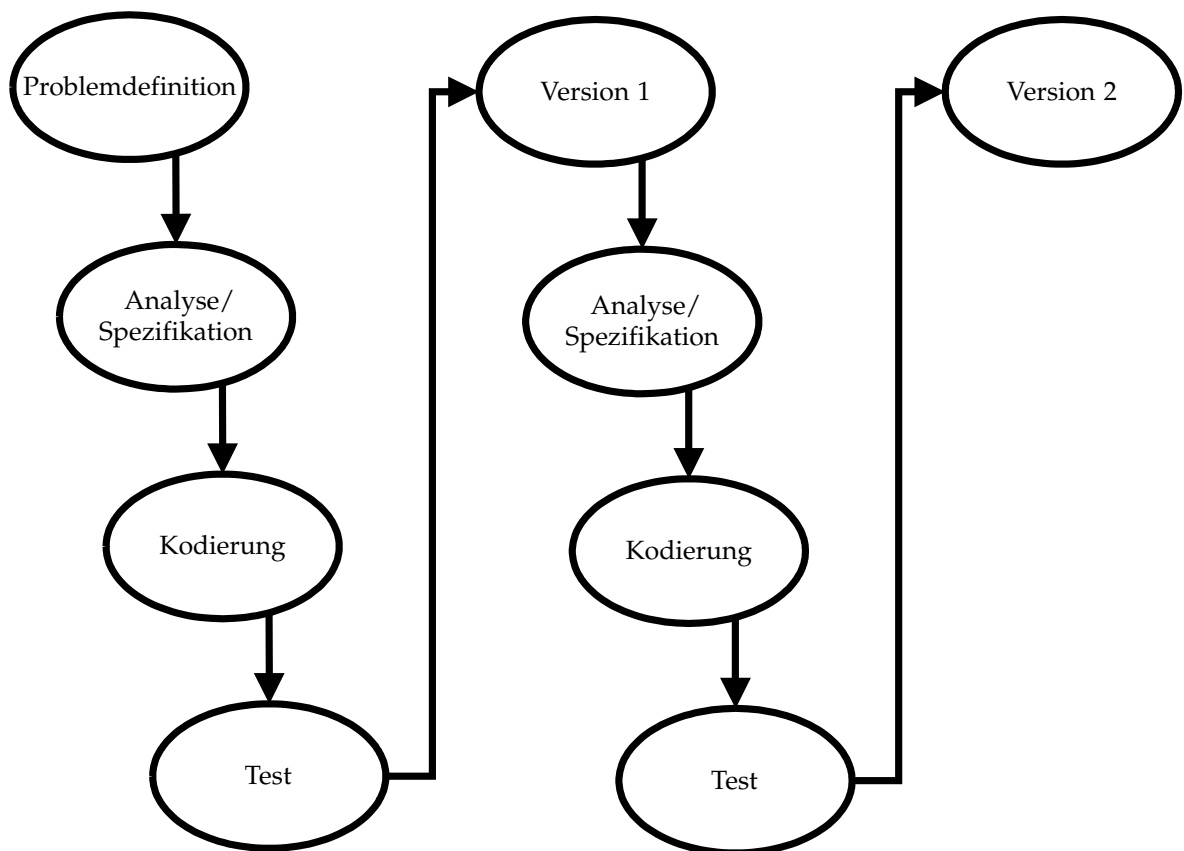


Abbildung 5.3: Zu Anfang der Entwicklung von MMT2 entsprach die Vorgehensweise der inkrementellen Software-Entwicklung.

implementiert. Zu Anfang dieser Arbeit wurde auf den Erfahrungen aufgebaut, die mit MMT1 und [Haunschild, 2001] gemacht wurden. Die softwaretechnische Vorgehensweise bei der Entwicklung von MMT2 kann man in der frühen Phase mit der inkrementellen Software-Entwicklung (Abb. 5.3) beschreiben. Die Laufzeit der Diplomarbeit könnte man in Abb. 5.3 mit dem ersten Durchlauf von „Problemdefinition“ zu „Version 1“ beschreiben, in der ein Prototyp erstellt wurde.

Der Stand der Entwicklung von MMT2 nach dem ersten Durchlauf (vgl. Abb. 5.3 Version 1) ist in der Diplomarbeit beschrieben. Im Rahmen dieser Arbeit konnte die Problemdefinition dann wesentlich weiter gefasst werden und entspricht dem Umfang, der in Abb. 1.3 dargestellt wird. Zu Anfang wurden die wichtigsten Features von MMT1 reimplementiert. Parallel dazu wurde und ein Konverter geschrieben, der die Modelle im MMT1 Datenbankformat in das MMT2 XML-Format übersetzt. Den erreichten Stand könnte man in Abb. 5.3 mit der Version 2 markieren.

Ab dieser Version konnten die Simulationsergebnisse von MMT1 und MMT2 direkt miteinander verglichen werden und auch die Anwender konnten die Unterschiede in der Vorgehensweise von MMT1 und MMT2 bewerten. Ab diesem Punkt kann die eingeschlagene Entwicklungsstrategie nicht mehr mit der inkrementellen Softwareentwicklung beschrieben werden, sondern eher mit einem Modell, dass dem Whirlpool-Modell [Dumke, 2003] ähnelt (Abb. 5.4).

Die zugrunde liegende Vision des Ablaufs einer softwaregestützten experimentellen Modellbildung, wie sie in Abb.1.3 gezeigt wird, lag von Anfang an dem Projekt zugrunde. Während der Entwicklung und engen Zusammenarbeit mit den Anwendern wurden die Konzepte im Detail ausgearbeitet. Das beinhaltet viele Durchläufe in Abb. 5.4, in der immer wieder Schwächen in der Vorgehensweise oder der Implementierung entdeckt und beseitigt wurden; neue Ideen zu Details in der Implementierung eingearbeitet wurden; sowie Verfeinerungen im Ablauf der Modellbildung realisiert wurden.

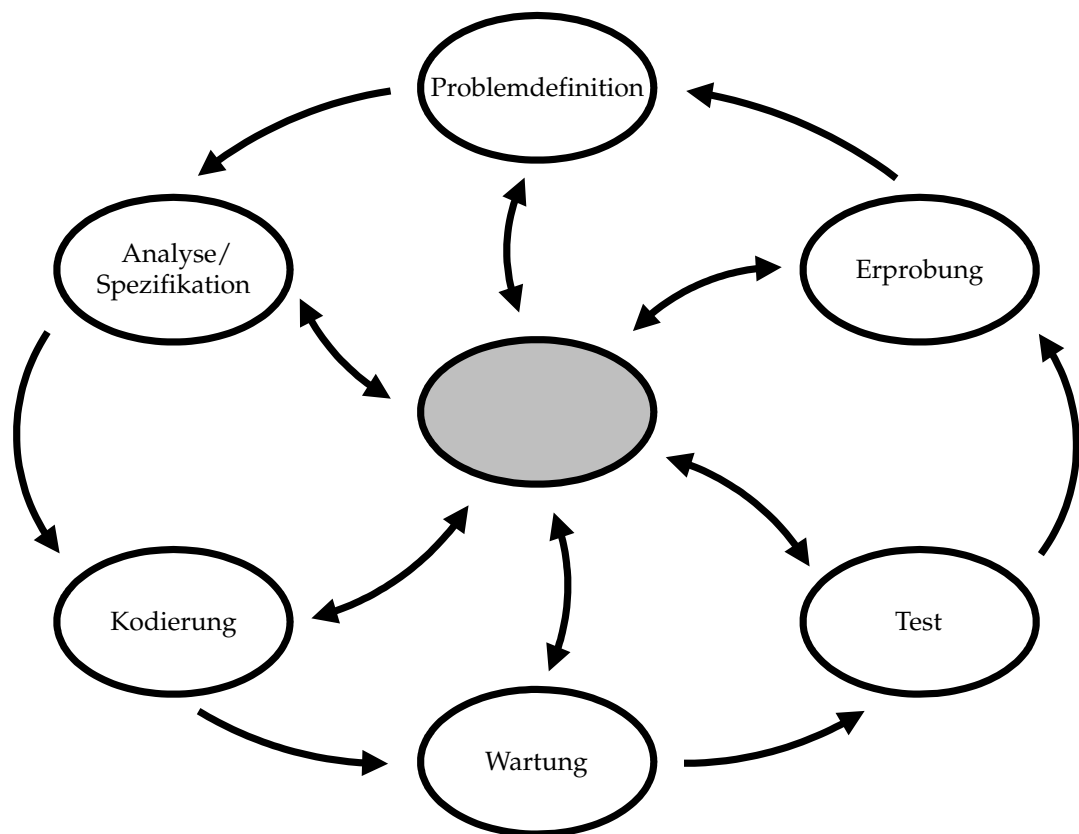


Abbildung 5.4: Als die Implementierung den definierten Stand erreicht hatte, wurde mit dem Whirlpool-Modell der Software-Entwicklung Verfeinerungen in MMT2 eingebaut.

Teil II

Methoden und Implementierung

Modellfamilien als Modellierungswerkzeug

In diesem Kapitel wird das neue Konzept der Modellfamilien informal an einem Beispiel vorgestellt. In Kapitel 12 wird dieses Konzept auf einem höheren Abstraktionsniveau beschrieben. Teile des Inhalts dieses Kapitels wurden in [Haunschild et al., 2005a] publiziert.

6.1. Modellbasierte Datenauswertung

Besonders bei biologischen Systemen ist ein wichtiger Aspekt, dass die Modellbildung ein iterativer Prozess ist, der in einer Abfolge von Experimenten, Simulationsläufen, modellbasierten Datenauswertungen und Experimentplanung besteht (Abb. 1.3) [Wiechert, 2002]. In diesem experimentellen Zyklus wird das Modell schrittweise verbessert bis schließlich eine zufrieden stellende Nachbildung der gemessenen

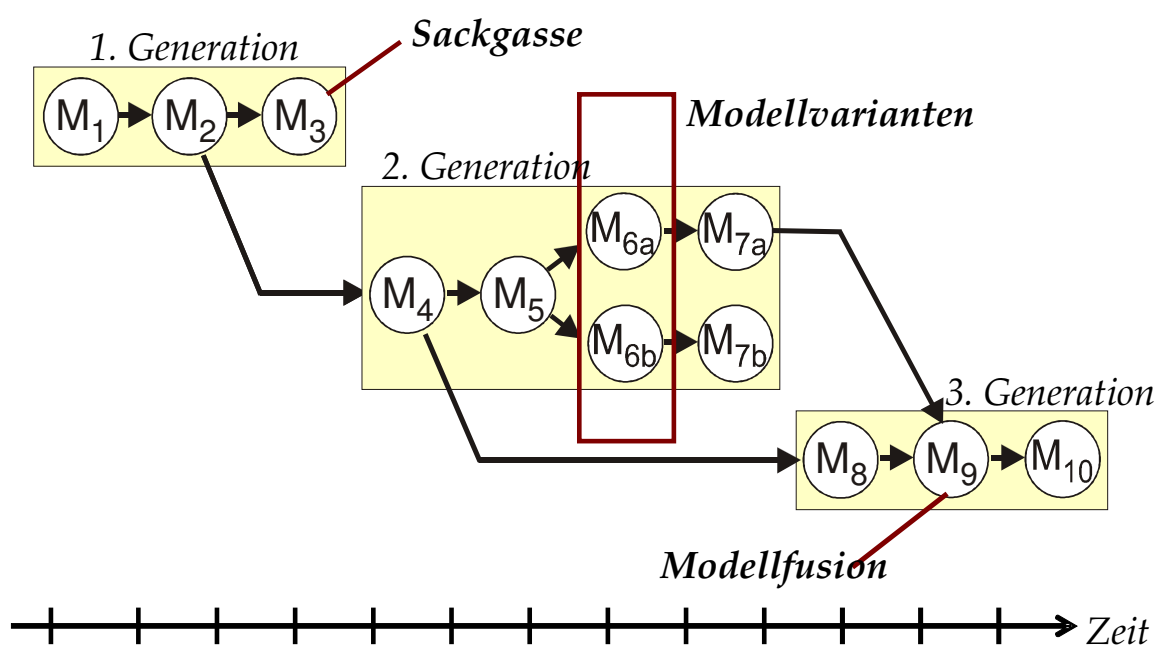


Abbildung 6.1: Während der Modellbildung entstehen viele Modelle, die voneinander abgeleitet sind.

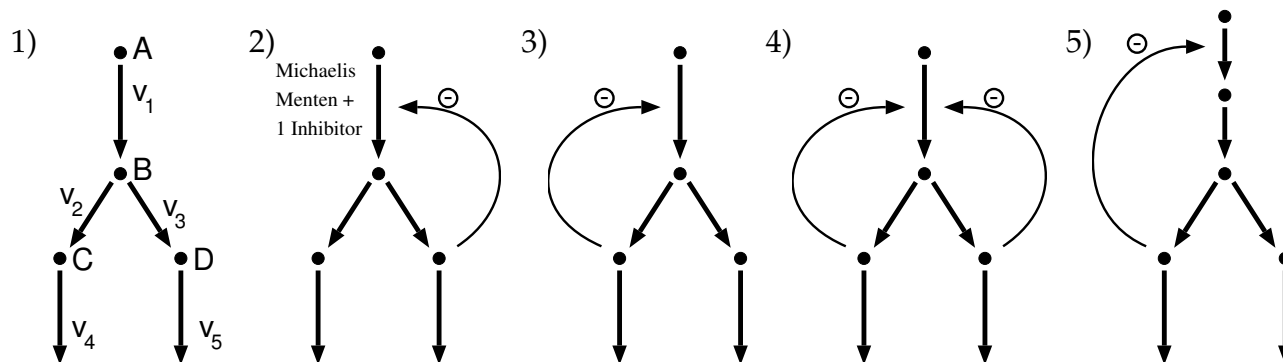


Abbildung 6.2: Eine Sequenz von Modellen aus einem Modellierungsprozess. Die Netzwerke 1-4 haben dieselbe Topologie und unterscheiden sich nur in den kinetischen Beschreibungen der Reaktionen. Netzwerk 1 ist ein initialer Modellansatz, dem in den Netzwerken 2-4 verschiedene Inhibitor-Wechselwirkungen hinzugefügt wurden. Netzwerk 5 zeigt eine Änderung in der Topologie durch einen neuen Reaktionsschritt. Der Weg von Netzwerk 1-4 zu Netzwerk 5 ist jedoch kein elementarer Schritt, wie Abb. 6.5 zeigt.

Daten erreicht wird. Dies ist aber keine lineare Abfolge (Abb. 6.1), sondern ein Prozess mit vielen Sackgassen, neuen Modellansätzen, Erprobung von Modellvarianten und Modellfusionen. Dies führt auf das Konzept der Modellfamilie. Literaturbeispiele für Modellfamilien finden sich in [Takors et al., 1997].

Beispiele aus der Literatur sind [Takors et al., 2001, Takors et al., 1997], in denen die Diskriminierung reaktionskinetischer Modelle in der Biotechnologie behandelt wird und [Leifheit and King, 2004, King et al., 2002], worin ein Framework zur (semi-) automatischen Identifizierung von chemischen Reaktionsmodellen beschrieben wird. Letzteres basiert auf einem allgemeinen kombinatorischen Formalismus, um eine große Anzahl von Black-Box Modellen zu erstellen, in dem die Beziehungen zwischen den Modellen aber nicht weiter berücksichtigt werden. In [Hoffmann et al., 2004] wird ein evolutionärer Algorithmus für die Diskriminierung von ökologischen Modellen vorgestellt, bei dem angenommen wird, dass ein einzelner Parametervektor für alle Modelle genommen werden kann. Dieses Konzept kann aber nicht für die biochemischen Modelle angewendet werden, die in Kapitel 3.4 eingeführt wurden. Alle genannten Beispiele sind aber noch nicht mit den nachfolgenden beschriebenen Strukturen angereichert, die eine automatische Navigation im Raum der Modellvarianten ermöglichen.

In dieser Arbeit wird ein neuartiger Ansatz erprobt, der die typische Vorgehensweise in der experimentellen Modellbildung softwaretechnisch unterstützen soll. In Kapitel 3 wurde für die Beschreibung metabolischer Netzwerke eine mathematisch formale Beschreibung in den Gleichungen 3.2 und 3.6 eingeführt, die ein einzelnes Modell be-

schreiben. Für die Beschreibung von Modellfamilien muss diese formale Beschreibung wie folgt erweitert werden.

$$\dot{\vec{x}}^i = \mathbf{N}^i \cdot \vec{v}^i(\vec{\alpha}^i, \mathbf{S}, \vec{x}^i), \quad i \in I \tag{6.1}$$

Der Zustandsvektor \vec{x} , Netzwerkstruktur \mathbf{N} , reaktionskinetische Terme \vec{v} sowie deren Parameter $\vec{\alpha}$ können von Modell zu Modell verschieden sein, während die externen Einflussgrößen \mathbf{S} dieselben bleiben. Hier ist I eine Indexmenge, die alle Modelle des sequentiellen Modellierungsprozesses aufzählt. Eine lineare Sequenz der Modelle spiegelt aber nicht die eigentlichen Beziehungen zwischen den Modellen wieder, weil es Verzweigungen, Varianten, Sackgassen oder Modellvereinigungen gegeben haben kann (Abb. 6.2). Daher muss die Indexmenge I durch eine zusätzliche Relation \prec erweitert werden, die die Beziehung zwischen den Modellen angibt. Die Relation $i \prec j$ wird verwendet, wenn Modell j durch eine elementare Netzwerkmodifikation von Modell i entstanden ist. Mit „elementar“ ist hier eine einfache Änderung an einer Kinetik oder der Netzwerktopologie gemeint, so wie im Weiteren definiert. Durch diese elementare Relation \prec wird ein gerichteter Graph erzeugt, der die Beziehungen zwischen allen Modellen repräsentiert (siehe „Netzwerk von Netzwerken“ in Abbildung 6.5).

In diesem Kapitel werden Modellfamilien als Modellierungswerkzeug anhand einer Beispielfamilie informell vorgestellt. In Kapitel 12 folgt dann eine formale Beschreibung der Modellfamilien.

6.2. Kinetische Varianten

.....

Ein einfaches Beispiel in Abbildung 6.2 soll das Konzept der Modellfamilie detaillierter illustrieren. Angenommen, das erste Modell (Abbildung 6.2, Netzwerk 1) im Modellierungsprozess ist ein einfaches verzweigtes Netzwerk mit fünf Reaktionsschritten vom Typ Michaelis-Menten (exemplarisch für v_1):

$$v_1^{MM} = v_{1,max} \cdot \frac{A}{A + k_{1,S}} \tag{6.2}$$

Ein Vergleich des Modells mit gemessenen Daten könnte darauf hinweisen, dass ein inhibierender Effekt des Pools D auf den ersten Reaktionsschritt v_1 vorhanden ist. Um diesen Effekt in das Modell zu übernehmen, muss der reaktionskinetische Term des ersten Reaktionsschrittes geändert werden, was das zweite Modell (Abb. 6.2, Netzwerk 2) erzeugt. Das zweite Modell nennen wir eine kinetische Variante des ersten Modells, weil nur eine Kinetik geändert wurde und die Netzwerktopologie gleich geblieben ist. Der Einfachheit halber wird die Inhibition als multiplikativer Term ausgedrückt. Das Softwaretool MMT2 lässt aber auch komplexere Ausdrücke zu, die die Reaktion exakter beschreiben können.

$$v_1^{MMI,C} = v_{1,max} \cdot \frac{A}{A + k_{1,S}} \cdot \frac{1}{1 + k_{1,I} \cdot D} \tag{6.3}$$

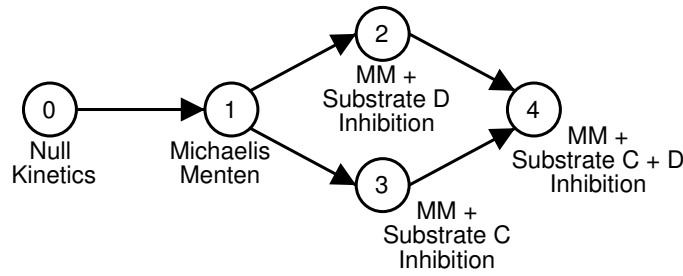


Abbildung 6.3: Darstellung der Beziehungen zwischen reaktionskinetischen Formeln durch einen Graphen am Beispiel der Formeln 6.2 bis 6.4.

Wie man gut erkennen kann ist nur ein Parameter hinzugefügt wurden (die Inhibierungskonstante k_I), die Michaelis-Menten Parameter $v_{1,max}$, $k_{1,m}$ haben weiterhin dieselbe biologische Bedeutung im zweiten Modell als im ersten Modell. Dies wird noch klarer wenn man den Grenzwert $\lim_{k_I \rightarrow 0}$ betrachtet, wodurch man sieht, dass dieser reaktionskinetische Ausdruck ein Spezialfall des ersten ist. Formal ausgedrückt wird diese elementare Erweiterung zwischen den ersten beiden Modellen durch die Relation $1 \prec 2$.

Nachdem das Inhibierungsmodell ausprobiert wurde, ist nach der Simulation das Ergebnis immer noch nicht zufrieden stellend und die Frage stellt sich, ob der andere Metabolit C deren Inhibitor sein könnte. Außerdem gibt es noch die Möglichkeit, dass beide Metabolite C und D einen inhibierenden Effekt auf die Reaktion v_1 ausüben. Daher müssen zwei weitere reaktionskinetische Terme ausprobiert werden, wodurch die Modelle 3 und 4 in der Sequenz entstehen (Abb. 6.2c und 6.2d). Eine kinetische Beschreibung der zweifachen Inhibierung könnte wieder durch einen multiplikativen Term ausgedrückt werden

$$v_1^{MMII} = v_{1,max} \cdot \frac{A}{A + k_{1,S}} \cdot \frac{1}{1 + k_{1,C} \cdot C} \cdot \frac{1}{1 + k_{1,D} \cdot D} \quad (6.4)$$

Die bisherige Betrachtung des Reaktionsschrittes v_1 brachte vier mögliche kinetische Beschreibungen hervor. Zunächst eine einfache Beschreibung durch die Michaelis-Menten Kinetik (Gleichung 6.2), erweitert durch eine inhibierende Wirkung eines der Metabolite C oder D (Gleichung 6.3) und schließlich die gemeinsame Wirkung der Metabolite C und D (Gleichung 6.4). Abb. 6.3 zeigt diese in Betracht gezogenen kinetischen Beschreibungen in Beziehung zueinander gestellt. Die abgebildete Null-Kinetik wird später eingeführt.

Die Zusammenhänge der vier kinetischen Varianten des Reaktionsschrittes v_1 sind in Abb. 6.4 dargestellt. Die Nummerierung in Abb. 6.3 sagt nichts über die logischen Zusammenhänge aus. Beispielsweise ist Modell 3 eine elementare Verallgemeinerung von Modell 1 in der gleichen Weise wie Modell 2 eine Erweiterung von Modell 1 ist. Modell 4 hingegen ist eine elementare Verallgemeinerung von den zwei Modellen 2 und 3. Es ist aber auch indirekt eine Verallgemeinerung des Modells 1 mit Modell 2 oder 3 als Zwischenschritt.

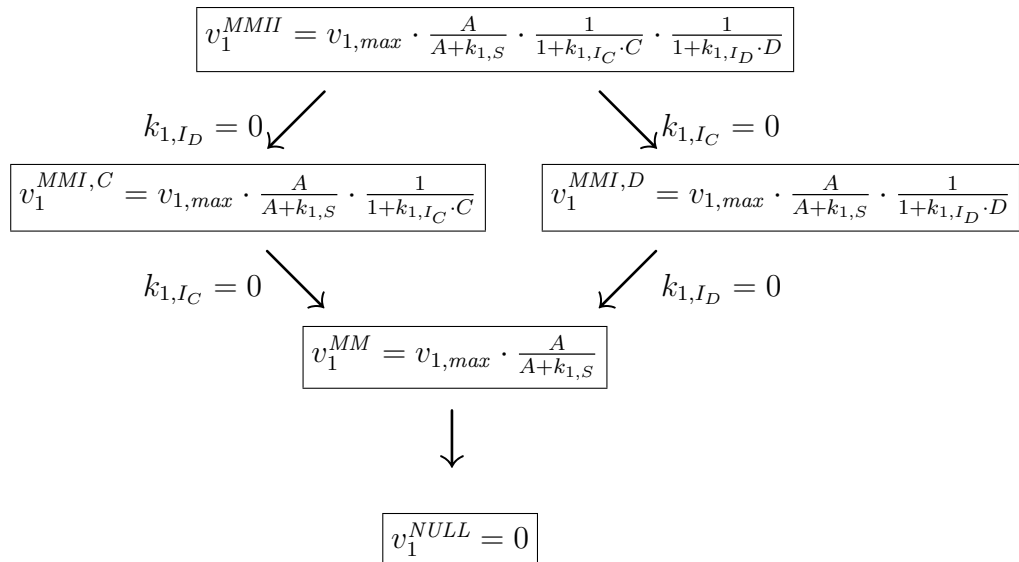


Abbildung 6.4: Vgl. Abb. 6.3. Alle Kinetiken sind ein Spezialfall der Kinetik v_1^{MMII} bei denen eine oder beide Inhibierungskonstanten $k_{1,IC}$ und $k_{1,ID}$ auf Null gesetzt wurden.

6.3. Netzwerkvarianten

.....

Bisher werden am laufenden Beispiel nur kinetische Varianten des Netzwerkes illustriert, die keinen Einfluss auf die Netzwerktopologie haben. Netzwerkvarianten sind die zweite Art von Modifikationen in einem Modellierungsprozess. In einer elementaren Modifikation des Netzwerkes wird ein neuer Reaktionsschritt eingefügt oder ein existierender kann herausgenommen werden.

Nehmen wir an, dass zusätzliche Messdaten verfügbar werden, mit denen man den zusammengefassten Pool A in die zwei Teile A und E aufteilen kann. Dadurch wird auch der Reaktionsschritt v_1 (Abb. 6.2d) in die zwei Reaktionsschritte v_6 and v_7 (Abb. 6.2e) unterteilt.

Der Schritt von Netzwerk 4 zu Netzwerk 5 ist keine elementare Erweiterung oder Verallgemeinerung, weil dort Modifikationen in drei Reaktionsschritten stattfinden (v_1 , v_6 and v_7). Daher müssen Hilfsmodelle eingeführt werden, um die Beziehung zwischen den Modellen 4 und 5 durch eine Sequenz von elementaren Modifikationen (Abb. 6.5) darstellen. Diese Hilfsmodelle (Modelle I-IV) müssen aber nicht unbedingt explizit von Modellierer formuliert werden. Der Weg von Modell 4 zu Modell 5 führt nun über die Sequenz von Modellen 4-I-II-5 oder alternativ über 4-III-IV-5 oder 4-V-VI-5.

Es stellt sich heraus, dass die Modelle I, III und IV praktisch sinnlos sind, weil sie isolierte Metabolite oder Sackgassen-Metabolite besitzen. Daher kann sich kein biologisch sinnvoller stationärer Zustand außer dem Null-Fluss einstellen, der in einem metabolischen Netzwerk keinen Sinn macht. Modell II repräsentiert das maximale Netzwerk, das die Modelle 4 und 5 verallgemeinert. Grundsätzlich sollte jedes metabolische

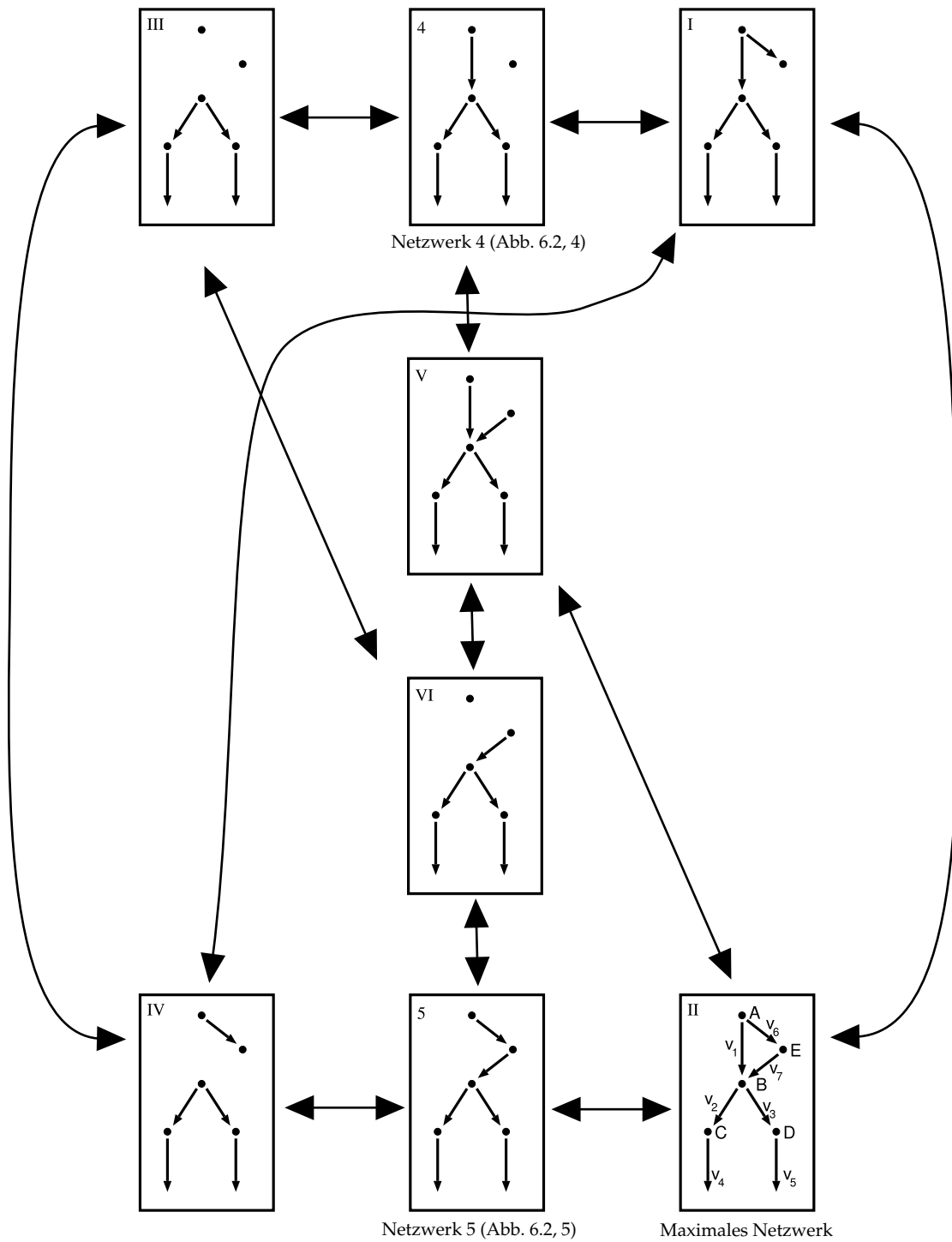


Abbildung 6.5: a) Einige Varianten des Beispielnetzwerkes und ihre Nachbarschaftsbeziehungen. Abb. 6.2 zeigt den Übergang von Netzwerk 4 nach Netzwerk 5 als einen Schritt, obwohl es dazwischen noch Netzwerke gibt.

Netzwerk die Bedingung erfüllen, dass eine stationäre Lösung ungleich null möglich ist (s. Abschnitt 6.6).

6.4. Kombinatorische Modellgenerierung

Bisher wurde der Modellbildungsprozess aus einer rückblickenden Perspektive beschrieben um die allgemeine Vorgehensweise der Modellbildung zu zeigen. Bei der Modellierung von kleinen metabolischen Netzwerken mit weniger als fünf Reaktionsschritten ist es normalerweise ausreichend, den Modellierungsprozess Schritt für Schritt mit elementaren Veränderungen durchzuführen. Die zu beobachtende Abweichung zwischen Modell und gemessenen Daten geben in einer solchen Situation gewöhnlich die nötigen Hinweise auf die Änderungen, die in der Netzwerkstruktur oder den Reaktionskinetiken zu machen sind.

Bei der Modellierung großer metabolischer Modelle hingegen wird es immer schwieriger die richtigen Ideen für eine Modellveränderung zu finden. Es gibt gewöhnlicherweise mehrere Alternativen wie das bestehende Modell verbessert werden kann, und erst eine Kombination dieser Alternativen könnte ein gutes Ergebnis sein.

Im Falle des laufenden Beispiels nehmen wir an, dass in die momentane Situation des Modellbildungsprozesses das Netzwerk 5 in Abb. 6.2 ist. Alle Schritte, die von Netzwerk 1 bis Netzwerk 5 geführt haben, werden jetzt zusammengefasst und bilden auf die folgende Weise kombinatorisch einen Modellsuchraum:

1. Alle Reaktionsschritte, die in den verschiedenen Modellen vorhanden sind, werden vereinigt in ein maximales Netzwerk II in Abb. 6.5. In diesem Netzwerk können die Reaktionsschritte v_1 , v_6 und v_7 ein- und ausgeschaltet werden. Im Falle einer Isolation eines Metabolitknotens wird dieser automatisch vom Netzwerk entfernt. Wenn beispielsweise die Reaktionsschritte v_6 und v_7 gleichzeitig ausgeschaltet werden, so wird auch Metabolit E entfernt. Das resultierende Netzwerk ist die Nr. 4.
2. Für andere Reaktionsschritte stehen mehrere kinetische Beschreibungen zur Verfügung, die vom Modellierer vorgegeben werden. Diese Alternativen müssen explizit durch kinetische Formeln gegeben werden und auch die Beziehungen untereinander müssen spezifiziert werden. Parametern mit der gleichen biologischen Bedeutung muss in allen kinetischen Varianten derselbe Name gegeben werden. Im Beispiel gibt es nur die Reaktionsschritte v_1 und v_6 , für die verschiedene kinetische Formeln zur Auswahl stehen. Die vier möglichen Varianten des Reaktionsschritts v_1 und ihre Beziehungen sind in Abb. 6.3 abgebildet. Analog dazu kann v_6 als Michaelis-Menten-Kinetik oder als inhibierte Michaelis-Menten-Kinetik mit Inhibitor C angenommen werden (vgl. Abb. 6.6).

Die Gesamtzahl der Modelle, die dadurch generiert wird, ist $4 + 8 + 8 + 2 + 2 + 1 + 4 + 1 = 30 = 30$ (vgl. Tab. 6.5). In dieser Situation wird es immer aufwändiger von Hand

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	kin. variants
4	1	1	1	1	1	0	0	4
I	1	1	1	1	1	1	0	8
II	1	1	1	1	1	1	1	8
5	0	1	1	1	1	1	1	2
IV	0	1	1	1	1	1	0	2
III	0	1	1	1	1	0	0	1
V	1	1	1	1	1	0	1	4
VI	0	1	1	1	1	0	1	1
								Σ 30

Tabelle 6.5: Tabellarische Darstellung des Modellraums durch Auflistung jeder Netzwerkvariante. Die Ziffern 0 und 1 zeigen an, ob eine Reaktion ein- oder ausgeschaltet ist.

jede der Modellvarianten zu formulieren und die Parameter an die experimentellen Daten anzupassen. Sogar eine kleine Anzahl möglicher Modellveränderungen kann in Kombination miteinander schnell zu einer Explosion der Zahl der Modellvarianten führen, bis ein manuelles Durchprobieren nicht mehr zu bewältigen ist.

Mit der Perspektive des Netzwerk/Cluster-Computings [Foster and Kesselman, 2003, Berman et al., 2003] ist dies eine ganz typische Anwendung, die automatisiert auf ein Computernetzwerk verteilt werden kann. Mit der Grid-Technologie ist es möglich, sogar Modellräume mit mehreren hundert metabolischen Netzwerken zu durchsuchen und sie an experimentellen Daten anzupassen.

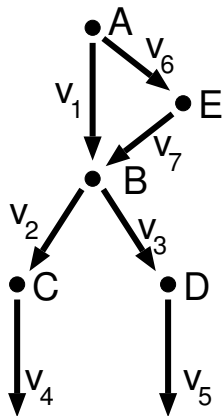
Trotzdem muss die Anzahl der zu testenden Modelle auf das Nötigste reduziert werden. Es sollte klar sein, dass eine große Anzahl von den generierten Modellen vom Modellierer von vornherein nicht gewollt ist oder biologisch unsinnig wäre. Durch die Methode der Elementarmoden und durch benutzerdefinierte Regeln wird der Modellraum minimiert, was in Kapitel 6.6 beschrieben wird.

6.5. Zusammenlegung kinetischer- und Netzwerk-Varianten

.....

Bisher wurde in diesem Kapitel das Konzept der Modellfamilien eingeführt. In den weiteren Abschnitten dieses Kapitels werden Teilaspekte der praktischen Umsetzung des Konzepts in MMT2 vorgestellt. Bei der Implementierung der Modellfamilien wird nicht zwischen kinetischen Varianten und Netzwerkvarianten unterschieden. Stattdessen werden sie formal in eine Beschreibung vereinigt durch Einführung einer Nullkinetik. Abb. 6.5 zeigt die Struktur des Modellvariantenraums, der sich kombinatorisch

1. Maximales Netzwerk



2. Kinetische Varianten

v ₁	Nullkinetik Michaelis-Menten Michaelis-Menten + 1 Inh (C) Michaelis-Menten + 1 Inh (D) Michaelis-Menten + 2 Inh (C+D)	5 Varianten
v ₂	Michaelis-Menten	1 Variante
v ₃	Michaelis-Menten	1 Variante
v ₄	Michaelis-Menten	1 Variante
v ₅	Michaelis-Menten	1 Variante
v ₆	Nullkinetik Michaelis-Menten Michaelis-Menten + 1 Inh (C)	3 Varianten
v ₇	Nullkinetik Michaelis-Menten	2 Varianten

Abbildung 6.6: Darstellung einer Modellfamilie in Form von 1. maximalem Netzwerk und 2. allen kinetischen Varianten aller Reaktionen.

aus der Beschreibung ergibt. Die Modelle sind identisch bis auf eine elementare Verallgemeinerung oder Spezialisierung, die durch die Pfeile angedeutet sind.

Alle Varianten des Modellraums werden in tabellarischer Form in den Zeilen der Tabelle 6.5 aufgelistet. Die erste Spalte bezeichnet die Netzwerke wie in der Abbildungen 6.5. Die folgenden sieben Spalten geben an, ob eine Reaktion ein oder ausgeschaltet ist. Die rechte Spalte in Tabelle 6.5 gibt an, wie viele mögliche kinetische Varianten es für das jeweilige Netzwerk gibt. Für die Reaktionen v₁ gibt es vier kinetische Varianten und für v₆ gibt es zwei mögliche kinetische Varianten.

In der ersten Zeile wird das Modell 4 repräsentiert, bei dem bis auf die Reaktion v₁ alle Reaktionen nur eine Variante besitzen. Daher ist die kombinatorisch mögliche Anzahl von kinetischen Varianten dieses Netzwerkes 4 (ohne die Nullkinetik), wie in der letzten Spalte angegeben. Die zweite Zeile steht für das Modell I, bei dem zwei Reaktionen (v₁ und v₆) kinetische Varianten besitzen. Hier ist die Zahl der kombinatorischen Möglichkeiten 4 · 2. Summiert man die letzte Spalte auf, so erhält man die Gesamtzahl der Modelle im Modellraum.

Netzwerkvarianten und kinetische Varianten können durch Einführung einer künstlichen Nullkinetik und durch ausschließliches Betrachten des maximalen Netzwerkes, leicht in eine Darstellung zusammen geführt werden. In Abb. 6.6 ist in Punkt 2 eine verallgemeinerte Darstellung der Tabelle 6.5 angegeben, in der nicht mehr zwischen den Netzwerkvarianten unterschieden wird, sondern nur noch die möglichen Varianten jeder Reaktion aufgelistet werden. Die Nullkinetik zählt dabei als mögliche kinetische Variante. Die Gesamtzahl der Modellvarianten ergibt sich jetzt viel direkter als:

$$\prod_{i=1}^7 \text{anzahl_varianten}(v_i) = 5 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 3 \cdot 2 = 30 \text{ (vgl. Abb. 6.6).}$$

6.6. Ausschluss biologisch sinnloser Modelle

Um den Aufwand beim kombinatorischen Durchtesten aller Modelle zu verringern, sind in MMT2 zwei Ausschlussmethoden implementiert. Ein automatischer Navigationsalgorithmus könnte aber trotzdem die ausgeschlossenen Modelle hilfsweise als Zwischenschritte verwenden, um von einer Modellvariante zur nächsten zu kommen (Abb. 6.5).

1. Jedes Modell im Suchraum muss biologisch sinnvoll sein. Eine mögliche Definition hierfür ist, dass eine stationäre Lösung ohne Null-Flüsse möglich sein muss. Ob solch ein Zustand möglich ist, kann durch die Berechnung der elementaren Flussmoden, die in Abschnitt 6.7 vorgestellt werden, effizient ermittelt werden. Durch diesen Mechanismus werden die Netzwerke I, III, IV, V und VI ausgeschlossen und die Anzahl der Modelle reduziert sich auf 14.
2. Weitere Einschränkungen können durch den Benutzer durch Bool'sche Ausdrücke formuliert werden, die beschreiben, welche Kombinationen von Kinetiken ausgeschlossen werden. Zum Beispiel erzwingt der folgende Ausdruck, dass genau einer der Reaktionsschritte v_1 und v_6 die Nullkinetik sein muss.

$$\text{isnull}(v_1) \quad \text{XOR} \quad \text{isnull}(v_6) \quad (6.5)$$

Damit werden die Netzwerke I, II, III, V und VI ausgeschlossen und der Modellraum reduziert sich (ohne Berücksichtigung des ersten Kriteriums) auf 8 Modelle.

Mit diesen beiden Ausschlussmechanismen wird der Modellraum auf die Netzwerke 4 und 5 reduziert. Wenn man der kombinatorischen Explosion mit den beschriebenen Ausschlussmechanismen entgegen wirkt, sollte es klar werden, dass das Durchsuchen des komplexen Modellsuchraumes mit der Unterstützung von Höchstleistungsrechnern zu bewältigen ist.

6.7. Elementare Flussmoden

Die Methode der elementaren Flussmoden [Schuster et al., 2000] teilt das komplette Netzwerk auf elementare Mengen von Reaktionen auf. Diese so genannten elementaren Flussmoden können als minimale Unternetzwerke interpretiert werden, die unabhängig vom Gesamtnetzwerk einen stationären Zustand einnehmen können. Abb. 6.7a listet die vier Elementarmoden des maximalen Netzwerkes aus dem Beispielnetzwerk II auf.

Eine einfach umzusetzende Vorgehensweise, die so genannten „failure modes“ zu berechnen, ist in [Klamt and Gilles, 2004] beschrieben. Durch Berechnung der Elementarmoden des maximalen Netzwerkes ist es möglich vorherzusagen, welche Reaktio-

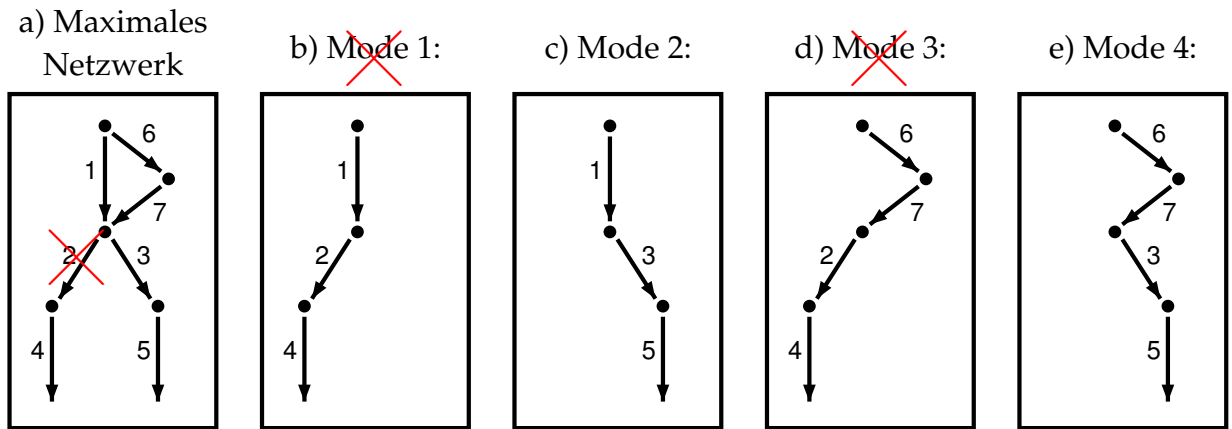


Abbildung 6.7: Die Elementarmoden des Beispiels. In a) wird das maximale Netzwerk gezeigt (Abb. 6.5, Netzwerk II), b)-e) zeigen jeweils eine der vier Elementarmoden, die eine minimale Menge an Reaktionen enthalten, bei der noch ein stationärer Zustand erreicht werden kann. Die roten Kreuze zeigen, welche Moden bei Ausschalten der Reaktion 2 verschwinden

Reaktion Nr.	#1	#2	#3	#4	#5	#6	#7
Mode nr. 1	1	1	0	1	0	0	0
Mode nr. 2	1	0	1	0	1	0	0
Mode nr. 3	0	1	0	1	0	1	1
Mode nr. 4	0	0	1	0	1	1	1

Tabelle 6.7: Tabellarische Notation der Elementarmoden von Abb. 6.7.

nen nicht mehr funktionieren können, wenn man eine bestimmte Reaktion ausschaltet (d. h. auf die Nullkinetik setzt). Wenn im Beispiel Reaktion #2 ausgeschaltet wird, werden alle Elementarmoden, die diese Reaktion besitzen (also Mode 1 und 3), nicht mehr funktionieren. In den verbleibenden Moden ist die Reaktion #4 immer ausgeschaltet. Dies zeigt an, dass diese Reaktion vom restlichen Netzwerk abgeschnitten ist.

Metabolic Modeling Markup Language - M3L

Dieses Kapitel beschäftigt sich mit M3L, dem Speicherformat von MMT2, das in Abb. 1.3 bei den Punkten C - F beteiligt ist. Beim Modell-Setup (Abb. 1.3C) müssen alle Informationen, die im graphischen Editor eingegeben wurden, im M3L-Format (Abb. 1.3D) abbildbar sein. Einer der wesentlichen Erweiterungen in M3L gegenüber vergleichbaren Sprachen ist die Unterstützung von Modellfamilien (Abb. 1.3E), die bereits informal in Kapitel 6 eingeführt wurden. Schließlich ist M3L die Eingabe zum Quelltextgenerator (Abb. 1.3F), der den Simulator erzeugt und kompiliert.

In diesem Kapitel wird zunächst ein kleiner Überblick über bereits vorhandene Sprachen zur Beschreibung von biologischen Modellen gegeben. Dann werden die wichtigsten Bestandteile von M3L kurz, anhand von XML-Fragmenten, vorgestellt. Für eine umfassende Beschreibung von M3L wird auf das MMT2 Benutzerhandbuch [Haunschild, 2005] verwiesen.

7.1. Überblick über existierende Modellierungssprachen

.....

In den Biowissenschaften ist XML zu einer Grundlage vieler Beschreibungssprachen geworden. In mehreren Forschungsprojekten wurden XML-basierte Sprachen entwickelt um Datenbestände aufzubauen, zu analysieren und zu durchsuchen. Viele dieser Sprachen beschreiben biologische Systeme auf einer anderen Ebene, als sie für das Softwaretool dieser Arbeit benötigt werden wie z. B.

- die BIOPolymer Markup Language (BIOML) [Fenyo, 1999] zur Beschreibung experimentell ermittelter Informationen über Proteine und andere Polymere;
- die Protein Sequence Database Markup Language (PSDML), zum Speichern von Informationen über Proteine, die in der *Protein Information Resource* (PIR, <http://pir.georgetown.edu/pirwww/>) Datenbank enthalten sind;
- die Bioinformatic Sequence Markup Language (BSML) (<http://www.bsml.org>), zur Speicherung des Codes und der graphischen Darstellung von DNA, RNA und Proteinsequenzen;
- die Genome Annotation Markup Elements (GAME) (<http://xml.coverpages.org/game.html>), als Beschreibungssprache, die in der Molekularbiologie verwendet wird um Anotationen für eine Biosequenz zu speichern;

- die Multiple Sequence Alignments Markup Language (MSAML) (<http://xml.coverpages.org/msaml.html>), zur Unterstützung der Manipulation und Extraktion von Multiple-Sequence-Alignment-Informationen;
- die Gene Expression Markup Language (GEML) (<http://xml.coverpages.org/geml.html>), zur Speicherung von DNA-Microarray- und Gen-Expressionsdaten;
- die Microarray Markup Language (MAML) (<http://xml.coverpages.org/maml.html>),
- und die Chemical Markup Language (CML) [Murray-Rust and Rzepa, 2003].

Zwei XML-basierte Sprachen zur Beschreibung zellulärer Stoffwechselmodelle, wie sie in dieser Arbeit vorkommen, sind

- CellML [Lloyd et al., 2004] (<http://www.cellml.org>) in der die zugrunde liegende Struktur und Modellgleichungen von zellulären Modellen in einer sehr allgemeinen Form beschrieben wird; und die
- Systems Biology Markup Language (SBML, [Hucka and et. al, 2003], <http://www.sbml.org>), die auf Pathway- und Reaktionsmodelle spezialisiert ist.

CellML wird von *Physiome Sciences, Inc.* entwickelt in Zusammenarbeit mit der *bioengineering research group* der University of Auckland und wird für biologische Modelle als Austausch- und Speicherformat verwendet. Für diese Arbeit würde die sehr allgemeine Form der Beschreibung zu einer komplexen und nicht mehr einfach nachvollziehbaren Dateistruktur führen, die durch ein anderes Format, wie z. B. SBML vermeidbar wäre.

SBML versteht sich in erster Linie als Austauschformat zwischen verschiedenen Applikationen für biochemische Netzwerkmodelle auf der Basis einer stöchiometrischen oder reaktionskinetischen Beschreibung. Die Entwicklung von SBML wurde begonnen mit der Vereinigung der wichtigsten Elemente der Softwaretools BioSpice (<http://biospice.lbl.gov>), DBSolve [Goryanin et al., 1999], E-Cell [Tomita et al., 2001], Gepasi [Mendes, 1997], Jarnac [Sauro, 2000], StochSim [Morton-Firth and Bray, 1998] und Virtual Cell [Loew and Schaff, 2001]. Mit Import/Export-Schnittstellen im SBML-Format konnte zwischen diesen Tools, die zu den bekanntesten und meistgenutzten gehören, Modelle ausgetauscht werden. Bis heute ist die Anzahl der Softwaretools, die SBML unterstützen, auf über 80 angewachsen (<http://www.sbml.org>). In SBML werden die Elemente eines Modells in Kategorien aufgeteilt, von denen die wichtigsten *model*, *compartment*, *specie*, und *reaction* sind. Die Strukturierung der Daten innerhalb des Dokuments ist übersichtlich und einfach gestaltet, was die Bearbeitung des Modells mit einem XML-Editor ermöglicht.

Das Softwaretool MMT2 lehnt sich wegen der einfachen Handhabung an SBML an. Leider werden in SBML nicht alle Features unterstützt, die für die Auswertung von Rapid-Sampling-Ergebnissen und die Betrachtung von Modellfamilien benötigt werden. Dies betrifft insbesondere die fehlende Möglichkeit in SBML Messdaten als Modellinput zu verwenden und Messdaten als Qualitätskriterium für eine Simulation

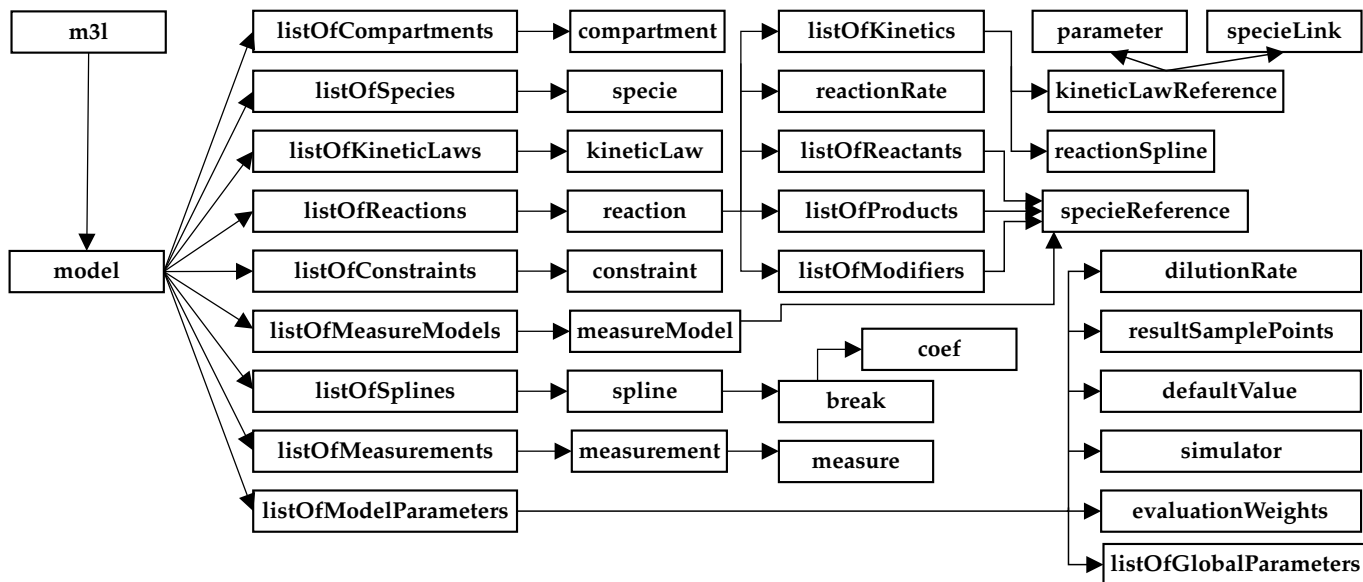


Abbildung 7.1: Hierarchische Darstellung der Elemente der XML-Sprache M3L als Baumstruktur. In Listing 7.1 ist erkennbar, dass M3L ein Dialekt der Sprache SBML ist.

```

1: <m3l>
2:   <model name="...">
3:     <listOfCompartments>
4:       ...
5:     </listOfCompartments>
6:     <listOfSpecies>
7:       ...
8:     </listOfSpecies>
9:     <listOfReactions>
10:      ...
11:    </listOfReactions>
12:    <listOfKineticLaws>
13:      ...
14:    </listOfKineticLaws>
15:    <listOfConstraints>
16:      ...
17:    </listOfConstraints>
18:    <listOfModelParameters>
19:      ...
20:    </listOfModelParameters>
21:    <listOfSplines>
22:      ...
23:    </listOfSplines>
24:    <listOfMeasurements>
25:      ...
26:    </listOfMeasurements>
27:    <listOfMeasureModels>
28:      ...
29:    </listOfMeasureModels>
30:  </model>
31: </m3l>

```

Listing 7.1: Die Hauptebene eines M3L-Baums in dem MMT2 ein Modell speichert. Die Elemente ab Zeile 12 sind in SBML nicht enthalten. Das Element listOfReactions hat eine andere Definition als in SBML, da die Veränderungen nicht als Erweiterung abzubilden sind.

während einer Parameteranpassung zu verwenden. Bei der Entwicklung des XML-Speicherformats Metabolic Modeling Markup Language (M3L) wurde aber der Aufbau nahe an SBML angelehnt. Eine reine Erweiterung von SBML ist M3L jedoch nicht, da neben Erweiterungen für die Features von MMT2 auch bestehende Sprachelemente ersetzt werden mussten (Abschnitte 7.2 und 7.4).

In den nächsten Abschnitten wird die Struktur von M3L erläutert und die Unterschiede zwischen SBML und M3L aufgezeigt. Diese werden bereits in Listing. 7.1, der Grundstruktur des SBML-Dialektes Metabolic Modeling Markup Language (M3L), deutlich.

7.2. Modellvarianten

.....

Das Modell wird unterhalb des Elements `model` gespeichert. Die Elemente `listOfCompartments`, `listOfSpecies` und `listOfReactions` wurden von SBML übernommen. Im Gegensatz zu SBML werden die kinetischen Reaktionsbeschreibungen nicht unter dem Element `listOfReactions` gespeichert, sondern wurden ausgelagert in das Element `listOfKineticLaws`. Durch einen eindeutigen Bezeichner für jede kinetische Reaktionsbeschreibung werden sie aus den Reaktionsdefinitionen referenziert. Diese Auslagerung verringert die Fehleranfälligkeit und vermeidet Redundanz, wenn Reaktionskinetiken oft wieder verwendet werden, wie es bei Modellfamilien der Fall ist.

Das Listing 7.2 zeigt am Beispiel aus Kapitel 6 (Abb. 6.5, Maximales Netzwerk, Reaktion v_1) eine Reaktion mit mehreren kinetischen Varianten. Die Reaktionskinetiken werden nicht explizit angegeben, sondern referenziert. Daher müssen die Formeln in Listing 7.3 in einer allgemeinen Form mit Platzhaltern angegeben werden, die dann in der Reaktionsdefinition aufgelöst werden (z. B. Listing 7.2, Zeilen 23-26).

Um eine M3L-Datei zu erzeugen und zu bearbeiten wurde in dieser Arbeit größtenteils der Open-Source XML-Editor Xerlin (Abb. 10.4) verwendet. Für die erstmalige Erstellung eines M3L-Modells kann auch ein Modell in SBML oder Metatoolformat erstellt werden und dann nach M3L konvertiert werden.

7.3. Messdaten

.....

Eine wichtige Erweiterung von M3L gegenüber SBML besteht in der Möglichkeit Messdaten und Splines zu definieren. Die experimentell ermittelten Zeitverläufe können entweder als Referenzverhalten für die Parameteranpassung dienen, oder als Bestandteil der Simulation in Form von Splines um die Komplexität des Modells zu reduzieren. Wird eine Messung als Referenzverhalten definiert, so muss sie unter dem Knoten `listOfMeasurements` (Listing 7.5) definiert werden. Für jede Messung ist es möglich neben dem genauen Zeitpunkt und der Konzentration die Standardabweichung


```

1: <listOfReactions>
2:   <reaction value="v1">
3:     <listOfReactants>
4:       <specieRef compartment="X" specie="A"/>
5:     </listOfReactants>
6:     <listOfProducts>
7:       <specieRef compartment="X" specie="B"/>
8:     </listOfProducts>
9:     <listOfKinetics>
10:      <kineticLawRef kineticLaw="null">
11:      </kineticLawRef>
12:      <kineticLawRef kineticLaw="mm">
13:        <parameter symbol="k_m" value="1"/>
14:        <specieLink specie="A" symbol="S0"/>
15:      </kineticLawRef>
16:      <kineticLawRef kineticLaw="mm_linh">
17:        <parameter symbol="k_m" value="1"/>
18:        <parameter symbol="k_I" value="1"/>
19:        <specieLink specie="A" symbol="S0"/>
20:        <specieLink specie="C" symbol="S1"/>
21:      </kineticLawRef>
22:      <kineticLawRef kineticLaw="mm_linh">
23:        <parameter symbol="k_m" value="1"/>
24:        <parameter symbol="k_I" value="1"/>
25:        <specieLink specie="A" symbol="S0"/>
26:        <specieLink specie="D" symbol="S1"/>
27:      </kineticLawRef>
28:    </listOfKinetics>
29:  </reaction>
30: </listOfReactions>
31: ...

```

Listing 7.2: Um die möglichen Varianten der Definition einer Reaktion abzubilden können mehrere *kineticLaws* angegeben werden. In diesem Abschnitt wird aus dem maximalen Netzwerk (Abb. 6.5, Netzwerk II) die Reaktion v_1 definiert, die vier kinetische Varianten besitzt (inklusive der Null-Kinetik).

```

1: <listOfKineticLaws>
2:   <KineticLaw name="mm" formula="v_m*S0/(S0+k_m)"/>
3:   <KineticLaw name="mm_linh"
4:     formula="v_m*S0/(S0+k_m)*1/(1+k_I)"/>
5:     <extends name="mm"/>
6:   </KineticLaw>
7: </listOfKineticLaws>

```

Listing 7.3: Reaktionskinetische Beschreibungen werden unter dem Element *kineticLaw* definiert und werden von den Reaktionsdefinitionen referenziert. Die Abhängigkeiten zwischen den Kinetiken (Abb. 6.4) werden durch das Element *extends* beschrieben.

```

1: <listOfConstraints>
2:   <constraint name="c1">
3:     (r.v1==k.null XOR r.v6==k.null)
4:   </constraint>
5: </listOfConstraints>

```

Listing 7.4: Durch Constraints kann der Benutzer ungewollte Modellvarianten ausschließen, die sich durch die Kombinatorik ergeben, aber nicht erwünscht sind. Diese Definition legt fest, dass genau eine der Reaktionen v_1 und v_6 auf die NULL-Kinetik gesetzt ist. Durch das Element *extends* wird beschrieben, wo die Kinetik ihren Platz im Abhängigkeitsbaum hat (Abb. 6.4).

```

1: <listOfMeasurements>
2:   <measurement name="mDAHP">
3:     <measure t="-3.942" value="0.0117749"
4:       stddev="6.6e-5"/>
5:     <measure t="-3.708" value="0.0471655"
6:       stddev="0.0004314"/>
7:     <measure t="-3.477" value="0.0379831"
8:       stddev="0.0003058"/>
9:     <measure t="-3.246" value="0.042373"
10:      stddev="0.0003632"/>
11:     <measure t="-3.012" value="0.0464297"
12:      stddev="0.0004205"/>
13:     <measure t="-2.781" value="0.0438399"
14:      stddev="0.0003834"/>
15:     ...
16:   </measurement>
17:   ...
18: </listOfMeasurements>

```

Listing 7.5: Messdaten können in das Modell aufgenommen werden, unter der optionalen Angabe der Standardabweichung für jede einzelne Messung.

chung *stddev* anzugeben. Dieser Wert wurde in den Kooperationsprojekten ermittelt, indem in der Analytik jede Messung drei Mal durchgeführt wurde (Abb. 2.7).

Manchmal ist es nicht möglich, dass zwei Metabolite analytisch getrennt werden können und daher nur eine Gesamtkonzentration mehrerer Metabolite angegeben werden kann. Um eine solche Messung im Modell berücksichtigen zu können, muss man entweder das Modell so konstruieren, dass diese Systemvariable in einen Gesamtmetaboliten modelliert wird, oder es muss softwareseitig eine Unterstützung dafür geben. In den Kooperationsprojekten trat ein solcher Fall auf (Abb. 2.7), bei dem in der Massenspektroskopie zwei Moleküle aufgrund gleich großer Massen nicht getrennt werden konnten, und nur ein Signal für beide Metabolite gemessen werden konnte. MMT2 bietet softwareseitige Unterstützung für solche Messungen und ermöglicht somit, dass die Modellbildung zunächst unabhängig von dieser Eigenschaft mancher experimentellen Daten stattfindet. Das Messmodell definiert dann, wie Daten und Modell zusammenhängen. In Listing 7.6 ist angegeben, dass das gemessene Signal aus 2/3 G6P und 1/3 F6P besteht.

```

1: <listOfMeasureModels>
2:   <measureModel name="mmG6P_F6P">
3:     <specieReference name="F6P" stoichiometry=1/>
4:     <specieReference name="G6P" stoichiometry=2/>
5:   </measureModel>
6:   ...
7: </listOfMeasureModels>

```

Listing 7.6: Mit dem Messmodell ist es möglich, Metabolite, die analytisch nicht getrennt werden konnten, aber getrennt im Modell vorkommen, für die Parameteranpassung verwenden zu können

```

1: <listOfSplines>
2:   <spline name="spl_ADP" t0="-3.942">
3:     <break tr="-3.708" degree="3">
4:       <coef c="0" value="0.119826413412"/>
5:       <coef c="1" value="0.0159524499127"/>
6:       <coef c="2" value="0"/>
7:       <coef c="3" value="0.195012665949"/>
8:     </break>
9:     <break tr="-3.477" degree="3">
10:      <coef c="0" value="0.126057965259"/>
11:      <coef c="1" value="0.0479867905228"/>
12:      <coef c="2" value="0.136898891496"/>
13:      <coef c="3" value="0.0910831719996"/>
14:    </break>
15:    ...
16:  </spline>
17:  ...
18: </listOfSplines>

```

Listing 7.7: Messdaten können auch als Spline in das Modell aufgenommen werden, um die Umgebungsbedingungen an der Systemgrenze zu simulieren.

7.4. Splines

.....

Wie im letzten Abschnitt schon angesprochen, besteht die zweite Möglichkeit experimentelle Daten zu verwenden darin, die Komplexität des Modells zu reduzieren. So kann beispielsweise der komplexe Umwandlungskreislauf von ATP und ADP, der Einfluss auf viele Reaktionen hat, einfach durch gemessene Daten ersetzt werden. Somit erhält der Modellierer die Möglichkeit, sich auf andere Eigenschaften zu konzentrieren und wesentliche Bestandteile des realen Systems unmodelliert zu lassen, ohne sie zu vernachlässigen. Ein Spline wird in M3L wie folgt dargestellt und bietet insbesondere eine freie Wahl des Grades der Teilpolynome.

$$s(t) = \begin{cases} s_1(t), & \text{falls } t_0 \leq t \leq t_1 \\ \vdots \\ s_n(t), & \text{falls } t_{n-1} \leq t \leq t_n \end{cases} \quad (7.1)$$

s_i , i -tes Polynom des Splines
 $c_{i,j}$, Polynomkoeffizient
 m_i , Grad des Polynoms

Die verrauschten Messdaten eignen sich in ihrer rohen Form nicht dazu als Input verwendet zu werden, weil sich dann das Rauschen in den Messdaten in die Simulation fortpflanzen würde. Daher bietet MMT2 die Möglichkeit, gesplinet Messdaten in das Modell einzufügen. Ein solcher glättender Spline, der den Verlauf der Messung beschreibt, muss von einem unabhängigen Tool erzeugt werden und nach MMT2 exportiert werden, was nicht Bestandteil dieser Arbeit war. Dieses Tool wurde von Aljoscha Wahl programmiert unter Verwendung der numerischen Bibliothek NAG (Numerical Algorithms Group) <http://www.nag.co.uk/>, [Pennington and Berzins, 1994].

Listing 7.7 zeigt ein Beispiel für einen Spline, der als stückweise definiertes Polynom n -ten Grades definiert wird. Ein zusätzliches Feature, das MMT2 bietet ist, dass der Spline Unstetigkeiten haben kann wie z. B. Sprünge, die einen externen Glucosepuls simulieren könnten. Da ODE-Löser Probleme bekommen, wenn Unstetigkeiten in den Modellgleichungen existieren, ist in MMT2 eine automatische Erkennung von Unstetigkeiten eingebaut, die den ODE-Löser an einer Stelle t_u anhält, bei der eine Unstetigkeit auftritt, und dann die Berechnung an dieser Stelle t_u neu startet.

In Listing 7.7 wird ab Zeile 2 ein Spline definiert, dessen linke Stützstelle sich am Punkt $t_0 = -3.942$ befindet. Die Stützstellen werden in den `<break>`-Knoten definiert, die Koeffizienten $c_{i,j}$ in `<coef>`. Das erste stückweise Polynom ist definiert von t_0 bis $t_r = -3.708$ (Zeile 3) mit den Koeffizienten aus den Zeilen 4 - 7.

```

1: <listOfSpecies>
2:   <specie compartment="Cyt" name="DAHP"
   initialAmount="0.04" measure="mDAHP" min="0"
   max="0.2"/>
3:   <specie compartment="Cyt" name="ADP"
   from_data="spl_ADAP"/>
4:   <specie compartment="Cyt" name="Phe"
   initialAmount="20" fixed="1"/>
5:   <specie compartment="Cyt" name="EPSP"
   initialAmount="0.2" min="0" max="5"/>
6:   ...
7: </listOfSpecies>

```

Listing 7.8: Die im Modell enthaltenen Metabolite werden in diesem Abschnitt definiert.

Hier wird u. a. festgelegt für welche Metabolite eine ODE aufgestellt wird und welche von den Daten genommen werden.

Nachdem Splines und Messdaten definiert wurden, müssen sie noch mit Systemvariablen in Zusammenhang gebracht werden. So zeigt Listing 7.8 in Zeile 2, dass der Metabolit DAHP bei der Parameteranpassung mit der Messung mDAHP verglichen werden soll. In Zeile 3 wird definiert, dass für die Systemvariable ADP keine Differentialgleichung aufgestellt wird, sondern der Zeitverlauf vom Spline `spl_ADP` übernommen wird. Systemvariablen können auch auf eine Konstante gesetzt werden, wie in Zeile 4, was ein Spezialfall für einen Spline mit konstantem Wert wäre. Für die Metabolite in Zeilen 2 und 5 wird jeweils eine ODE aufgestellt, für die in den Attributen `min` und `max` Konzentrationsgrenzen definiert werden, die eingehalten werden sollen. M3L bietet weiterhin die Möglichkeit, dass Splines und gemessene Daten nicht nur für Metabolite verwendet werden können, sondern auch für Reaktionen [Haunschild, 2005].

7.5. Verfahrensparameter

.....

```

1: <listOfModelParameters>
2:   <defaultValue flux_max="10" flux_min="-10"/>
3:   <dilutionRate value="0"/>
4:   <resultSamplePoints begin="-3.0" end="20"
   interval="0.1"/>
5:   <listOfModelParameters>
6:     <parameter symbol="cNADP_NADPH" value="1.0"
       min="0.35" max="3"/>
7:   </listOfModelParameters>
8: </listOfModelParameters>

```

Listing 7.9: Die Verfahrensparameter, wie z. B. die Simulationszeit, werden in diesem Abschnitt festgelegt.

Listing 7.9 zeigt den Ausschnitt aus einer M3L-Datei, welche die Verfahrensparameter definiert. Neben den Defaultwerten für Fluss- und Konzentrationsgrenzen (Zeile 2) kann eine sog. Auswaschungsrate angegeben werden (Zeile 3), die den Vorgang modelliert, dass Metabolite mit der Zeit in die Biomasse übergehen. Ist dieser Wert definiert, so wird jeder ODE ein Term

$$-dx/dt * d_r \quad (7.2)$$

hinzugefügt mit x als dem jeweiligen Metabolit und d_r der Auswaschungsrate.

Zeile 4 legt fest, in welchem Zeitfenster die Simulation stattfindet und ab Zeile 5 können globale Parameter definiert werden. Im Gegensatz zu den lokalen Parametern (Listing 7.2) sind globale Parameter nicht nur auf eine Reaktion beschränkt, sondern können in jeder beliebigen Reaktion verwendet werden.

Source-Code Generierung

Bezogen auf die Übersicht aus Abb. 1.3 befasst sich dieses Kapitel mit dem Punkt F, der in Abb. 8.1 nochmals dargestellt ist. Nachdem ein Modell erstellt und in der Sprache M3L kodiert wurde, kommt nun im Ablauf der Abb. 1.3 der Schritt, in dem mit Hilfe von Source-Code Generierung ein auf das Modell spezialisierter Simulator erzeugt wird.

Kern des Modells sind ODEs (Kap. 3.4), die von einem ODE-Löser gelöst werden müssen. Die ODE-Lösung ist die Rechenoperation, die bei der Auswertung von Experimenten mit Abstand am häufigsten ausgeführt wird. Daher haben dort bereits kleine Rechenzeiteinsparungen eine große Auswirkung auf die Laufzeit des ODE-Lösers, der sich wiederum in einer Schleife für die Parameteranpassung befindet.

Es gibt zwei allgemeine Varianten, wie die Differentialgleichungen dem ODE-Löser zur Verfügung gestellt werden können. Es gibt die Source-Code Generierung, die z. B. bei dem Mehrzweck-Simulationswerkzeug Modelica eingesetzt wird. Viel häufiger ist aber die Methode eines Interpreters zu finden, bei der der Simulator selber ein Modellfile einliest (SBML oder eigenes Format) und sofort in der Lage ist den zeitlichen Verlauf des Modells zu berechnen.

Bei MMT2 hingegen wird erstere Variante verwendet, bei der ein - im Vergleich zur einzelnen Simulation - relativ zeitintensiver Prozess an Source-Code Generierung

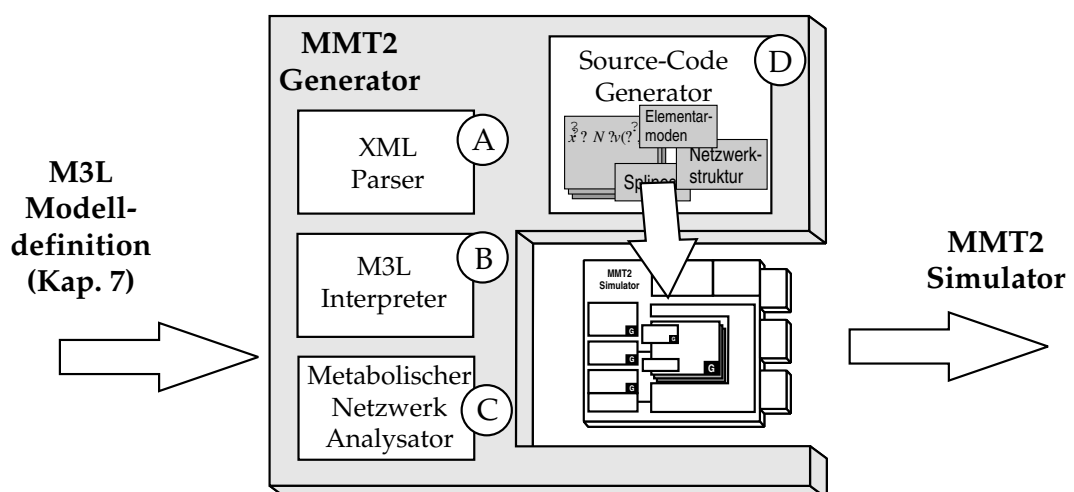


Abbildung 8.1: Architektur des Quelltext-Generators. Nach der Generierung und Kompilierung ist der erzeugte Simulator hochoptimiert auf das Modell, dass in der M3L-Datei spezifiziert wurde.

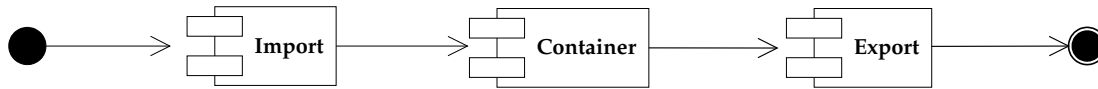


Abbildung 8.2: Architektur des Quelltextgenerators. Die Komponenten Import und Export können ausgetauscht werden wodurch der Quelltextgenerator auch als Konverter zwischen verschiedenen Dateiformaten dienen kann.

in C/C++ und Kompilierung vorgeschaltet werden muss. Der Grund für diese Designentscheidung ist, dass der Simulator in erster Linie für Parameteranpassungen verwendet wird, die durch einen optimierten Simulationscode wesentlich beschleunigt wird. Im Vergleich zur Laufzeit der Parameteranpassung ist die Source-Code Generierung und -kompilierung vernachlässigbar kurz.

8.1. Architektur

.....

Die Anforderungen an MMT2 waren anfangs relativ vielfältig, konnten aber mit einer geeigneten Architektur des Generators in der Weise berücksichtigt werden, dass der Generator nicht ausschließlich Quelltext erzeugt, sondern auch als Konverter zwischen verschiedenen Formaten einsetzbar ist. Diesen Realisierungsansatz zeigt Abb. 8.2, bei dem die Komponenten „Import“ und „Export“ austauschbar sind. Um den Generator als Werkzeug zu realisieren, das für viele Aufgaben verwendbar ist, wurden drei verschiedene „Import“-Komponenten implementiert, die Modelle aus verschiedenen Quellen beziehen können. Dies sind

1. das neue Speicherformat von MMT2 (M3L) als XML-Datei, oder
2. Modelle, die mit anderen Tools erstellt wurden und im SBML-Format vorliegen.

Folgende „Export“-Komponenten wurden implementiert:

1. Ausgabe des Simulators im Quelltext als C/C++-Dateien,
2. Ausgabe des Modells als Matlab M-File (zum Überprüfen der Simulationssoftware),
3. Ausgabe des Modells im M3L-Format, und
4. Ausgabe des Modells im SBML-Format.

Der eigentliche Quelltextgenerator besteht aus der „Import“-Komponente Nr. 1 und der „Export“-Komponente Nr. 1. Weil gerade zu Anfang der Entwicklung von MMT2 die Modellierung noch komplett mit MMT1 durchgeführt wurde, mussten zum Testen von MMT2 die Modelle zunächst aus der MMT1-Datenbank geholt werden. Dafür wurde ein Generator (Abb. 8.2) gebaut, der als „Import“-Komponente Nr. 1 implementiert hatte und als „Export“-Komponente Nr. 3. Die Modelle lagen dann für die weitere Verwendung im M3L-Format vor, was auch unbedingt notwendig wurde, nachdem MMT1 durch MMT2 ersetzt wurde. Im Gegensatz zu der Speicherung in einem RDBS

(Relationales Datenbanksystem) ist die M3L-Datei wesentlich handlicher und lesbarer. Bei MMT1 wurden beim Datenbankdesign konsequent Redundanzen vermieden was zu vielen Transitionstabellen geführt hat. In dieser Speicherform war es nicht mehr möglich von Hand ein Modell nachzuvollziehen. Nach der Konvertierung in M3L ist es ohne weiteres möglich das Modell ohne die Software MMT2 mit einem XML-Editor einzusehen (Kapitel 7).

Die Abbildung 8.3 zeigt eine abstrakte Sicht der Implementierung der Import- und Exportkomponenten. In einer Basisklasse wird die Grundfunktionalität implementiert, die die spezialisierten Klassen gemeinsam haben.

8.2. Implementierung

.....

Der Teil des Quelltextgenerators in Abb. 8.2 ist als Linux-Shared-Object realisiert, dass von einem Steuerprogramm verwendet wird. Er besteht aus der „Import“ -Komponente Nr. 1 (Kap. 8.1) und der „Export“ -Komponente Nr. 1 (Kap. 8.1). Der Ablauf der Source-Code Generierung kann grob in 4 Phasen (Abb. 8.1A-D, Abb. 8.4) eingeteilt werden.

- Zunächst kümmert sich `mmt2.mks` in Phase 1 (Abb. 8.4) darum, die Kommandozeile zu parsen, die Klassenobjekte des Generators für Import, Export und Container zu erzeugen und die Verzeichnisse anzulegen, in die der Simulator geschrieben werden soll.
- In der Phase 2 wird der Importklasse die M3L-Datei übergeben, welche dann geparkt wird (Abb. 8.1A). Als XML-Parser verwendet die Klasse `import_m3l` die Gnome XML-Library, die als `libxml2` unter den Linux-Distributionen erhältlich ist. Während des Parsens des XML-Baums werden die Modellteile extrahiert (Abb. 8.1B) und in der Klasse Container zwischengespeichert.
- In Phase 3 wird das gespeicherte Modell analysiert und die Elementarmoden werden mit Hilfe eines Algorithmus des Softwarewerkzeugs Metatool [Schuster et al., 2000] berechnet (Abb. 8.1C).
- Phase 4 ruft die Klasse `export_m3l` auf, welche das Modell aus der Klasse Container ausliest und als C/C++ Quelltext speichert (Abb. 8.1D). Insbesondere der erzeugte numerische Quelltext, in dem die Modell-ODEs berechnet werden, wird so erzeugt, dass er noch gut lesbar bleibt und Veränderungen sowie Validierungen nach der Source-Code Generierung leicht möglich sind.

Kern des Quelltextgenerators ist die Komponente Container, die als Abstraktionsschicht zwischen den Komponenten Import und Export zu sehen ist. Während der Laufzeit dieser Arbeit wurden der Funktionalität von MMT2 Features hinzugefügt bzw. geringfügig geändert. Beispielsweise gab es zu Beginn keine Unterscheidung zwischen Messdaten und Splines, was zur Folge hatte, dass diese Unterstützung sowohl M3L als auch dem Quelltextgenerator hinzugefügt werden musste. Änderungen an den Features ziehen sich daher meistens vom M3L-Format über

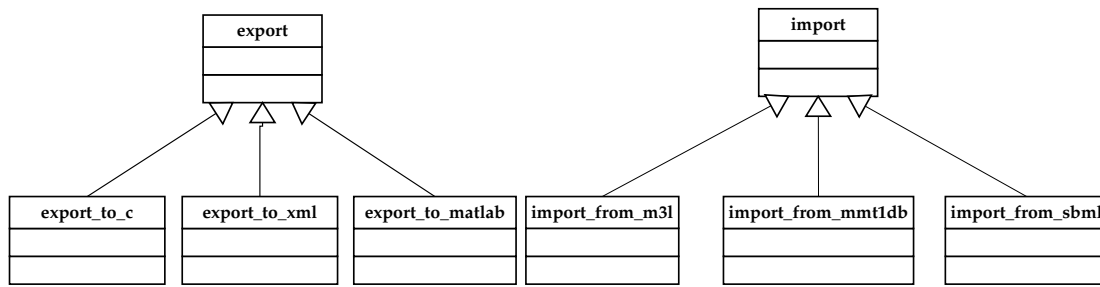


Abbildung 8.3: Links: Die Exportkomponente besteht aus einer Basisklasse und drei Klassen, die die Funktionalität spezialisieren. Hauptsächlich genutzt wird `export_c` während die anderen Klassen nur Anfangs zur Validierung (`export_matlab` und zur Konvertierung (`export_m3l`) verwendet wurden. Rechts: Die Importkomponente ist analog zur Exportkomponente aufgebaut.

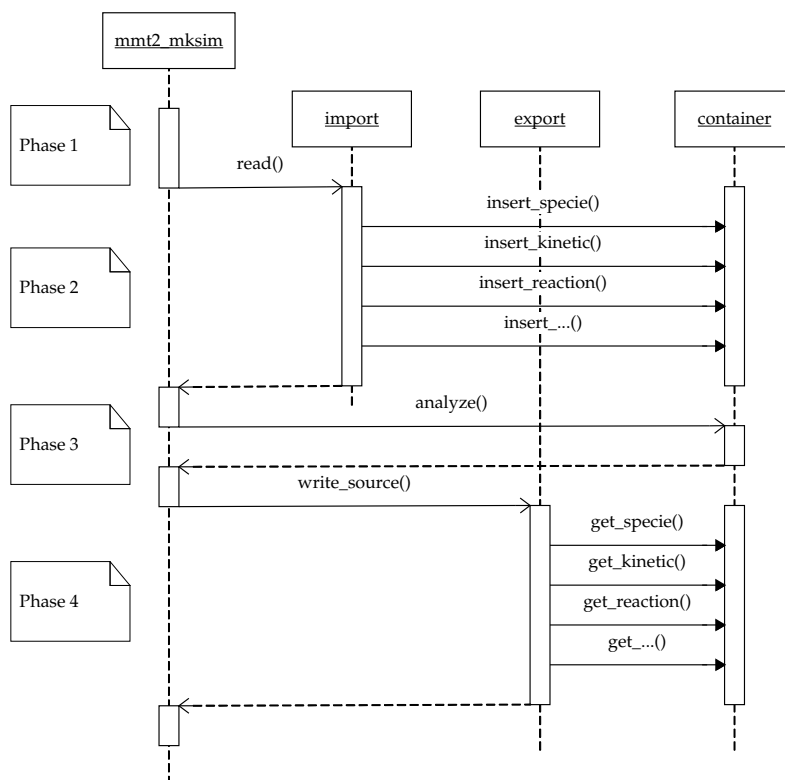


Abbildung 8.4: UML-Sequenzdiagramm für den Ablauf der Source-Code-Erzeugung aus einem Modell im M3L-Format. Phase 1: Kommandozeile parsen, Objekte anlegen, Verzeichnisse anlegen; Phase 2: M3L-Datei parsen und Modellinformation in Containerobjekt eintragen; Phase 3: Modellinformation optimieren; Phase 4: Source-Code erzeugen.

Import/Container/Export bis hin zum Simulator. Insbesondere muss der Container alle Features unterstützen, die auch in M3L spezifiziert werden können. Abb. 8.5 zeigt, wie im Container die Struktur von M3L nachgebildet wird.

8.3. Zusammenstellung von Hilfsprogrammen

.....

Während der Laufzeit dieser Arbeit wurde die Software MMT2 entwickelt und möglichst zeitnah den Projektpartnern, die für die Modellierung zuständig waren, zum Installieren als RPM-Paket (Redhat Package Manager) zugänglich gemacht. Dabei ergab sich einige Male Bedarf an einfachen Hilfsprogrammen, die bei der Modelerstellung und Kodierung im M3L-Format behilflich sind. Mit der Zeit entstanden immer mehr kleine Hilfsprogramme, sodass die Entscheidung getroffen wurde alle in ein Programm Namens `mmt2_converter` zu vereinen. Das Programm besteht einfach aus einem C-Programm, das in der `main()`-Funktion mit der `Getopt`-Bibliothek die Kommandozeile parst und dann in die Hauptroutine des Hilfsprogramms verzweigt.

8.3.1 Konverter für Messdaten im CSV-Format in ein M3L-Fragment

Eines dieser Hilfsprogramme ist ein Konverter von Messdaten im CSV-Format (Comma-Separated-Values) in das von M3L benötigte Format (Kapitel. 7.3). Ein solcher Konverter lässt sich leicht als Scanner mit GNU Flex [Herold, 2003] erstellen. Das benötigte Lex-Programm (Listing 8.1) ist nur wenige Zeilen lang und liefert nach der Übersetzung mit Flex einen Scanner, der CSV einliest und ein M3L-Fragment ausgibt.

8.3.2 Konverter vom Metatool-Format nach M3L

Ein weiteres Hilfsprogramm ist ein Konverter vom Metatool-Format nach M3L für eine schnellere Erstellung eines initialen Modells. Im Metatool-Format werden die Reaktionen im Textformat in einer kurzen Notation angegeben, was das schnelle Eintippen vieler Reaktionen ermöglicht (Listing 8.2) im Gegensatz zum Anlegen aller Reaktionen in einem XML-Editor. Auch dieses Hilfsprogramm konnte mit einem Scanner gelöst werden, dessen Implementierung in einem kurzen Lex-Programm realisiert werden kann.

8.3.3 Konverter zwischen verschiedenen Versionen von M3L

Während der Entwicklung war es öfters nötig das Format M3L zu erweitern oder zu verändern. Wenn es sich um eine Veränderung handelte, war die neue Version meist inkompatibel zu anderen und benötigte einen Konverter, der Modelle in der alten Version in die neue übersetzt. Da es sich meist um kleinere Änderungen handelt, bei denen nur Knoten oder Teilbäume verschoben werden mussten oder Attributnamen geändert werden mussten, war auch wieder nur ein kleines Programm nötig, das mit der Gnome XML Library die M3L-Datei in den Speicher lädt, mit ein paar Befehlen und Schleifen die Änderungen vornimmt und dann wieder in eine Datei schreibt.

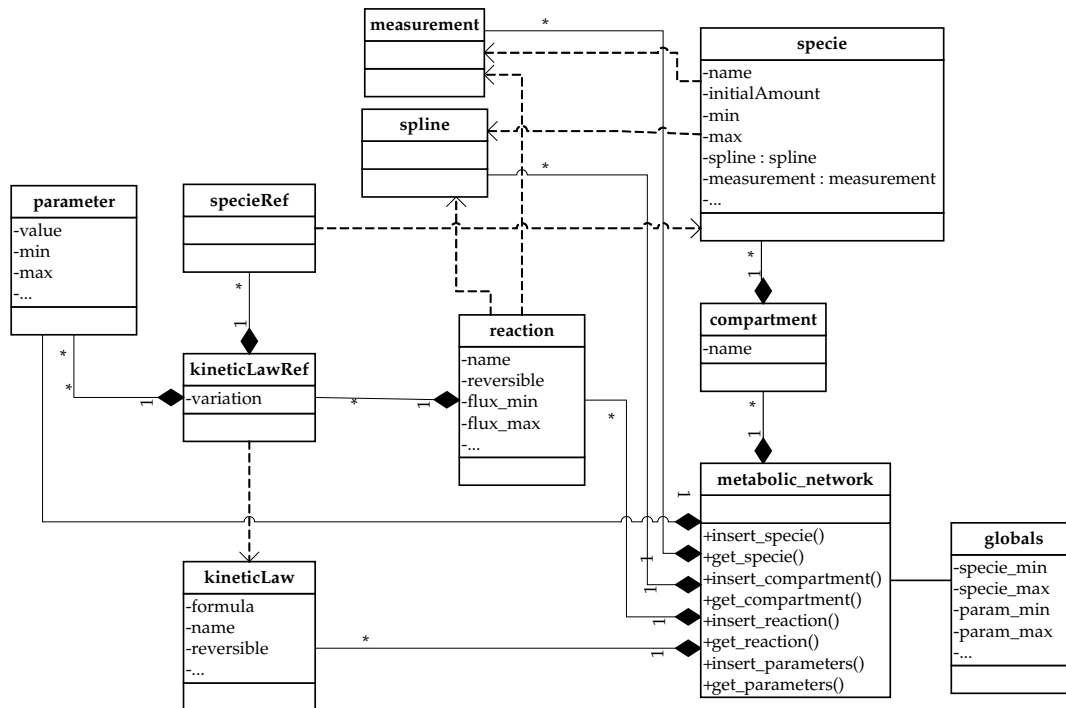


Abbildung 8.5: UML-Klassendiagramm des Containerobjekts aus Abb. 8.4. Der Container muss in der Lage sein, alles zu speichern, was in der M3L-Datei gespeichert ist.

```

1: %{
2:     using namespace std;
3:     #include<string>
4:     string v1, v2, v3;
5:     extern FILE* measure_xml_out;
6:     #define PRINTF_LINE() \
7:     fprintf(measure_xml_out, "\t\t\t<measure
8:     t=\"%s\" value=\"%s\" weight=\"%s\">\n", v1.c_str(),
9:     v2.c_str(), v3.c_str());
10: }%
11: %x tab1 tab2 tab3 tab4 tab5
12: %option noyywrap
13: %%
14: -?[-.eE0-9]+ v1=yytext; BEGIN(tab1);
15: <tab1>[\t\ ;,]+ BEGIN(tab2);
16: <tab2>-?[-.eE0-9]+ v2=yytext; BEGIN(tab3);
17: <tab3>[\t\ ;,]+ BEGIN(tab4);
18: <tab4>-?[-.eE0-9]+ v3=yytext; BEGIN(tab5);
19: <tab5>[\t\ ;,]\n PRINTF_LINE(); BEGIN(INITIAL);
20: %%
  
```

Listing 8.1: Flex-Programm zur Erstellung eines Scanners, der Messdaten im CSV-Format in ein M3L-Fragment konvertiert.

8.3.4 Übernehmen von optimierten Parametersätzen in die M3L-Datei

Während einer Modellstudie werden viele Parameteranpassungen durchgeführt mit dem Ziel bessere Parametersätze zu finden. Die Anzahl der Parameter kann dabei leicht die 100 überschreiten. Damit die Parameter nicht von Hand zurück in das M3L-Modellfile übernommen werden müssen, wurde auch hierfür ein simples Hilfsprogramm geschrieben, das den angepassten Parametervektor im kompakten Format (MMT2 Referenzhandbuch [Haunschild, 2005]) auf der Kommandozeile als Parameter übergeben bekommt und die Werte dann in das M3L-Modellfile einträgt.

Bei diesem Hilfsprogramm musste sehr darauf geachtet werden, dass die Parameterwerte auch in die richtigen Elementknoten geschrieben werden. Die Schwierigkeit ist, dass der Generator aus implementierungstechnischen Gründen die Parameter innerhalb einer Reaktion alphabetisch sortiert. Es ist also nicht einfach damit getan die Werte der Reihe nach in das M3L-Modellfile zu übernehmen. Abb. 8.6 zeigt, wie der Ablauf implementiert ist. Die Einheiten `scanner` und `libxml2` sind im `mmt2_converter` enthalten. `mmt2_sim` ist das Executable, das der Generator erzeugt hat und das die Parameteranpassung durchgeführt hat. Dieses Executable wird dazu verwendet die Parameter im kompakten Format den Parameternamen zuzuordnen.

Der Ablauf in Abb. 8.6 beginnt damit, dass die Kommandozeile von `mmt2_converter` geparkt wird, in der sich die zu setzenden Parameter befinden, und dann das angegebene M3L-Modellfile. Mit dem Aufruf `load_m3l_file()` wird das Modellfile in den Speicher geladen. Anschließend wird der Parametervektor, so wie er von der Kommandozeile gelesen wurde, per Kommandozeilenaufruf dem Simulator-Executable übergeben. Die Standardausgabe von `mmt2_sim` wird per `popen()` auf die Eingabe von `scanner` umgeleitet, der dann die einzelnen Parameterwerte samt zugehörigem Namen, Reaktion und Variante geliefert bekommt. Der `scanner` ist wieder ein Programm wie in Listing 8.1, das die Ausgabe von `mmt2_sim` versteht und für jeden Parameter eine Funktion aufruft, die `libxml2` veranlasst den Parameter im M3L-Modellfile zu ändern.

```

1: -ENZIRREV
2: v1 v2 v3
3:
4: -METINT
5: A B C D
6:
7: -CAT
8: v1: A = B
9: v2: B = C
10: v3: C = D

```

Listing 8.2: Eine Modellbeschreibung im Metatool-Format ist kompakt, beschreibt nur die Modellstruktur und kann schnell in einem Texteditor eingegeben werden. `mmt2_convert` konvertiert eine solche Datei in ein M3L-Gerüst, das man dann mit einem XML-Editor weiterbearbeiten kann. Dieses Beispiel ist aus dem MMT2-Tutorial entnommen.

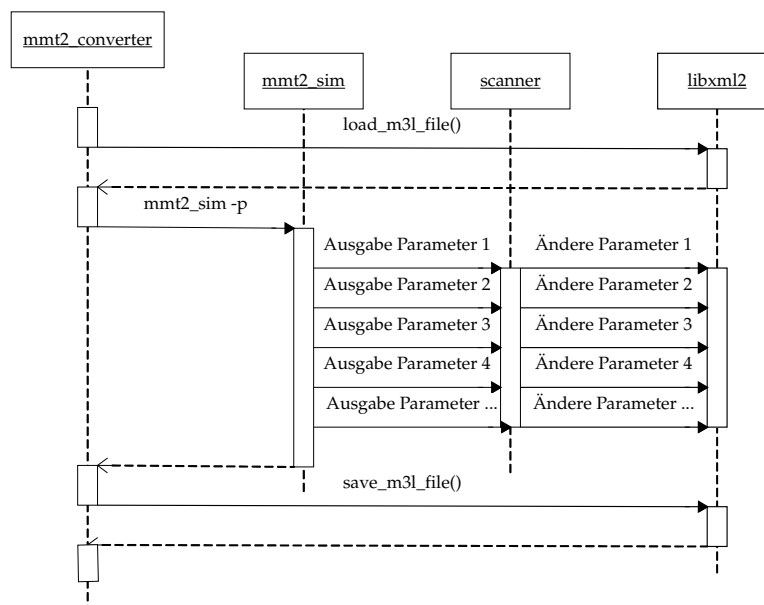


Abbildung 8.6: UML-Sequenzdiagramm des Ablaufs der Übernahme eines Parametervektors im kompakten Format in die zugehörige M3L-Datei.

Numerische Methoden

In der experimentellen Modellbildung wird zunächst ein Modell aufgestellt (Abb. 1.3C), das dann analysiert werden muss. Durch Simulation (Abb. 1.3F) kann zwar das Verhalten des Modells sichtbar gemacht werden, aber eine grundlegende Antwort auf die Frage, ob das Modell gut auf die Daten passt, lässt sich allein mit der Simulation nicht finden. Ein passendes Modell mit falschen Parametern kann ein völlig falsches Verhalten zeigen - mit dem richtigen Parametersatz jedoch würde das Modellverhalten gut mit den Daten übereinstimmen. Gleich nach der Erstellung des Modells und ggf. Plausibilitätsprüfungen wird durch eine oder mehrere Parameteranpassungen (Abb. 1.3I) getestet, ob das Modell an die Daten anpassbar ist. Können die Algorithmen keinen passenden Parametersatz finden, so geht man davon aus, dass das Modell unpassend ist und weitere Veränderungen nötig sind, damit das Modell in der Lage ist das experimentell beobachtete Verhalten zu reproduzieren. In dieser Arbeit wird eine Parameteranpassung unter Nebenbedingungen durchgeführt:

$$\min_{\vec{\alpha} \in \text{Feas}} \text{Crit}(F(\vec{\alpha})) \quad (9.1)$$

In diesem Kapitel werden grundlegende Algorithmen des MMT2-Simulators vorgestellt (Abb. 9.1). Die in Abb. 9.1 mit einem **G** versehenen Blöcke wurden vom Source-Code-Generator so erzeugt, dass sie auf die Modellfamilie spezialisiert sind. Auf der rechten Seite in Abb. 9.1 sind drei Schnittstellen aufgeführt, über die der Benutzer den Simulator bedienen kann.

In Abschnitt 9.1 wird beschrieben, wie die Funktion $F(\vec{\alpha})$ berechnet wird, die die rechte Seite der Modell-ODEs festlegt. Abschnitt 9.2 befasst sich mit dem Bewertungskriterium Crit , das die Qualität der Anpassung an die gemessenen Daten und die Einhaltung von Nebenbedingungen bewertet. Verwendete Algorithmen zur Parameteranpassung $\min_{\vec{\alpha}}$ werden in Abschnitt 9.3 vorgestellt.

Ist ein passender Parametersatz α gefunden worden, so geht es weiter mit der Modellanalyse (Abb. 1.3J) deren grundlegendes Werkzeug die Sensitivitätsanalyse ist mit der sich Abschnitt 9.4 beschäftigt. Teile des Inhalts dieses Kapitels wurden in [Haunschild and Wiechert, 2003] publiziert.

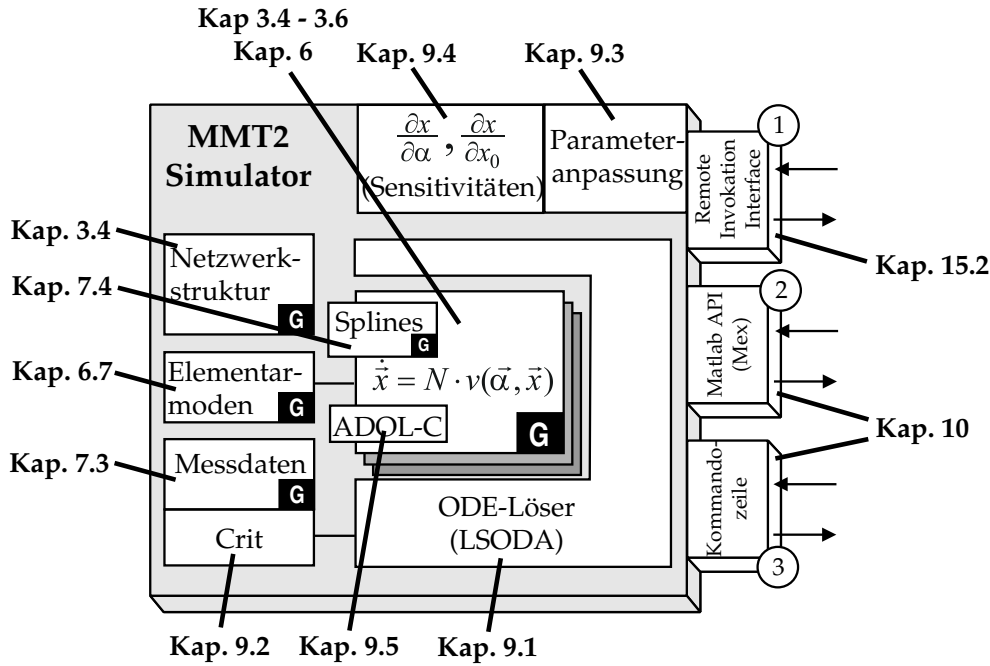


Abbildung 9.1: Architektur des MMT2 Simulators. Durch Source-Code Generierung und Kompilierung ist der erzeugte Simulator hochoptimiert auf die spezifizierte Modellfamilie.

9.1. ODE-Löser

Wie bereits in Kapitel 3 vorgestellt, sieht die Struktur der Modelle, welche die Software aus dem metabolischen Netzwerk erstellt, auf mathematischer Ebene wie folgt aus:

$$\dot{\vec{x}} = N \cdot \vec{v}(\vec{\alpha}, \vec{x}), \quad \vec{x}(0) := \vec{x}_0 \quad (9.2)$$

Die Stoffflüsse v_i des Stoffflussvektors \vec{v} sind üblicherweise nichtlineare Funktionen der Konzentrationen, wodurch eine analytische Lösung unmöglich wird. Daher ist ein numerischer Algorithmus zum Lösen des Systems nötig, der als Eingabe das System von Differentialgleichungen (ODEs) erhält. Die ODEs liefern keine explizite Beschreibung des wahren Verlaufs der Funktion, nur eine implizite, die man als Geschwindigkeitsfeld visualisieren kann (Abb. 9.2A). Ein Prototyp eines numerischen Algorithmus ist das Eulerverfahren (Abb. 9.2B), das ausgehend von einem Punkt im Zustandsraum den Verlauf der ODE-Lösung mit einer Schrittweite Δt linear approximiert und diesen Punkt wieder als Startpunkt nutzt.

$$\mathbf{x}(t + \Delta t) := \mathbf{x}(t) + \Delta t \cdot \dot{\mathbf{x}}(t) \quad (9.3)$$

Eine Verringerung von Δt erhöht die Genauigkeit mit der der wahre Verlauf rekonstruiert wird, erhöht aber auch die Anzahl der Funktionsauswertungen und damit die Rechenzeit. Geschickte Verfahren wie das Runge-Kutta Verfahren approximieren den Verlauf nicht linear, sondern mit einer höheren Ordnung, wodurch der Fehler zwar verringert wird, aber auch wieder mehr Funktionsauswertungen nötig sind. Moderne

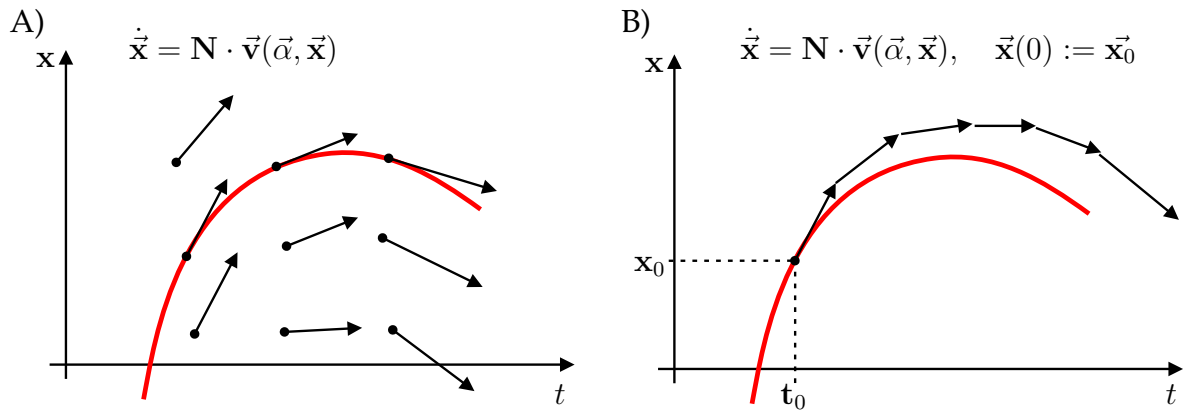


Abbildung 9.2: A) Geschwindigkeitsfeld einer Differentialgleichung. B) Eulerverfahren für zum Lösen einer Differentialgleichung.

Algorithmen sind in der Lage, sowohl die Ordnung als auch die Schrittweite, dynamisch anzupassen, um ein gutes Verhältnis zwischen Rechenzeit und Fehler zu erhalten. Dieses Verhältnis kann man üblicherweise bei den Algorithmen durch Angabe einer relativen und absoluten Toleranz einstellen.

Eine weitere problematische Eigenschaft von Differentialgleichungen ist ihre Steifigkeit, die beschreibt wie weit die langsamste und schnellste Dynamik im System auseinander liegen. Von steifen Systemen redet man, wenn diese Dynamiken um mehr als etwa vier Größenordnungen differieren und die langsame Dynamik das Verhalten beschreibt während die schnelle Dynamik sich im eingeschwungenen Zustand befindet und das System vernachlässigbar gering beeinflusst. Bei einem solchen System würden explizite Algorithmen, wie die Runge-Kutta Verfahren, ihre Schrittweite der schnellen Dynamik anpassen und damit eine sehr hohe Rechenzeit benötigen. Algorithmen, die mit steifen Systemen umgehen können, sind in der Lage die schnellen und langsamen Dynamiken voneinander zu trennen und somit auch solche Systeme in akzeptabler Zeit zu lösen. Ein Prototyp eines solchen Algorithmus ist das implizite Euler-Verfahren:

$$\mathbf{x}(t + \Delta t) := \mathbf{x}(t) + \Delta t \cdot \dot{\mathbf{x}}(t + \Delta t) \tag{9.4}$$

Wegen der guten Erfahrungen im Einsatz mit MMT1 [Hurlebaus et al., 2002] wurde auch für diese Arbeit der frei erhältliche Algorithmus LSODA [Hindmarsh, 1983] gewählt. Dieser Algorithmus löst Probleme der Form

$$\dot{x} = f(t, x) \tag{9.5}$$

LSODA ist eine Kombination des Adams-Verfahrens für nicht-steife ODEs und des BDF-Verfahrens für steife ODEs. Beide Verfahren sind mit automatischer Schrittweiten- und Ordnungssteuerung implementiert und werden vom Algorithmus basierend auf der Steifigkeit der ODEs automatisch ausgewählt.

9.2. Modellbewertung

.....

In dieser Arbeit wird eine Parameteranpassung unter Nebenbedingungen vorgenommen, die dem Optimierungsalgorithmus durch eine Straffunktion mitgeteilt werden. In diesem Abschnitt wird die Berechnung des Bewertungskriteriums (Gleichung 9.1) erläutert, das sich aus mehreren Einzelkriterien zusammensetzt, die im Folgenden vorgestellt werden. Nach der Bestimmung der Einzelkriterien werden sie mit unterschiedlichen Gewichten zusammenaddiert (Abschnitt 9.2.5).

9.2.1 Fehlerquadratsumme

Zur Bewertung, wie gut die Simulation auf die Messdaten passt, wird die Fehlerquadratsumme (FQS) verwendet, die weite Ausreißer härter bestraft als geringe Abweichungen. Mit dieser Eigenschaft bewirkt die FQS im Optimierungsalgorithmus das Verhalten, dass große Abweichungen (Abb. 9.3A, t_a) vorrangig behandelt werden gegenüber kleineren. In der Literatur findet man viele Varianten für die Berechnung der FQS, darunter die gewichtete FQS, die robuste FQS und auch Bayes-Ansätze um die Abweichungen zu quantifizieren. In dieser Arbeit wurde die gewichtete FQS verwendet:

$$\text{FQS} = \sum_{i=1}^n \left(\frac{m_i - s_i}{\sigma_i} \right)^2, \quad \begin{array}{l} m_i \text{ } i\text{-te Messung} \\ s_i \text{ simulierter Wert zur } i\text{-ten Messung} \\ \sigma_i \text{ Standardabweichung der } i\text{-ten Messung} \end{array} \quad (9.6)$$

Bei einem typischen Modellierungsprozess wird nicht nur ein Metabolitpool an die Daten angepasst, sondern mehrere gleichzeitig. Für die einzelnen Metabolitpools existieren aber nicht immer gleich viele Messpunkte. U. U. gibt es einen Messpunkt, für den wesentlich weniger Messdaten zur Verfügung stehen als bei den restlichen. Solche Pools, für die mehr Messdaten zur Verfügung stehen, sollen dann auch höher gewichtet werden. Aus diesem Grunde sind in \vec{m} alle Zeitpunkte aller simulierten Metabolitpools enthalten, zu denen ein Messwert existiert.

9.2.2 Nebenbedingungen für Metabolite und Flüsse

Bevor ein metabolisches Modell simuliert wird, ist dem Modellierer bereits klar, in welchen Größenordnungen sich die einzelnen Metabolite und Reaktionsraten aufhalten müssen. Dieses Wissen bezieht er entweder aus Überlegungen zur Plausibilität, aus den Messdaten oder der Literatur. Bewegt sich der Zeitverlauf des Modells nicht innerhalb dieser Grenzen, so fällt das Modell durch die Plausibilitätsprüfung. Diese Nebenbedingung, die noch wichtiger ist als die Fehlerquadratsumme, wird durch ein entsprechend hohes Gewicht g_2 und g_3 (Gleichung 9.11) priorisiert.

Die Nebenbedingung lautet, dass sich die Konzentrationen der Metabolitpools in einem bestimmten Fenster $x_{min} < c < x_{max}$ befinden sollen. Dies wird über die Penalty-Funktion P_m realisiert, die so gestaltet sein muss, dass ihr Einfluss $Crit$ dominiert (Glei-

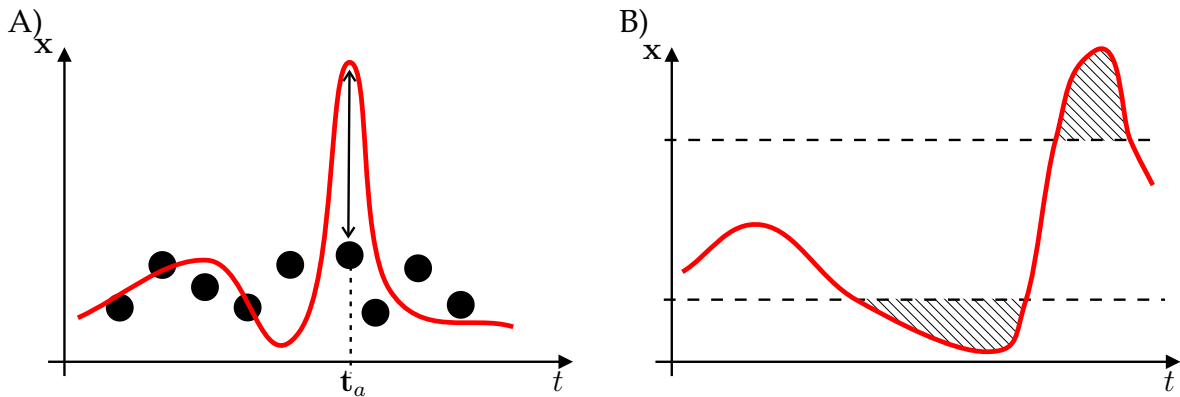


Abbildung 9.3: A) Vorgehensweise Optimierer B) Straffunktion P_p für Parameterwerte

chung 9.11), solange die Nebenbedingung nicht erfüllt ist. Weiterhin muss sie stetig abnehmen, wenn sich der Optimierer in die richtige Richtung bewegt. Sei $\mathbf{x}(t, \vec{\alpha})$ der Wert der ODE-Lösung an der Stelle t . Überschreitet der Zeitverlauf des Modells das Konzentrationslimit, so entsteht zwischen dem Limit und dem Zeitverlauf des Modells eine Fläche, die in P_p aufintegriert wird.

$$P_m = \int_{t_0}^{t_{end}} G_m(t) dt, \quad \text{mit} \quad G_m(t) = \begin{cases} \mathbf{x}_{max} - \mathbf{x}(t, \vec{\alpha}) & \text{falls } \mathbf{x}(t, \vec{\alpha}) < \mathbf{x}_{max} \\ \mathbf{x}(t, \vec{\alpha}) - \mathbf{x}_{min} & \text{falls } \mathbf{x}(t, \vec{\alpha}) > \mathbf{x}_{min} \\ 0 & \text{sonst} \end{cases} \quad (9.7)$$

Analog zu P_m soll es auch eine Nebenbedingung für die Reaktionsraten geben, die mit der Penalty-Funktion P_r realisiert wurde. Analog zu $\mathbf{x}(t, \vec{\alpha})$ gibt die Funktion $\mathbf{v}(\mathbf{x}, \vec{\alpha})$ den Verlauf der Reaktionsraten an.

$$P_r = \int_{t_0}^{t_{end}} G_f(t) dt, \quad \text{mit} \quad G_f(t) = \begin{cases} \mathbf{v}_{max} - \mathbf{v}(\mathbf{x}, \vec{\alpha}) & \text{falls } \mathbf{v}(\mathbf{x}, \vec{\alpha}) < \mathbf{v}_{max} \\ \mathbf{v}(\mathbf{x}, \vec{\alpha}) - \mathbf{v}_{min} & \text{falls } \mathbf{v}(\mathbf{x}, \vec{\alpha}) > \mathbf{v}_{min} \\ 0 & \text{sonst} \end{cases} \quad (9.8)$$

9.2.3 Nebenbedingungen für Parameterwerte

Genauso wie die Reaktionsraten und die Konzentrationsverläufe der Metabolite innerhalb vernünftiger Grenzen verlaufen müssen, so müssen auch die Parameter des Modells innerhalb vernünftiger Grenzen liegen.

- Wenn es sich um ein mechanistisches Modell handelt, so ist mit jedem Parameter eine biologische Bedeutung verknüpft, die keinen Sinn ergibt, wenn der Parameter beispielsweise negativ oder exorbitant groß ist.
- Der Wert eines Parameters kann beeinflussen wie gut der ODE-Löser mit dem ODE-System zurecht kommt und wie lange das Lösen dauert. Abb. 9.4B zeigt ein Beispiel in dem die Parameter α_1 und α_2 die Dynamik des Systems bestimmen

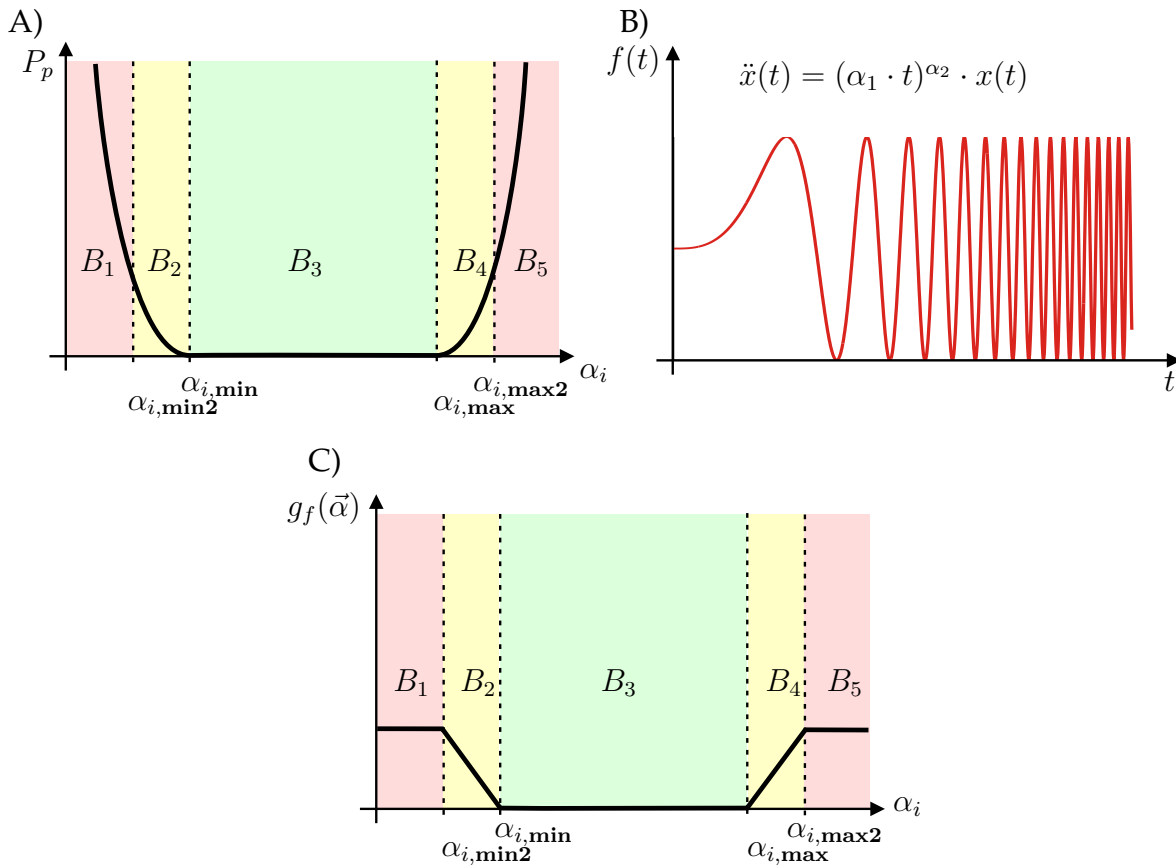


Abbildung 9.4: A) Verhalten der Straffunktion P_p in Abhängigkeit der Entfernung zum gültigen Parameterbereich. B) Beispiel für einen Parameter, der sich bei steigendem Wert nicht gutartig verhält. C) Verhalten der Straffunktion $g_f(\vec{\alpha})$.

und damit auch die Zeit, die der ODE-Löser für das Lösen braucht. Des Weiteren kann es sein, dass durch ungünstige Parameterwerte das ODE-System zu steif wird oder Singularitäten auftreten, womit das System dann überhaupt nicht mehr lösbar ist.

- Viele Parameter sind linear voneinander abhängig, wodurch man Parameter eliminieren könnte. Durch die Beschränkung der Parameter auf einen Bereich kann es aber sein, dass sich eine Kombination von vernünftigen Werten durch die Parameteranpassung einstellt.

Viele Optimierungsalgorithmen unterstützen die zusätzliche Angabe von Grenzen für Parameter, manche aber nicht. Damit diese Algorithmen dennoch genutzt werden können, müssen die Grenzen über eine Penalty-Funktion P_p definiert werden. Sie hat die Aufgabe am Anfang den Optimierer in den gültigen Parameterbereich zu führen und während der Parameteranpassung dafür zu sorgen, dass der gültige Bereich nicht verlassen wird.

Wegen der Problematik, die in Abb. 9.4B veranschaulicht wurde, muss der Optimierer zunächst in einen Bereich valider Parameterwerte gesteuert werden, bevor die Simulation erlaubt wird. Abb. 9.4A+C zeigen die möglichen Bereiche, in der sich ein

Parameter befinden kann. In den Bereichen B_1 und B_5 ist der Parameter so weit im verbotenen Bereich, dass die Penaltyfunktion P_p den Qualitätswert in diesem Bereich dominiert, womit die Bewertung der Simulation vernachlässigbar gering ist und weggelassen werden kann.

Befinden sich alle Parameter im gültigen Bereich B_3 so ist $P_p = 0$ und die Penaltyfunktion hat ihre erste Aufgabe erledigt, den Optimierer in den gültigen Parameterbereich zu leiten. Passiert es, dass der Optimierer den Bereich B_3 verlassen möchte, so muss die Penaltyfunktion den Weg sanft in Richtung gültigen Bereich zurückweisen. Die Bereiche B_2 und B_4 haben daher sowohl die Aufgabe, die Penaltyfunktion dort ansteigen zu lassen, als auch die Bereiche B_1 und B_5 , in denen nicht simuliert wird und der Qualitätswert keinen Anteil an der Bewertung zur Simulation hat, mit denen zu verbinden (B_2 und B_4), in denen nur die Simulation in den Qualitätswert eingeht. Zur Behandlung dieser Bereiche wird die Schalterfunktion $g_f(\vec{\alpha})$ eingeführt, die wie folgt definiert ist:

$$g_f(\vec{\alpha}) = \begin{cases} 1, & \text{falls } \alpha_i < \alpha_{i,min2} & (\text{Abb. 9.4C, } B_1) \\ \frac{\alpha_i - \alpha_{i,min}}{\alpha_{i,min} - \alpha_{i,min2}}, & \text{falls } \alpha_{i,min2} \leq \alpha_i < \alpha_{i,min} & (\text{Abb. 9.4C, } B_2) \\ 0, & \text{falls } \alpha_{i,min} \leq \alpha_i < \alpha_{i,max} & (\text{Abb. 9.4C, } B_3) \\ \frac{\alpha_i - \alpha_{i,max}}{\alpha_{i,max2} - \alpha_{i,max}}, & \text{falls } \alpha_{i,max} \leq \alpha_i < \alpha_{i,max2} & (\text{Abb. 9.4C, } B_4) \\ 1, & \text{falls } \alpha_i > \alpha_{i,max2} & (\text{Abb. 9.4C, } B_5) \end{cases} \quad (9.9)$$

Für den Optimierer ist es sehr wichtig, dass die resultierende Qualitätsfunktion Crit_2 an keiner Stelle einen Sprung nach unten macht (z. B. durch Abschalten des Anteils Crit_2), da dies sonst ein lokales Minimum darstellen kann, in dem sich der Optimierer verfängt. Die Qualitätsfunktion, die die Penaltyfunktion P_p mit einbezieht, braucht für $g_f(\vec{\alpha}) = 1$ nicht mehr die Simulation durchzuführen.

Die Penaltyfunktion P_p selber muss nur eine Funktion sein, die schnell genug ansteigt und damit auch Berge im Bereich B_3 überragt. Es darf nicht vorkommen, dass in den Bereichen B_2 und B_4 durch das lineare Überblenden von Crit_2 auf P_p ein künstliches lokales Minimum entsteht.

$$P_p = \begin{cases} \left(\frac{\alpha_i - \alpha_{i,min}}{\alpha_{i,min} - \alpha_{i,min2}} \cdot g_{p1} \right)^{g_{p2}} & \text{falls } \alpha_i < \alpha_{i,min} & (\text{Abb. 9.4A, } B_2) \\ 0 & \text{falls } \alpha_{i,max} \geq \alpha_i \geq \alpha_{i,min} \\ \left(\frac{\alpha_i - \alpha_{i,max}}{\alpha_{i,max2} - \alpha_{i,max}} \cdot g_{p1} \right)^{g_{p2}} & \text{falls } \alpha_{i,max} < \alpha_i \end{cases} \quad (9.10)$$

9.2.4 Parameterwahl

Die in den letzten Abschnitten eingeführten Gewichte sollen das Verhalten des Optimierers so beeinflussen, dass mit höherer Priorität die Einhaltung der Nebenbedingungen sichergestellt wird. Die Gewichte g_1 , g_2 und g_3 werden dafür so gewählt, dass der Optimierer zunächst einen Parametersatz findet, so dass gilt $P_m=0$ und $P_f=0$. In der Implementierung von MMT2 wurde gewählt $g_1=1$, $g_2=g_3=10^6$.

Im Falle der Verwendung von Crit_2 als Kriterium für Optimierer, die keine Nebenbedingungen unterstützen, sorgt die Funktion $g_f(\vec{\alpha})$ zusammen mit den Parametern g_{p_1} und g_{p_2} , dass der Optimierer die Priorität auf die Einhaltung der Nebenbedingungen $\alpha_{i,\min} < \alpha_i < \alpha_{i,\max}, i = 1 \dots n$ setzt. Im Einsatz von MMT2 haben sich die Einstellungen $g_{p_1}=1000$ und $g_{p_2}=4$ bewährt.

9.2.5 Bewertungskriterium

Nach der Definition der Teilbewertungskriterien wird das Gesamt-Bewertungskriterium Crit wie folgt definiert:

$$\text{Crit}_1 = (g_1 \cdot \text{FQS} + g_2 \cdot P_m + g_3 \cdot P_f) \quad (9.11)$$

Mit Gewichten g_1, g_2 und g_3 für die Fehlerquadratsumme, die Strafffunktion für die Nebenbedingungen für Metabolite P_m und die Strafffunktion für die Nebenbedingungen für Flüsse P_f . Das Kriterium Crit_1 bewertet die Qualität der Simulation, und kann nur dann eingesetzt werden, wenn der Optimierungsalgorithmus Nebenbedingungen für Parameter unterstützt. Andernfalls wird Crit_1 entsprechend erweitert:

$$\text{Crit}_2 = (1 - g_f(\vec{\alpha})) \cdot \text{Crit}_1 + g_f(\vec{\alpha}) \cdot P_p, \quad \text{mit } g_f \mapsto [0, 1]. \quad (9.12)$$

Die Funktion $g_f(\vec{\alpha})$ hat die Aufgabe den Einfluss von Crit_1 langsam auszublenden, je weiter der Parametervektor $\vec{\alpha}$ im verbotenen Bereich ist und den Optimierungsalgorithmus zunächst dazu zu bewegen, einen gültigen Parameterbereich zu suchen. Der verbotene Parameterbereich beschreibt hier allerdings nicht die Situation, dass eine Masse oder eine Quantität negativ wird, sondern lediglich benutzerdefinierte Einschränkungen.

9.3. Parameteranpassung

.....

Dieser Abschnitt befasst sich mit der Berechnung von $\min_{\vec{\alpha}}$ aus der Gleichung 9.1, die man auch Parameteranpassung oder Quadratsummenminimierung nennt. Die gängige Vorgehensweise der Optimierungsverfahren ist es, eine Folge von Punkten im Parameterraum zu erzeugen, die gegen ein lokales oder globales Minimum konvergiert. Im Allgemeinen kann man aber nicht davon ausgehen, dass die Folge das Minimum erreicht. Ganz besonders wenn es sich um hochdimensionale Probleme handelt, ist das Streben, das globale Optimum zu erreichen, utopisch. Diese Erkenntnis teilt die Verfahren in zwei Kategorien ein:

Die lokalen Optimierungsverfahren zeichnen sich dadurch aus, dass sie von einem gegebenen Startpunkt im Parameterraum eines der nahe gelegenen lokalen Minima suchen. Die globalen Optimierungsverfahren sind dagegen so ausgerichtet, dass sie von einem beliebigen Startpunkt das globale Optimum suchen, ohne Garantie dieses in endlicher Zeit zu erreichen. Dies ist auch gleichzeitig ein großer Nachteil dieser Verfahren. Man kann zu keinem Zeitpunkt eine definitive Aussage darüber machen, ob

das beste gefundene Minimum auch das globale Minimum ist. Daher kann das Abbruchkriterium nur empirisch festgelegt werden. In den folgenden Abschnitten werden die Verfahren kurz vorgestellt, die in MMT2 implementiert wurden.

9.3.1 Subplex-Verfahren

Die Subplex-Methode gehört zu den lokalen Optimierungsverfahren und ist eine Weiterentwicklung der NMS-Methode (Nelder-Mead-Simplex), die ein Optimierungsalgorithmus für Probleme ohne Nebenbedingungen ist, bei denen nur Funktionsauswertungen und keine Parameterableitungen benötigt werden. NMS verwendet einen Simplex, der definiert ist als die konvexe Hülle von $n + 1$ Punkten in einem n -dimensionalen Raum. Dieser Simplex bewegt sich durch die Optimierungslandschaft und verändert dabei automatisch seine Form und Größe. In [Rowan, 1990] wird der Simplex-Algorithmus im Detail erklärt und folgende Charakteristika werden aufgezählt:

1. Im Vergleich zu anderen Methoden kommt NMS gut mit verrauschten Funktionen zurecht.
2. Die Effizienz von NMS ist abhängig von der Problemgröße n . Bei kleiner Dimension ($n < 5$) ist sie hoch, wird aber bei größeren n ineffizient.
3. Der benötigte Speicherplatz von NMS beträgt wegen des Simplex mit $n+1$ Kanten, die jeweils durch einen Vektor der Länge n dargestellt werden, $O(n^2)$.

Die Subplex-Methode erhält Eigenschaft 1 und kompensiert die Eigenschaften 2 und 3 durch Zerlegung des Suchraum R^n in Subräume $R_i^{n_i}$. In jedem dieser Subräume $R_i^{n_i}$ wird mit dem NMS-Verfahren nach dem Minimum gesucht, was durch die niedrigere Dimension wieder effizienter erfolgt. Am Ende eines Iterationsschrittes setzt sich der Parametervektor aus den Einzelvektoren jedes Subraums zusammen. Die Subräume werden in jedem Iterationsschritt so ausgewählt, dass die Parameter mit der höchsten Sensitivität auf das Ergebnis in einem Subraum liegen.

Im Einsatz von MMT2 wurden genauso wie bereits in MMT1 gute Erfahrungen mit dem Subplex-Verfahren gemacht. Bei manchen Optimierungsproblemen kam es allerdings vor, dass die Konvergenzgeschwindigkeit sehr gering war. Startete man das Verfahren von einem anderen Startpunkt, so war die Konvergenzgeschwindigkeit wesentlich schneller. Auch während der Optimierung wurden lokale Minima gefunden, die nicht zufrieden stellend waren, so dass man sich wünschte, dass das Verfahren nach anderen Minima weitersucht. Das konnte dadurch erreicht werden, dass der Parametervektor des lokalen Minimums durch Zufallszahlen leicht verändert wurde und das Verfahren von diesem Punkt neu gestartet wurde. Dies war auch die Motivation dafür dieses Vorgehen zu automatisieren, was zu dem Verfahren führte, das im nächsten Abschnitt beschrieben wird.

9.3.2 Erweiterung des Subplex-Verfahrens

Um dieses lokale Optimierungsverfahren in eine Art globales Optimierungsverfahren zu überführen wurde das Verfahren erweitert. Eine der Beobachtungen beim Ein-

satz von Subplex war, dass die Konvergenzgeschwindigkeit maßgeblich von dem gewählten Startpunkt im Parameterraum abhängt. Gerät der Optimierer gleich zu Anfang in ein lokales Minimum, oder in einen Bereich in dem eine schnelle Konvergenz erschwert wird, so ist das Ergebnis u. U. unbefriedigend. Mit der Absicht die Konvergenzgeschwindigkeit des Subplex zu beschleunigen wurde das Verfahren wie folgt erweitert:

1. Initialisiere den Parametervektor α_{cur} mit dem Default-Parametersatz $\alpha_{default}$.
2. Erzeuge mit einem Zufallsgenerator n Punkte im Parameterraum. Diese Punkte werden mit einem maximalen Radius r_n um den Parametervektor α_{cur} gestreut, beim ersten Durchlauf mit dem Radius r_0 . Die Streuweite für r_0 kann auch als unendlich gewählt werden, wodurch nur noch zufällige Werte im Parameterraum erzeugt werden, die nicht mehr mit α_d korreliert sind.
3. Berechne für alle diese Punkte das Bewertungskriterium $Crit$ und wähle den besten Punkt $P_{default}$ aus.
4. Wähle den Punkt $P_{default}$ als Startpunkt für das Subplex-Verfahren und lasse den Algorithmus solange laufen, bis die Funktion n mal aufgerufen wurde oder das Verfahren konvergiert ist.
5. Speichere den erreichten Punkt im Parameterraum in α_{cur} und fahre fort mit Punkt 2.

Mit dieser Vorgehensweise wird zum Einen die Wahl des Startpunktes verbessert. Zum Anderen wird versucht beim Erreichen eines lokalen Minimums oder einer zu langsamen Konvergenzgeschwindigkeit den Subplex-Algorithmus aus diesem Gebiet herauszuführen. Dieses Verfahren wurde fast ausschließlich für Parameteranpassungen verwendet. Es hatte bei den Optimierungsproblemen, die sich in den Anwendungen ergaben (Kap. 11) stets die höchste Konvergenzgeschwindigkeit.

9.3.3 Evolutionsstrategie (ES)

Die Evolutionsstrategie ist ein Optimierungsverfahren, dass sich an der Evolutionstheorie orientiert und deren grundlegenden Mechanismen Mutation, Selektion und Vererbung auf allgemeine Probleme abbildet. Die ES ist aber mehr ein Konzept als ein feststehender Algorithmus. Es wird im Folgenden kurz skizziert:

1. Initalisierung einer ersten Generation ($i = 0$) von μ Eltern, d. h. μ Parametersätze werden erzeugt.
2. Die Eltern erzeugen λ Nachkommen. Die Mischungszahl ρ gibt an, wie viele Eltern ihre Eigenschaften bei der Erzeugung eines Nachkommens weiter vererben. Durch die Wahl der Eltern kann die Konvergenzgeschwindigkeit beeinflusst werden. Neben der standardmäßigen Strategie die Partner stochastisch auszuwählen wäre denkbar, dass einer der Partner immer aus der n % Elite stammt (z. B. $n = 20$) oder dass die Hälfte der Eltern aus der n % Elite stammen.

3. Nach der Erzeugung der Nachkommen, d. h. der Rekombination der Eigenschaften der ρ beteiligten Eltern werden die Nachkommen mutiert. Der Parametervektor wird in diesem Schritt durch Zufallszahlen leicht verändert.
4. Alle Individuen der Generation werden durch eine Fitnessfunktion bewertet. D. h. der Parametersatz, den das Individuum repräsentiert, wird durch das Bewertungskriterium $Crit$ ausgewertet.
5. In die nächste Generation ($i + 1$) werden μ Individuen übernommen, die restlichen sterben. Bei der (μ, ρ, λ) -ES stehen nur die erzeugten Kinder zur Auswahl; bei der $(\mu + \rho, \lambda)$ -ES sowohl Kinder als auch Eltern und bei der $(\mu/\rho, \lambda)$ -ES werden in der neuen Generation ($i + 1$) die Eltern durch die besten Vertreter der jeweiligen Nachkommengruppen ersetzt.
6. Anschließend wird das Abbruchkriterium überprüft, d. h. ob ein Individuum die zu erreichende untere Fehlerschranke unterschritten hat, oder die maximale Anzahl der Generationen erreicht ist.
7. Falls das Abbruchkriterium nicht erfüllt ist, weiter bei Punkt 2.

In MMT2 wird eine $(\mu/\rho, \lambda)$ -ES eingesetzt, die im Rahmen von [Weitzel, 2003] implementiert wurde. Im Einsatz mit MMT2 wurde die Erfahrung gemacht, dass das Verfahren meist nach einer längeren Anlaufphase eine relativ hohe Konvergenzgeschwindigkeit erreicht, die zu einem lokalen Minimum führt, die in der Größenordnung liegt, die auch das erweiterte Subplex-Verfahren gefunden hat. In den meisten Fällen hatte zeitlich gesehen das erweiterte Subplex-Verfahren immer die Nase vorn und fand auch geringfügig bessere Optima.

9.3.4 BFGS Verfahren

Im Gegensatz zum BFGS-Verfahren, das auch zu den lokalen Optimierungsverfahren zählt, arbeiten alle anderen hier aufgeführten Verfahren ableitungsfrei. BFGS [Zhu et al., 1997] ist ein Quasi-Newton-Verfahren für Probleme ohne Nebenbedingungen und stützt sich auf das Newton-Verfahren, dass wie folgt definiert ist:

$$x_{k+1} = x_k - \mathbf{H}^{-1}(x_k) \nabla f(x_k) \tag{9.13}$$

Ist die Berechnung der inversen Hesse-Matrix \mathbf{H}^{-1} nicht möglich oder zu zeitintensiv, so schafft das Quasi-Newton-Verfahren Abhilfe, bei dem die Berechnung von \mathbf{H}^{-1} durch eine Approximation ersetzt wird. Eine der Approximationen ist die Broyden-Fletcher-Goldfarb-Shanno (BFGS) Formel, nach der das Verfahren benannt ist.

In MMT2 wird der L-BFGS-B Algorithmus aus der Netlib eingesetzt. Dieser Algorithmus benötigt neben dem Funktionswert f an der Stelle x_k auch den Gradienten ∇f , dessen Berechnung in Abschnitt 9.4 beschrieben wird. Im Einsatz mit MMT2 findet der Algorithmus bei manchen Problemen bei günstig gewählten Startpunkten ein brauchbares Minimum. Im Vergleich zum Subplex-Verfahren schneidet er aber eigentlich immer schlechter ab. Auch der Versuch den Algorithmus dafür einzusetzen lokale Minima, die durch einen der anderen Algorithmen gefunden wurden, weiter zu verbessern, war nicht erfolgreich.

9.3.5 Problematik der Parameteranpassung

Die u. U. mehrfach durchgeführte Parameteranpassung eines oder mehrerer Modelle soll Erkenntnisse darüber liefern, ob ein Modell überhaupt geeignet ist die Daten zu reproduzieren. Gelingt es nicht die Parameter eines Modells anzupassen, so kann dies mehrere Gründe haben:

1. **Underfitting:** Das Modell ist zu einfach oder eingeschränkt und kann deshalb nicht angepasst werden. Dies kann auch daran liegen, dass die Nebenbedingungen (Abschnitte 9.2.2 und 9.2.3) zu restriktiv eingestellt sind.
2. **Overfitting:** Kommen mehrere Parameteranpassungen desselben Modells zu einer zufrieden stellenden Qualität, aber zu unterschiedlichen Parametersätzen, so ist das Modell überangepasst. Dies kann daher kommen, dass das Modell zu viele Parameter besitzt, oder dass die Nebenbedingungen zu locker eingestellt sind. Eine Möglichkeit dem Problem entgegenzuwirken ist, Parameter mit niedriger Sensitivität (Abschnitt 9.4) von der Anpassung auszuschließen.
3. **Falsches Modell:** Der häufigste Fall ist, dass das Modell nicht angepasst werden kann, obwohl die Punkte 1 und 2 nicht zutreffen. Das Modell ist falsch und ist somit nicht in der Lage das System korrekt zu beschreiben.
4. **Zu hohe Komplexität:** Schließlich kann es noch sein, dass Punkte 1, 2 und 3 nicht zutreffen. Das Modell beschreibt das System korrekt und trotzdem wird kein passender Parametersatz gefunden. Dies kann daran liegen, dass entweder der Parameteranpassungsprozess nicht genug Zeit bekommen hat, ein für das Problem ungünstiges Verfahren gewählt wurde, oder die Startpunkte ungünstig waren.

In Kapitel 11 werden drei experimentelle Arbeiten vorgestellt, bei denen MMT2 zum Einsatz kam. Dort ist die Komplexität der Modelle meist so hoch, dass sich die Frage stellt, ob das Modell einfach nur falsch ist (Punkt 3) oder ob die Parameteranpassung durch die Optimierungslandschaft nicht in der Lage ist zu einem zufrieden stellenden Minimum vorzudringen (Punkt 4).

Das Fehlschlagen der Parameteranpassung muss aber nicht unbedingt daran liegen, dass das Optimierungsverfahren, oder der gewählte Startpunkt ungünstig gewählt sind. Es kann auch daran liegen, dass das Bewertungskriterium Crit (Abschnitt 9.2) ungünstig konstruiert wurde und eine Optimierungslandschaft erzeugt, die für ein Optimierungsverfahren nur schwer zu bewältigen ist.

9.4. Sensitivitätsanalyse

.....

Im Prozess der experimentellen Modellbildung (Abb. 1.3) spielt die statistische Analyse eine große Rolle bei der Auswertung von Simulationen. Auf der Berechnung der Sensitivitäten basieren weitere Analysemethoden wie z. B. Parameteranpassung, Ana-

lyse der Parameterbestimmtheit und Stabilitätsanalyse. Gleichung 9.14 zeigt die dazu üblicherweise benötigten Formen von Sensitivitäten:

$$\begin{aligned} \frac{\partial \vec{x}}{\partial \vec{\alpha}}(t), & \text{ Parametersensitivitäten} \\ \frac{\partial \vec{x}}{\partial \vec{x}_0}(t), & \text{ Startwertsensitivitäten} \\ \frac{\partial \dot{\vec{x}}}{\partial \vec{x}}(t), & \text{ lokale Sensitivitäten} \end{aligned} \tag{9.14}$$

Es gibt verschiedene Methoden wie diese Sensitivitäten berechnet werden können. Die zwei bekanntesten sind die Methode der finiten Differenzen und die Methode der Sensitivitätsgleichungen, die am Beispiel der Parametersensitivitäten vorgestellt werden sollen.

9.4.1 Finite Differenzen

Sei $\vec{x}(t, \vec{\alpha}, \vec{x}_0)$ die Lösung von Gleichung 9.2 mit dem Parametervektor $\vec{\alpha}$ und den Startwerten \vec{x}_0 . Die Methode der finiten Differenzen mit Schrittweitenparameter $\Delta\alpha_i$ liefert eine Approximation um die partielle Ableitung $\frac{\partial \vec{x}}{\partial \alpha_i}$ zu berechnen.

$$s_i(t) \approx \frac{\vec{x}(t, (\alpha_0, \dots, \alpha_i(1 + \Delta\alpha_i), \dots, \alpha_n), \vec{x}_0) - \vec{x}(t, (\alpha_0, \dots, \alpha_i, \dots, \alpha_n), \vec{x}_0)}{\alpha_i \cdot \Delta\alpha_i} \tag{9.15}$$

Abb. 9.5B zeigt zwei Vorgehensweisen bei der Bestimmung der ersten Ableitung. Für die Berechnung der normalen finiten Differenz (FD nach Gl. 9.15) im Punkt t wird die Steigung zwischen den Werten t und $t + \Delta\alpha_i$ bestimmt. Bei der symmetrischen FD wird die Steigung zwischen den Punkten $t - \Delta\alpha_i$ und $t + \Delta\alpha_i$ bestimmt, wodurch das Ergebnis bei gleicher Schrittweite um eine Fehlerordnung genauer ist. Diese Methode und Methoden höherer Ordnung zeigen zwar ein besseres Konvergenzverhalten, jedoch auf Kosten weiterer Funktionsauswertungen. So kann bei der Berechnung der unzentrierten FD der erste Wert aller Parameter mit einer einzigen Referenzsimulation durchgeführt werden was bei der Methode der symmetrischen FD nicht möglich ist und die Komplexität auf mindestens $O(2 \cdot \dim(\vec{\alpha}))$ erhöht.

Nach dieser Referenzsimulation $\vec{x}(t, \vec{\alpha}, \vec{x}_0)$, die den ersten Punkt für alle Parameter liefert, muss dann für den zweiten Punkt eine Simulation pro Parameter durchgeführt werden. Der Aufwand ist dafür nur $O(n + 1)$, $n = \dim(\vec{\alpha})$, jedoch hängt die Genauigkeit der Resultate von der Wahl von $\Delta\alpha_i$ ab. Wählt man $\Delta\alpha_i$ zu groß so dominiert der Schrittweitenfehler (Abb. 9.5A). Theoretisch gilt $\lim_{\delta \rightarrow 0} s_i = \frac{\partial \vec{x}}{\partial \alpha_i}$, aber in der Praxis führen zu kleine Werte von $\Delta\alpha_i$ in der Größenordnung der numerischen Genauigkeit zu Rundungsfehlern, die das Ergebnis verfälschen.

9.4.2 Sensitivitätsgleichungen

Bei dieser Methode werden zunächst die Modell-ODEs (Gleichung 9.2) differenziert. Durch Ausnutzen des Wissens über die zugrunde liegende Modellstruktur, dass sich hauptsächlich in der stöchiometrischen Matrix N befindet, lässt sich der Teil, der differenziert werden muss, minimieren. Die Sensitivitätsgleichungen in Gleichung 9.16

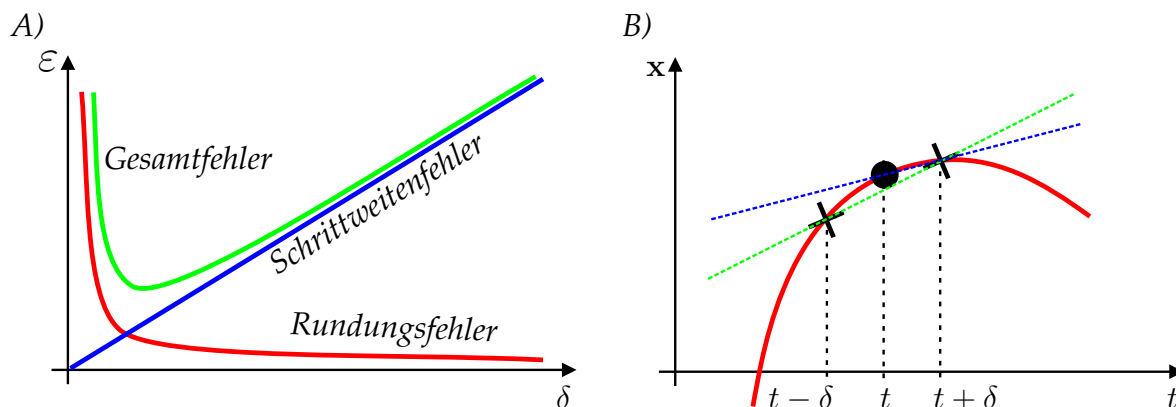


Abbildung 9.5: A) Typischer Verlauf des Gesamtfehlers in Abhängigkeit der Schrittweite δ . B) Verfahren zur Bestimmung der ersten Ableitung. Finite Differenz und die symmetrisch-finite Differenz.

werden in [Hurlebaus et al., 2002] hergeleitet. Zu beachten ist, dass die linke Seite matrixwertig ist und für jeden Modellparameter $\dim(\vec{v})$ ODEs existieren, die den Einfluss des Parameters auf einen bestimmten Metabolit beschreiben. Somit entstehen $\dim(\vec{\alpha}) \times \dim(\vec{v})$ ODEs, die durch den ODE-Löser gelöst werden müssen.

$$\begin{aligned} (\mathbf{D}_{\vec{\alpha}} \dot{\vec{x}}) &= \mathbf{N} \cdot \left(\frac{\partial \vec{v}}{\partial \vec{\alpha}} + \frac{\partial \vec{v}}{\partial \vec{x}} \cdot (\mathbf{D}_{\vec{\alpha}} \vec{x}) \right), \text{ mit } \mathbf{D} \text{ als totales Differential.} \\ (\mathbf{D}_{\vec{x}_0} \dot{\vec{x}}) &= \mathbf{N} \cdot \left(\frac{\partial \vec{v}}{\partial \vec{x}} \cdot (\mathbf{D}_{\vec{x}_0} \vec{x}) \right) \end{aligned} \quad (9.16)$$

Im Gegensatz zur FD-Methode gibt es hier kein Genauigkeitsproblem, was man aber mit einem wesentlich höheren Implementierungsaufwand bezahlen muss. Eine mögliche Implementierungsvariante wäre die Sensitivitätsgleichungen 9.16 mit einer computeralgebraischen Software aufzustellen und die erzeugten Gleichungen in C-Quelltext zu speichern, zu kompilieren und während der Laufzeit durch einen ODE-Löser zu lösen. Der Vorteil sind schnelle Funktionsauswertungen. Nachteil ist, dass die Formeln sehr groß und unhandlich werden können. Beispielsweise müssen für 100 Parameter und 10 Metabolite 1000 ODEs erzeugt werden. Dieser Algorithmus wurde in [Hurlebaus et al., 2002] implementiert. Um aber die computeralgebraische Aufstellung eines so großen Differentialgleichungssystems zu vermeiden wurde in MMT2 der folgende Implementierungsansatz gewählt.

9.5. Automatische Differenzierung

Eine Alternative zur symbolischen Berechnung der Ableitungen der Modellgleichungen ist die Automatische Differentiation (AD). Die verschiedenen Implementierungen der Methode des AD kann man grob in Source-To-Source-Transformation und Operatorüberladung einteilen, wovon letztere in dieser Arbeit verwendet wurde. MMT2 verwendet die C++ Bibliothek ADOL-C [Griewank et al., 1996], die durch Opera-

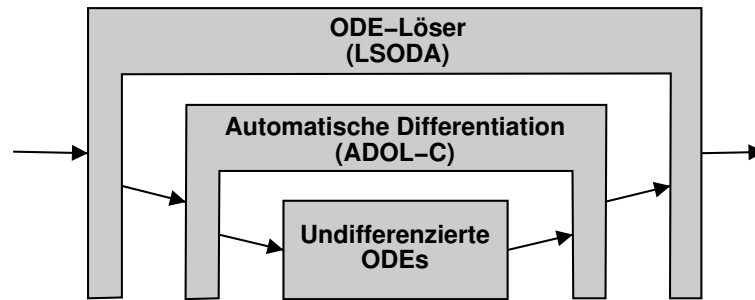


Abbildung 9.6: Berechnung der Sensitivitätsgleichungen ohne explizite Berechnung der Terme mit partiellen Ableitungen. Durch automatische Differentiation der Modell-ODEs kann der ODE-Löser den zeitlichen Verlauf der Sensitivitäten berechnen.

torüberladung aus einer gegebenen Funktion zusätzlich die Ableitungen berechnet, ohne dass diese explizit angegeben werden müssen.

Um die Sensitivitätsgleichungen zu berechnen muss in MMT2 nur noch die Gleichung 9.16 implementiert werden, wobei mit Hilfe von ADOL-C die Berechnungen der Ableitungen $\frac{\partial \vec{v}}{\partial \vec{\alpha}}$ und $\frac{\partial \vec{v}}{\partial \vec{x}}$ durchgeführt werden. Der große Vorteil bei dieser Vorgehensweise ist, dass im Quelltext nur die undifferenzierte Form der ODEs auftaucht, wodurch der Quelltext kompakt und überschaubar bleibt. Abb 9.6 zeigt die nötige Verschachtelung von ODE-Löser, AD und den Modell-ODEs.

9.5.1 Impelementierung

In [Lee and Hovland, 2001] wird ein Tool namens SensPVIDE vorgestellt. Es unterstützt die Lösung allgemeiner Gleichungen (wie ODEs und DAEs) und berechnet die Sensitivitäten mit AD. Dazu werden die Modell-ODEs 9.2 und die Sensitivitäts-ODEs 9.16 zu einem großen Gleichungssystem zusammengefasst, das der ODE-Löser dann lösen muss.

Dieser Ansatz wurde in dieser Arbeit probeweise für die metabolischen Netzwerke implementiert. Leider stellte sich heraus, dass die Zeit, die für die Lösung der Sensitivitäts-ODEs benötigt wurde, wesentlich höher war, als mit der FD-Methode. Eine wesentliche Beschleunigung der Berechnung stellte sich ein, als die Modell-ODEs (Gleichung 9.2) und die Sensitivitäts-ODEs (Gleichung 9.16) entkoppelt voneinander berechnet wurden. Abbildung 9.7a zeigt ein Flussdiagramm dieses Implementierungsansatzes. Dort werden zunächst in den Schritten 1 und 2 die Modell-ODEs gelöst und dann durch das Ergebnis der Berechnung ein Spline gezogen um in der späteren Berechnung darauf zurückzugreifen. In Schritt 3 wird die AD der Ableitungen $\frac{\partial \vec{v}}{\partial \vec{\alpha}}$ und $\frac{\partial \vec{v}}{\partial \vec{x}}$ initialisiert. In den Schritten 4 bis 7 wird die rechte Seite der Sensitivitäts-ODEs (Gleichung 9.16 und Abb. 9.6) berechnet.

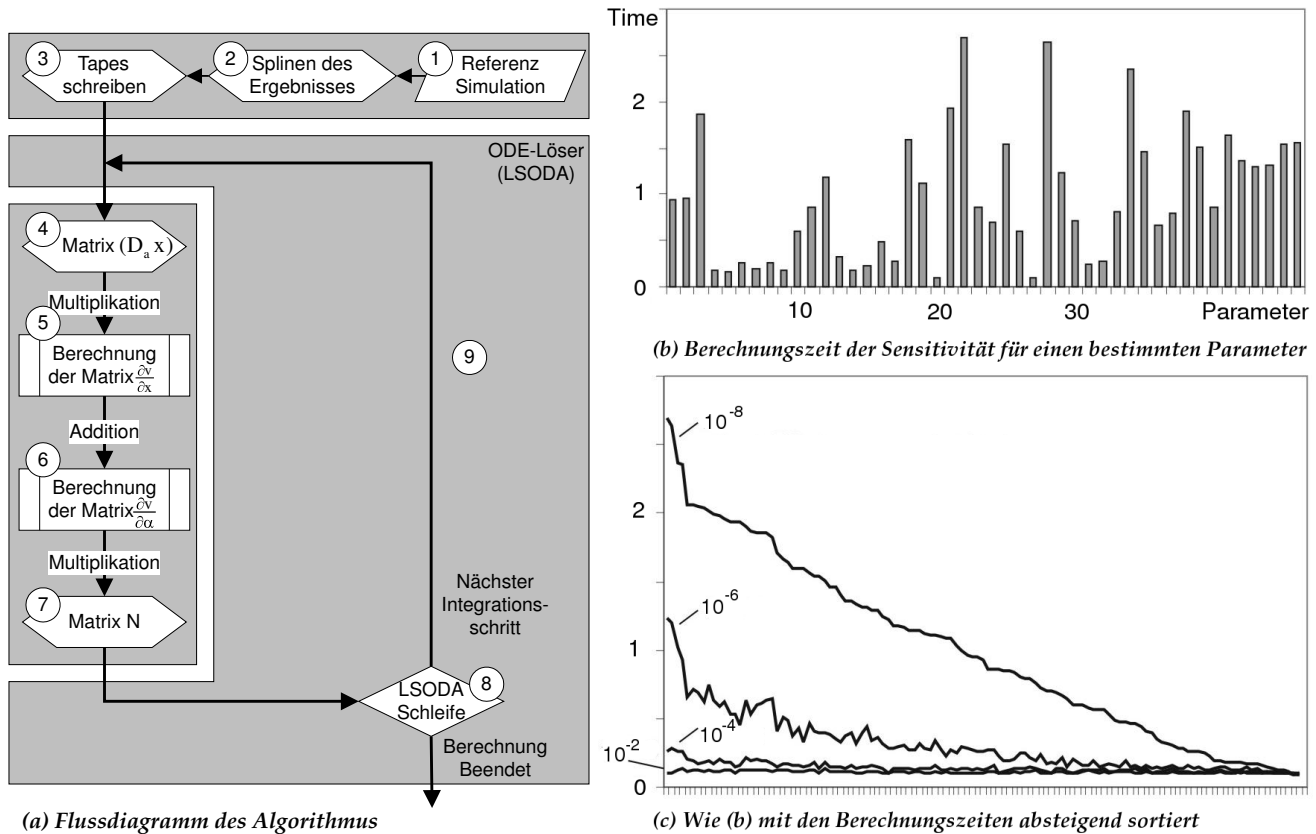


Abbildung 9.7: Berechnung der Parametersensitivitäten mit automatischer Differentiation. Analysiert wurde die Berechnungszeit für das Modell #5 aus Tab. 9.7. In c) sind die Berechnungszeiten mit der Toleranzeinstellung des ODE-Lösers beschriftet.

9.5.2 Modelle für Benchmark-Messungen

Um die zwei erwähnten Algorithmen zur Berechnung der Parametersensitivitäten zu testen wurden verschiedene Modelle aus laufenden Forschungsprojekten als Benchmark verwendet. Alle Benchmark-Messungen sind in Tabelle 9.7 zusammengestellt und wurden auf einem Athlon 1900+ durchgeführt. Bei den Berechnungen mit der Methode der Sensitivitäts-ODEs wurde die schnellere Variante gewählt, die in Abb. 9.7a als Flussdiagramm dargestellt ist.

9.5.3 Analyse und Diskussion

Bei der Berechnung der Sensitivitäten fiel auf, dass die Berechnungszeit sehr stark von der Einstellung der Fehlertoleranzen des ODE-Lösers abhängt. Bei dem Benchmark-Modell betrug die Berechnungszeit bei einer Fehlertoleranz von 10^{-8} mehr als 30 Minuten, bei einer Fehlertoleranz von 10^{-2} hingegen dauerte es nur 22 Sekunden. Im Ggs. zu den Modell-ODEs wird bei der Berechnung der Sensitivitäts-ODEs keine hohe Genauigkeit benötigt, weshalb eine Fehlertoleranzeinstellung von 10^{-2} ausreichend wäre.

Um zu untersuchen, warum die Berechnungszeit so stark von der Einstellung der Fehlertoleranzen des ODE-Lösers abhängt, wurde jede Parametersensitivität separat berechnet. Abb. 9.7b zeigt die Berechnungszeit für jeden einzelnen Parameter in Sekunden. Um einen Besseren Überblick zu erhalten wurden die Parameter nach ihrer Zeit

	dim($\vec{\alpha}$)	dim(\vec{x})	dim(\vec{v})	r.s.	f.d.	s.e.			
						10^{-2}	10^{-4}	10^{-6}	10^{-8}
#1	21	9	11	0.044s	0.47s	1.68s	1.76s	2.41s	1.26s
#2	91	10	15	0.33s	14.6s	6.70s	3.06s	0.97s	3.35s
#3	92	9	16	0.13s	9.5s	6.07s	0.55s	1.01s	2.04s
#4	99	10	16	0.31s	14.7s	7.53s	0.39s	1.08s	3.61s
#5	102	10	17	0.68s	60.6s	21.4s	49.9s	348s	1852s
#6	105	10	17	0.31s	15.0s	7.01s	0.43s	0.96s	3.56s
#7	122	10	22	0.32s	14.0s	4.00s	0.38s	1.48s	3.60s

Tabelle 9.7: Benchmarkresultate der verschiedenen Algorithmen. r.s. = Referenzsimulation, f.d. = FD-Methode s.e. = Methode der Sensitivitätsgleichungen (mit verschiedenen Einstellungen der Fehlertoleranz des ODE-Lösers).

absteigend sortiert. Ohne diese Reihenfolge zu ändern wurde die Berechnung dann nochmals durchgeführt mit einer anderen Einstellung der Fehlertoleranz (Abb. 9.7c). Aus dieser Visualisierung geht klar hervor, dass die Berechnung der Parametersensitivitäten, deren Berechnung sehr lange dauert, durch die Herabsetzung der Fehlertoleranz maßgeblich beschleunigt wird, wogegen bei den schnell zu berechnenden Sensitivitäten die Beschleunigung nicht so groß ausfällt. Das liegt vermutlich an der Steifigkeit der Sensitivitätsgleichungen, deren schnelle Dynamik bei der hohen Fehlertoleranz noch mit berücksichtigt wird. Wird die Fehlertoleranz geringer gewählt, so scheinen die schnellen Dynamiken vernachlässigt zu werden, was beschleunigenden Einfluss auf die Berechnung hat. Wenn einige Sensitivitäts-ODEs sehr steife Eigenschaften aufweisen, wirkt sich das sehr ineffizient auf die Berechnung aus, da wenige steife ODEs die Berechnung der vielen unsteifen ODEs ausbremsen.

Ein weiterer interessanter Effekt ist, dass bei einigen Modellen die Berechnungszeit bei einer Einstellung der Fehlertoleranz von 10^{-2} höher ist, als bei der präziseren Einstellung von 10^{-4} . Hier wird angenommen, dass dies durch die Schrittweitensteuerung des ODE-Lösers hervorgerufen wird, die durch die niedrige Genauigkeit irritiert wird. Über einer bestimmten Schwelle der Einstellung der Fehlertoleranz besteht dann aber die intuitiv erwartete Regelmäßigkeit, dass eine höhere Genauigkeit auch eine längere Berechnungszeit fordert.

Teil III

Anwendung von MMT2

Exemplarische Modellstudie mit MMT2

In diesem Kapitel soll am Beispiel eines einfachen Modells die Handhabung von MMT2 demonstriert werden. Der Inhalt dieses Kapitels ist an das MMT2 Reference Manual [Haunschild, 2005] angelehnt.

Als Hilfsprogramme werden in diesem Kapitel vor allem der Open-Source XML-Editor Xerlin (<http://www.xerlin.org>) und Matlab (<http://www.mathworks.com>) verwendet, aber auch die Kommandozeile von Linux, sowie ein Texteditor. Als einfaches Beispielmmodell wird eine lineare Abfolge von drei Reaktionen genommen.

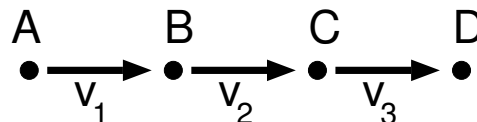


Abbildung 10.1: Beispielmmodell

10.1. Erstellung der Modellstruktur

.....

Das Modell muss zunächst in der XML-Markupsprache M3L (Kap. 7) formuliert werden. Zur erstmaligen Erstellung einer M3L-Datei gibt es die Möglichkeiten

1. mit einem Texteditor das XML-Dokument zu erzeugen und beispielsweise aus verschiedenen Modellen im M3L-Format ein neues zusammenzukopieren,
2. mit Xerlin den XML-Baum Schritt für Schritt zu erstellen,
3. mit dem vollgraphischen Editor Metvis (Metabolic Visualizer) [Qeli et al., 2003] das Modell zu erstellen und im M3L-Format zu exportieren,
4. mit dem SBML-Konverter ein Modell im SBML-Format in das M3L-Format zu konvertieren, oder
5. mit dem Metatool-Konverter (Kapitel 8.3.2) ein Modell vom Metatool-Format nach M3L zu konvertieren.

Sieht die Vorgehensweise des Modellierers so aus, dass er sich auf einem Blatt Papier die Reaktionsschritte zunächst notiert und dann erst in den Computer eingibt, würde Möglichkeit Nr. 5 eine schnelle Eingabe ermöglichen, die im Folgenden gezeigt werden soll.

Im Metatool-Format lassen sich die Reaktionen kompakt notieren. Dort kann zwar nur die stöchiometrische Modellstruktur definiert werden, aber nachdem diese in einem Texteditor eingegeben wurde, kann `mmt2_convert` diese Beschreibung in ein M3L-Gerüst umwandeln, bei dem sich die restlichen Elemente schnell hinzufügen lassen.

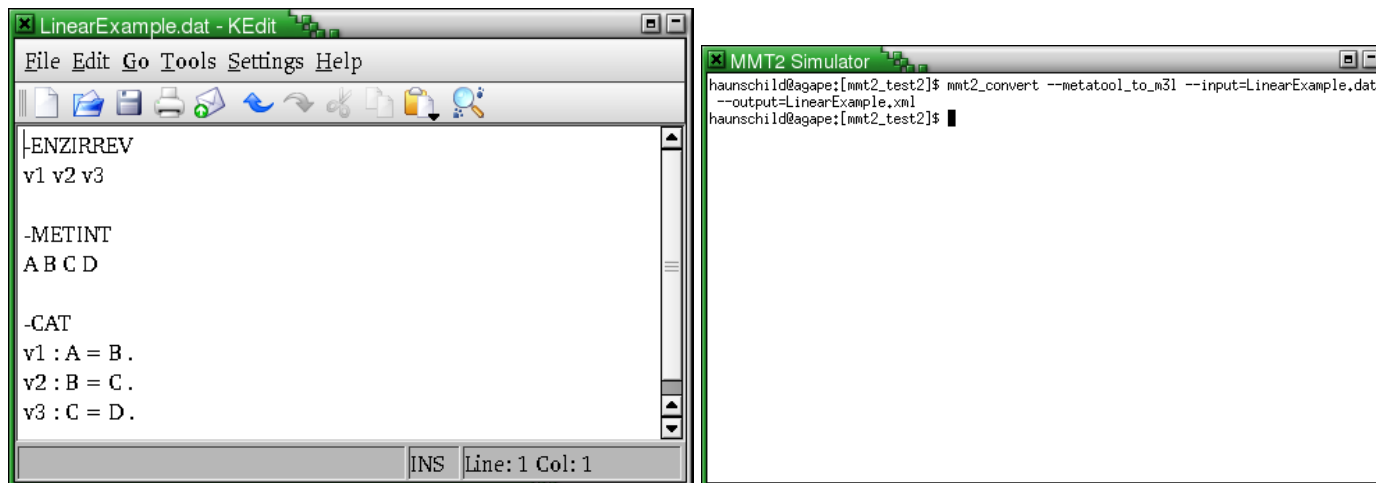


Abbildung 10.2: Links: Modell im kompakten Metatoolformat vgl. Alg. 8.2, Rechts: Der Konverter erzeugt ein M3L-Gerüst aus der Metatoolbeschreibung

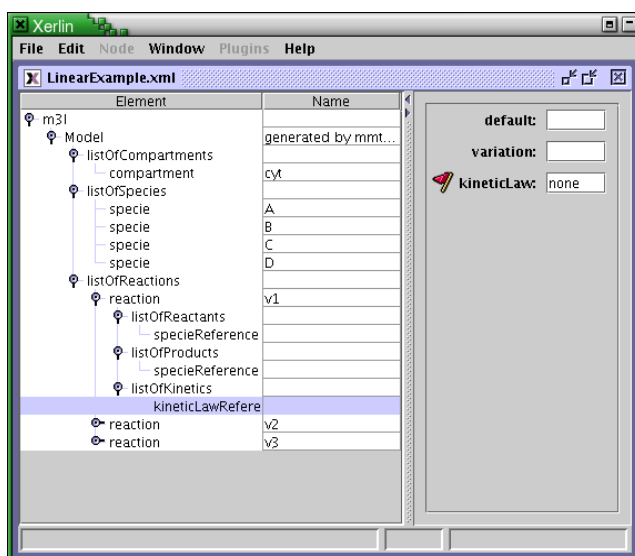


Abbildung 10.3: Erzeugtes M3L-Gerüst aus der Metatoolbeschreibung in Abb. 10.2

10.2. Vervollständigen der Modellstruktur

Nachdem die Stöchiometrie des Reaktionsnetzwerks definiert wurde, ist der nächste Schritt das Modellgerüst um kinetische Informationen zu erweitern. Dazu muss zunächst ein Knoten `listOfKineticsLaws` mit einem Unterknoten `kineticLaw` hinzugefügt werden (Abb. 10.4 links). Hier wird beispielhaft ein kinetisches Gesetz

definiert, dass `linear` genannt wird. Als Formel wird $k * S$ eingegeben. k und S sind Platzhalter, die in den Reaktionen, in denen diese Kinetik verwendet wird, an einen Wert oder Metaboliten gebunden werden muss. In Abb 10.4 rechts wird gezeigt wie für die Reaktion v_1 definiert wird, dass die Kinetik `linear` für diese Reaktion gilt. Dies muss auch für die anderen Reaktionen v_2 und v_3 entsprechend eingetragen werden.

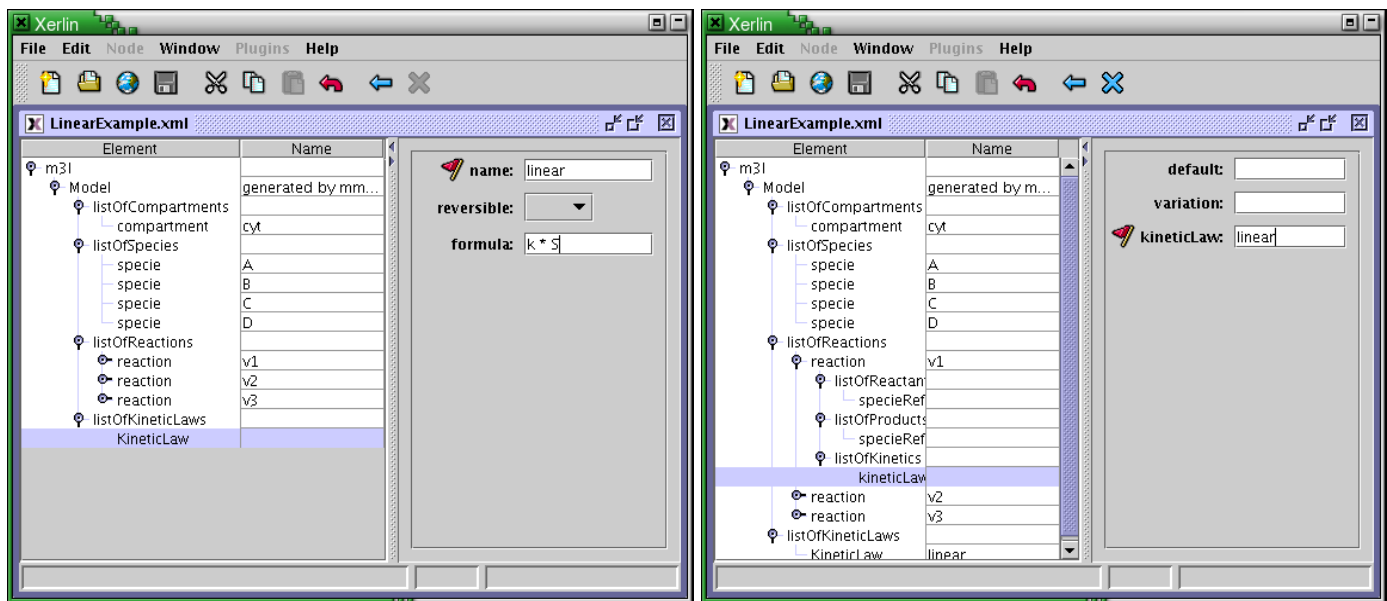


Abbildung 10.4: Links: Definieren eines kinetischen Gesetzes. Rechts: Festlegung, dass diese Kinetik in der Reaktion v_1 verwendet wird.

Nachdem definiert wurde, welches kinetische Gesetz für die einzelnen Reaktionen gilt, muss die Verwendung der Platzhalter spezifiziert werden. In Abb. 10.5 links wird der Platzhalter k als Parameter mit dem Wert 4 definiert und in Abb. 10.5 rechts wird der Platzhalter S an den Metaboliten A gebunden. Analog dazu muss in Reaktion v_2 der Parameter k auf 2 und S auf Metabolit B gesetzt werden und in Reaktion v_3 der Parameter k auf 1 und S auf Metabolit C gesetzt werden.

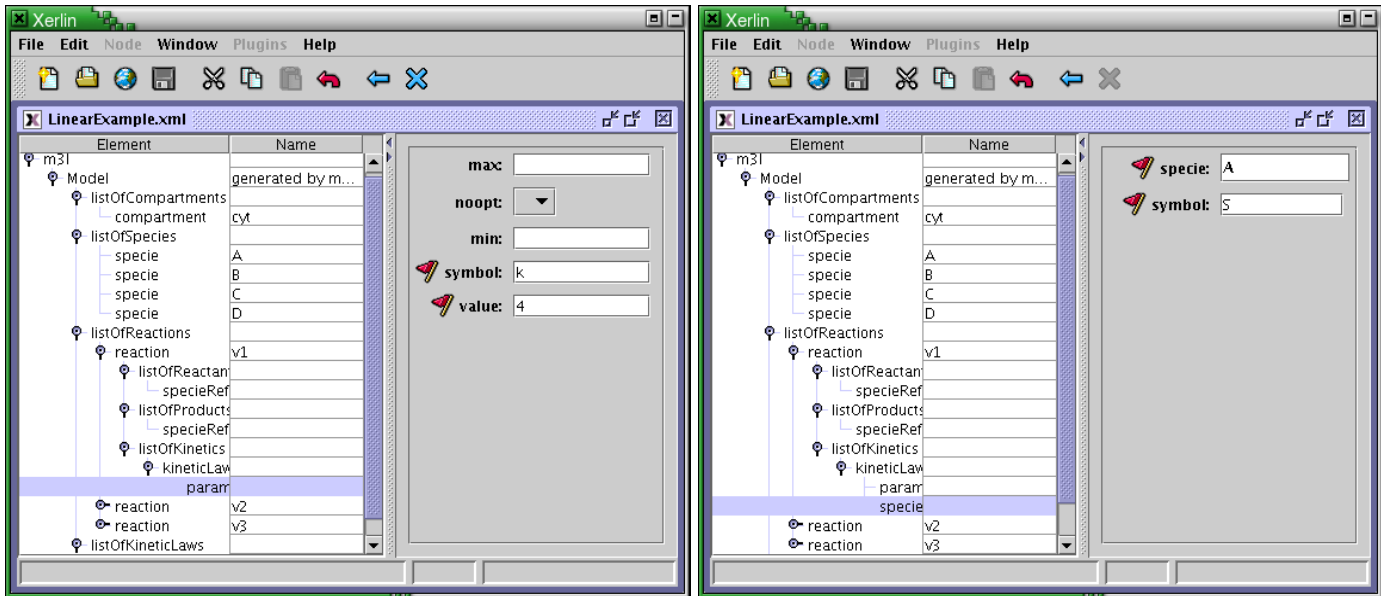


Abbildung 10.5: Links: Binden des Platzhalters k an den Parameterwert 4. Rechts: Binden des Platzhalters S an den Metaboliten A.

In Abb 10.6 links werden folgende Startwerte der Metabolite definiert: $A=4$, $B=0$, $C=0$, $D=0$. Schließlich müssen noch einige Verfahrensparameter für die Simulation gesetzt werden, wie in Abb 10.6 rechts gezeigt. Dazu muss ein Knoten `listOfModelParameters` angelegt werden mit einem Unterknoten `resultSamplePoints` in dessen Attributen die Simulationszeit angegeben wird.

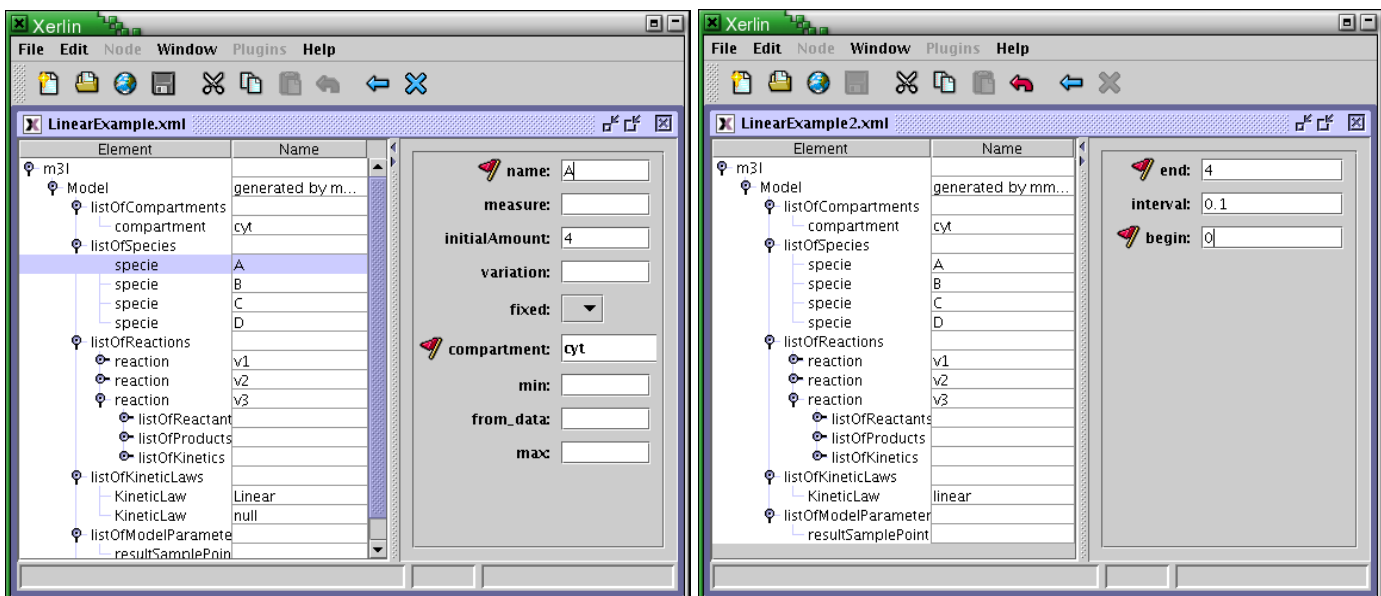
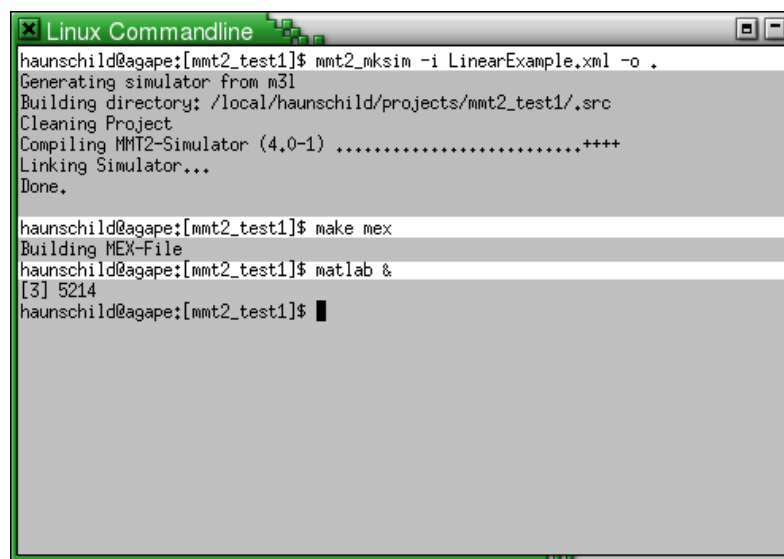


Abbildung 10.6: Links: Setzen der Startwerte der einzelnen Metabolite. Rechts: Setzen der Verfahrensparameter.

10.3. Simulatorerzeugung und Simulationsläufe

.....



```
Linux Commandline
haunschild@agape:[mmt2_test1]$ mmt2_mkxsim -i LinearExample.xml -o .
Generating simulator from m3l
Building directory: /local/haunschild/projects/mmt2_test1/.src
Cleaning Project
Compiling MMT2-Simulator (4.0-1) .....++++
Linking Simulator...
Done.

haunschild@agape:[mmt2_test1]$ make mex
Building MEX-File
haunschild@agape:[mmt2_test1]$ matlab &
[3] 5214
haunschild@agape:[mmt2_test1]$
```

Abbildung 10.7: Der Simulator wird aus der M3L-Beschreibung generiert und kompiliert.

Jetzt ist das Modell komplett definiert und kann simuliert werden. Abb. 10.7 zeigt, wie mit einem Aufruf `mmt2_mkxsim` des Generators auf der Kommandozeile der Simulator erzeugt und kompiliert wird. Um das Interface zu Matlab zu erstellen muss noch `make mex` aufgerufen werden. In Abb. 10.8 links ist zu sehen, dass es jetzt in Matlab einen Befehl `mmt2` gibt, über den auf MMT2 zugegriffen werden kann. Hier wird eine Simulation durchgeführt, die in einer Matlab-Struktur gespeichert wird. Abb. 10.8 rechts zeigt das Matlab-Ausgabefenster.

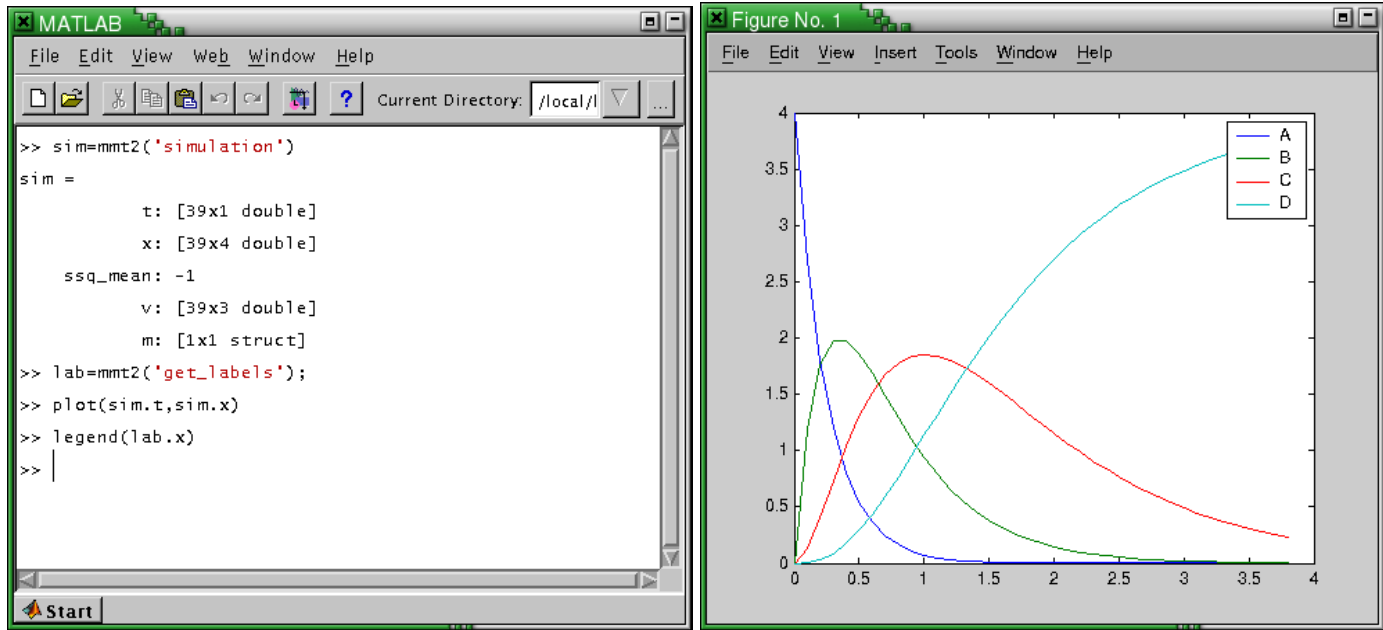


Abbildung 10.8: Links: Zugriff auf MMT2 von der Matlab-Kommandozeile. Rechts: Simulierter zeitlicher Verlauf der Konzentrationen der Metabolite des Beispielmodells.

10.4. Messdaten hinzufügen

Um in diesem Beispiel eine Parameteranpassung durchzuführen soll jetzt aus dem Originalverhalten des Modells ein Zeitverlauf genommen werden, mit etwas Rauschen überlagert werden und anschließend in die Modellbeschreibung als Messdaten aufgenommen werden.

Abb. 10.9 links zeigt, wie mit Matlab eine solche Referenzsimulation durchgeführt, mit Rauschen überlagert und schließlich als CSV-Datei abgespeichert wird. Für die Umwandlung dieser Daten in ein M3L Fragment steht wieder das Tool `mmt2_convert` zur Verfügung, das die CSV-Datei einliest und als M3L-Fragment abspeichert (Abb. 10.9 rechts).

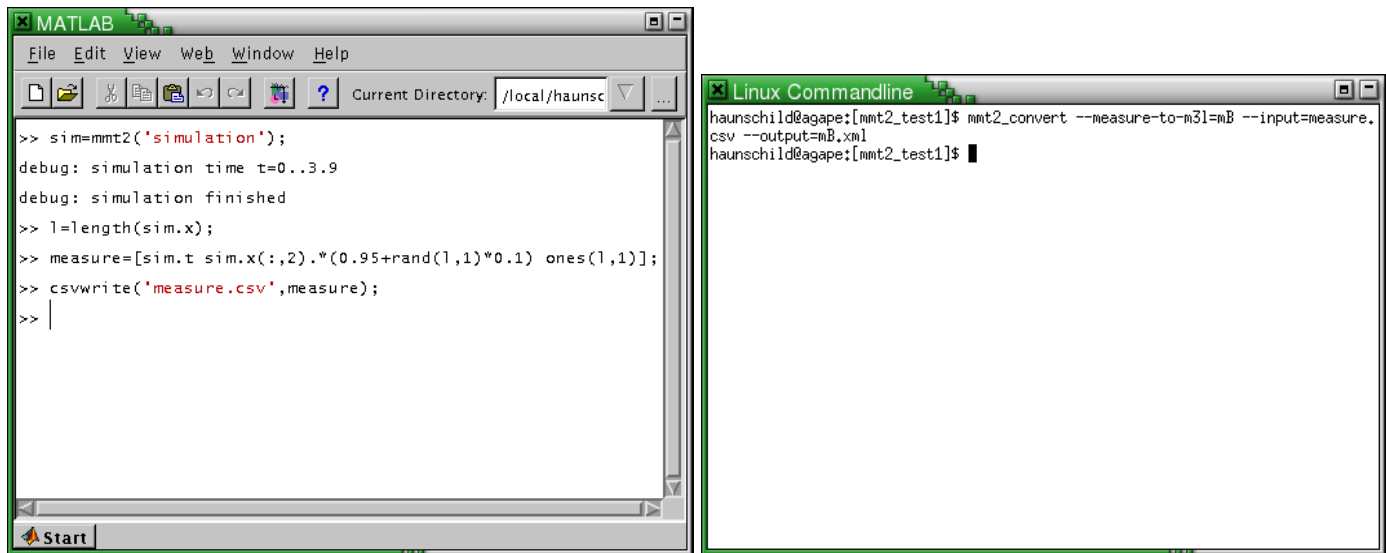


Abbildung 10.9: Erzeugung von Messdaten aus dem Originalverhalten des Modells

Abb. 10.10 zeigt, wie die Messdaten in das bestehende Modell im M3L-Format aufgenommen werden können. Mittels eines Texteditors wird ein neuer Knoten im XML-Baum angelegt (Abb. 10.10 links), in den dann das abgespeicherte M3L-Fragment eingefügt wird (Abb. 10.10 rechts).

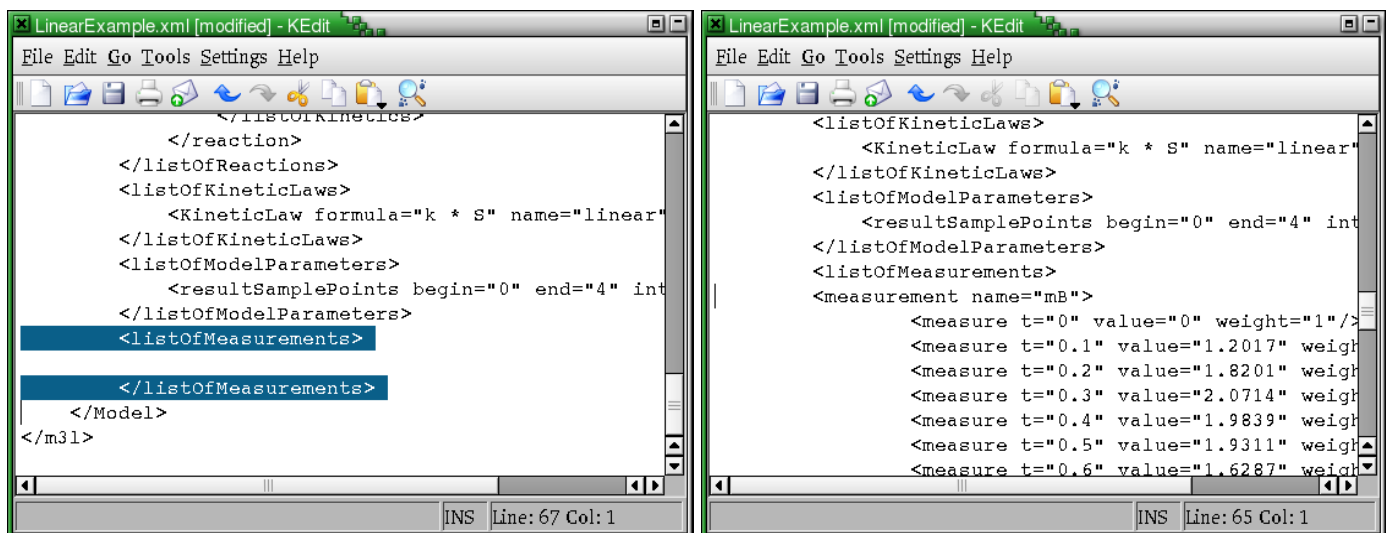


Abbildung 10.10: Einfügen von Messdaten in die Modellbeschreibungdatei im M3L Format

Nun sind die Messdaten im Modell definiert und müssen an einen Metaboliten gebunden werden (Abb. 10.11 links). Zusätzlich wird der Startwert des Metaboliten B noch verändert, damit die Simulation von den Messdaten abweicht. Nach der Veränderung der Modellbeschreibung muss der Simulator neu generiert und kompiliert werden (Abb. 10.11 rechts). In Abb. 10.12 links wurden neben dem Zeitverlauf des Modells die erzeugten Daten visualisiert.

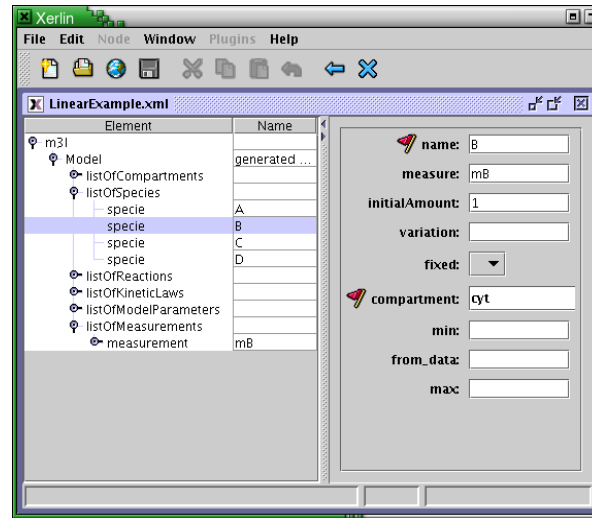


Abbildung 10.11: Binden der Messdaten an den Metabolit B.

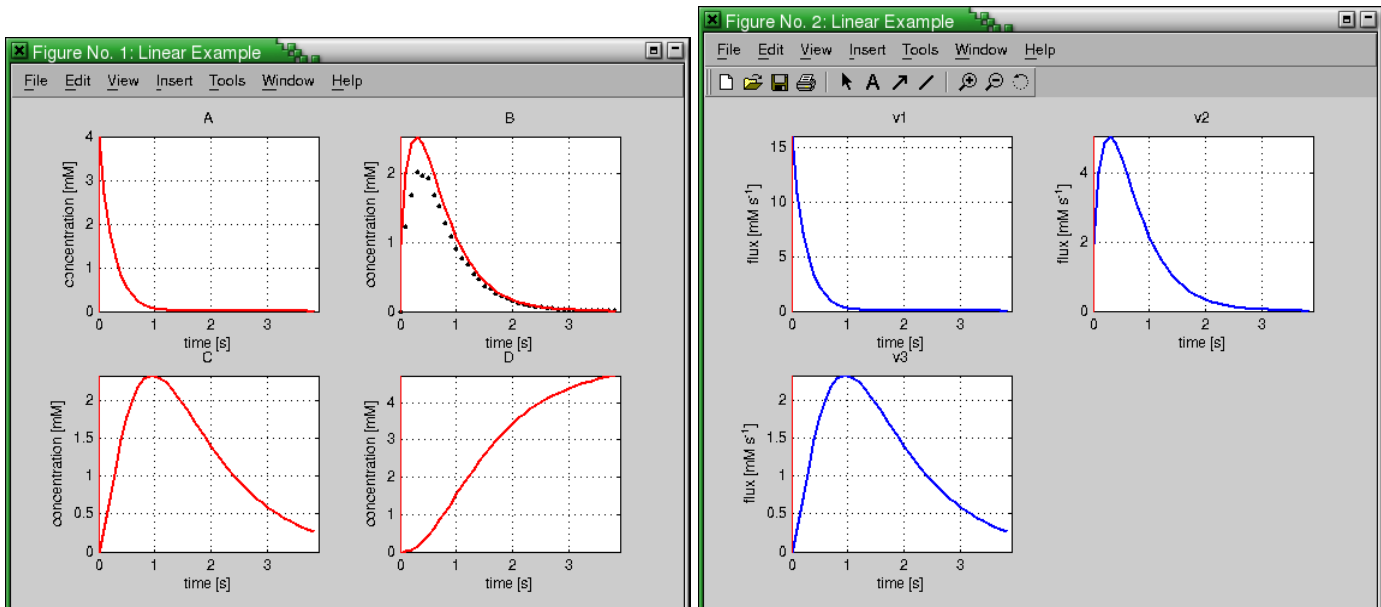


Abbildung 10.12: Links: Simulation des Modells und Ausgabe der zeitlichen Verläufe der Metabolitpools. Rechts: Geschwindigkeit der Flüsse über der Zeit.

10.5. Parameteranpassung

Mit dem Vorhandensein von Messdaten kann nun eine Parameteranpassung durchgeführt werden. Da der Startwert des Metaboliten B verändert wurde, passt der Verlauf von B nicht mehr exakt auf die erzeugten Messdaten. In Abb. 10.13 links wird die Parameteranpassung gestartet, die innerhalb weniger Sekunden zu einer Verbesserung kommt. Mit CTRL-C kann der Anpassungsprozess angehalten werden, woraufhin der beste gefundene Parametersatz ausgegeben wird. Dieser Parametersatz wird kopiert und in Abb. 10.13 rechts entsprechend eingefügt (neben `set_model_a`). Das Fenster

aus Abb. 10.12 ist noch offen und mit dem Befehl `plot_simulation` werden die Ergebnisse der Simulation mit dem neuen Parametersatz in einer anderen Farbe darüber geplottet (Abb. 10.14).

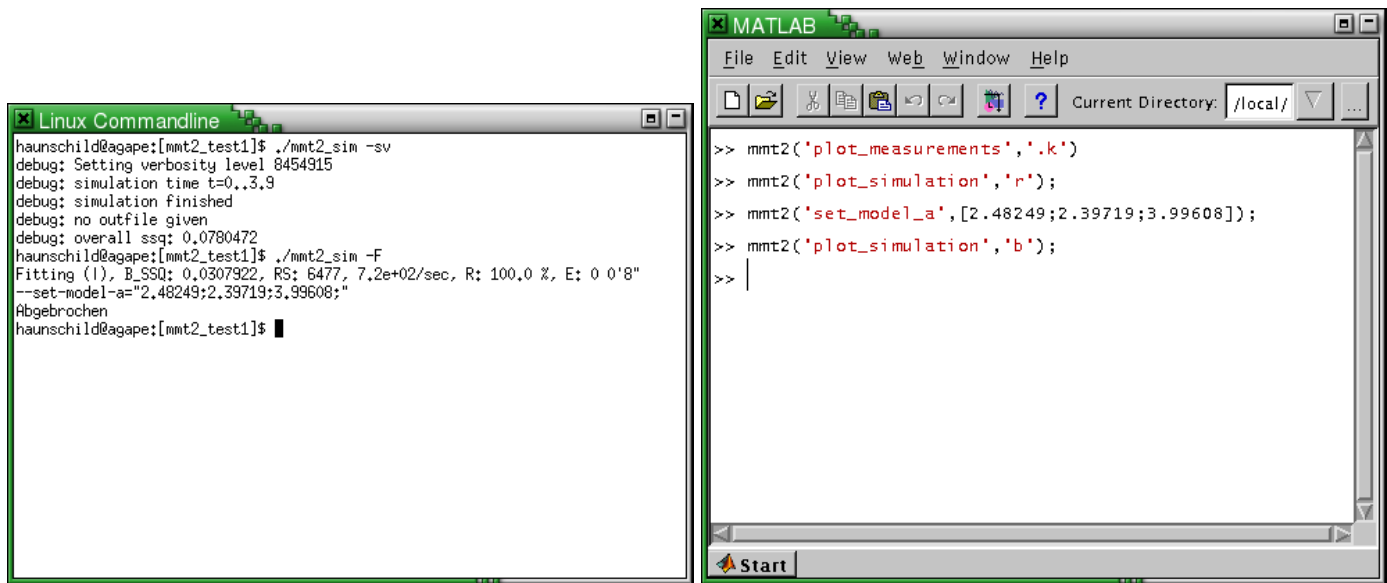


Abbildung 10.13: Durchführung der Parameteranpassung

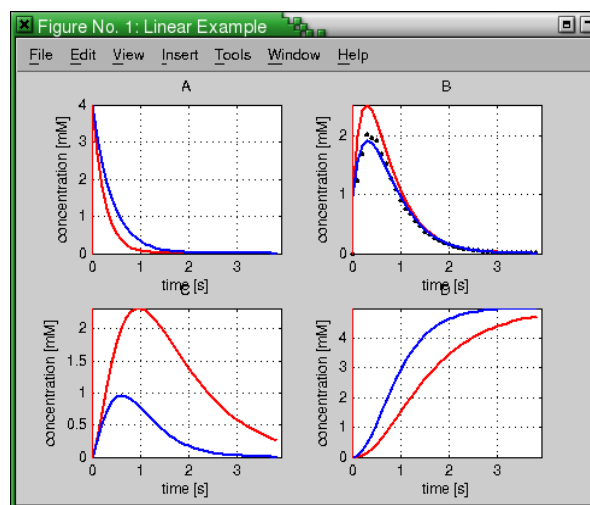


Abbildung 10.14: Zeitverlauf vor und nach der Parameteranpassung

10.6. Splines hinzufügen

Der Zeitverlauf eines Metaboliten kann auch durch einen Spline vorgegeben werden. In diesem Abschnitt soll ein weiterer Metabolit dem Modell hinzugefügt werden, der zu einem bestimmten Zeitpunkt einen Impuls auf das System gibt. Die Splinedefinition in M3L ist so gestaltet, dass die Teilpolynome des Splines unstetig sein können und einen beliebigen Grad haben können. So kann man Teilpolynome von Grad 0 wählen

um einen Impuls zu einem bestimmten Zeitpunkt zu erzeugen wie in Abb. 10.15 rechts gezeigt.

Um einen Impuls auf das System zu geben wird ein zusätzlicher Metabolit hinzugefügt dessen Zeitverlauf anschließend auf den des Splines gesetzt wird (Abb. 10.15 links)

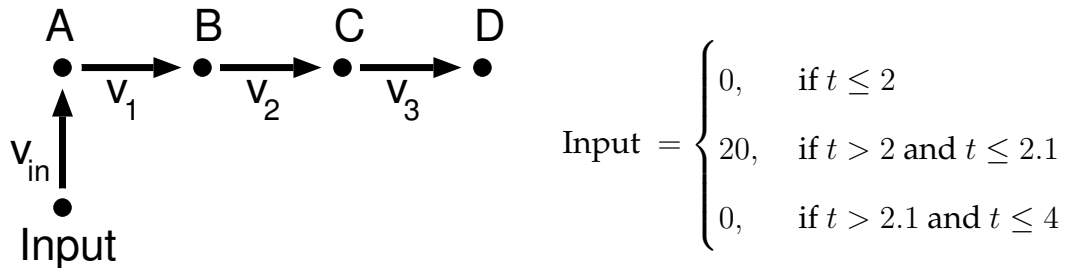


Abbildung 10.15: Erweiterung des Beispielmodells um einen Metaboliten, dessen Konzentrationsverlauf von einem Spline vorgegeben wird.

Zunächst muss der Spline in der Modellbeschreibung definiert werden. Dazu muss ein Knoten `listOfSplines` mit einem Unterknoten `spline` erzeugt werden. Wie in Abb. 10.15 rechts definiert werden die Stützstellen des Splines im M3L Format definiert (Abb. 10.16 links). Abb. 10.16 rechts zeigt die Ansicht aus dem XML-Editor. Hier muss jetzt der zusätzliche Metabolitpool hinzugefügt werden. Im Attribut `from_data` wird der Zeitverlauf des Splines dem Metaboliten zugewiesen.

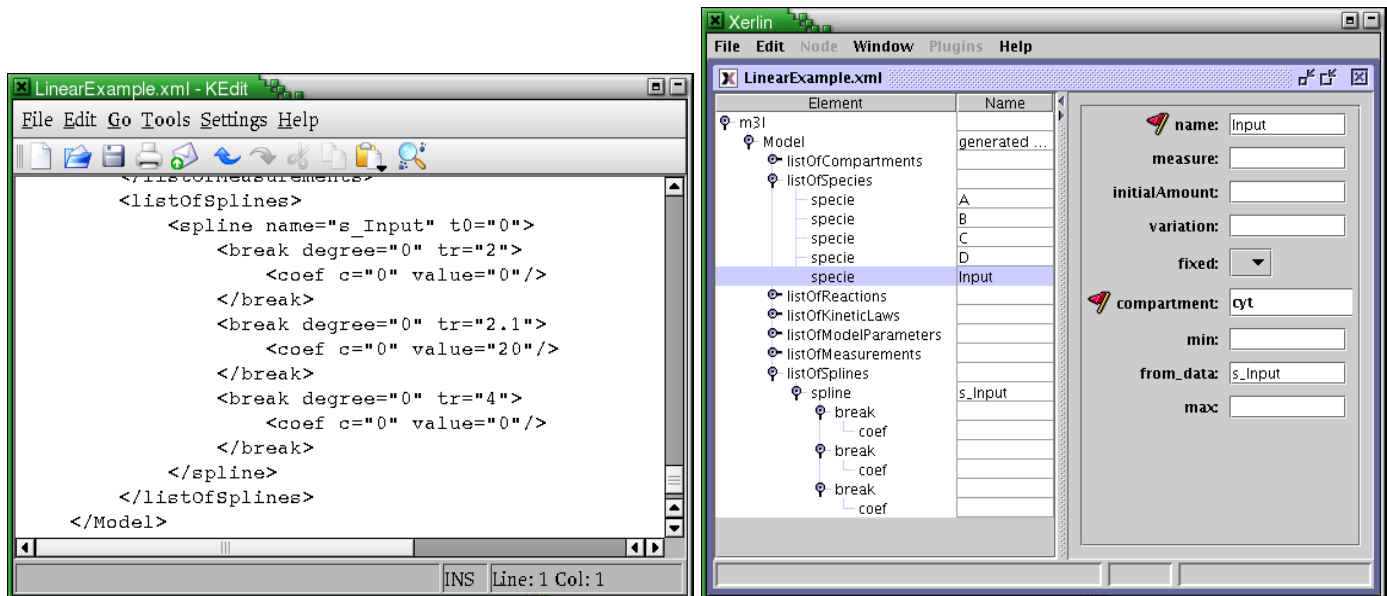


Abbildung 10.16: Links: Definition des Splines. Rechts: Hinzufügen eines zusätzlichen Metaboliten auf den der Spline zugewiesen wird.

Jetzt muss noch die Reaktion v_{in} definiert werden, die den Impuls des Metaboliten Input auf den Metaboliten A überträgt (Abb. 10.17 links). Nach dem Neuerstellen des Simulators und der Simulation aus Matlab heraus stellt sich der Zeitverlauf aus Abb. 10.17 rechts ein.

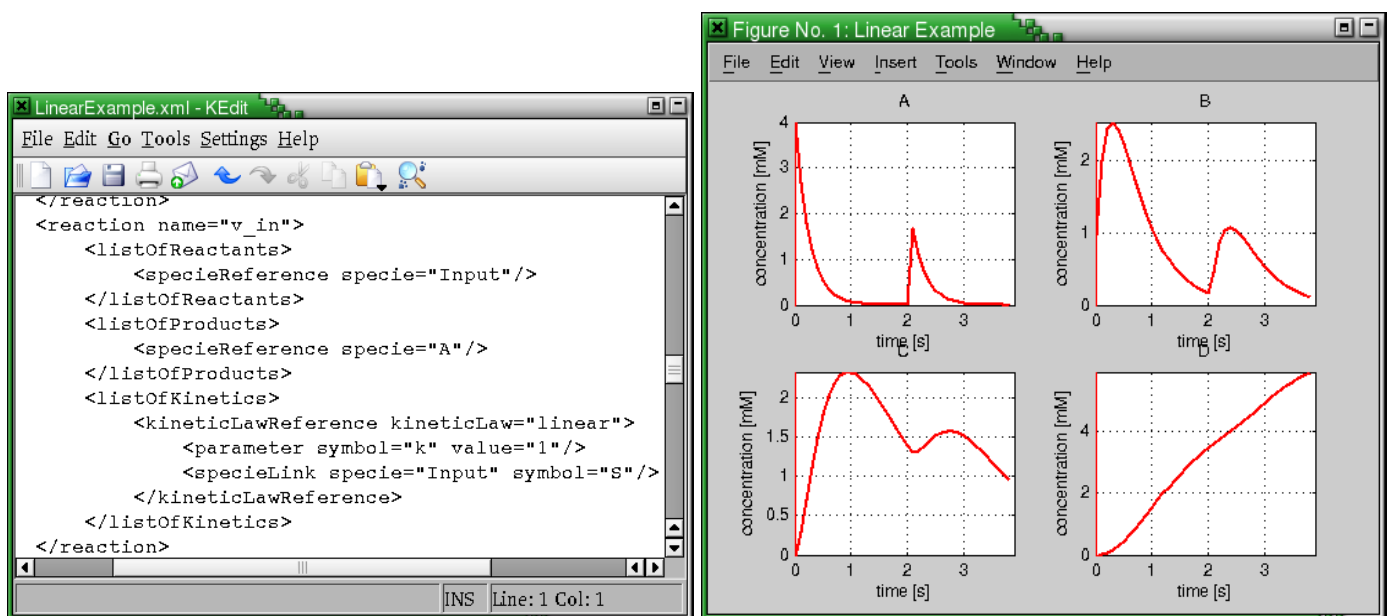


Abbildung 10.17: Links: Definition der Reaktion v_{in} . Rechts: Simulationsergebnis des erweiterten Modells.

10.7. Modellfamilie erstellen

Nun sollen die zwei bisher erstellten Modelle in eine Modellfamilie aufgenommen werden. Dazu muss das maximale Netzwerk in der Modellbeschreibung definiert werden, was mit dem zweiten Modell bereits der Fall ist. Der Unterschied zwischen dem ersten und dem zweiten Modell ist die Reaktion v_{in} , die im ersten Modell auf die Null-Kinetik gesetzt sein muss und beim zweiten Modell auf die Kinetik `linear` gesetzt sein muss.

Um die Null-Kinetik zu verwenden muss zunächst ein neues kinetisches Gesetz angelegt werden, bei dem die Formel auf konstant 0 gesetzt wird (Abb. 10.18 links). Dann muss für die Reaktion v_{in} neben der Kinetik `linear` die Null-Kinetik als Alternative spezifiziert werden (Abb. 10.18 rechts).

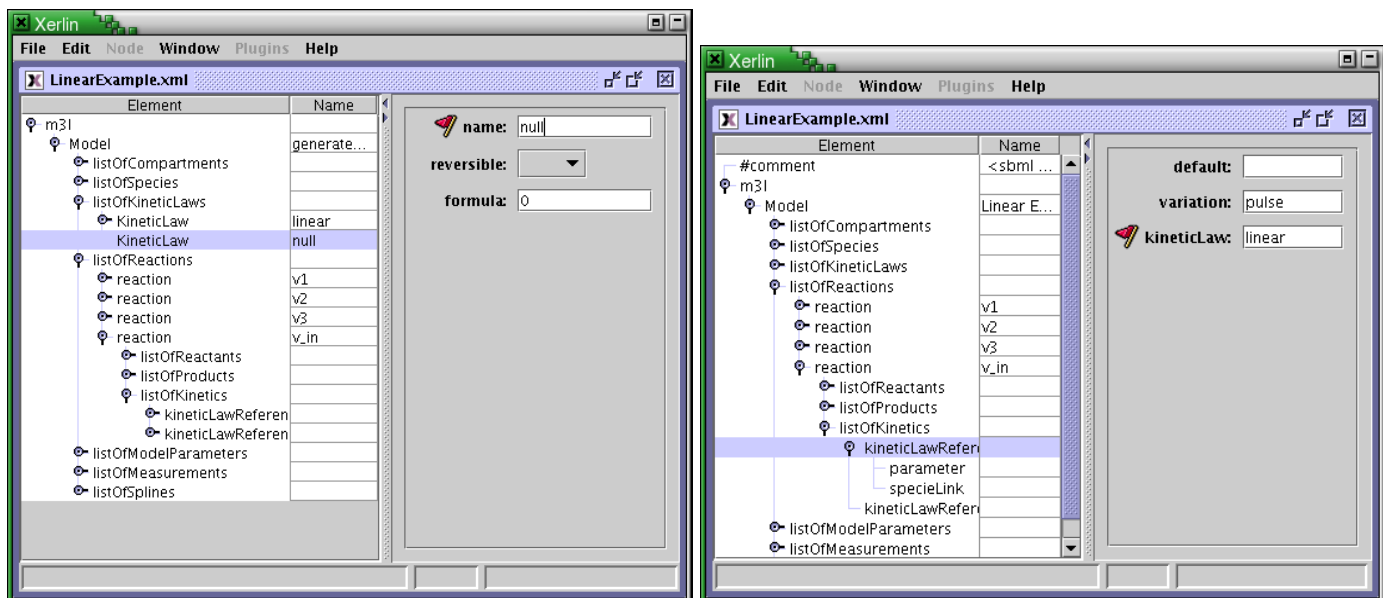


Abbildung 10.18: Einführung der Null-Kinetik und Anwendung auf eine Reaktion.

Nachdem der Simulator neu erzeugt und kompiliert wurde kann man auf der Kommandozeile mit dem Parameter `--get-model-info` in den Informationen sehen, dass der erzeugte Simulator zwei Varianten des Modells gespeichert hat.

```

Linux Commandline
haunschild@agape:[mmt2_test1]$ ./mmt2_sim --get-model-info
mmt2_sim 4.0
Name:          'Linear Example'
Species:       4
Parameters:    4
Reactions:     4
Variants:     2
Splines:       1
Measurements:  39 (Species: 1)
Simulation time: 0..4
Building machine: agape, i686 Linux 2.6.10
haunschild@agape:[mmt2_test1]$ ./mmt2_sim -p --set-variation-no=1
lfdrnr:      name:      current;      initial;      freedom
model.x0:
0;           A;         4;           4;           0.00 %
1;           B;         1;           1;           0.00 %
2;           C;         0;           0;           0.00 %
3;           D;         0;           0;           0.00 %
model.a:
Reaction: v1, linear (enabled)
0;         k_v0;      4;           4;           0.00 %
Reaction: v2, linear (enabled)
1;         k_v0;      2;           2;           0.00 %
Reaction: v3, linear (enabled)
2;         k_v0;      1;           1;           0.00 %
Reaction: v_in, pulse (3=1) (enabled)
3;         k_v1;      1;           1;           0.00 %
haunschild@agape:[mmt2_test1]$

```

Abbildung 10.19: Aufruf des neuen Simulators auf der Kommandozeile.

Auf der Matlab-Kommandozeile hat man nun Zugriff auf alle Modellvarianten. In Abb. 10.20 links wird zunächst die Modellvariante 0 geplottet, bei der die Reaktion v_{in} auf die Null-Kinetik gesetzt wird. Daher sieht man in Abb. 10.20 links keinen Impuls. In Abb. 10.20 rechts wird nun über die bereits vorhandene Modellvariante 0 die Modellvariante 1 geplottet, bei der die Reaktion v_{in} aktiv ist. Daher sieht man in Abb. 10.20 rechts im Zeitverlauf des Modells 1, der in Rot geplottet ist, den Impuls, der auf das System ausgeübt wird.

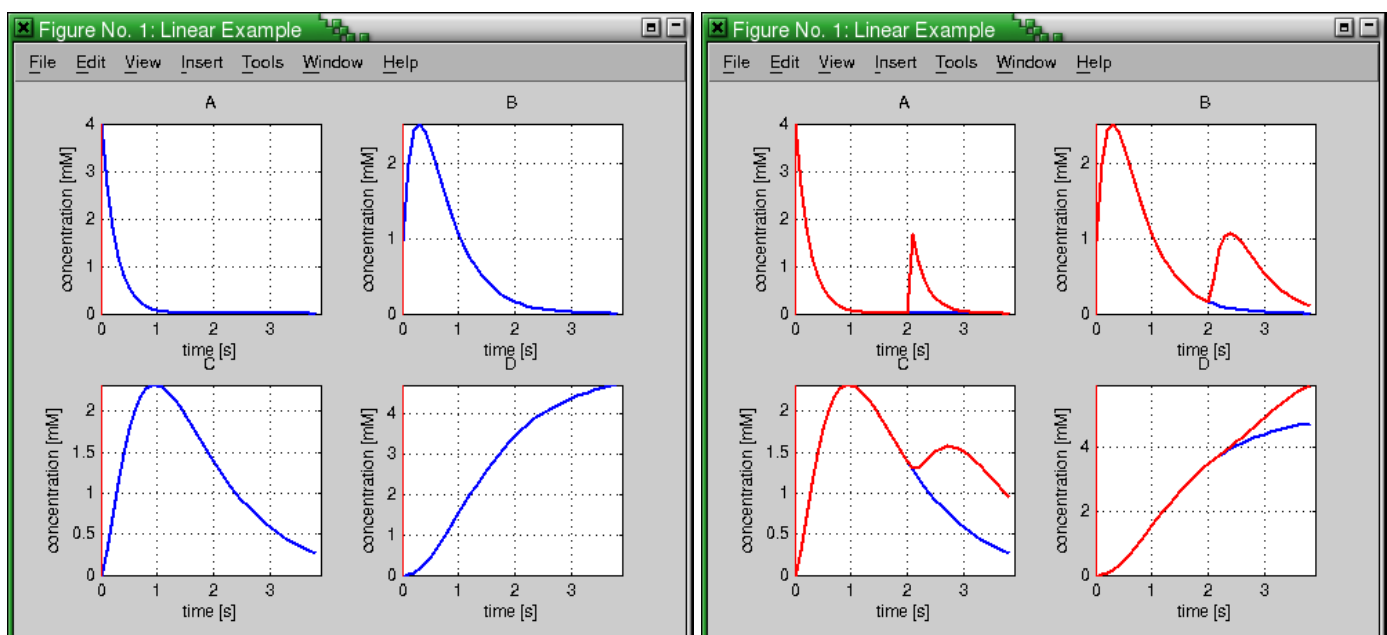


Abbildung 10.20: Plotten des Zeitverlaufs von Modellvariante 0

Komplexe Anwendungsbeispiele

Dieses Kapitel stellt drei biotechnologische Projekte vor, in denen das Softwarewerkzeug MMT2 zur Simulation und Datenauswertung verwendet wurde. In diesem Kapitel soll exemplarisch gezeigt werden, in welchen Projektkontexten MMT2 bereits zum Einsatz gekommen ist. Diese wissenschaftlichen Arbeiten befassen sich im einzelnen mit

- einem Auswertungsprozess für Rapid-Sampling-Experimente [Degenring, 2004],
- einem *E. coli* L-Phenylalanin Produktionsstamm [Wahl, 2005] und
- einem *C. glutamicum* L-Valin Produktionsstamm [Magnus, 2005].

Alle drei Arbeiten haben gemeinsam, dass sie am Institut für Biotechnologie des Forschungszentrums Jülich durchgeführt wurden und experimentelle Daten auswerten, die im Biotechnikum des IBT (Abb. 11.1) gemessen wurden.



Abbildung 11.1: Das Biotechnikum des Instituts für Biotechnologie (IBT), das zu den Großgeräten des Forschungszentrums Jülich zählt. Hier wurden die Stimulus-Response-Experimente durchgeführt, für die MMT2 zugeschnitten wurde.

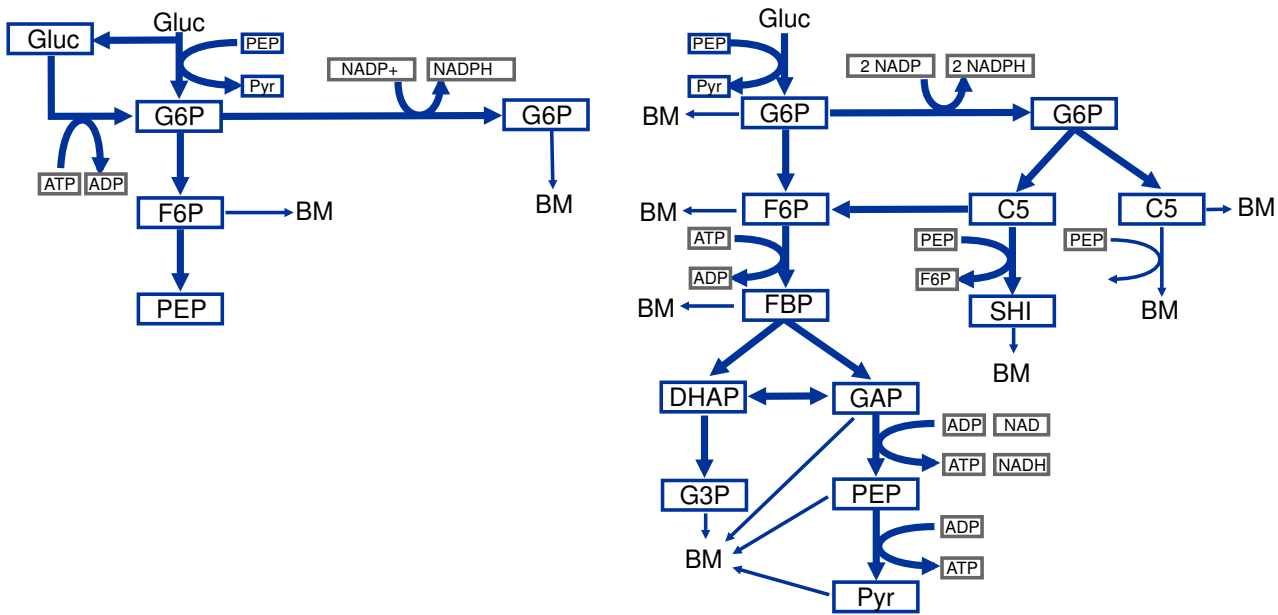


Abbildung 11.2: In der Arbeit von [Degenring, 2004] wurden für ein Experiment ca. 130 Modelle erstellt, die auf sechs verschiedenen Modell-Topologien basieren. Links die Ausgangsstruktur, die in den folgenden Modellen erweitert wurde. Rechts eine der umfangreichsten der sechs erstellten Topologien.

```

1: function bootstrap_test(Modell, OrigDaten, b)
2:   ...
3:   for i = 1 to b do
4:     BootDaten ← Erzeuge Bootstrapdaten;
5:     XMLDatei ← Vereinige(Modell, BootDaten);
6:     /* Compilieren des Modells: mmt2_sim entsteht */
7:     MMT2_MKSIM(XMLDatei);
8:     StartParaVektor ← RANDOMIZE(UntereGrenze,
9:     ObereGrenze, OrigParaVektor);
10:    /* Anpassen des Modells an die Bootstrap-Daten */
11:    OptParaVektor ← MMT2_SIM --FIT(StartParaVektor);
12:    /* Simulation mit optimierten Parametern */
13:    OptErgebnis ← MMT2_SIM --SIM(OptParaVektor);
14:    ErgebnisListe ← (BootDaten, OptErgebnis);
15:  end
16:  ...
17: end

```

Listing 11.1: In [Degenring, 2004] wurde ein auf Bootstrap-Datensätzen basierender statistischer Entscheidungstest implementiert. Durch das einfache Handling konnte MMT2 in dieser Implementierung über seine XML-Schnittstelle und Programmaufrufen in dieser Berechnung verwendet werden.

11.1. Auswertungsprozess für Rapid-Sampling-Experimente

.....

Erstmalig kam MMT2 in diesem Projekt zum Einsatz. Das Ziel dieser Arbeit war es, einen Modellierungs- (Kapitel 3) und Validierungsprozess für die Auswertung von Datensätzen aus Stimulus-Response-Experimenten (SRE) (Kapitel 2.3) zu entwickeln. Die einzelnen Schritte der experimentellen Modellbildung, wie sie in Abb. 1.3 aufgeführt sind, finden mit Ausnahme von Abb. 1.3E, G und H auch in [Degenring, 2004] Anwendung.

Die Datensätze (Abb. 1.3A) kamen aus einem Experiment mit einem *E. coli* K12 Wildstamm [Buchholz et al., 2001]. Gegen Ende dieser Arbeit wurden die Datensätze eines weiteren Experiments verfügbar, das auf zwei genetisch modifizierten *E. coli*-L-Phenylalanin-Produktionsstämmen [Oldiges, 2005] basiert.

Für den Validierungsprozess wurden verschiedene Methoden ausgearbeitet um die Qualität der Datensätze aus den Experimenten zu untersuchen. Dazu gehören u. a. eine Größtfehlerabschätzung, Plausibilitätstests, Nachweis-, Erfassungs- und Bestimmungsgrenzen und verschiedene Datenglättungsmethoden um das Rauschen in den Messungen zu eliminieren. Neben der Validität der Daten wurden auch die Softwarewerkzeuge MMT1 und MMT2 mit Hilfe von vielen Beispielmotellen auf Richtigkeit der Rechenergebnisse überprüft.

11.1.1 Modellbildung

Für das Experiment am *E. coli* K12 Wildstamm wurden ca. 130 Modelle erstellt, um die Stoffwechselforgänge in der Glykolyse und des PPP zu beschreiben. Die Modelle lassen sich in sechs Topologien aufteilen von denen in Abb. 11.2 zwei abgebildet sind. Innerhalb dieser Topologien wurden für die Reaktionen verschiedene kinetische Beschreibungen durchgetestet.

Da diese Modellierung hauptsächlich mit MMT1 durchgeführt wurde, während MMT2 sich noch im frühen Entwicklungsstadium befand, konnte das Konzept der Modellfamilien noch nicht eingesetzt werden, was in diesem Falle zu einer erheblichen Entlastung bei der Modellerstellung geführt hätte.

Nach der Erstellung eines Modells wurde nach der Parameteranpassung die Wiedergabequalität der Messdaten bewertet. Diese Bewertung wurde anschließend als Entscheidungskriterium verwendet, ob die Hypothesen und Vereinfachungen, die in dem Modell gemacht wurden geeignet sind. Am Ende des Modellierungsprozesses wurden aus den 130 erstellten Modellen 12 ausgewählt, die den zeitlichen Verlauf der Messdaten angemessen gut wiedergeben. Diese Modelle liegen in der Größenordnung von 10-12 ODEs und 90-130 Parametern [Degenring, 2004].

11.1.2 Modelldiskriminierung

Für die Modelldiskriminierung wurden in [Degenring, 2004] verschiedene Kriterien untersucht und miteinander verglichen. Diese Kriterien sind die Fehlerquadratsumme (die auch von MMT2 verwendet wird - Kapitel 9.2), Log-Likelihood-Funktion, Akaike-

Kriterium und Anpassungstest auf Basis der Bootstrap-Datensätze. Es stellte sich dabei heraus, dass all diese Kriterien geeignet sind um die Anpassungsqualität der Stoffwechselmodelle zu bewerten. Als bestes Kriterium wird das Verfahren auf Basis der Bootstrap-Datensätze ermittelt, das eine eindeutige Entscheidung liefert, ob ein Modell akzeptiert oder verworfen werden sollte. Die Implementierung dieses Verfahrens ist in Listing 11.1 dargestellt.

Zunächst werden aus den Messdaten die Bootstrapdaten erzeugt (Zeile 4), die mit den Modell-ODEs zusammengesetzt und als XML-Datei (Zeile 5) ausgegeben werden. Danach wird mit `mmt2_mksim` der Simulator generiert (Zeile 6), der von einem zufälligen Startpunkt im Parameterraum (Zeile 7) eine Parameteranpassung startet (Zeile 8). In den verbleibenden Zeilen 9 und 10 der For-Schleife werden die Ergebnisse aufbereitet.

Teil 3 ordnet die Ergebnisse aus Teil 2 und vergleicht sie mit den Prüfgrößen. Daraus werden Kennzahlen berechnet, aufgrund derer die Entscheidung getroffen wird, ob das Modell akzeptiert oder verworfen werden sollte.

11.2. *E. coli* L-Phenylalanin Produktionsstamm

.....

Die Aminosäure Phenylalanin ist unter Anderem für die Synthese von Zuckerersatzstoffen ein wichtiger Ausgangsstoff und wird in großen Mengen benötigt. Im Rahmen des Projekts „Aromatensynthese auf Zuckerbasis“ wurde basierend auf dem Wildtyp *E. coli* LJ110 der gentechnisch veränderte Produktionsstamm *E. coli* 4 entwickelt. Zur Analyse der genetischen Eingriffe und zur Beurteilung der Auswirkungen kommen verschiedene Methoden zum Einsatz. Neben den genetischen Untersuchungen wird vor allem auf der Ebene des Metaboloms, konkret der Flüsse geforscht. Im Rahmen der experimentellen Arbeiten eines weiteren Projekts [Oldiges, 2005] wurden mehrere Stimulus-Response-Experimente (Abb. 1.3A und Kapitel 2.3) durchgeführt. Die Messergebnisse aus den Experimenten sind in Abb. 11.3 enthalten.

In den entsprechenden Modellierungsarbeiten [Wahl, 2005] wird mit Hilfe der experimentellen Modellbildung (Abb. 1.3) das Verhalten des Aromatenbiosynthesewegs untersucht. Als Werkzeug zur Modellbildung wurde MMT2 verwendet, dessen neue Algorithmen und Vorgehensweisen in dieser Arbeit getestet wurden. Des Weiteren wurden im Rahmen dieser Arbeit Ideen erarbeitet, durch die das Softwarewerkzeug den Prozess der Modellbildung noch besser unterstützen kann. Diese Ideen flossen direkt in die Erstellung von MMT2 ein und konnten zeitnah getestet und ggf. abgeändert werden.

11.2.1 Modellbildung

Die Modellbildung konzentriert sich nicht wie in Abschnitt 11.1 auf den Zentralstoffwechsel, sondern auf den Shikimat-Syntheseweg (Abb. 11.4), dem gemeinsamen Reaktionsweg der aromatischen Aminosäuren (L-Phe, L-Tyr, L-Trp).

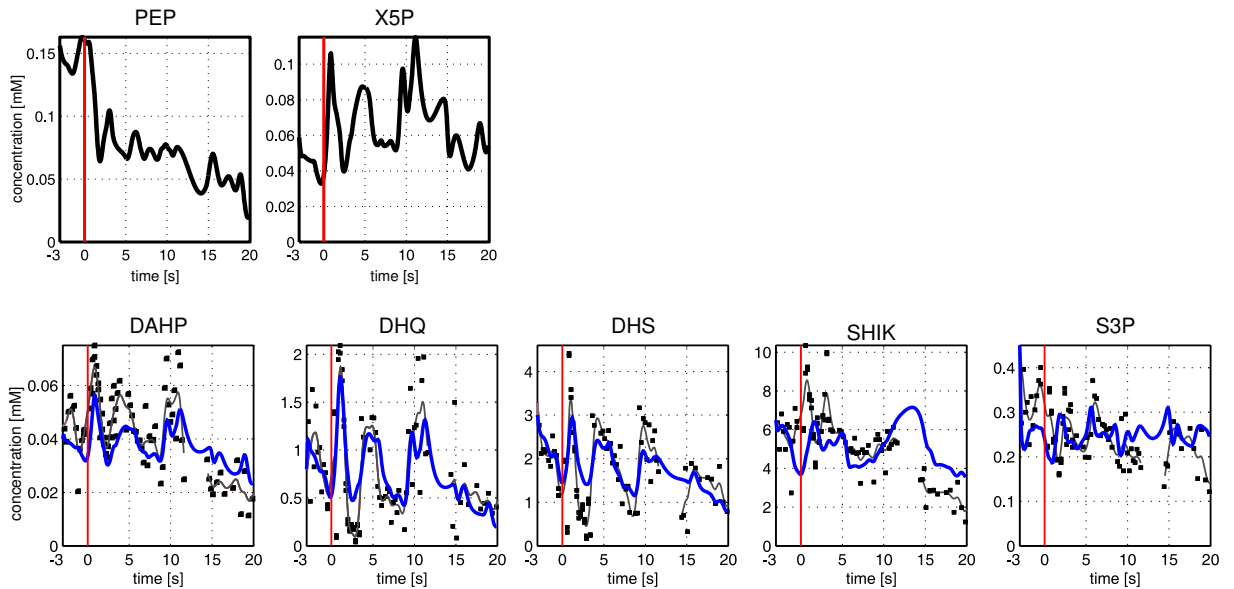


Abbildung 11.3: Messergebnisse der experimentellen Arbeiten in [Oldiges, 2005]. Der rote Balken gibt den Zeitpunkt des Substrat-Pulses an ($t = 0$), die durchgezogenen Linien die Modellvorhersage und die Punkte die gemessenen Daten. Durch die Messdaten von PEP und X5P wurde ein geglätteter Spline gezogen, der in die Simulation mit einfließt.

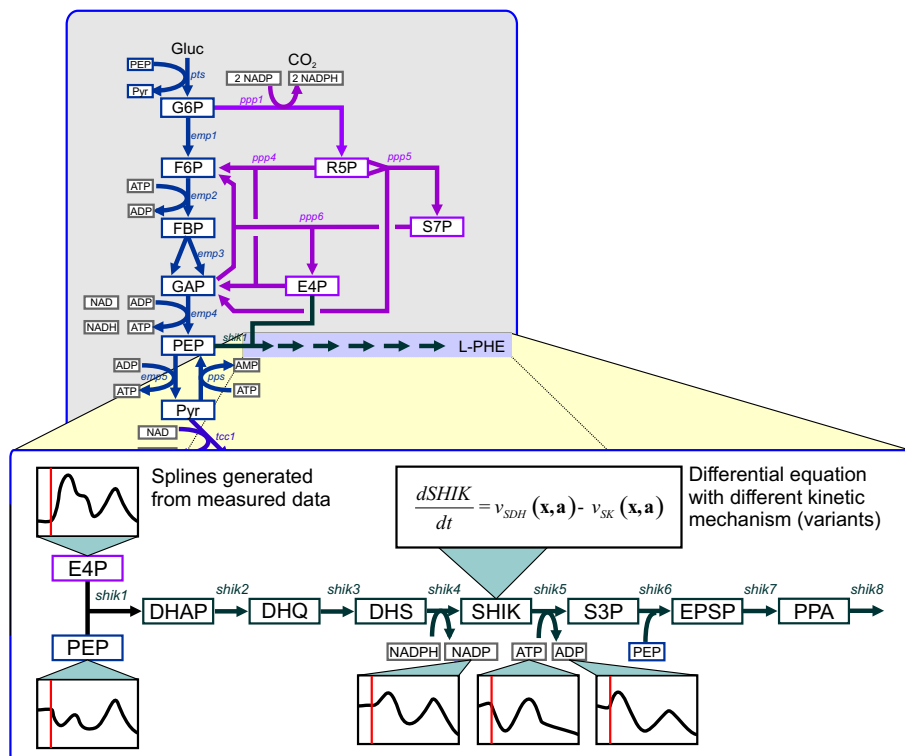


Abbildung 11.4: Übersicht über das metabolische Netzwerk eines L-Phenylalanin produzierenden *E. coli* Stammes. Der Fokus bei der Modellierung liegt auf dem Shikimat-Syntheseweg, der hier aus dem Zentralstoffwechsel heraus vergrößert dargestellt wird.

DHQ DH DHQ \rightleftharpoons DHS	SHIK DH DHS+NADPH \rightleftharpoons SHIK+NADP	SK SHIK+ATP \rightarrow S3P+ADP	EPSP S S3P+PEP \rightleftharpoons EPSP	CHA S,M PPA \rightleftharpoons L-PHE	S3P dephos S3P \rightarrow SHIK
+	+	- PPA - AMP, PPA	- ADP	NADPH dep.	\rightarrow
\rightleftharpoons	\rightleftharpoons	- AMP, PPA + PEP \rightleftharpoons	\rightleftharpoons	\rightleftharpoons	\rightleftharpoons

- Inhibitor \rightleftharpoons Kinetik ohne Effektoren
+ Aktivator ~~\rightleftharpoons~~ Null-Kinetik

Abbildung 11.5: Auflistung der Reaktionsschritte, für die mehrere kinetische Beschreibungen im Modell (Abb. 11.4) existieren.

Die metabolischen Vorläufer im Zentralstoffwechsel sind PEP und E4P. Für die Synthese von L-Phenylalanin werden zusätzlich noch NADPH, ATP und ein weiteres Molekül PEP benötigt. In der Modellierung wurde ein auf diesen Reaktionsweg fokussiertes Modell erstellt. Die Metabolite PEP, E4P, NADPH und ATP, die sich außerhalb des Systems befinden, werden durch geglättete Messdaten in Form von Splines als Eingabe in das System modelliert.

11.2.2 Modellfamilien

Für den Shikimat-Syntheseweg wurden mit Hilfe von Modellfamilien (Kapitel 6) 128 verschiedene Lin-Log Modelle [Visser and Heijnen, 2003] erstellt und angepasst. Eines der Resultate war, dass der zeitliche Verlauf der Metabolite SHIK und S3P nur von solchen Modellen angemessen gut reproduziert werden konnte, in denen die Shikimatkinase-Reaktion durch ADP/AMP und PPA inhibiert wurde. Abb. 11.5 listet die sechs Reaktionsschritte auf, für die mehrere kinetische Beschreibungen existieren.

Im Gegensatz zum Projekt in Abschnitt 11.1 lag in diesem Projekt MMT2 mit der Unterstützung von Modellfamilien vor. Während vorher alle Modellvarianten einzeln von Hand erstellt, simuliert und gegenüber gestellt werden mussten, konnten in diesem Projekt bereits die Automatismen der Modellfamilien genutzt werden. Der Modellbildungsprozess findet hier nicht mehr Modell für Modell statt, sondern Modellfamilie für Modellfamilie. Dadurch ist es möglich eine viel größere Anzahl von Modellen in der gleichen Zeit zu analysieren.

11.2.3 Anpassung an mehrere Experimente

In den experimentellen Arbeiten des Projekts [Oldiges, 2005] wurden verschiedene L-Phenylalanin-Produzenten untersucht. Zwei dieser Datensätze eignen sich zur modellgestützten Auswertung. Um ein Modell zu erstellen, das beide Experimente wiedergeben kann, wurde ein Modell zur gleichzeitigen Anpassung erstellt. Dabei wird angenommen, dass die Enzymeigenschaften, d.h. Substrataffinitäten (K_m -Werte) und Inhibitoraffinitäten (K_I -Werte) in beiden Experimenten identisch sind. Unterschiede

ergeben sich in der Enzymexpression, d.h. der gebildeten Enzymmenge, die sich proportional zum r_{max} Wert verhält.

Dieser Modellierungsansatz konnte mit globalen Parametern (Kap. 7.5) umgesetzt werden und bringt im Vergleich zu den Einzelmodellen (Kap. 11.2.2) einen signifikanten Anstieg der Parameterbestimmtheit.

11.3. *C. glutamicum* L-Valin Produktionsstamm

.....

Das Ziel eines weiteren Projektes [Magnus, 2005] war es, zunächst ein Stimulus-Response-Experiment mit *C. glutamicum* durchzuführen, die analytischen Methoden zur Quantifizierung der Zellmetabolite zu verbessern und schließlich die Messergebnisse mit einem Modell zu beschreiben.

11.3.1 Experiment

Im Rahmen der experimentellen Arbeiten wurden in diesem Projekt erstmals alle Metabolite im Valin- und Leucin-Reaktionsweg (Abb. 11.6) erfolgreich quantifiziert. Des Weiteren wurden auch alle Metabolite und Co-Metabolite der Glykolyse sowie einige Metabolite des TCA-Zyklus und des PPP quantifiziert. Mit diesen Messdaten konnte beobachtet werden, wie sich der Glucose-Puls durch das Reaktionsnetzwerk fortpflanzt, angefangen von der Glucose-Aufnahme über die Glykolyse in den Valin-Weg bis zum Endprodukt Valin.

11.3.2 Modellbildung

Eine weitere Aufgabe war, die Messdaten mit einem Modell nachzubilden. Erstmalig wurde mit den Methoden der experimentellen Modellbildung in dieser Arbeit ein Modell für die Stoffwechselwege erstellt, die für die Valin- und Leucinproduktion zuständig sind. Das Modell simuliert die Reaktionen und Metabolite dieses Weges angefangen vom Pyruvat (Abb.11.7) bis zum Endprodukt.

Die Modellbildung wurde von Anfang an mit MMT2 durchgeführt und wurde anfangs dazu eingesetzt das Verhalten verschiedener reaktionskinetischer Formeln und Mechanismen wie Inhibitionen, Aktivatoren und alternative Reaktionswege auszutesten. Nach dieser Anfangsphase wurde MMT2 zur Modellidentifizierung und Parameteranpassung verwendet. In dieser Arbeit wurde versucht alle verfügbaren Informationen in der Literatur über die jeweiligen Reaktionsschritte dieses Organismus in die Modellbildung mit einzubeziehen. Daher entstanden weniger Modelle als theoretisch durch Kombination denkbar.

Am Ende dieser Reihe von Testmodellen stehen ein mechanistisches- und ein LinLog-Modell, wie in Abb. 11.7 abgebildet. Betrachtet man die Fortschritte in den Testmodellen als Entwicklungsschritte, so wurden für das mechanistische Modell ca. 10 - 15 Entwicklungsschritte durchgeführt und für das LinLog-Modell ca. 25 Ent-

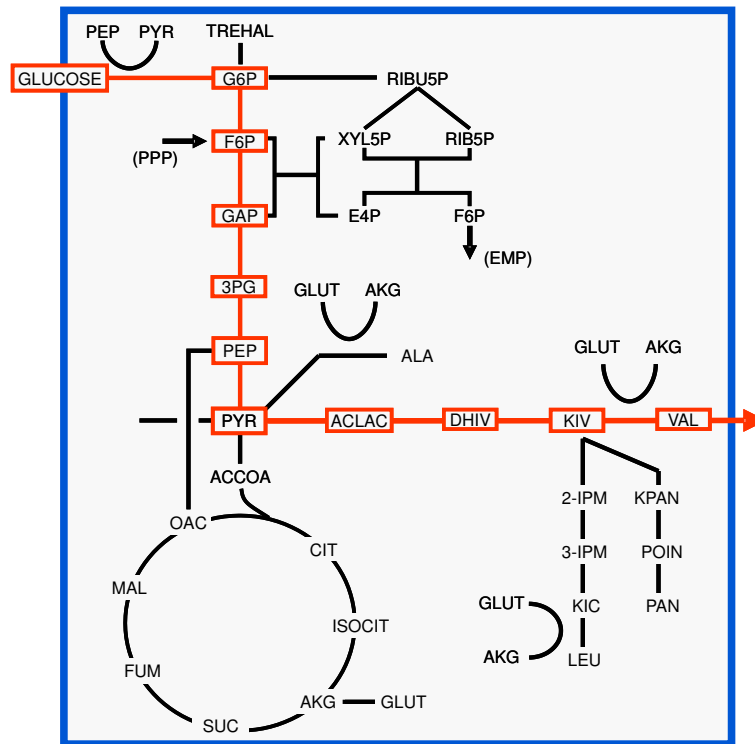


Abbildung 11.6: Vereinfachtes metabolisches Netzwerk eines L-Valin produzierenden *C. glutamicum* Stamm. Der Fokus bei der Modellierung liegt auf dem L-Valin- und L-Leucin-Syntheseweg, von denen der erstere hier aus dem Zentralstoffwechsel heraus rot dargestellt wird.

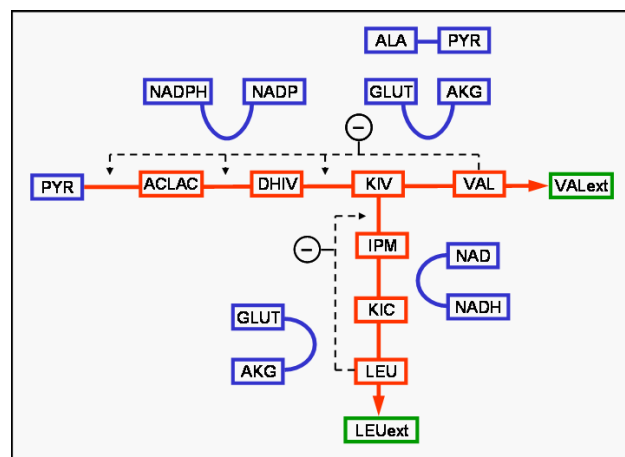


Abbildung 11.7: Das erstellte Modell simuliert sowie die L-Valin- als auch die L-Leucin-Synthese. Die blau dargestellten Metabolite sind von den Messdaten über Splines in das Modell mit aufgenommen worden, die roten sind als Differentialgleichung aufgestellt und die grünen sind die externen Metabolite.

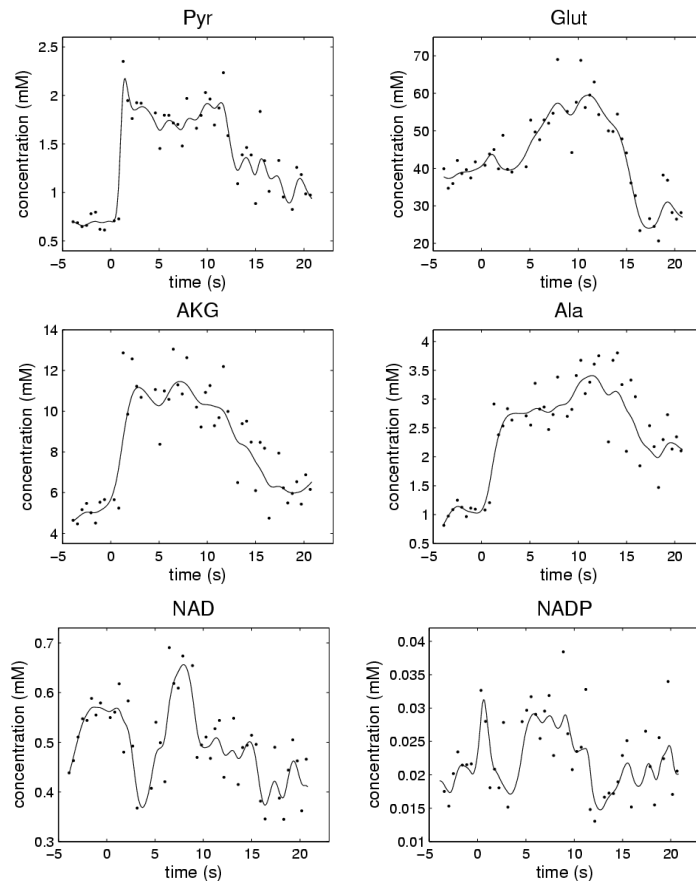


Abbildung 11.8: Mit Hilfe von Splines werden Messdaten in das Modell aufgenommen und die Komplexität des Modells reduziert. Die Punkte stellen die Messergebnisse dar, die in einem Stimulus-Response-Experiment mit einem *C. glutamicum* L-Valin Produktionsstamm gemessen wurden. Die durchgezogenen Linien sind ein geglätteter Spline, der im Modell als Input verwendet wird.

wicklungsschritte. Zählt man alle diese Testmodelle und Kombinationen zusammen, so kommt man auf eine dreistellige Zahl.

Wie auch schon in Abschnitt 11.2.1 wird nur ein Teilbereich (Abb. 11.7) des Metabolismus (Abb. 11.6) modelliert. Dazu werden die Metabolite, die an den Teilbereich angrenzen, im Modell durch einen Spline repräsentiert, der aus den Messdaten erstellt wurde. In Abb. 11.7 sind dies die blauen Metabolitknoten. Für die roten Metabolitknoten wurden aus den reaktionskinetischen Beschreibungen des Modells jeweils eine ODE generiert, die den Konzentrationsverlauf nachbildet. Dieser Verlauf wird nach der Simulation mit den gemessenen Daten verglichen und bewertet. Die Simulation und Parameteranpassung wurden automatisch von MMT2 durchgeführt und waren damit eines der zentralen Hilfsmittel der Modellbildung.

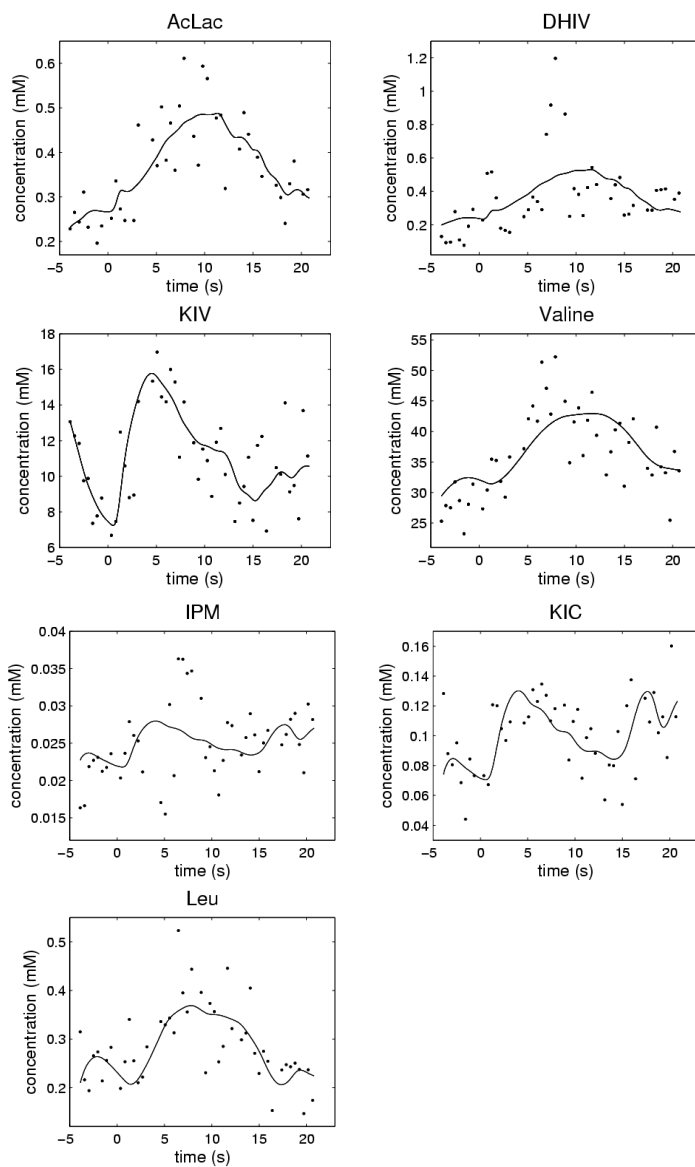


Abbildung 11.9: Wie in Abb. 11.8 stellen die Punkte die Messergebnisse dar, die durchgezogenen Linien sind das vom Modell simulierte Verhalten der Metabolite.

Teil IV

Modellsuche

Formale Beschreibung von Modellfamilien

Im Kapitel 11 wurden drei Projekte vorgestellt, in denen die Modellbildung ein wesentlicher Bestandteil war. Dabei entstanden große Modellfamilien aus vielen ähnlichen Modellen. In dem Projekt aus Kapitel 11.2 kam die Unterstützung von Modellfamilien durch MMT2 bereits zum Einsatz.

In Kapitel 6 wurden Modellfamilien informal anhand eines Beispiels vorgestellt. In diesem Kapitel wird nun ein allgemeiner Formalismus zur Definition von Modellfamilien eingeführt und am Beispiel einer Modellfamilie von Polynomen veranschaulicht. Teile des Inhalts dieses Kapitels wurden in [Haunschild et al., 2005b] und [Haunschild and Wiechert, 2004] publiziert.

12.1. Grundlegende Annahmen

.....

12.1.1 Modelldiskriminierung

Die Modelldiskriminierung befasst sich mit der Auswahl eines mathematischen Modells aus einer Menge von Kandidaten, das eine gegebene Menge an experimentellen Daten am besten beschreibt [Burnham and Anderson, 1998, Borowiak, 1990, Linhart and Zucchini, 1986, Box and Hill, 1967]. Im Allgemeinen wird das am besten passende Modell ausgewählt durch die Minimierung eines statistisch motivierten Auswahlkriteriums.

In Kapitel 9 wurde in Gleichung 9.1 bereits ein Minimierungskriterium vorgestellt, das nur ein einzelnes Modell betrachtet. Hier wird hingegen nicht mehr nur ein konkretes Modell betrachtet, sondern eine ganze Familie von Modellen. Das Modelldiskriminierungsproblem ist dann ein diskret- / kontinuierliches Optimierungsproblem der Art

$$\min_i \min_{\alpha_i} \text{Crit}(M^i, \vec{\alpha}^i, m) \quad (12.1)$$

mit M^i als den i -ten Modell der Menge, $\vec{\alpha}^i$ den Parametervektor dieses Modells, Messdaten m und Crit als Auswahlkriterium. Im Allgemeinen sind die Modelle M^i nicht völlig verschieden, sondern haben viele Gemeinsamkeiten. Die Beziehungen zwischen

den Modellen sollte von einem Optimierungsalgorithmus ausgenutzt werden. Die mathematische Struktur des Modellraums ist das Hauptthema dieses Kapitels.

Das Auswahlkriterium muss einerseits berücksichtigen, dass das Modell möglichst gut auf die Daten passt (Kap. 9), andererseits soll die Komplexität des gewählten Modells niedrig gehalten werden, um ein Overfitting (Kap. 9.3.5) zu vermeiden. Dies erreicht man dadurch, dass die Einführung neuer Parameter in das Modell bestraft wird. Ein bekanntes Auswahlkriterium für Regressionsmodelle ist das Akaike Informationskriterium, das die gewichtete Fehlerquadratsumme eines angepassten Modells berücksichtigt, die Anzahl m der Daten und die Anzahl n der angepassten Parameter des Modells [Burnham and Anderson, 1998]. Wenn die Standardabweichung der Messungen σ bekannt ist, dann gilt [Wiechert and Takors, 2004]:

$$Crit = AIC = \sum_{i=1}^m \left(\frac{y_i - \hat{y}_i(\hat{\alpha})}{\sigma} \right)^2 + 2n \quad (12.2)$$

Mit y_i als das i -te Messdatum, $\hat{\alpha}$ als die Parameterschätzung und \hat{y}_i als Vorhersage der Messungen durch das angepasste Modell. Dies ist nur ein mögliches Kriterium, für das es mehrere Weiterentwicklungen und Alternativen gibt, die in der Literatur vorgeschlagen werden: [Burnham and Anderson, 1998, Borowiak, 1990, Linhart and Zucchini, 1986, Box and Hill, 1967]. Daher beschäftigt sich dieses Kapitel nicht mit einem Auswahlkriterium, sondern mit der allgemeinen Sicht auf das Optimierungsproblem.

Dies zeigt auch die Anwendung vorhandener Modelldiskriminierungskriterien [Burnham and Anderson, 1998], was hier am Beispiel des Akaike Kriteriums, für eine Menge von $m = 1000$ Datenpunkten und einer Modellfamilie mit etwa $n = 100$ Parametern pro Modell gezeigt werden soll. Für ein gut passendes Modell muss die Summe $\sum (y_i - \hat{y}_i)^2 / \sigma^2$ in der Größenordnung von $m - n$ liegen. Aus Gleichung (12.2) wird klar, dass eine Änderung in der Anzahl der Parameter um den Wert 10 viel weniger relevant ist als eine Verbesserung der Fehlerquadratsumme. Daher machen statistische Modelldiskriminierungskriterien keinen großen Unterschied zwischen großen Modellen mit ähnlich großer Anzahl an Parametern. In dieser Situation macht es Sinn erst eine Anzahl gut passende Modelle zu generieren und dann die Entscheidungen durch weitergehende Analyse der Modelle aufgrund systemspezifischen Wissens fortzusetzen.

12.1.2 Standardbeispiele

Die meisten Auswahlkriterien in der statistischen Literatur beschäftigen sich mit kleinen Familien von Modellen. Das am häufigsten diskutierte Beispiel ist das Anpassen von Polynomen

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 \quad (12.3)$$

mit aufsteigendem Grad n an ein gegebenen Datensatz. Aus Sicht der benötigten Rechenzeit auf heutigen Computern ist dies eine eher einfache Aufgabe, weil das Anpas-

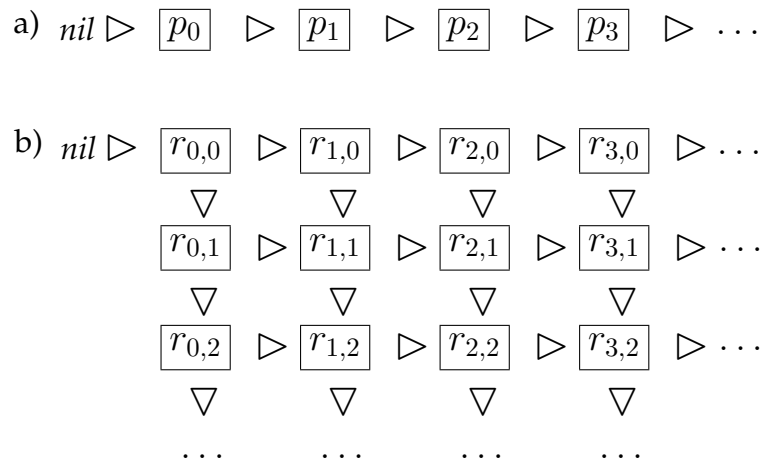


Abbildung 12.1: Struktur eines Modellraums für Modellfamilien von a) Polynomen und b) rationalen Polynomen

sen von Polynomen ein lineares Regressionsproblem ist und daher die Berechnungszeit für eine verschachtelte Sequenz

$$p_0 \triangleright p_1 \triangleright p_2 \triangleright \dots \tag{12.4}$$

von Polynomen fast vernachlässigt werden kann. Die Situation ist eine völlig andere, wenn große Mengen an Messdaten und hunderte von Modellkandidaten zur Auswahl stehen, die zudem nichtlinear und nicht sequentiell verschachtelt sind. Ein bekanntes aber immer noch einfaches Beispiel einer nicht sequentiellen Modellverschachtelungsstruktur ist gegeben durch rationale Polynome

$$r_{n,m}(x) = c \cdot \frac{x^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0}{x^m + b_{m-1}x^{m-1} + \dots + b_1x + b_0} \tag{12.5}$$

mit einer Verschachtelungsstruktur, die durch die folgende Relation gegeben ist:

$$r_{n,m} \triangleright r_{n+1,m} \quad \text{und} \quad r_{n,m} \triangleright r_{n,m+1} \tag{12.6}$$

Die Verallgemeinerungsrelation \triangleright sagt aus, dass das Modell auf der rechten Seite als eine Erweiterung des Modells auf der linken Seite betrachtet werden kann und daher komplexer ist. Es gibt Modelle wie $r_{2,1}$ und $r_{1,2}$, die keine direkte Relation verbindet, obwohl ein Modell $r_{1,1}$ existiert, die beide spezialisiert. Dies zeigt, dass es sinnvoll ist, einen gerichteten Modellverallgemeinerungsgraphen für die betrachtete Modellfamilie einzuführen (Abb. 12.1a und 12.1b). In den folgenden Abschnitten wird dies im Detail beschrieben.

12.1.3 Übergang zu großen Modellen

Rationale Polynome sind immer noch ein einfaches Beispiel nichtlinearer Modelle. Obwohl bekannt ist, dass das Anpassen rationaler Funktionen ein heikles Optimierungsproblem ist [Maehly, 1960], ist der Rechenaufwand zur Anpassung und Auswahl eines Modells für heutige Verhältnisse nicht mehr hoch. Die Situation ändert sich aber erheb-

lich, wenn das System kein Blackbox-Modell darstellt; wie einfache, rationale, trigonometrische oder exponentielle Polynome, Splines, radiale Basisfunktionen usw.; sondern die Modellkandidaten aus einem strukturierten Modellierungsansatz entstanden sind. In dieser Situation können die Modelle sehr komplex werden, hochdimensional, nichtlinear und gewöhnlicherweise implizit definiert. Ein Beispiel einer Modellfamilie von biochemischen Netzwerken wurde bereits in Kapitel 6 betrachtet. In dieser Situation ist nicht mehr klar, wie genau eine Verallgemeinerungsrelation aussehen muss und wie in dem Raum der möglichen Modelle navigiert werden sollte. Des Weiteren kann der Rechenaufwand erheblich sein - selbst für die Auswertung eines einzelnen Modells als Teil der Modelldiskriminierung.

Ein weiteres gut bekanntes Problem ist, dass nichtlineare Parameteranpassungen üblicherweise schlecht konditionierte Probleme darstellen, weil viele der Parameter durch die Daten nicht gut bestimmt sind. Ganz besonders bei Modellen, die nicht gut auf die Daten passen, besteht die Optimierungslandschaft aus vielen lokalen Optima. Es liegt in der Natur der Modelldiskriminierungsprozesse, dass schlecht passende Modelle eher die Regel als die Ausnahme sind. Dies muss bei der Entwicklung von Algorithmen bedacht werden.

Beispiele aus der Literatur sind [Takors et al., 2001], wo die Diskriminierung reaktionskinetischer Modelle in der Biotechnologie behandelt wird und [Leifheit and King, 2004, King et al., 2002], wo ein Framework zur (semi) automatischen Identifizierung von chemischen Reaktionsmodellen beschrieben wird. Dieses basiert auf einem allgemeinen kombinatorischen Formalismus um eine große Anzahl von Modellen zu erstellen, in dem die Beziehungen zwischen den Modellen aber nicht weiter berücksichtigt werden.

In [Hoffmann et al., 2004] wird ein evolutionärer Algorithmus für die Diskriminierung von ökologischen Modellen vorgestellt, bei dem angenommen wird, dass ein einzelner Parametervektor für alle Modelle gewählt werden kann. Dies kann aber nicht auf die biochemischen Modelle angewendet werden, die in Kapitel 3.4 (Kapitel 6.3) eingeführt wurden, weil jede Modellvariante eine Menge an inaktiven Modellparametern hat, die nur in anderen Modellvarianten aktiv sind. Um die Dimension des kontinuierlichen Optimierungsproblems niedrig zu halten und auch um die inaktiven Parameter nicht unnötigerweise zu verändern, muss jede Modellvariante einen zugeschnittenen Parametervektor haben (Abschnitt 12.3.2).

Aus diesen Betrachtungen wird klar, dass für hochdimensionale Modelldiskriminierungs-Algorithmen eine hohe Rechenleistung benötigt wird. Andererseits ist das Problem wie geschaffen für die Verteilung auf ein Computecluster oder ein Grid (Kap. 15), denn die einzelne Aufgabe der Modellauswertung ist eine Primitivoperation im Modelldiskriminierungsprozess.

Wegen der Komplexität, der hohen Dimensionalität und der schlechten Konditionierung des Problems ist es klar, dass es nicht möglich ist innerhalb einer vertretbaren Rechenzeit das globale Minimum zu finden. Das bedeutet auch, dass Heuristiken entwickelt werden müssen um wenigstens ein paar gut passende Modelle mit den vor-

handenen Ressourcen zu finden. Die Aufgabe solcher Algorithmen sollte zudem eher eine bewertete Rangfolge der besten Kandidaten sein, als nur eine einzelne Lösung auszugeben.

12.2. Allgemeine Struktur von Modellfamilien

.....

12.2.1 Familien von Regressionsmodellen

Eine allgemeine Form nichtlinearer Regressionsmodelle ist gegeben durch [Seber and Wild, 1989]

$$M : y = f(\alpha, u) + \epsilon \quad (12.7)$$

mit den bekannten Systemvariablen u , unbekanntem Systemparametern α , gemessenen Größen y und Messrauschen ϵ . Diese Beschreibung deckt explizite Regressionsmodelle, ODEs, DAEs und PDEs ab, sowohl in der stationären, als auch in der zeitabhängigen Situation. Betrachtet man nicht mehr einzelne Modelle, sondern Modellfamilien, gibt es eine diskrete Indexmenge Ind , so dass die Modellfamilie gegeben ist durch

$$M^i : y = f^i(\alpha^i, u) + \epsilon, \quad i \in Ind \quad (12.8)$$

mit den Vektoren α^i , die verschiedene Dimensionen $\dim \alpha^i$ haben können. Später wird es wichtig sein, dass die Indexmenge ein Element nil enthält, so dass das korrespondierende Modell M^{nil} ein leeres Modell ohne Parameter darstellt:

$$nil \in Ind, \quad \dim \alpha^{nil} = 0 \quad (12.9)$$

Das Modell M^{nil} kann als das einfachste Modell in der Modellfamilie betrachtet werden, das bezüglich der Verallgemeinerungsrelation das minimale Element darstellt. Die Einführung des leeren Modells ist nur aus technischen Gründen erforderlich, die später klarer werden.

12.2.2 Modellabhängigkeitsgraph

Die Indexmenge Ind wird erweitert um die Struktur eines gerichteten Graphen - den Modellabhängigkeitsgraphen - mit der Erweiterungsrelation:

$$\triangleright \subset Ind \times Ind. \quad (12.10)$$

Im Modellabhängigkeitsgraph bedeutet $i \triangleright j$, dass das Modell M^j eine direkte Erweiterung des Modells M^i ist. „Direkt“ bedeutet, dass es kein Modell in der Modellfamilie gibt, das sich „zwischen“ M^i und M^j befindet und damit sowohl M^i verallgemeinert als auch M^j vereinfacht.

Das nil -Modell sollte zumindest mit einem anderen Modell verbunden sein und sollte kein anderes Modell verallgemeinern. Es ist das minimale Element:

$$\begin{aligned} nil \triangleright i & \text{ für mindestens ein } i \in Ind - \{nil\} \\ i \triangleright nil & \text{ für kein } i \in Ind - \{nil\} \end{aligned} \quad (12.11)$$

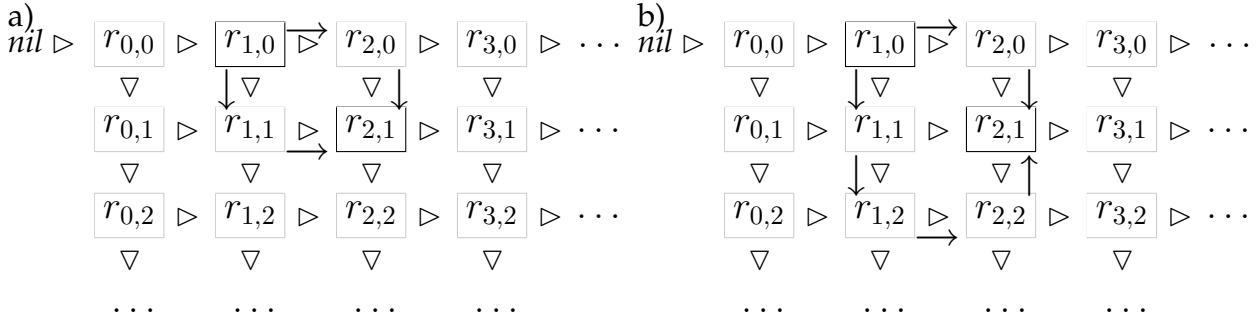


Abbildung 12.2: Navigation im Modellraum von (1,0) nach (2,1) mit a) Modellverallgemeinerungen und b) sowie Modellverallgemeinerungen und Modellvereinfachungen zusammen

Die reflexive und transitive Hülle der direkten Abhängigkeitsrelation wird ausgedrückt durch \triangleright^* . Damit beschreibt $i \triangleright^* j$ die Existenz eines Pfades $i \triangleright k_1 \triangleright k_2 \triangleright \dots \triangleright k_n \triangleright j$ im Modellabhängigkeitsgraphen. Das Modell M^j verallgemeinert Modell M^i über eine Folge von direkten Verallgemeinerungsschritten. Zwei einfache Beispiele sollen dieses Konzept illustrieren:

1. In der Menge der Polynome (Gleichung 12.3) wird das Polynommodell vom Grad n verallgemeinert durch das Polynommodell vom Grad $n + 1$:

$$\begin{aligned} \text{Ind} &= \{nil, 0, 1, 2, \dots\} \\ \text{mit } \begin{cases} i \triangleright i' & \text{iff } i' = i + 1 \vee (i = nil \wedge i' = 0) \\ nil \triangleright 0 \end{cases} & \quad (12.12) \end{aligned}$$

Damit gilt,

$$i \triangleright^* i' \quad \text{iff } i = nil \vee (i, i' \neq nil \wedge i < i'), \quad (12.13)$$

ein Polynom wird also verallgemeinert von jedem anderen Polynommodell gleichem oder höherem Grades (Abb. 12.1a).

2. Im Fall der rationalen Polynome (Gleichung 12.5) ist die Indexmenge gegeben durch

$$\begin{aligned} \text{Ind} &= \{nil\} \cup \{0, 1, 2, \dots\} \times \{0, 1, 2, \dots\} \\ \text{mit } \begin{cases} (i, j) \triangleright (i', j') & \text{iff } (i' = i + 1 \wedge j' = j) \vee (i' = i \wedge j' = j + 1) \\ nil \triangleright (0, 0) \end{cases} & \quad (12.14) \end{aligned}$$

Der korrespondierende Abhängigkeitsgraph wird in Abb. 12.1b gezeigt. Im Gegensatz zur Familie der Polynommodelle gibt es verschiedene Wege ein Modell vom Grad (i', j') als Verallgemeinerung eines anderen Modells vom Grad (i, j) zu beschreiben:

$$\begin{cases} (i, j) \triangleright^* (i', j') & \text{iff } (i \leq i' \wedge j \leq j') & \forall (i, j), (i', j') \in \text{Ind} - \{nil\} \\ nil \triangleright^* (i, j) & & \forall i', j' \in \text{Ind} \end{cases} \quad (12.15)$$

Als Beispiel kann das rationale Polynommodell vom Grad (2,1) vom Modell (1,0) über zwei Sequenzen von direkten Verallgemeinerungsschritten erreicht werden (Abb. 12.2a):

$$(1, 0) \triangleright (2, 0) \triangleright (2, 1) \quad \text{und} \quad (1, 0) \triangleright (1, 1) \triangleright (2, 1) \quad (12.16)$$

12.2.3 Bidirektionale Navigation

Für einen Suchalgorithmus wird es wichtig sein in zwei Richtungen durch den Modellabhängigkeitsgraphen navigieren zu können. Sowohl in der Richtung von Modellverallgemeinerungen als auch in der Richtung von Modellvereinfachungen. Aus diesem Grund wird das Symbol \diamond eingeführt für die symmetrische nichtgerichtete Relation

$$i \diamond j \quad \text{iff} \quad i \triangleright j \vee j \triangleright i \quad (12.17)$$

die bedeutet, dass M^i entweder eine direkte Vereinfachung oder eine direkte Verallgemeinerung von M^j ist. Genauso wird \diamond^* definiert als transitive und reflexive Hülle von \diamond .

Um ein Modell von einem anderen Modell aus zu erreichen, sollte der Modellabhängigkeitsgraph spezielle Verbindungseigenschaften aufweisen. Es sollte möglich sein jedes Modell M^i von jedem anderen Modell M^j über eine Sequenz von Verallgemeinerungs- und Vereinfachungsschritten zu erreichen, d. h. er sollte also zusammenhängend sein:

$$i \diamond^* j \quad \forall i, j \in \text{Ind} \quad (12.18)$$

Eine ausreichende Bedingung für Gleichung 12.18 ist, dass das *nil*-Modell jedes andere Modell vereinfacht.

$$\text{nil} \triangleright^* i \quad \forall i \in \text{Ind} \quad \Rightarrow \text{eq. (12.18)} \quad (12.19)$$

12.2.4 Gültige Parametersätze, Inklusions- und Projektionsfunktion

Über die Spezifikation des Modellabhängigkeitsgraphen hinaus wird eine weitere strukturierende Information benötigt, um während der Navigation im diskreten Raum der Modelle, die Parametersätze α^i mitzunehmen. Diese Information bildet den Parameterraum eines Modells auf den Parameterraum eines benachbarten Modells ab. Der Raum von möglichen Parametersätzen des Modells M^i wird gegeben durch eine Teilmenge

$$\text{Par}^i \subset \mathbb{R}^{\text{dim}\alpha^i} \quad (12.20)$$

die von der Anwendung abhängig ist. Der Raum möglicher Parametersätze kann durch verschiedene Weisen definiert werden durch implizite oder explizite Relationen. Dies hat natürlich einen Einfluss auf die Wahl eines Optimierungsalgorithmus für Probleme mit Nebenbedingungen und muss daher auch formal beschrieben werden, was

an dieser Stelle aber nicht im Detail ausformuliert wird, um den Formalismus einfach zu halten.

Eine wichtige Voraussetzung für jeden Suchalgorithmus ist, dass das Umschalten von einem Modell M^i zu einer direkten Verallgemeinerung M^j oder umgekehrt von einem M^j zu einer direkten Vereinfachung M^i mit einer Abbildung des Parametervektors von α^i zu α^j und umgekehrt verbunden ist. Diese Parameterabbildung soll nun formal beschrieben werden durch:

$$\text{eine Inklusionsfunktion } \iota_{i,j} : \text{Par}^i \rightarrow \text{Par}^j \quad (12.21)$$

$$\text{und eine Projektionsfunktion } \pi_{j,i} : \text{Par}^j \rightarrow \text{Par}^i \quad (12.22)$$

Unter Ausnutzung dieser Struktur kann ein Suchalgorithmus die Information über bereits gefundene „gute“ Parameter des Modells M^i als Startwerte für ein einfacheres oder komplexeres Modell übernehmen. Die zwei Funktionen $\iota_{i,j}$ und $\pi_{j,i}$ müssen überall dort definiert sein, wo die Relation $i \triangleright j$ gilt. Natürlich soll die Abbildung jedes gültigen Parametersatzes wieder gültig sein. In den Polynombeispielen können die Inklusions- und Projektionsfunktion wie folgt definiert werden:

1. Für einfache Polynome:

$$\iota_{\text{nil},0} : () \rightarrow (a_0^{\text{default}}) \quad (12.23)$$

$$\iota_{n,n+1} : (a_0, \dots, a_n) \rightarrow (a_0, \dots, a_n, a_{n+1}^{\text{default}}) \quad (12.24)$$

$$\pi_{0,\text{nil}} : (a_0) \rightarrow () \quad (12.25)$$

$$\pi_{n+1,n} : (a_0, \dots, a_n, a_{n+1}) \rightarrow (a_0, \dots, a_n) \quad (12.26)$$

2. Im Fall der rationalen Polynome:

$$\iota_{\text{nil},(0,0)} : () \rightarrow (c^{\text{default}}) \quad (12.27)$$

$$\begin{aligned} \iota_{(n,m),(n+1,m)} : (c, a_0, \dots, a_{n-1}, b_0, \dots, b_{m-1}) \rightarrow \\ (c, a_0, \dots, a_{n-1}, a_n^{\text{default}}, b_0, \dots, b_{m-1}) \end{aligned} \quad (12.28)$$

$$\begin{aligned} \iota_{(n,m),(n,m+1)} : (c, a_0, \dots, a_{n-1}, b_0, \dots, b_{m-1}) \rightarrow \\ (c, a_0, \dots, a_{n-1}, b_0, \dots, b_{m-1}, b_m^{\text{default}}) \end{aligned} \quad (12.29)$$

$$\pi_{\text{nil},(0,0)} : (c) \rightarrow () \quad (12.30)$$

$$\begin{aligned} \pi_{(n+1,m),(n,m)} : (c, a_0, \dots, a_{n-1}, a_n, b_0, \dots, b_{m-1}) \rightarrow \\ (c, a_0, \dots, a_{n-1}, b_0, \dots, b_{m-1}) \end{aligned} \quad (12.31)$$

$$\begin{aligned} \pi_{(n,m+1),(n,m)} : (c, a_0, \dots, a_{n-1}, b_0, \dots, b_{m-1}, b_m) \rightarrow \\ (c, a_0, \dots, a_{n-1}, b_0, \dots, b_{m-1}) \end{aligned} \quad (12.32)$$

In den zwei Beispielen werden die Funktionen auf triviale Weise definiert durch Abbildung der Parameter des einen Modells auf die entsprechenden Parameter des

anderen Modells. Bei der Inklusionsfunktion müssen bei dem größeren Modell für die hinzugekommenen Parameter neue Werte erzeugt werden. Hier werden Defaultwerte eingefügt, die von der Anwendung her sinnvoll vorgegeben sein müssen. An dieser Stelle wird es klar, warum der Modellabhängigkeitsgraph ein minimales Element *nil* hat. Zusammen mit der Bedingung, dass der Graph zusammenhängend sein muss, wird dadurch ermöglicht, dass jedes Modell M^i zu Beginn einer Modellsuche durch Defaultparameter parametrisiert werden kann.

12.2.5 Mögliche Erweiterungen für das Konzept

Sicherlich sind intelligentere Parameterabbildungsstrategien denkbar, wenn weitere Information über das Modellverhalten verfügbar ist. Eine Möglichkeit wäre, dass die Inklusions- und Projektionsfunktionen nicht nur von den Parameterwerten abhängen, sondern auch vom gegenwärtigen Stand und Fortschritt des Optimierungsalgorithmus. Weiterhin könnte es Sinn machen bei der Anwendung eines evolutionären Optimierungsalgorithmus nicht nur einen einzelnen Parametervektor, sondern eine Menge von Vektoren durch die Inklusions- und Projektionsfunktionen zu erzeugen. Diese Details sollen aber nicht formalisiert werden um die Notation nicht zu überladen.

Wie bereits erwähnt wurde, kann ein Optimierungsalgorithmus von einem Modell M^i zu einem anderen Modell M^j über mehrere Wege gelangen (Abb. 12.2b). Es macht Sinn zu fordern, dass die daran beteiligten Inklusions- und Projektionsfunktionen zum gleichen Resultat führen müssen. Zum Beispiel sollte für die zwei Pfade in Abb. 12.2b gelten:

$$\iota_{(2,0),(2,1)} \circ \iota_{(1,0),(2,0)} = \pi_{(2,2),(2,1)} \circ \iota_{(1,2),(2,2)} \circ \iota_{(1,1),(1,2)} \circ \iota_{(1,0),(1,1)} \quad (12.33)$$

Solche Bedingungen sind natürlich erfüllt in den Beispielen aus Abschnitt 12.1.2 mit einfachen und rationalen Polynomen. Für die weiterhin beschriebenen Methoden sind diese pfadabhängigen Resultate jedoch nicht unbedingt notwendig. Daher wird diese Forderung als optional angesehen obwohl sie wichtig sein könnte für zukünftige Erweiterungen.

12.2.6 Strukturell gültige Modelle

Das letzte strukturierende Element, das noch eingeführt werden muss, ist die Beschränkung des Modellsuchraums $M^i, i \in Ind$ analog zu der Beschränkung des Parameterraums durch die Menge Par^i für jedes einzelne Modell. Es wird später noch klarer, dass es für komplexe Modelle häufig der Fall ist, dass ein Modell M^i formal konstruiert werden kann, es aber möglich ist ohne Simulation zu entscheiden, ob dieses Modell bereits physikalisch falsch ist. Dann macht es keinen Sinn dieses Modell überhaupt in den Suchraum mit aufzunehmen.

Solche Modelle können aber trotzdem als „Relaisstationen“ in Suchalgorithmen verwendet werden wie auch ungültige Punkte bei kontinuierlichen Optimierungsalgorithmen mit Nebenbedingungen. Dadurch wird es möglich von einem Modell M^i zu einem anderen Modell M^j über möglicherweise ungültige Hilfsmodelle zu navigieren,

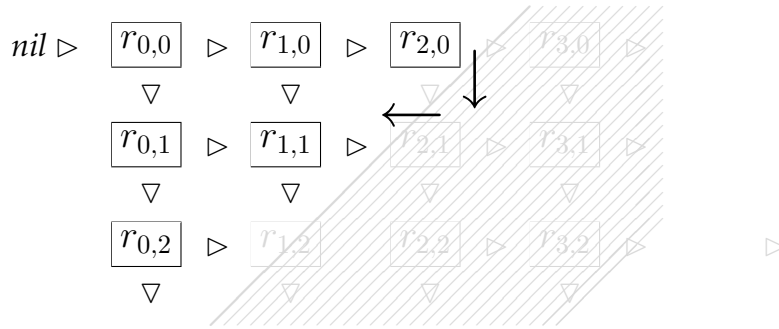


Abbildung 12.3: Navigation durch das Gebiet von ungültigen Modellen

wenn M^i und M^j nicht direkt miteinander verbunden sind. Die Menge von ungültigen Modellen wird formal beschrieben durch eine Teilmenge

$$Feas \subset Ind. \tag{12.34}$$

Das nil -Modell ist natürlich immer ein ungültiges Modell:

$$nil \notin Feas \tag{12.35}$$

Im Fall von einfachen oder rationalen Polynomen ist das Konzept der ungültigen Modelle eher akademisch wie auch das folgende Beispiel, in dem die Gesamtzahl der Parameter α beschränkt ist, damit der Suchraum endlich bleibt. Dann gilt

1. im Falle von einfachen Polynomen

$$Feas = \{0, 1, 2, \dots, N - 1\} \tag{12.36}$$

2. und im Falle von rationalen Polynomen

$$Feas = \{(i, j) \mid i + j \leq N - 1\} \tag{12.37}$$

Abb. 12.3 illustriert den Modellraum für rationale Polynome mit $N = 3$. Es gibt hier die Möglichkeit von $(2, 0)$ über $(2, 1)$ nach $(1, 1)$ zu navigieren. Ansonsten würde $(2, 0)$ eine „Sackgasse“ im Modellabhängigkeitsgraphen darstellen.

12.3. Anwendungsbeispiel: Metabolische Modelle

12.3.1 Varianten von metabolischen Modellen

Bisher wurden die Konzepte an sehr simplen und klassischen Beispielen von einfachen und rationalen Polynomen veranschaulicht. Um zu beschreiben wie das Framework im Zusammenhang mit komplexen und großen Modellen funktioniert, wird es anhand eines Beispiels von metabolischen Modellen veranschaulicht. Die Vielfalt der Modelle ergibt sich durch die folgenden strukturellen Elemente:

1. **Strukturelle Varianten:** Einzelne Reaktionsschritte könnten im Netzwerk vorhanden sein, oder nicht vorhanden sein (Variation der Netzwerkstruktur). Daher wird zunächst ein maximales Netzwerk definiert, das alle möglichen Netzwerke vereint.

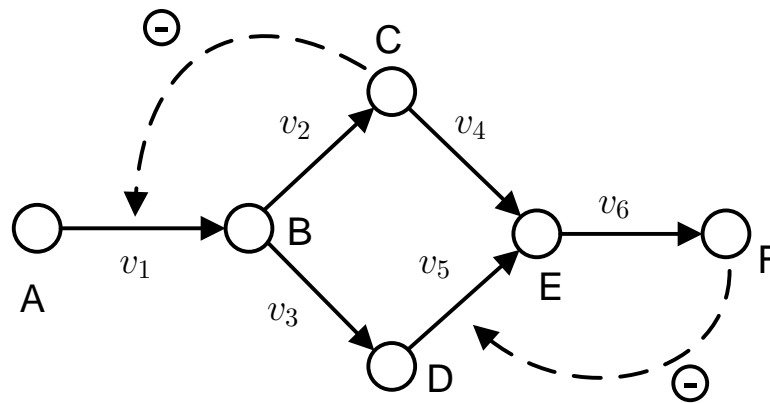


Abbildung 12.4: Ein metabolisches Netzwerk mit zwei alternative Pfaden. Die Kreise stellen die Metabolite dar, durchgezogene Linien die Reaktionen und gestrichelte Linien die regulatorischen Beziehungen.

2. **Stöchiometrische Gültigkeit:** Ein einzelnes Reaktionsnetzwerk sollte zumindest eine sinnvolle Lösung für einen stationären Zustand besitzen, der ungleich Null ist und bei dem alle Flüsse in ihre vorgegebene Richtung fließen. Eine Ausnahme bilden bidirektionale Flüsse (die spezifiziert werden müssen), welche die Möglichkeit haben vorwärts oder rückwärts zu fließen.
3. **Benutzerdefinierte Einschränkungen:** Der Benutzer kann zusätzliche Einschränkungen für die Netzwerktopologie spezifizieren um einer kombinatorischen Explosion, die durch die Anwendung von Punkt 1 entstehen kann, entgegenzuwirken. Beispielsweise zeigt Abb. 12.4 zwei alternative Pfade von Metabolit B nach E. Der Modellierer möchte aber nur jeweils einen dieser Pfade aktivieren, wodurch das maximale Netzwerk ungültig wird.
4. **Regulatorische Struktur:** Die regulatorischen Einflüsse von Metaboliten auf Reaktionsschritte werden üblicherweise durch Effektoranten im Reaktionsnetzwerk ausgedrückt (Abb. 12.4). Diese Einflüsse können inhibierend oder aktivierend sein. Das Ein- und Ausschalten dieser Einflüsse wird über die kinetische Beschreibung der Reaktion vorgenommen.
5. **Reaktionskinetiken:** Die Flussrate jeder Reaktion wird durch eine reaktionskinetische Formel $v_i(x)$ ausgedrückt mit x als Vektor aller Metabolitkonzentrationen (Gleichung 9.2). Die Flussraten v_i werden dabei nur von den jeweiligen Effektoranten der Reaktion beeinflusst.

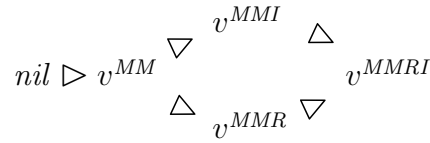


Abbildung 12.5: Abhängigkeiten zwischen kinetischen Beschreibungen

12.3.2 Formale Beschreibung der Reaktionskinetiken

Die allgemeine Struktur reaktionskinetischer Terme suggeriert die Einführung einer Familie von kinetischen Formeln für jeden Reaktionsschritt im Netzwerk. Kinetik j für den Reaktionsschritt i wird beschrieben durch ein Modell

$$M^{i,j}, \quad \begin{array}{l} i \in \{1, \dots, N\} \\ j \in \text{Ind}^i = \{\text{nil}\} \cup \{1, \dots, N^i\} \end{array} \quad (12.38)$$

Hier ist N die Anzahl der Reaktionsschritte im Netzwerk und N_i die Anzahl von Alternativen für Reaktionsschritt i . Die zugehörigen Parametervektoren sind gegeben durch

$$\alpha^{i,j} \in \text{Par}^{i,j} \subset \mathbb{R}^{\dim \alpha^{i,j}}. \quad (12.39)$$

Für jeden Reaktionsschritt wird eine *nil*-Kinetik eingeführt, mit der die Reaktion ein- oder ausgeschaltet werden kann. Definiert werden müssen zudem die Verallgemeinerungsrelationen

$$M^{i,j} \triangleright_i M^{i,j'}, \quad (12.40)$$

und die Inklusions- und Projektionsfunktionen

$$\iota_{j,j'}^i : \text{Par}^{i,j} \rightarrow \text{Par}^{i,j'} \quad (12.41)$$

$$\pi_{j',j}^i : \text{Par}^{i,j'} \rightarrow \text{Par}^{i,j}. \quad (12.42)$$

Sie müssen die gleichen Bedingungen erfüllen wie die Funktionen in Abschnitt 12.2.4. Hier muss der ausgeschaltete Reaktionsschritt (*nil*-Kinetik) mit mindestens einer anderen Kinetik verbunden werden, damit eine Parameterisierung durch Defaultwerte vorgenommen werden kann, wenn kein Vorgängermodell existiert.

Am Beispiel der Familie der Michaelis-Menten Kinetiken wird in Abb. 12.5 die Struktur des Raumes veranschaulicht. Die Projektionsoperationen werden definiert durch Weglassen des Parameters, der in der einfacheren Kinetik nicht enthalten ist. Die Inklusionsoperation wird definiert durch Belegung der neuen Parameter mit Defaultwerten.

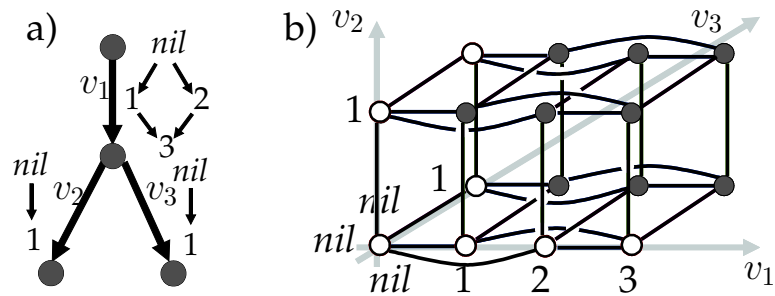


Abbildung 12.6: Modellraum visualisiert als Hypercube. a) Metabolisches Netzwerk mit kinetischen Varianten für jeden Reaktionsschritt. b) Zugehöriger Modellabhängigkeitsgraph. Gefüllte Kreise stellen gültige Modelle dar, leere Kreise stellen ungültige Modelle dar.

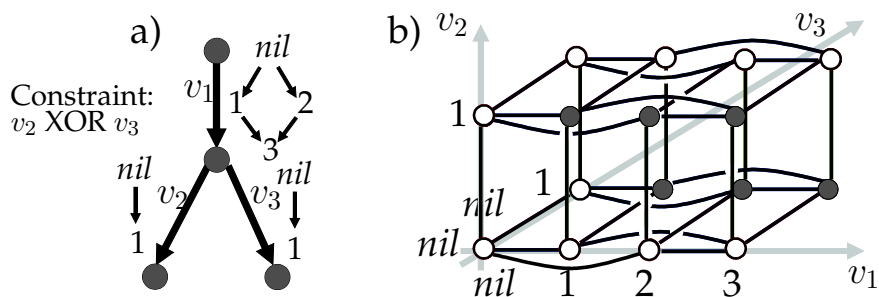


Abbildung 12.7: Die gleiche Situation wie in Abb. 12.6, mit einer zusätzlichen benutzerdefinierten Einschränkung, die festlegt, dass genau einer der Reaktionsschritte v_2 und v_3 aktiv sein muss (Gleichung. 6.5).

12.3.3 Formale Beschreibung der metabolischen Modellvarianten

Mit der Familie der Reaktionskinetiken für jeden Reaktionsschritt ist es jetzt möglich den Raum der Netzwerkmodelle als kartesisches Produkt zu definieren (Abb. 12.6):

$$Ind \stackrel{def}{=} Ind^1 \times \dots \times Ind^N \quad (12.43)$$

$$nil \stackrel{def}{=} (nil, \dots, nil) \quad (12.44)$$

$$(j_1, \dots, j_N) \triangleright (j'_1, \dots, j'_N) \text{ iff } \begin{cases} j_k \triangleright_k j'_k & \text{für genau ein } k \\ j_k = j'_k & \text{sonst.} \end{cases} \quad (12.45)$$

$$l_{(j_1, \dots, j_N), (j'_1, \dots, j'_N)}^i \stackrel{def}{=} l_{j_1, j'_1}^i \times \dots \times l_{j_N, j'_N}^i \quad (12.46)$$

$$\pi_{(j'_1, \dots, j'_N), (j_1, \dots, j_N)}^i \stackrel{def}{=} \pi_{j'_1, j_1}^i \times \dots \times \pi_{j'_N, j_N}^i \quad (12.47)$$

Der resultierende Modellraum wird normalerweise sehr groß und enthält viele ungültige Modelle. Zwei Restriktionsmechanismen werden eingeführt um einen Raum von gültigen Modellen zu definieren:

1. Jede irreversible Reaktionskinetik spezifiziert eine nominelle Richtung im Netzwerk. Beispielsweise schließt die Michaelis-Menten Kinetik (3.7) automatisch die Rückwärtsrichtung aus, während die reversible Michaelis-Menten Kinetik (3.8)

sie zulässt. Eine weitere notwendige Bedingung für die physikalische Gültigkeit eines Modells ist, dass das Netzwerk zumindest einen stationären Zustand besitzt, in dem insbesondere jede irreversible Reaktion in ihre nominelle Richtung fließt. Dies kann relativ einfach über die stöchiometrische Gleichung

$$N \cdot \vec{v} = 0 \quad (12.48)$$

ausgedrückt werden, so dass gilt

$$\begin{aligned} a. \quad & \vec{v} \neq 0 \\ b. \quad & v_i \geq 0 \quad \text{für jeden irreversiblen Reaktionsschritt} \\ c. \quad & v_i = 0 \quad \text{wenn Reaktionsschritt } i \text{ deaktiviert ist.} \end{aligned} \quad (12.49)$$

Diese Bedingung kann in einfacher Weise durch Berechnung der so genannten Elementarmoden [Schuster et al., 1999, Klamt and Gilles, 2004] geprüft werden, die bereits in Kapitel 6.7 vorgestellt wurden. Diese Moden müssen nur einmalig für das maximale Netzwerk berechnet werden und können dann auf jedes Subnetzwerk angewandt werden. Abb. 12.6b zeigt den resultierende Netzwerkgraphen für ein einfaches Beispiel (Abb. 12.6a).

2. Weitere Einschränkungen können durch den Modellierer spezifiziert werden um bestimmte Netzwerktopologien auszuschließen, die zwar stöchiometrisch sinnvoll sind, aber nicht gewollt werden. Boolesche Ausdrücke werden verwendet um diese ungewollten Topologien zu spezifizieren. In Abb. 12.7 soll genau eine der Reaktionen v_2 und v_3 aktiv sein, was formal wie folgt ausgedrückt wird.

$$(i_1, \dots, i_N) \in Feas \Rightarrow (i_2 = nil \text{ XOR } i_3 = nil) \quad (12.50)$$

Optimierungsalgorithmus für die Modellselektion

Im Kapitel 12 wurde in Gleichung 12.1 das Optimierungsproblem vorgestellt, das bei den Modellfamilien von metabolischen Modellen aufkommt. Mit der Struktur des Modellraums, der Menge der sinnvollen Modelle ($Feas$) und eines Parameterraums Par^i , der auf diese Modellvariante passt, lässt sich das Optimierungsproblem weiter konkretisieren:

$$\min_{i \in Feas} \min_{\alpha^i \in Par^i} \text{Crit}(M^i, \alpha^i, m) \quad (13.1)$$

Zu Beachten ist, dass das Minimierungsproblem bzgl. i ein diskretes Problem ist, das Minimierungsproblem bzgl. α^i aber ein kontinuierliches. Darauf aufbauend kann ein Suchalgorithmus entwickelt werden, der die Berechnungsprimitive der Modellfamilien nutzt (Kap. 12).

Für das kontinuierliche Optimierungsproblem $\min_{\alpha^i \in Par^i}$ gibt es eine Vielzahl an Algorithmen, von denen in Kapitel 9.3 diejenigen vorgestellt wurden, die in MMT2 zum Einsatz kommen. Nimmt man das Problem $\min_{i \in Feas}$ hinzu, so ergibt sich ein diskret-kontinuierliches Optimierungsproblem, für das in diesem Kapitel ein Algorithmus vorgestellt wird. Dieser Algorithmus ist Bestandteil von [Haunschild et al., 2005b].

In Kapitel 6.1 wurden im Hinblick auf die modellbasierte Datenauswertung einige Beispiele aus der Literatur genannt, die auf Probleme in der Art von Gleichung 13.1 ausgerichtet sind. Der Optimierungsalgorithmus, der in diesem Kapitel vorgestellt wird, nutzt zusätzlich den Formalismus der Modellfamilien (Kap. 12).

13.1. Eigenschaften des Modellraums

.....

In Kapitel 12.1.2 wurde am Beispiel einer Modellfamilie von rationalen Polynomen der Raum der Modelle gezeichnet (Abb. 12.1). Bei metabolischen Modellen gestaltet sich die Visualisierung des Modellraumes wesentlich schwieriger, da die zugrunde liegende Struktur nicht mehr zweidimensional ist, sondern so viele Dimensionen hat (Abb. 12.6), wie es Reaktionen mit Varianten gibt. In Abb. 13.1 ist daher der Raum einer Modellfamilie skizziert, die hier nicht weiter definiert wird. Von Interesse ist, wie die Modelle miteinander zusammen hängen und wie der Optimierungsalgorithmus diese Struktur ausnutzen kann.

Aufgrund der Definition, dass benachbarte Modelle sich in nur einer Reaktion unterscheiden, wird angenommen, dass sich in der Nachbarschaft eines gut auf die Messdaten passenden Modells, weitere Modelle befinden, die ebenfalls gut passen. Diese Eigenschaft soll vom Optimierungsalgorithmus ausgenutzt werden.

Abb. 13.1 zeigt neben der Modellraumstruktur Bereiche, die visualisieren, wie gut die jeweiligen Modelle auf die Messdaten passen. Untersucht der Algorithmus das Modell 64 (Abb. 13.2 links), dessen Parameteranpassung die Mindestqualität $Crit < q_1$ erfüllt, so soll auch die Nachbarschaft durchsucht werden. Dabei wird dann Modell 60 gefunden (Abb. 13.2 rechts), das noch etwas besser auf die Messdaten passt (Höhenlinien in Abb. 13.1).

13.2. Vorgehensweise

.....

Zunächst werden einige allgemeine Überlegungen aufgestellt, um das Problem näher zu charakterisieren und die Anforderungen an den Algorithmus näher zu spezifizieren.

1. Das in Gleichung (13.1) verwendete Bewertungskriterium $Crit$ wurde in Kapitel 9.2 beschrieben. Es drückt aus, wie gut die Simulation auf die gemessenen Daten und weitere Nebenbedingungen passt.
2. Die Berechnung von $\min_{\alpha^i \in Par^i}$ kann sehr zeitintensiv sein. Daher muss verhindert werden, dass der Algorithmus zu viel Zeit dafür verwendet, Modelle an die Messdaten anzupassen, die ohnehin nicht passen. Die Zeit, die für die Parameteranpassung eines Modells verwendet wird, sollte vom Fortschritt der Parameteranpassung abhängig sein.
3. Die Parameteranpassung sollte für jede Variante zumindest einige Male gestartet werden, weil die Konvergenz der Optimierung $\min_{\alpha^i \in Par^i}$ u. A. maßgeblich vom Startpunkt α_0^i abhängt. Ein einziger zufällig ausgewählter Startpunkt im Parameterraum könnte die Parameteranpassung in ein schlechtes lokales Minimum leiten, was die Variante schlecht aussehen lässt, wohingegen ein anderer Startpunkt zu einem Minimum mit gut passenden Parametern führen könnte.
4. Zu Beginn ist keine Information darüber vorhanden wo gut passende Modelle im Modellraum liegen könnten. Daher ist am Anfang nur ein zufälliges Durchlaufen des Modellraums möglich, was fortan als Modellraum-Sweep bezeichnet wird.
5. Wird eine gut passende Variante gefunden, so macht es Sinn auch die Modellvarianten in der Nachbarschaft zu testen. Der Durchsuchung der Nachbarschaft sollte dann eine höhere Priorität gegeben werden, als dem Modellraum-Sweep.
6. Mit dem Ziel die Parameter eines bereits gut passenden Modells noch genauer zu bestimmen, wird die Parameteranpassung mehrere Male parallel mit zufälligen

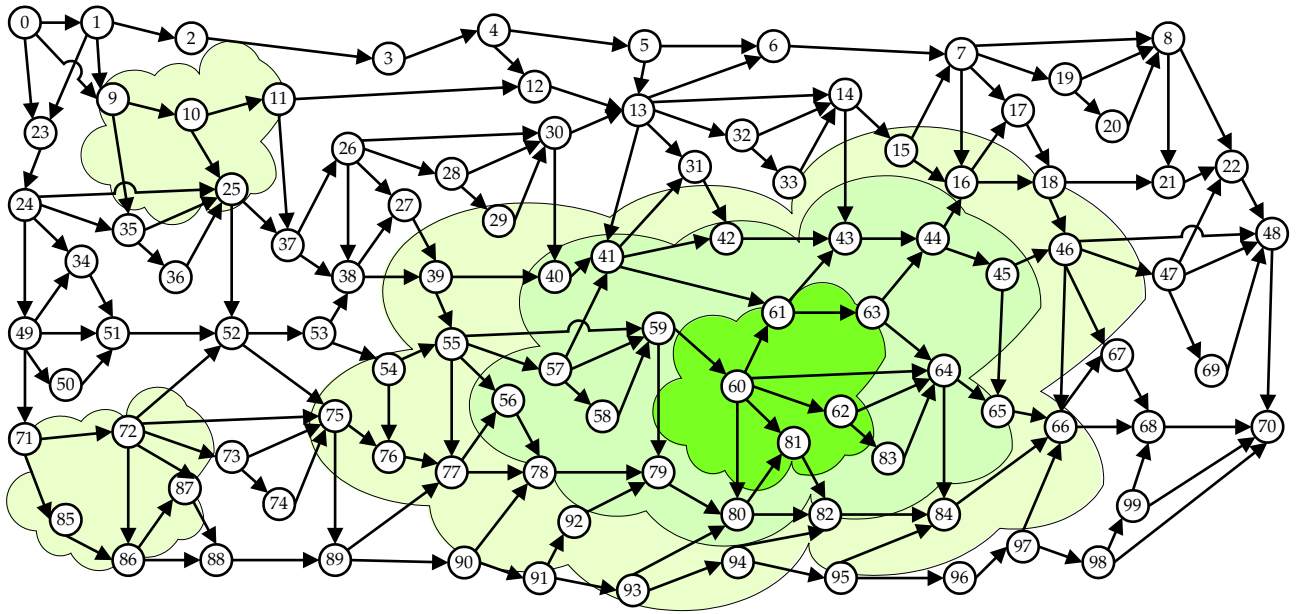


Abbildung 13.1: Skizze eines Modellraums mit fiktiven „Höhenlinien“ der Qualität der Modelle. Abb. 12.1 zeigte einen Modellraum, dessen Verbindungsnetzwerk, wegen der zugrunde liegenden zweidimensionalen Struktur, übersichtlich eingezeichnet werden konnte. Bei Modellfamilien aus der Praxis ist dies üblicherweise nicht möglich, da die Dimension der Struktur gleich der Anzahl der Reaktionen ist, die Varianten besitzen.

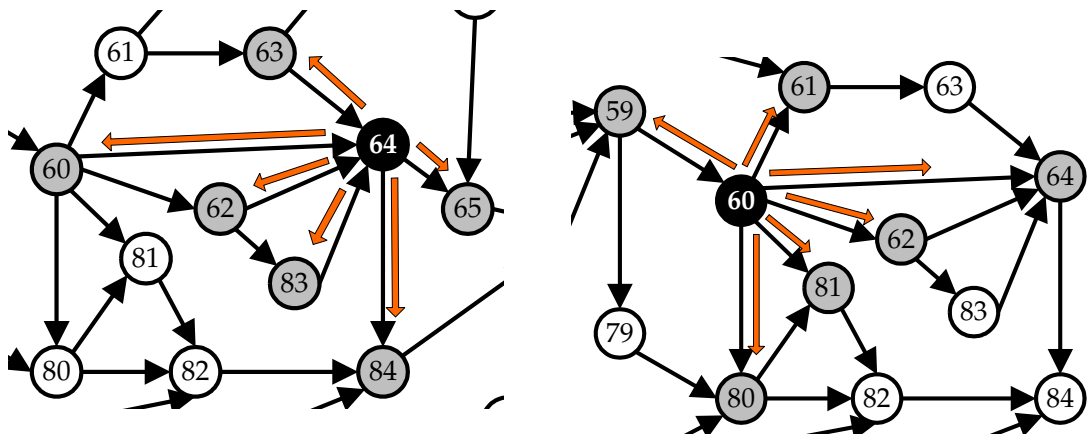


Abbildung 13.2: Ausschnitt aus der Modellraumskizze. Links: Die Parameteranpassung für Modell 64 lieferte $\text{Crit} < q_1$. Daher wird auch die Nachbarschaft dieses Modells durchsucht, bei der sich für Modell 60 ebenfalls $\text{Crit} < q_1$ ergibt. Rechts: Daher wird auch dessen Nachbarschaft durchsucht. Mit diesem Prinzip kann der Algorithmus in die Richtung besserer Modelle konvergieren.

Startwerten α_0 gestartet. Da der Algorithmus an diesem Punkt besonders nahe an seiner Zielstellung ist, nämlich gut passende Modelle zu finden, wird dieser Aktion die höchste Priorität gewährt.

13.3. Struktur des Algorithmus

.....

In diesem Abschnitt wird zunächst die Struktur des Algorithmus für das diskret-kontinuierliche Optimierungsverfahren vorgestellt, sowie das Verhalten anhand einer tabellarischen Beschreibung (Tabelle 13.3).

13.3.1 Multi-Threaded Manager-Worker Schema

Der Algorithmus operiert auf einem Cluster von Computern (Kap 15.1), oder auf einem Grid (Kap 15.2). Abb. 13.3 skizziert die Abhängigkeiten zwischen den einzelnen Komponenten des Algorithmus. Die Manager befinden sich solange im schlafenden Zustand, bis der Controller einen von ihnen aktiviert. Daraufhin erzeugt der Manager einen Job und wird wieder in den schlafenden Zustand versetzt. Der Controller-Prozess ist maßgeblich daran beteiligt, wann und wie oft die Manager-Threads zum Zuge kommen.

13.3.2 Verhaltensbeschreibung

Tabelle 13.3 zeigt eine Übersicht über das Verhalten der Manager-Threads. Sie werden in drei Kategorien unterteilt (im Weiteren Stufen genannt), die sich sehr ähnlich Verhalten, jedoch mit anderen Verhaltensparametern. Im Folgenden wird die Bedeutung der Zeilen der Tabelle 13.3 erläutert:

- **Parameter Priorität:** Ein Verhaltensparameter, der durch den Controller-Prozess gesteuert wird, ist in der Zeile Priorität notiert. Er gibt an, wie oft die Prozesse der angegebenen Stufe aufgeweckt werden und damit einen Job erzeugen. Die Vorgehensweise wird im Abschnitt über den Controller erläutert.

Wird ein Manager-Thread geweckt, so ruft er zur Auswahl einer Modellvariante die Prozedur `select_variant` auf, die je nach Stufe des Manager Threads eine Modellvariante auswählt. Manager-Threads der Stufe 1 wählen für einen Job ein beliebiges Modell aus dem Modellraum aus. Bei den Manager-Threads der Stufen 2 und 3 liegt zusätzlich die Modellvariante zugrunde, deren Erfolg die Erzeugung des jeweiligen Manager-Threads war.

- **Parameter Redundanz:** Über die Parameter $red_i, i = 1 \dots 3$ wird eingestellt, dass eine Modellvariante nicht nur einmal angepasst wird, sondern mehrere Male. Damit soll dem entgegengewirkt werden, dass ein einzelner Anpassungsprozess auch einmal fehlschlagen kann, obwohl es für die Modellvariante vielleicht doch einen gut passenden Parametersatz gibt (vgl. Abschnitt. 13.2 Punkt 3).
- **Parameter Zeitintervall int_i :** Für die einzelnen Anpassungsprozesse, die die Worker durchführen, gilt eine Maximalzeit, die in den Parametern Zeitintervall

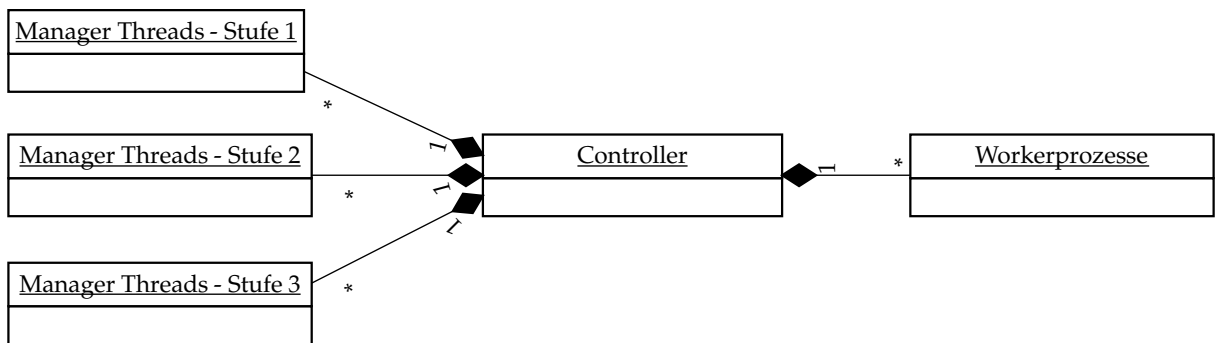


Abbildung 13.3: Multi-Threaded Manager/Worker Schema

	Stufe 1: Modellraum- Sweep	Stufe 2: Nachbarschafts- Sweep	Stufe 3: Parameter- raum-Sweep
Priorität	niedrig	mittel	hoch
Job-Erzeugung	zufälliges Modell aus Modellraum	zufälliges Modell aus Nachbarschaft	Initiales Modell
Redundanz	red_1	red_2	red_3
Zeitintervall t_i	int_1	int_2	int_3
Max. Laufzeit	$t_{max_1} \cdot t_i$	$t_{max_2} \cdot t_i$	$t_{max_3} \cdot t_i$
Aktion pro t_i für alle Jobs	$Crit < q_1 \Rightarrow$ erzeuge Thread (Stufe 2) setze Job auf Stufe 2	$Crit < q_2 \Rightarrow$ erzeuge Thread (Stufe 3) erzeuge Thread (Stufe 2) setze Job auf Stufe 3	keine

Tabelle 13.3: Diese Tabelle zeigt in ihren Spalten die drei möglichen Arten von Threads und ihr Verhalten.

t_i und maximale Laufzeit steckt. Nach jeder Zeiteinheit der Größe Zeitintervall wird der Fortschritt der Parameteranpassung überprüft. Erfolgt sie mit gutem Erfolg, so wird die Aktion in der letzten Zeile durchgeführt. Schließlich wird durch den Parameter „Aktion“ festgelegt, welche Aktion durchgeführt wird, falls eine Anpassung mit gutem Erfolg fortschreitet.

- **Parameter Aktion pro Zeitintervall für Stufe 1:** Im Falle des Modellraum-Sweep kommt es erst einmal nur darauf an, den Raum der Modellvarianten nach einigermaßen passenden Varianten zu durchsuchen. Ist eine solche Variante gefunden, so soll die Nachbarschaft durchsucht werden, in der sich vielleicht eine Modellvariante findet, die noch besser passt. Daher wird bei gutem Erfolg eines Jobs j der Stufe 1 ein Manager-Thread der Stufe 2 erzeugt, der nur Modellvarianten aus der Nachbarschaft von j auswählt. Da dieser Manager-Thread mit einer höheren Priorität läuft, wird ein größerer Anteil der zur Verfügung stehenden Rechner zur Nachbarschaftssuche verwendet, als zum weiteren Modellraum-Sweep.
- **Parameter Aktion pro Zeitintervall für Stufe 2:** Führt eine Anpassung der Stufe 2 (Nachbarschafts-Sweep) zu sehr gutem Erfolg (Abb. 13.2 links), so wird für die angepasste Modellvariante wiederum ein Thread der Stufe 2 erzeugt, der dessen Nachbarschaft durchsucht (Abb. 13.2 rechts). Dadurch ist der Algorithmus in der Lage den Weg von schlechten zu besser passenden Modellen zu beschreiten.

In den beiden Nachbarschaft-Sweeps in Abb. 13.2 links und rechts ist zu sehen, dass einige Modelle von beiden Sweeps abgedeckt werden. Durch die Prozedur `select_variant` wird sichergestellt, dass jede Modellvariante nur so oft angepasst wird, wie im Parameter Redundanz $red_i, i = 1 \dots 3$ festgelegt ist. Stehen keine Modellvarianten in der Nachbarschaft zur Verfügung, die noch nicht angepasst wurden, so beendet sich der entsprechende Manager-Thread.

- **Parameter Aktion pro Zeitintervall für Stufe 3:** Bei sehr gutem Erfolg einer Anpassung einer Modellvariante der Stufe 2 wird weiterhin ein Manager-Thread der Stufe 3 erzeugt, der eine weitere Feinanpassung der Modellvariante durchführen soll. Dazu wird die Parameteranpassung mehrmals mit zufälligen Startwerten im Parameterraum gestartet. Diese Threads haben die höchste Priorität und entsprechend viele Worker werden daher mit dieser Aufgabe beschäftigt.

13.4. Beschreibung des Algorithmus

.....

Nach dieser informalen Vorstellung des Verhaltens des Algorithmus werden in den folgenden Abschnitten die einzelnen Bestandteile durch Pseudocode genauer beschrieben.

13.4.1 Worker

Die in Abb. 13.3 vorgestellten Workerprozesse sind wie in Listing 13.1 implementiert und besitzen eine Hauptschleife, in der sie zunächst auf die Zuweisung eines Jobs warten. Wird ein Job zugeteilt, so wird die Parameteranpassung der Modellvariante gestartet, die in Job j spezifiziert ist. Die meiste Zeit verbringt der Workerprozess damit, die Parameteranpassung durchzuführen. Der Gesamtalgorithmus ist so spezifiziert, dass er auf einem Cluster von Rechnern läuft, auf denen jeweils ein Workerprozess arbeitet.

```

1: worker: process
2: begin
3:   while not termination signal received do
4:     wait until a job  $j$  is assigned
5:     start parameter fitting with  $j$ 
6: end

```

Listing 13.1: Die Workerprozesse sind zuständig für die Parameteranpassung einer bestimmten Modellvariante. Die Jobs werden von Manager-Prozessen erzeugt (Listing 13.2)

13.4.2 Manager

Die Manager-Threads (Listing 13.2) warten die meiste Zeit darauf, dass der Controller ein Signal zum Aufwachen schickt (Zeile 6). Dann erzeugen sie einen Job für einen Worker und werden wieder in den Wartezustand versetzt, falls noch Varianten erzeugt werden können. Durch die For-Schleife (ab Zeile 5) wird der Parameter Redundanz (Tabelle 13.3) implementiert. Die Anzahl der Manager-Threads hängt davon ab, ob und wie viele gute Modellvarianten untersucht werden. Zu Anfang existiert nur ein Manager-Thread, der den Modellraum-Sweep durchführt.

```

1: manager: thread
2: begin
3:   while variants available do
4:     variant = select_variant
5:     for red times do
6:       wait
7:       create job ( $j$ )
8: end

```

Listing 13.2: Die Manager-Threads sind dafür zuständig Modellvarianten zu selektieren, für die eine Parameteranpassung durchgeführt werden soll. Sind keine Varianten mehr vorhanden endet der Thread.

13.4.3 select_variant

Alle Manager-Threads rufen zur Auswahl einer Modellvariante die Prozedur `select_variant`. Dadurch soll sichergestellt werden, dass Modellvarianten nur so oft angepasst werden, wie im Parameter Redundanz (Tabelle 13.3) vorgegeben. Denn es kann sein, dass mehrere Manager-Threads in der Lage sind eine bestimmte Modellvarian-

te auszuwählen. Diese Prozedur teilt dem Manager mit, wenn keine Modellvariante mehr zur Verfügung steht, woraufhin dieser sich beendet.

```

1: select_variant:  procedure
2: begin
3:   switch level do
4:     case 1:
5:       | return random variant
6:     case 2:
7:       | return variant of neighborhood(j)
8:       | if no variant is left then return -1
9:     case 3:
10:    | return j.variant
11: end

```

Listing 13.3: Abhängig vom Level des Managers wird eine Modellvariante ausgewählt

13.4.4 Controller

Der Controller ist das Hauptprogramm des Algorithmus. Hier findet die Initialisierung statt, sowie das Beenden des Algorithmus. Der Controller-Prozess steuert den Ablauf der Manager-Threads.

Der Controllerprozess (Listing 13.4) sorgt zunächst für die Initialisierung des Algorithmus, in der ein Manager-Thread der Stufe 1 erzeugt wird. Da direkt nach der Initialisierung alle Worker noch keinen Job in Arbeit haben, wird die Schleife ab Zeile 6 nicht durchlaufen. Die Schleife ab Zeile 8 dagegen wird solange durchlaufen, bis allen Workern ein Job zugewiesen wurde. Dazu wird der eine Manager-Thread aufgeweckt, der dahingehend einen Job erzeugt und dann wieder in den schlafenden Zustand versetzt wird. Da es zu diesem Zeitpunkt nur einen Thread gibt, wird dieser in der Schleife solange zur Erzeugung eines Jobs aufgeweckt, bis allen Workern ein Job zugewiesen wurde.

```

1: controller:  process
2: begin
3:   p.level=1
4:   create manager thread (p)
5:   while number of managers > 0 do
6:     | foreach job in joblist do
7:     | | check.job(ticks, job)
8:     | while count(workers) > count(joblist) do
9:     | | signal_process
10:    | | sleep ( $t_i$ )
11: end

```

Listing 13.4: Der Controller-Prozess steuert die Optimierung. Die kontinuierliche Parameteranpassung der Worker wird überwacht und ein Managerthread wird aktiviert, wenn ein Worker idle wird.

Der Controllerprozess in Listing 13.4 überwacht in den Zeilen 6 bis 9 den Fortschritt der einzelnen Jobs. Listing 13.4 zeigt die Vorgehensweise. Erweist sich eine Modellvariante als gut anpassbar, so wird zum Einen der Anpassung mehr Zeit gewährt, zum

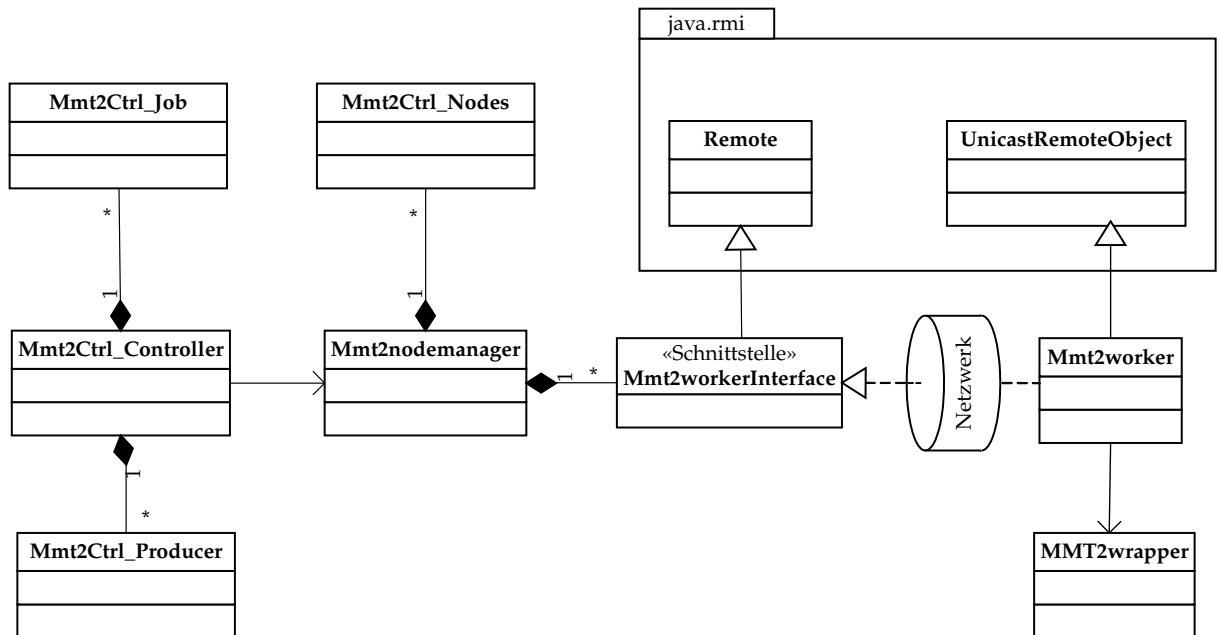


Abbildung 13.4: UML-Klassendiagramm der wichtigsten Klassen, des diskret-/kontinuierlichen Optimierungsalgorithmus und deren Abhängigkeiten.

Anderen wird ein Manager-Thread (Tabelle 13.3, Stufe 3) zur genaueren Untersuchung dieser Variante erzeugt.

```

1: check_job: procedure;
2: begin
3:   if job.ssq < q1 then
4:     p.level=2;
5:     create process (p, job);
6:   else if job.ssq < q2 then
7:     p.level=3;
8:     create process (p, job);
9:   else if job.t.run > t.max then
10:    remove_job;
11: end
  
```

Listing 13.5: Abhängig vom Fortschritt der Parameteranpassung wird dem Vorgang mehr Zeit zugeteilt oder er wird abgebrochen.

13.5. Implementierung

Der im letzten Abschnitt beschriebene Algorithmus wurde in Java implementiert. Abb. 13.4 zeigt die wichtigsten Klassen und deren Abhängigkeiten. Die Kommunikation der Manager und Worker zwischen den Rechnern wurde mit der Bibliothek Java RMI (Remote Method Invokation) realisiert.

In Abb 13.3 wurde das grobe Schema vorgestellt, auf dem der Algorithmus basiert. Die Manager-Threads sind in der Klasse MMT2.CtrlProducer implemen-

tiert, der Controller in der Klasse `MMT2_CtrlController` und der Worker in der Klasse `MMT2_wrapper`. Sie verwenden den MMT2-Simulator als Legacy-Code zur Parameteranpassung. Die Klassen `MMT2worker` und `MMT2workerInterface` sind für die Kommunikation über das Netzwerk zuständig, das von der Klasse `MMT2nodemanager` initialisiert und überwacht wird.

Während der Implementierung des Algorithmus wurden viele Testläufe durchgeführt, von denen im nächsten Kapitel drei vorgestellt werden. Während dieser Testläufe wurden umfangreiche Logfiles geschrieben, in denen die Kommunikation zwischen Managern und Worker dokumentiert wurde, welche Manager-Threads wann welche Jobs erzeugt haben sowie das Verhalten der Worker, die jeweils eine Instanz von MMT2 steuern.

Eine Erfahrung, die bei den Testläufen gemacht wurde, ist, dass das Hochskalieren des Algorithmus von 10 Workern auf 100 Worker einen erheblichen Einfluss auf das Verhalten des Algorithmus haben kann. Bei dieser Implementierung traten solche Engpässe in Listing 13.2, Zeile 7 und in Listing 13.3, Zeile 6+7 auf, in denen die Kommunikation mit den Workern synchron durchgeführt wurde, was in Listing 13.2, Zeile 7 dazu führte, dass der Aufruf in Zeile 7 erst dann fertig ist, wenn auf dem entfernten Worker der Wrapper-Prozess für den Simulator gestartet wurde.

Während dies bei 10 Workern noch keine erheblichen Verzögerungen verursacht, entsteht bei 100 Workern ein Kommunikationsflaschenhals, der die Jobzuweisung so ineffizient macht, dass die Worker nicht mehr zeitnah bei Bedarf neue Jobs zugewiesen bekommen und so unnötig eine Vielzahl der Worker keine Berechnungen durchführt. Dies ist aber kein generelles Problem des Algorithmus, sondern liegt in der Implementierung, die durch leichte Änderungen zu einer konstanten Auslastung aller Worker geführt hat.

Analyse des Modellselektionsalgorithmus

Der Algorithmus und seine Implementierung, die in Kapitel 13 vorgestellt wurde, soll in diesem Kapitel analysiert und getestet werden. Dazu wurde ein Referenzmodell erstellt (Abschnitt 14.1) aus dem durch Hinzufügen von kinetischen Varianten eine Testmodellfamilie erstellt wurde (Abschnitt 14.2). Die anwendungsspezifischen Einstellungen der Verfahrensparameter werden in Abschnitt 14.3 diskutiert. Drei der aufgetretenen Laufzeitverhaltensweisen des Algorithmus werden in Abschnitt 14.4 vorgestellt und in Abschnitt 14.5 diskutiert. Abschließend werden Kapitel 13 und 14 zusammengefasst.

14.1. Referenzmodell

Um eine Modellfamilie zum Test des Optimierungsalgorithmus zu erstellen, muss zunächst ein Referenzmodell M^{ref} erstellt werden, aus dessen Verhalten künstliche Messdaten erzeugt werden. Aus M^{ref} wird dann durch das Hinzufügen von Varianten eine Modellfamilie erstellt. Dabei sollen die entstehenden Modelle strukturell hinreichend verschieden sein.

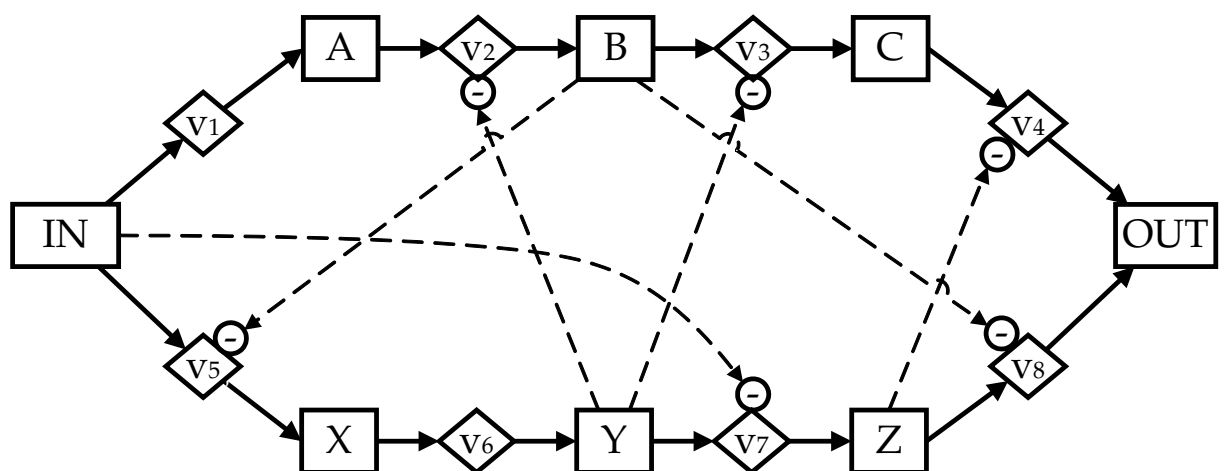


Abbildung 14.1: Das Metabolische Netzwerk, das zum Testen der Modellfamilien und des Optimierungsalgorithmus verwendet wurde. Die zwei parallelen Stoffflusswege verzögern sich durch inhibierte Kinetiken gegenseitig.

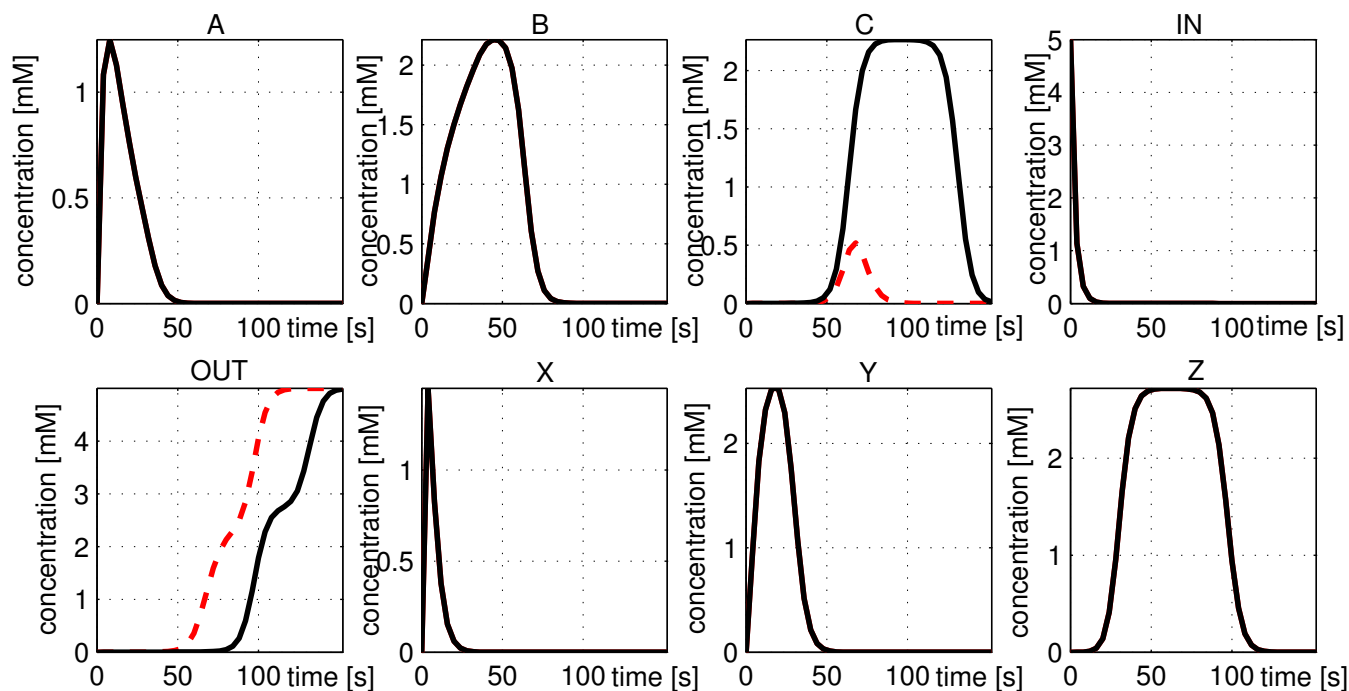


Abbildung 14.2: Schwarze Kurven: Verhalten des Referenzmodells M^{ref} aus Abb. 14.1.

Die zwei parallelen Stoffwechselwege verzögern ihren Stofffluss gegenseitig. Gestrichelte rote Kurven: Entfernung der inhibierenden Wirkung von Z auf v_4 (M^{1471}).

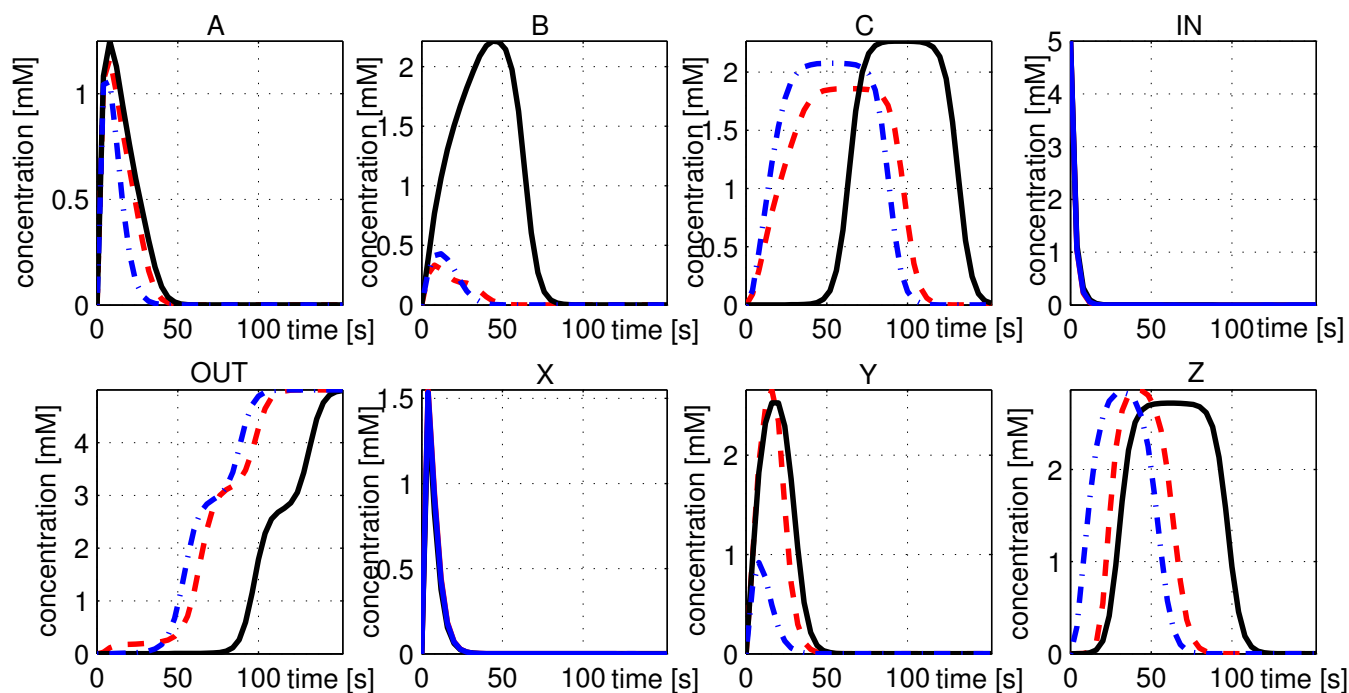


Abbildung 14.3: Schwarze Kurven: Referenzmodell M^{ref} ; Rote, gestrichelte Kurven:

M^{1477} (M^{ref} ohne v_3 Inhibition); Blaue, strich-punktierte Kurven:

M^{1195} (M^{ref} ohne v_7 Inhibition).

Als Referenzmodell M^{ref} wurde das Modell in Abb. 14.1 verwendet. Es besitzt $\dim(x)=8$ Metabolitpools und $\dim(\vec{v})=8$ Reaktionen vom Typ Michaelis-Menten (eq. 3.7) oder Michaelis-Menten-Inhibiert (eq. 3.9). Die zwei parallelen Stoffwechselwege (v_1, \dots, v_4 und v_5, \dots, v_8) beeinflussen sich gegenseitig, was einen verzögerten Stofffluss von IN nach OUT zur Folge hat.

Abb. 14.2 (schwarze, durchgezogene Kurven) zeigt den Zeitverlauf der Konzentrationen von M^{ref} . Bis auf die Metabolite IN, OUT und X haben alle Zeitverläufe die Eigenschaft, dass es einen Zeitraum gibt, in dem sich Stoffe ansammeln, durch die Einwirkung eines Inhibitors, der den Abfluss hemmt. Weiterhin gibt es einen weiteren Zeitraum, in dem die Stoffe abfließen können, weil die inhibierende Wirkung verschwindet.

Verändert man die Struktur der Inhibitoren, so stellen sich andere Zeitverläufe ein. Abb. 14.2 (rote, gestrichelte Kurven) zeigt eine Variante des Referenzmodells M^{1471} (Abschnitt 14.2), die sich dadurch zu M^{ref} unterscheidet, dass die inhibierende Wirkung von Z auf v_4 entfernt wurde. Dadurch wird der Abfluss von C nicht mehr gehemmt und nur noch von der Michaelis-Menten Kinetik bestimmt. Dieses veränderte Verhalten von C beeinflusst in diesem Fall nur OUT.

Abb. 14.3 zeigt den Einfluss zweier anderer Modifikationen an M^{ref} . Die roten, gestrichelten Kurven zeigen das Verhalten von Modell M^{1477} , bei dem im Gegensatz zu M^{ref} die inhibierende Wirkung von Y auf v_3 entfernt wurde. Diese Modifikation veranschaulicht die komplexen Wechselwirkungen zwischen den Reaktionen. Durch die Entfernung der Inhibierung stauen sich in B wesentlich weniger Metabolite, weil sie früher nach C abfließen können. Aber auch der Stofffluss durch Z findet dadurch früher und schneller statt.

Ein weiteres Beispiel einer Modifikation gibt Modell M^{1195} , das im Gegensatz zu M^{ref} keine Inhibierung von IN auf v_7 besitzt. Der resultierende Zeitverlauf wird in Abb. 14.3 (blaue, strich-punktierte Kurven) gezeigt.

14.2. Testmodellfamilie

.....

Im letzten Abschnitt wurde ein Referenzmodell eingeführt und gezeigt, wie bereits eine vereinzelte Veränderung eines Inhibitors merkliche Auswirkung auf den Stofffluss verursacht. Daher wurde der Verlauf der Stoffkonzentrationen als Referenz genommen (Kapitel 10.4) und für dieses Modell kinetische Varianten erstellt, um eine Modellfamilie zu erstellen.

Die folgende Aufstellung listet die Varianten aller Reaktionen auf. Die unterstrichenen Kinetiken zeigen das Referenzmodell M^{ref} , das identisch ist mit M^{1483} . Insgesamt beinhaltet der Modellraum 2880 Modellvarianten.

- Reaktion v_1 : \underline{v}^{MM}
- Reaktion v_2 : $\underline{v}^{MM}, \underline{v}^{MMI,Y}, \underline{v}^{MMI,Z}$
- Reaktion v_3 : $\underline{v}^{MM}, \underline{v}^{MMI,X}, \underline{v}^{MMI,Y}, \underline{v}^{MMI,Z}$
- Reaktion v_4 : $\underline{v}^{MM}, \underline{v}^{MMI,Z}$
- Reaktion v_5 : $\underline{v}^{MM}, \underline{v}^{MMI,B}$
- Reaktion v_6 : $\underline{v}^{MM}, \underline{v}^{MMI,A}, \underline{v}^{MMI,IN}$
- Reaktion v_7 : $\underline{v}^{MM}, \underline{v}^{MMI,A}, \underline{v}^{MMI,IN}, \underline{v}^{MMI,X}$
- Reaktion v_8 : $\underline{v}^{MM}, \underline{v}^{MMI,A}, \underline{v}^{MMI,B}, \underline{v}^{MMI,C}, \underline{v}^{MMI,Z}$

14.3. Verfahrensparameter

.....

Der im letzten Kapitel beschriebene Algorithmus wurde auf einem Computecluster (Kap. 15.1) mit der im letzten Abschnitt beschriebenen Modellfamilie getestet. Die Verfahrensparameter wurden wie folgt gewählt (Tabelle 13.3):

- **Redundanz:** Für jedes Modell, das untersucht wird, soll die Parameteranpassung von zumindest vier verschiedenen zufälligen Parameterstartvektoren $\vec{\alpha}$ gestartet werden. Für das Finetuning der Parameter von Modellen, die sehr gut auf die Daten passen, soll die Parameteranpassung 20 Mal mit jeweils zufälligem $\vec{\alpha}$ gestartet werden. Für die entsprechenden Parameter wird also gewählt: $red_1=4$, $red_2=4$ und $red_3=20$.
- **Zeitintervall:** Damit der Algorithmus schnell auf gute Ergebnisse bei der Parameteranpassung reagieren kann, wird der Zeitintervall entsprechend klein eingestellt, aber groß genug, dass die Kommunikationszeit zwischen Master und Workern nicht unnötig hoch wird. Für den entsprechenden Parameter wird also gewählt: $t_i=1$ min.
- **Qualitätskriterien:** Die Parameter q_1 und q_2 lassen sich leider erst dann sinnvoll setzen, wenn bereits einige Parameteranpassungen durchgeführt wurden und damit bekannt ist, bei welchen Werten von Crit es sich um eine gute Anpassung handelt. Bei diesem Testlauf sind die Werte gut bekannt und werden somit wie folgt gesetzt: $q_1=1e^{-3}$, $q_2=1e^{-4}$.
- **maximale Laufzeit:** Jede Parameteranpassung soll zumindest 10 Minuten Rechenzeit erhalten. Stellt sich ein guter Fortschritt ein ($Crit < q_1$), so wird die Aktion entsprechend Tabelle 13.3 ausgeführt und die Rechenzeit auf 20 Minuten erweitert. Ist der Fortschritt der Parameteranpassung so gut, dass ein Finetuning der Parameter durchgeführt werden soll, so wird die Rechenzeit auf 30 Minuten erhöht. Für die entsprechenden Parameter wird also gewählt: $t_{max_1}=10$, $t_{max_2}=20$, $t_{max_3}=30$.

Benchmark Nr.	#1	#2	#3
1. Zeit	3 Std.	1 Std.	1 Std.
2. Worker	250	240	240
3. Jobs Stufe 3	147 (+11)	12 (+0)	0 (+82)
4. Jobs Stufe 2	848 (+156)	56 (+50)	58 (+126)
5. Jobs Stufe 1	666 (+83)	848 (+189)	914 (+32)
6. Jobs Summe	1661(+250)	916 (+239)	972 (+240)
7. Ranking	1-9: M ²³⁵⁹	1-12: M ¹³³⁴	-*
8. Erfolg	36 Mal Crit<1e ⁻³ (5 Modelle)	5 Mal Crit<2e ⁻³ (1 Modell)	-*
9. Untersuchte Modelle	363	277	275
10. Manager Threads (3/2/1)	30/17/1	1/5/1	32/7/1

* s. Abschnitt 14.4.2

Tabelle 14.3: Statistische Angaben über die Benchmarks. „Jobs Stufe n“ gibt an wieviele Jobs die Stufe n erreicht haben. In Klammern die Jobs, die nach Ablauf der Gesamtberechnungszeit abgebrochen wurden.

14.4. Analyse des Laufzeitverhaltens

Die Benchmarks wurden auf dem Rubens-Cluster (Kapitel 15.1.2) auf 240 bzw. 250 CPUs durchgeführt. Der Benchmark wurde drei Mal durchgeführt, wobei sich gezeigt hat, dass wie bei der kontinuierlichen Parameteranpassung, der Erfolg des Optimierungslaufs stark von den Zufallszahlen im Modellraum-Sweep (Stufe 1) abhängig ist. In Tabelle 14.3 sind statistische Angaben über die Benchmarks aufgeführt.

14.4.1 Benchmark 1

Benchmark Nr. 1 wurde auf eine maximale Gesamtlaufzeit von 3 Stunden beschränkt, nach der alle in Berechnung befindlichen Jobs abgebrochen werden. In Tabelle 14.3 werden in den Zeilen 3-6 die Anzahl der abgebrochenen Jobs in Klammern angegeben. Die zeitliche Verteilung dieser Jobs in der Gesamtlaufzeit ist in Abb. 14.4 dargestellt. Dort kann man gut erkennen, dass der verteilte Algorithmus sich gut darauf einstellen kann, ob viele gut passende Modelle untersucht werden können oder nicht.

Sind noch keine gut passenden Modellvarianten gefunden worden, so kann der Algorithmus nur mit Jobs der Stufe 1 (Modellraum-Sweep) nach solchen Modellvarianten suchen. Bereits nach zehn Minuten trifft der Algorithmus auf ein gut passendes Modell, in dessen Nachbarschaft auch viele gut passende Modelle vorhanden sind. Aufgrund der höheren Priorität der Stufe 2 reagiert der Algorithmus so, dass der Modellraum-Sweep (Stufe 1) eingeschränkt wird und überwiegend Jobs der Stufe 2

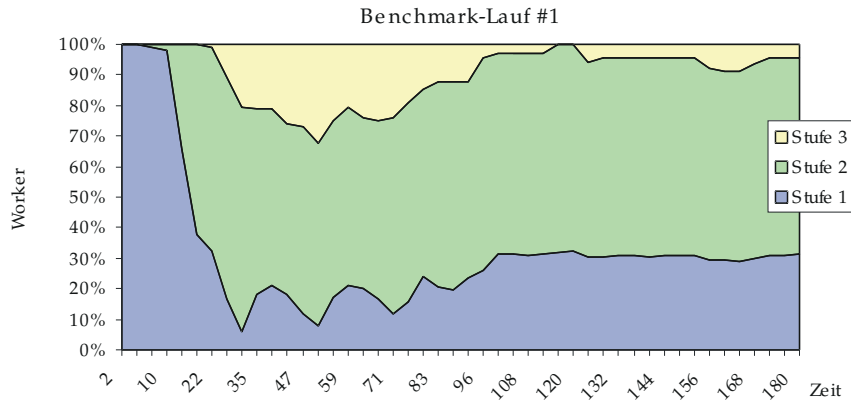


Abbildung 14.4: Zeilicher Verlauf der Verteilung der Job-Stufen. Zu Anfang wird im Modellraumsweep (Stufe 1) ein Gebiet getroffen, in dem sich viele passende Modelle befinden, daher wird der Modellraumsweep zeitweise auf unter 10 % heruntergefahren.

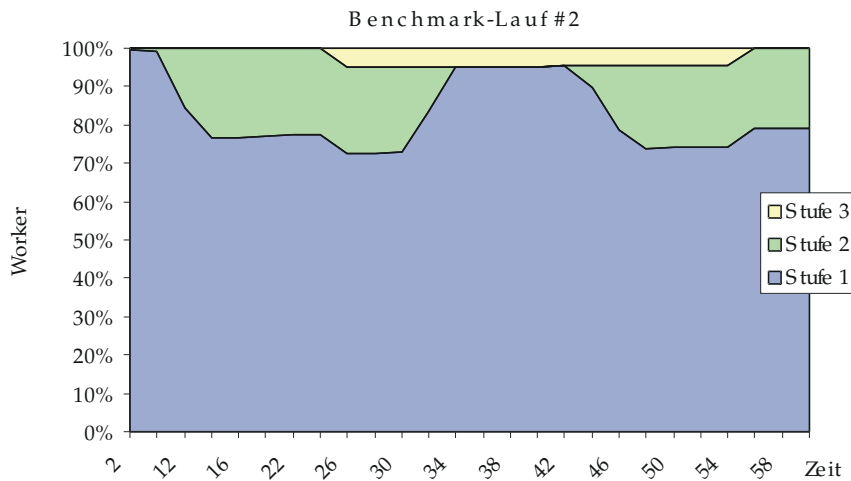


Abbildung 14.5: Wie in 14.4 wird Anfangs im Modellraum-Sweep (Stufe 1) ein Gebiet gut passender Modelle getroffen, das aber nicht so groß ist, um viele Worker damit zu beschäftigen.

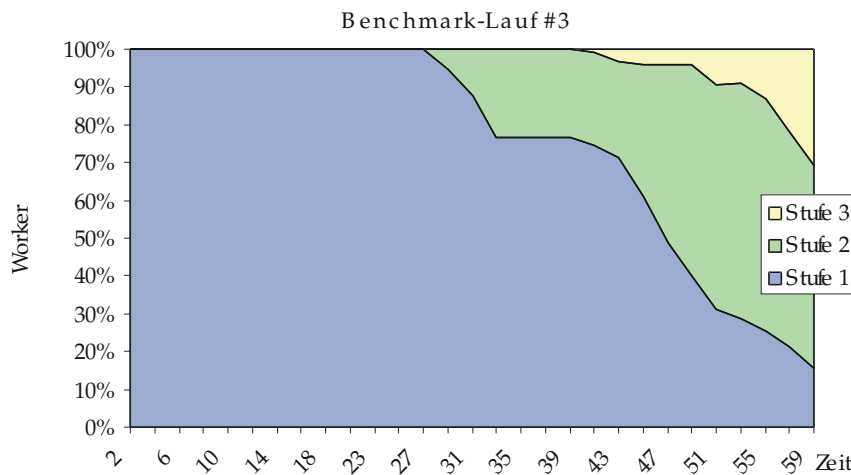


Abbildung 14.6: In diesem Benchmark wird nach etwa 30 Minuten ein gut passendes Modell gefunden in dessen Umgebung dann weitergesucht wird.

erzeugt werden. Analog dazu reagiert der Algorithmus auf sehr gut passende Modelle, für die Jobs der Stufe 3 (Parameterraum-Sweep) erzeugt werden.

Nach Ablauf der Gesamtlaufzeit werden alle Jobs abgebrochen und die Modelle werden in einer Tabelle aufsteigend bzgl. Crit sortiert. In diesem Ranking befindet sich auf den ersten neun Plätzen die Modellvariante M^{2359} (Tabelle 14.3, Zeile 7). Insgesamt befindet sich der erreichte Wert für Crit auf den ersten 36 Plätzen, die sich 5 verschiedene Modellvarianten teilen, unter dem Wert von $1e^{-3}$ (Tabelle 14.3, Zeile 8).

Die Anzahl der untersuchten Modelle (Tabelle 14.3, Zeile 9) ist im Vergleich zu den Benchmarks 2 und 3 deshalb nicht so groß, weil viel Zeit damit aufgewendet wurde Jobs der Stufe 3 zu bearbeiten, denen jeweils eine Rechenzeit von 30 Minuten zugewiesen ist.

Die Anzahl der erzeugten Manager-Threads ist in Tabelle 14.3, Zeile 10 angegeben. Es wurden 30 Manager der Stufe 3 (Parameterraum-Sweep) erzeugt, unter denen viele auch Jobs für dieselbe Modellvariante erzeugen. Insgesamt erzeugen sie aber nicht mehr als im Verfahrensparameter red_3 angegeben. Übersteigt die Anzahl red_3 , so werden diese Manager-Threads beendet. Analog gilt dies für die Manager-Threads der Stufe 2.

14.4.2 Benchmarks 2 und 3

In Benchmark 2 wurde zwar auch, wie in Benchmark 1, zu Anfang ein gut passendes Modell gefunden (Abb. 14.5), aber offensichtlich befanden sich unter den Modellen in der Nachbarschaft nicht so viele, die gut passen. Weiterhin kann man in Abb. 14.5 gut erkennen, wie ab Minute 10 gut passende Modelle gefunden werden, die zu weiteren Nachbarschaftssuchen (Stufe 2) führen, bis in Minute 36 die Nachbarschaftssuche abgeschlossen ist. Eines der Modelle lässt sich sehr gut anpassen, woraufhin ein Manager-Thread der Stufe 3 gestartet wird, der 20 Workern und eine halbe Stunde Zeit bekommt für ein Finetuning der Parameter des Modells. Aufgrund der wenigen Manager-Threads auf Stufe 2 wird der Modellraum-Sweep (Stufe 1) durch die Nachbarschaftssuche nicht sehr weit gedrosselt.

In Benchmark 3 (Abb. 14.6) wurde erst nach etwa 30 Minuten ein gut passendes Modell gefunden, in dessen Nachbarschaft offensichtlich auch viele gut passende Modelle vorhanden sind. Daher wurde auch hier, wie in Benchmark 1, der Modellraum-Sweep erheblich gedrosselt.

Nach Ablauf der Zeit für die Modellsuche waren alle gut passenden Modelle noch in Berechnung. Nach dem derzeitigen Stand der Implementierung werden bei Ablauf der Optimierungszeit die Ergebnisse der noch laufenden Parameteranpassungen nicht mit ins Ranking übernommen. Daher sind in Tabelle 14.3, Spalte 3, die Zeilen 7 und 8 keine Angaben gemacht worden.

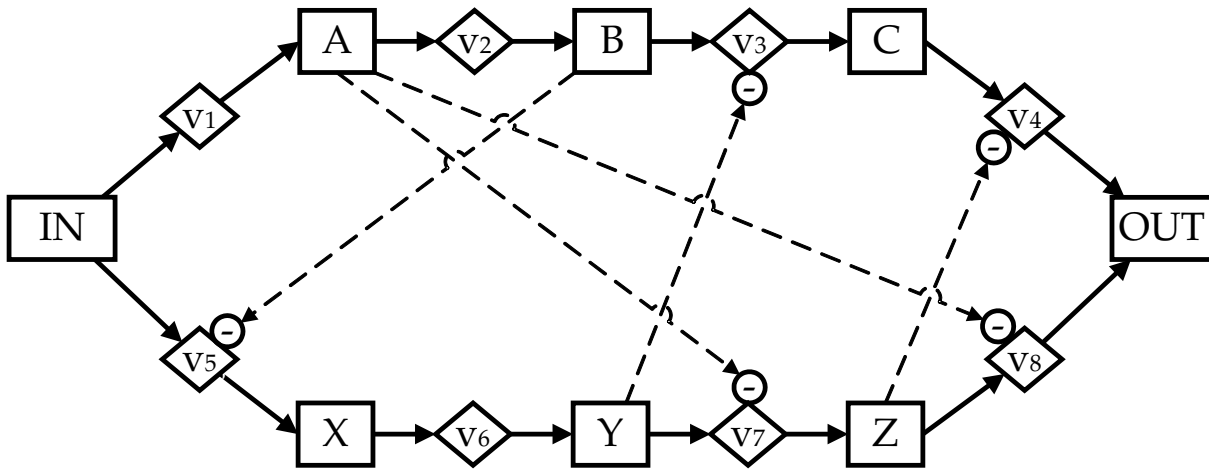


Abbildung 14.7: In Benchmark 1 erreichte dieses Modell den ersten Platz im Ranking. Interessant ist, dass es sich nicht um das Referenzmodell handelt, dass mit den richtigen Parametern perfekt auf die Daten passt.

14.5. Analyse des Ergebnisses

14.5.1 Bestes Modell

Es wurde erwartet, dass der Algorithmus während der diskret- kontinuierlichen Optimierung das Referenzmodell als bestes Modell wieder findet. Interessanterweise lieferte der Optimierungslauf aber in allen drei Benchmarks nicht das Referenzmodell als besten Kandidaten, obwohl es in Benchmark 1 und 2 mit zu den getesteten Modellen gehörte.

Der Grund für dieses Verhalten ist beim kontinuierlichen Optimierer zu suchen. Die Parameteranpassungen wurde nicht mit dem Default-Parametersatz begonnen, sondern an einem zufällig ausgewählten Punkt im Parameterraum. Von dort aus war der Optimierer wegen der hohen Komplexität (Kapitel 9.3.5) nicht in der Lage den Default-Parametersatz, und damit das globale Optimum zu finden.

14.5.2 Unterschiede und Gemeinsamkeiten mit dem Referenzmodell

Abbildung 14.7 zeigt das Modell M^{2359} , das im Benchmark 1 als bestes Modell die Plätze 1-9 (d. h. neun verschiedene Parametersätze wurden gefunden) belegen konnte. In diesem Modell werden zwei Reaktionen von einem anderem Inhibitor beeinflusst, als beim Referenzmodell M^{ref} . v_7 wird anstatt von IN von A inhibiert und v_8 wird anstatt von B von A inhibiert. Es ist also kein direkter Nachbar von M^{ref} , befindet sich aber im Modellabhängigkeitsgraphen nur zwei Kanten entfernt.

Abb. 14.8 zeigt den Zeitverlauf des Modells M^{2359} mit unangepassten Parametern (blaue, gestrichelte Linien) und mit angepassten Parametern (Rote Linien). Auch bei dem angepassten Parametersatz kann man erkennen, dass das Modell nicht perfekt in

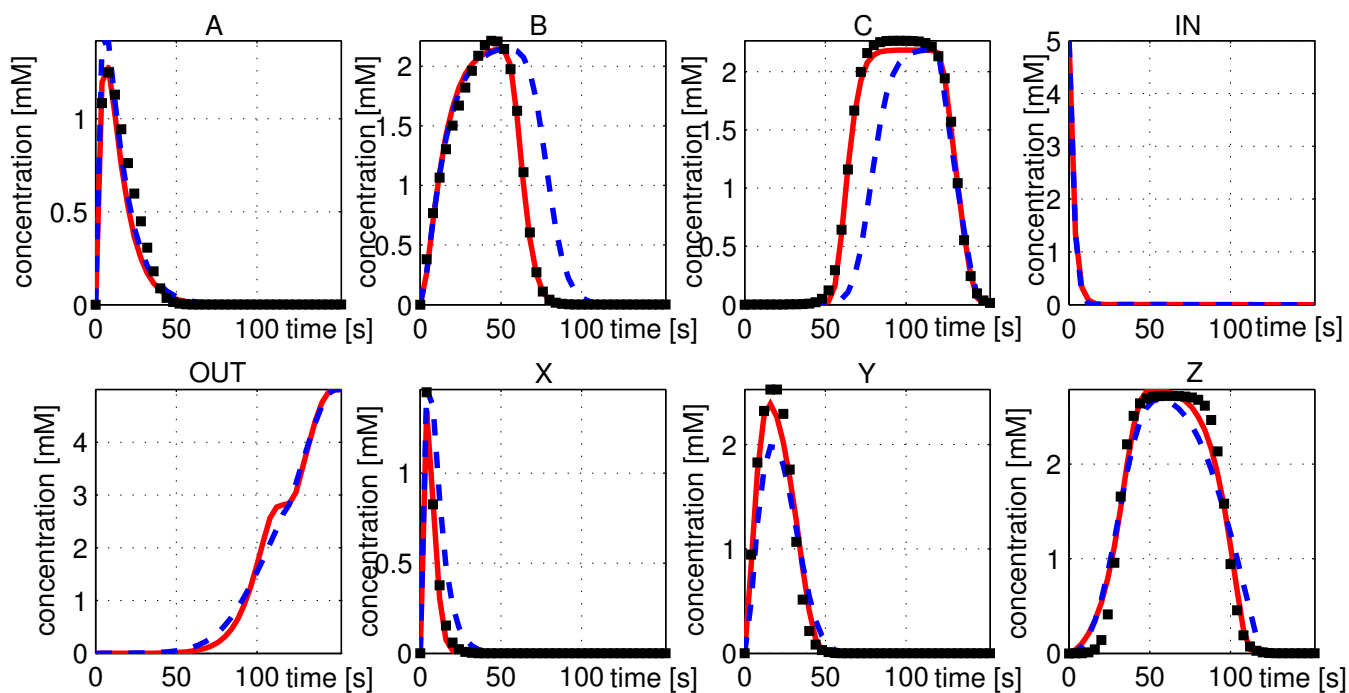


Abbildung 14.8: Verhalten eines der metabolischen Netzwerke im Ranking der besten Modelle nach dem Optimierungslauf. Es unterscheidet sich vom Referenzmodell in nur zwei Inhibitoren.

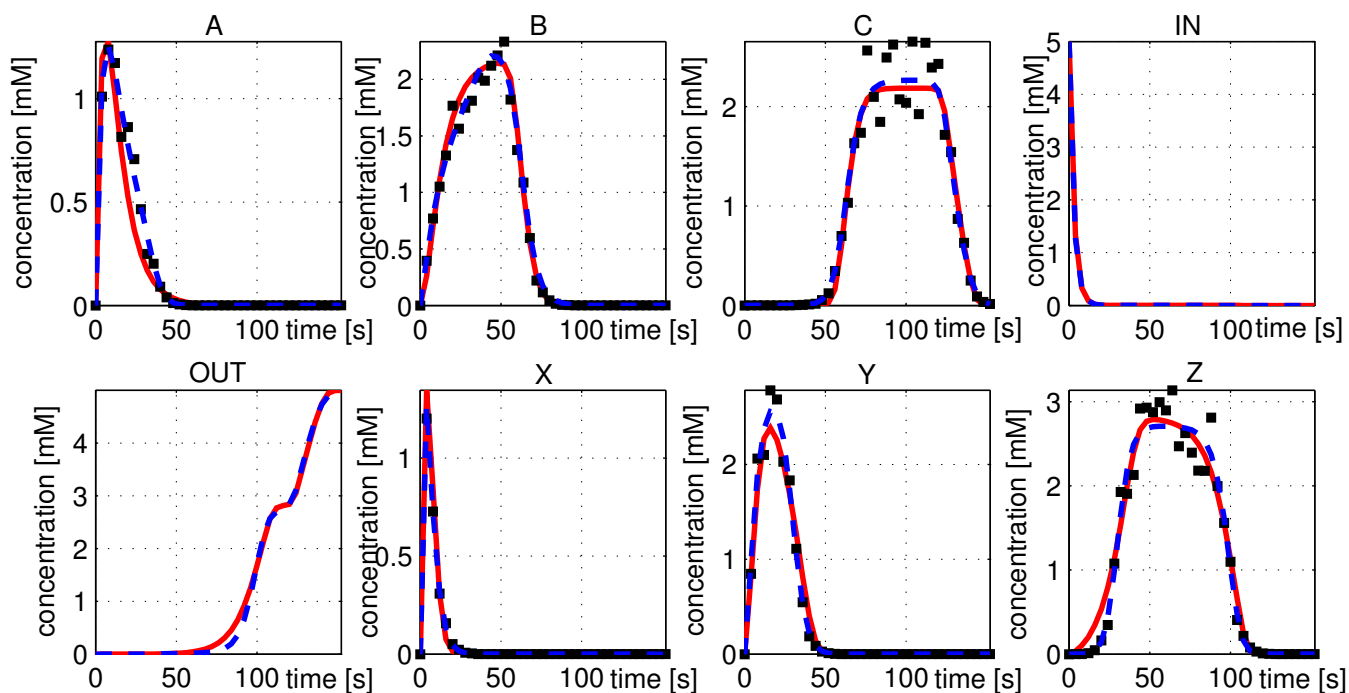


Abbildung 14.9: Zeitlicher Verlauf der Modelle M^{2359} und M^{ref} nach einer Parameteranpassung an verrauschte Daten.

der Lage ist den Verlauf der zugrunde liegenden Daten zu reproduzieren (insbesondere bei Pool C und Z).

14.6. Verrauschte Datenbasis

.....

Dieses Kapitel untersucht das Verhalten des Optimierungsalgorithmus an einem idealisierten Anwendungsfall. Die idealisierenden Eigenschaften, die sich in der Praxis nicht finden, sind:

- Es existiert ein Modell in der Modellfamilie, das exakt auf die Daten passt.
- Die Daten für die Parameteranpassung sind nicht verrauscht.

Beide Eigenschaften beeinflussen zwar indirekt das Verhalten und die Effektivität des Optimierungsalgorithmus, sind aber im Modell begründet. Würde man die Modellfamilie in diesem Kapitel an einen verrauschten Datensatz anpassen (Abb. 14.9), so würde das Verhalten des Optimierungsalgorithmus ein anderes sein, was aber in den Modelleigenschaften begründet ist. Genauso, wie die Verfahrensparameter q_1 , q_2 (Kap. 14.3) vom Modell und den Daten abhängen, und damit zunächst empirisch bestimmt werden müssen, so würde ein Verrauschen der Datenbasis eine weitere Untersuchung des Modellverhaltens erfordern um die Verfahrensparameter darauf einzustellen.

Im Falle der Modellfamilie dieses Kapitels wäre zu erwarten, dass durch das Verrauschen der Datenbasis die besten zu findenden Parametersätze der jeweiligen Modellvarianten sich noch weniger unterscheiden würden, was zur Folge hätte, dass die Platzierung der Modelle im Ranking näher zusammenrücken würde, und damit eine eindeutige Auswahl einer Menge von besten Kandidaten noch schwerer fiel.

14.7. Schlussfolgerungen

.....

1. Der beste Kandidat, der in der Optimierung gefunden wurde, muss nicht das Modell der Wahl sein. Der Modellierer muss immer über die Liste der am besten passenden Modellkandidaten schauen und anhand der erzeugten Zeitverläufe selber bewerten wie gut die Messdaten wiedergegeben werden und ob die gefundene Modellkonstellation plausibel ist.
2. Der Optimierungslauf gibt Hinweise, welche Modellkonstellationen gut passen könnten. Im Falle dieser Benchmark-Optimierung stellte sich heraus, dass beim Modell die Inhibitoren zweier Reaktionen verändert werden konnten, was durch eine Anpassung der Parameter ausgeglichen werden konnte und sich diese Veränderung daher nur wenig auf das Ergebnis ausgewirkt hat.

14.8. Zusammenfassung

.....

Die Kapitel 12 und 13 beschäftigten sich mit der Definition von Modellfamilien und eines Algorithmus der darauf aufbauend Modellfamilien auf gute Kandidaten durchsuchen kann.

Um nicht auf eine alleinige Brute-force-Durchsuchung des Modellraums angewiesen zu sein, wurde ein Algorithmus entwickelt, der die strukturellen Eigenschaften der Modellfamilien ausnutzt. Es wurde demonstriert, dass dies ganz allgemein gelöst werden kann, ohne die konkret betrachtete Modellfamilie im Algorithmus berücksichtigen zu müssen.

Der Hauptvorteil der abstrakten Repräsentation des Modellauswahlproblems, die durch Abhängigkeiten zwischen den Modellen erweitert wurde, ist, dass ein Algorithmus auf dieser Struktur aufbauen kann. Der im letzten Kapitel beschriebene Algorithmus wurde in diesem Kapitel an einer nichttrivialen Modellfamilie getestet und war in der Lage gut passende Modelle zu finden, in dem er die strukturellen Eigenschaften der Modellfamilie ausnutzte. Der Erfolg dieses Algorithmus wird dahingehend gewertet, dass der vorgeschlagene Formalismus der Modellfamilien es prinzipiell ermöglicht, arbeitsintensive Phasen während des iterativen Modellbildungsprozesses zu automatisieren.

High-Performance Computing

Durch die Steigerung der Rechenleistung der PCs in den letzten Jahrzehnten wurde ein Niveau erreicht, das es möglich macht, durch die Zusammenstellung und Vernetzung ausreichend vieler PCs ein kostengünstiges System aufzubauen, dessen Rechenleistung für High-Performance Computing ausreichend ist. In den Projekten, bei denen das Softwarewerkzeug MMT2 zum Einsatz kommt, besteht der Bedarf an High-Performance Computing bei den Parameteranpassungen und der Modellsuche. In diesem Kapitel werden zwei Hochleistungsrechner vorgestellt, auf denen während der Laufzeit dieses Projektes MMT2 zum Einsatz kam.

15.1. Compute-Cluster

.....

Grundsätzlich kann man bereits dann schon von einem Compute-Cluster reden, wenn es aus zwei Computern besteht, die miteinander vernetzt gemeinsam Rechenaufgaben bewältigen können. Ein Compute-Cluster für High-Performance Computing besteht aber üblicherweise aus mehreren hundert Prozessoren. Für die Entwicklung des Optimierungsalgorithmus aus Kapitel 13, sowie für die Projekte aus den Kapiteln 11.2 und 11.3 konnten die Cluster-Systeme genutzt werden, die in den Abschnitten 15.1.2 und 15.1.3 vorgestellt werden.

15.1.1 Brute-Force Optimierung

Um solche Compute-Cluster softwareseitig bei der Administration, Rechenzeitverteilung und Monitoring zu unterstützen, gehört das Beowulf-Projekt [Sterling, 2001], welches bereits 1994 für Compute-Cluster eingesetzt wurde, zu den Pionieren. Typischerweise werden solche Systeme von einer Vielzahl von Nutzern für Berechnungen genutzt. Um Nutzungskonflikte zu vermeiden ist ein sog. Job-Scheduler nötig, der die verfügbare Rechenleistung koordiniert.

Beim Einsatz von MMT2 wurde dieser Job-Scheduler dazu genutzt eine Brute-Force Optimierung für Modellvarianten durchzuführen, die der Modellierer ausgewählt hat. Dazu wird für die selektierten Modelle jeweils n Mal ein Parameterraum-Sweep (Tab. 13.3) zum Finetuning der Modellparameter durchgeführt. Dem Job-Scheduler werden die Kommandozeilenaufrufe für MMT2 übermittelt, die jeweils eine Parameteranpassung starten, deren Ergebnis in eine Datei geschrieben wird. Der Job-Scheduler übernimmt dann die Rolle des Managers, der auf die freien Worker die Jobs

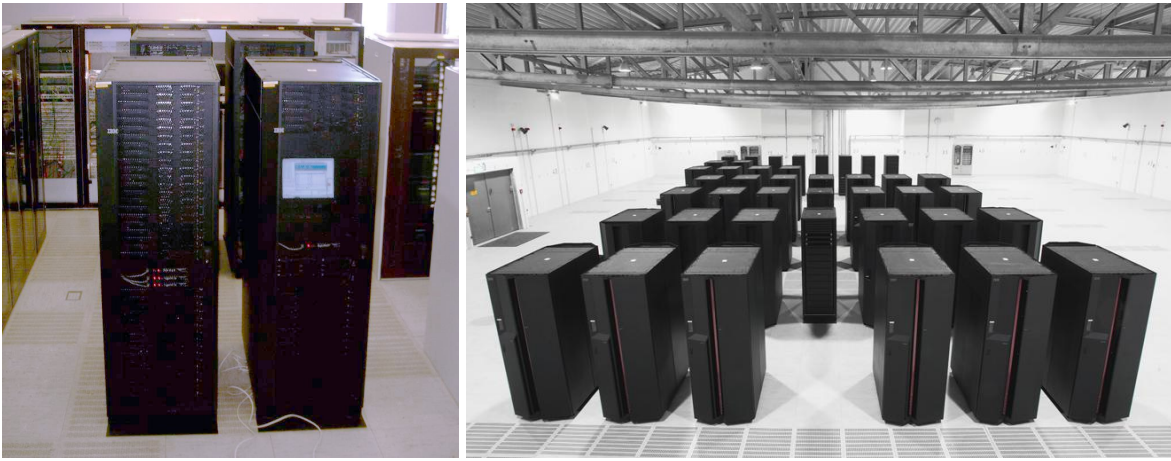


Abbildung 15.1: Links: Rubens-Cluster an der Universität Siegen mit 256 Opteron Prozessoren. Rechts: Höchstleistungsrechner Jump der Forschungszentrum Jülich GmbH mit 1312 Power4+ Prozessoren.

zuteilt und nach Beendigung von Jobs neue aus der Warteschlange zuweist. Für diese Vorgehensweise ist überhaupt keine zusätzliche Implementierung für den diskreten Teil des Optimierungsproblems nötig.

15.1.2 Rubens-Cluster

Der Rubens-Cluster der Universität Siegen (Abb. 15.1 links) erreichte nach seiner Einführung den Platz 464 auf der Top500-Liste vom Juni 2004, auf der die weltweit leistungsstärksten Hochleistungsrechner geführt werden. Der Standort des Clusters ist das HRZ (Hochschulrechenzentrum) der Universität Siegen und wird hauptsächlich im Rahmen von Forschungsprojekten der ansässigen Lehrstühle eingesetzt. Durch die Anbindung an den Rechnerverbund NRW haben auch externe Lehrstühle die Möglichkeit den Rubens-Cluster zu nutzen.

Benannt wurde der Rubens-Cluster nach dem Maler Peter Paul Rubens (1577-1640), der gebürtig aus Siegen stammt. Seine Vorgehensweise bei der Erstellung von Gemälden, durch Delegation von Teilaufgaben an seine Gehilfen, lässt sich gut mit der Manager-Worker Strategie des verteilten Rechnens vergleichen. Der Rubens-Cluster ist zusammengestellt aus 128 IBM eServer 325, die jeweils mit

- 2 AMD Opteron-64bit-Prozessoren (Model 246) mit 2 GHz und
- 2 GB RAM (2x1GB PC2700 ECC DDR SDRAM)

ausgestattet sind. Insgesamt stehen damit für Berechnungen 256 Prozessoren, 256 GB Hauptspeicher und 20 Terabyte Festplattenspeicher zur Verfügung. Gemessen mit dem Linpack-Benchmark beträgt die theoretische Spitzenleistung 1,0 Teraflop/s (R_{peak}) von denen effektiv 0,651 TFlop/s (R_{max}) nutzbar sind.

15.1.3 JUMP

Der Höchstleistungsrechner JUMP (JUelich MultiProzessor, Abb. 15.1 rechts) ist zur Zeit einer der leistungsstärksten Großrechner in Deutschland. Er erreichte nach seiner Inbetriebnahme im Februar 2004 den Platz 21 auf der Top500-Liste vom Juni 2004 und lag damit auf Platz 2 der leistungsstärksten Rechner in Europa. Der Standort des JUMP ist im NIC (John von Neumann-Institut für Computing) der Forschungszentrum Jülich GmbH und steht für Projekte der Wissenschaft, Forschung und Industrie auf dem Gebiet der Modellierung und Computersimulation zur Verfügung. Der JUMP besteht aus 41 IBM p690-Server (Regatta), die mit jeweils

- 32 Power4+ RISC-Prozessoren mit 1,7 GHz und
- 128 Gigabyte Hauptspeicher

ausgestattet sind. Insgesamt stehen damit für Berechnungen 1312 Prozessoren und 5,2 Terabyte Hauptspeicher zur Verfügung. Gemessen mit dem Linpack-Benchmark beträgt die theoretische Spitzenleistung 8,9 Teraflop/s (R_{peak}) von denen effektiv 5,6 Teraflop/s (R_{max}) nutzbar sind.

15.2. Grid Computing

.....

15.2.1 Grid-Struktur

Unter einem Compute-Cluster versteht man üblicherweise ein homogenes Netzwerk von Computern, das auf einem Standort konzentriert untergebracht ist. Grid-Computing kann man als Erweiterung des Cluster-Computing verstehen, bei dem ein heterogenes Netzwerk zugrunde liegt, das räumlich verteilt ist und sich nicht mehr auf einem einzigen Standort konzentriert. Ein Compute-Cluster kann eine Recheneinheit in einem Grid darstellen.

15.2.2 Grid-Services

Im Vergleich zum Cluster-Computing wird beim Grid-Computing durch die Heterogenität und die komplexere Netzwerkstruktur mehr Aufwand bei der Koordinierung benötigt, die durch die sog. Grid-Middleware geleistet werden soll. Das Globus Toolkit [Foster and Kesselman, 1997], das den de facto Standard für Grid-Middleware darstellt, stellt folgende Services zur Koordinierung eines Grids zur Verfügung:

- Der Monitoring and Discovery Service (MDS) stellt Informationsdienste zum Auffinden von Grid-Ressourcen bereit, sowie zum Protokollieren von Rechenzeit und Speicher durch die Nutzung.
- Das Grid Resource Allocation & Management Protocol (GRAM) regelt das Resource-Management, das zum Anfordern von Grid-Ressourcen benötigt wird.

- Die Grid Security Infrastructure (GSI) stellt Sicherheitsdienste für die Authentifizierung von Nutzern, den Schutz der Kommunikation und die Festlegung der Nutzungsrechte.
- Der Global Access to Secondary Storage (GASS) und GridFTP zur Unterstützung für das Verschieben der Daten zu den Grid-Ressourcen und zurück.

15.3. MMT2 als Grid-Service

.....

15.3.1 Wrapper zur Realisierung von Grid-Services

Die technische Realisierung eines Grid-Frameworks für die Modellbildung in der Systembiologie ist nicht trivial. Üblicherweise existieren die Softwarewerkzeuge als Einzelplatzversion mit einer GUI (Kap. 4), die nicht für Grid-Computing vorbereitet sind. Für einen Einsatz im Grid müssen für solche Softwarewerkzeuge spezielle Wrapper geschrieben werden, wie am Beispiel der Software E-Cell (Kap. 4.1), bei der der E-Cell SessionManager (ESM) [Sugimoto et al., 2005] nötig ist, um E-Cell im Grid einzusetzen. Für MMT2 wurde auch ein entsprechender Wrapper geschrieben, um den Einsatz im Grid zu ermöglichen. Abb. 15.2 zeigt die Architektur des Wrappers.

15.3.2 Ein Wrapper für MMT2

MMT2 verfügt über verschiedene Schnittstellen, die es ermöglichen MMT2 als Softwarekomponente oder als Standalone-Applikation zu betreiben. Die Schnittstellen des Simulators sind in Abb. 9.1, auf der rechten Seite abgebildet. Nach dem Compilieren des Simulators muss für den Link-Prozess eine dieser Schnittstellen ausgewählt werden. Jede Schnittstelle gibt eine andere Art des Zugriffs auf den Simulator. Bei Auswahl der Kommandozeilen-Schnittstelle (Abb. 9.1, Nr. 3) wird der Simulator als Executable erzeugt. Schnittstelle Nr. 2 ermöglicht die Steuerung von MMT2 über die Matlab Kommandozeile. Über Matlab's Mex-Interface werden Daten und Kommandos zwischen Matlab und MMT2 ausgetauscht.

Für den Einsatz im service-orientierten Grid wurde MMT2 als Grid-Service zugänglich gemacht. Dazu wurde Schnittstelle Nr. 3 verwendet, die über das JNI (Java Native Interface, Abb. 15.2) den Austausch von Daten und Befehlen von C und Java erlaubt. Über diese Schnittstelle kommuniziert MMT2 mit der Grid-Middleware um die Funktionalität von MMT2 als Grid-Service zur Verfügung zu stellen.

Der MMT2 Grid-Service wurde mit dem auf Java basierenden Globus Toolkit 3.0 (GT3) [Foster and Kesselman, 1997] realisiert. Die Grid-spezifischen Teile des Service wurden als Java-Wrapper implementiert, der den in C programmierten MMT2-Simulator über das JNI anspricht. Abb. 15.2 zeigt die von GT3 verwendeten Werkzeuge die verwendet wurden um die Grid-Middleware zu implementieren.

MMT2 ist auf Managerseite in die Grid-Middleware eingebettet (Abb. 15.2), auf der MMT2 für die Sicht auf den diskreten Teil der Optimierung verwendet wird

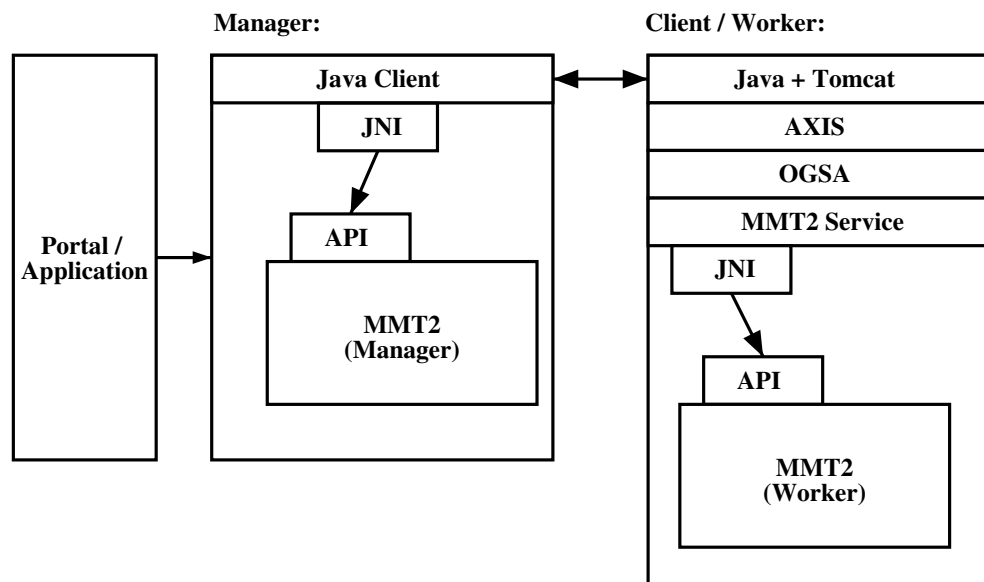


Abbildung 15.2: Der MMT2 Simulator aus Abb. 9.1 als Grid-Service eingebettet.

(Ermittlung der Nachbarschaft eines Modells). Auf der Workerseite ist MMT2 dafür zuständig, zu einem gegebenen Modell die Parameter anzupassen.

15.3.3 Workflow

Der MMT2-Grid-Service übernimmt im Prozess der Modellbildung (Abb. 1.3) den rechenintensiven Teil, während der restliche Teil der Modellbildung auf die gleiche Art und Weise vorgenommen wird, wie in Kapitel 10 vorgestellt. Das Modell muss zunächst mit MetVis und Xerlin definiert werden. Die XML-Beschreibung des Modells kann dann über ein Web-Portal (Abb. 15.2 links) abgeschickt werden, das dann mit dem MMT2-Generator den MMT2-Simulator durch Source-Code-Erzeugung erstellt und kompiliert. Schließlich muss der erstellte Simulator auf die Grid-Knoten verteilt werden damit der Grid-Service zur Verfügung steht.

Teil V

Schluss

Zusammenfassung und Diskussion

16.1. Zielsetzung

.....

Ziel der vorliegenden Arbeit war es, ein Softwarewerkzeug namens MMT2 (Metabolic Modeling Tool 2) zu implementieren, das darauf spezialisiert ist, bei der Modellbildung metabolischer Netzwerke auf Basis von Stimulus-Response-Experimenten unterstützend behilflich zu sein. Dazu konnte auf den Erfahrungen einer vorhergehenden Arbeit [Hurlebaus, 2001] aufgebaut werden. Zusätzlich zu den Vorgehensweisen in [Hurlebaus, 2001] wurden an die Architektur von MMT2 zu Beginn folgende Anforderungen gestellt:

- Definition einer XML-Sprache (M3L, Metabolic Modeling Markup Language) zur Spezifizierung von Modellen um eine einfache Handhabung, Archivierung und Bearbeitung zu ermöglichen.
- Spline Unterstützung (mit Sprungstellen) zur Eingabe von geglätteten Messdaten für die Simulation, um auf eine einfache Weise die Systemgrenzen setzen zu können, und/oder um die Komplexität des Modells zu reduzieren.
- Unterstützung von Messdaten für simulierte Metabolite um eine Parameteranpassung zu ermöglichen. Im Gegensatz zu den Splines, die die Messdaten in geglätteter Form repräsentieren, liegen hier Messpunkte zugrunde, die typischerweise sehr verrauscht sind.
- Automatische Code-Generierung um einen hochoptimierten Simulator für die Parameteranpassung zu erhalten. Dies sollte ohne den Einsatz proprietärer Software (im ggs. zu [Hurlebaus, 2001]) realisiert werden.
- Automatisches Differenzieren (AD) der Modell-ODEs zur statistischen Analyse als Alternative zum symbolischen Durchdifferenzieren der Modell-ODEs. Einerseits um keine proprietäre Software einsetzen zu müssen (wie es in [Hurlebaus, 2001] der Fall war), andererseits um die Benutzbarkeit von AD in diesem Kontext zu testen.
- Cluster- und Grid-Computing für Parameteranpassung und Modelldiskriminierung. Insbesondere bei größeren Modellen kann eine Parameteranpassung mehrere Stunden pro Modell dauern und sollte mehrmals wiederholt werden.

16.2. Verlauf des Projekts

.....

MMT2 wurde komplett neu implementiert und in der Anfangsphase eng an die Vorgehensweisen von [Hurlebaus, 2001] angelehnt. Nach einer ersten Implementierungsphase wurde MMT2 bereits freigegeben zur Benutzung in experimentellen Arbeiten zur Erstellungen von Modellen. In der praktischen Anwendung durch die Zielbenutzergruppe wurden die implementierten Vorgehensweisen in der Praxis getestet und neue Vorgehensweisen erarbeitet, von denen sich insbesondere die Folgenden als sehr nützlich erwiesen haben:

- Optional nutzbare Schnittstelle zu Matlab zum Postprocessing des simulierten Modells. Dadurch kann der Modellierer auf eine einfache Weise den Simulator durch Skripte steuern und eigene statistische Analysen programmieren, ohne dass auf die Implementierung von MMT2 Einfluss genommen werden muss. Dazu exportiert MMT2 über die MEX-Schnittstelle von Matlab alle Datenstrukturen, die zur Weiterverarbeitung interessant sein könnten.
- Unterstützung von Messdaten für simulierte Metabolite mit Angaben über die Genauigkeit jedes Messpunktes, um Messunsicherheiten in die Parameteranpassung mit einfließen lassen zu können. Diese Genauigkeitsangaben wurden in der Arbeit auch dazu verwendet, um bei der Modellbildung die Möglichkeit zu haben aktiv in die Parameteranpassung einzugreifen, indem bestimmten Messpunkten ein wesentlich höheres Gewicht gegeben wurde.
- Einfaches Handling der Installation von neuen Versionen durch Bereitstellen von Installationspaketen (RPM, Redhat Package Manager)

Darüber hinaus wurden viele kleine Features implementiert, die zusammen mit den Nutzern während der Laufzeit dieses Projektes erarbeitet wurden. Es gehört zu der allgemeinen Erfahrung in der Softwareentwicklung, dass der Einfluss auf Bedienbarkeit und Ergonomie nicht ausschließlich von den oben aufgezählten Features dominiert wird, sondern dass

1. arbeitsunterstützende Konzepte, wie oben aufgeführt, genauso wichtig sind wie
2. kleine Features zur Arbeitserleichterung und
3. Stabilität der Software.

Weist einer der Punkte Schwächen auf, so wirkt sich dies negativ auf Bedienbarkeit und Ergonomie aus. Leider kann nur Punkt 1 wissenschaftlich publiziert werden, während in den Punkten 2 und 3 mindestens genauso viel Arbeit investiert werden musste.

16.3. Modellfamilien und Modelldiskriminierung

.....

Im Vergleich zu anderen Simulationswerkzeugen bietet MMT2 neben den bereits aufgeführten Punkten eine besondere Unterstützung des Prozesses der Modellbil-

dung, die typischerweise immer ähnlich durchgeführt wird. Ohne softwareseitige Unterstützung ist dies ein sehr langwieriger Prozess, in dem viele Modellvarianten und Kombinationen von Eigenschaften durchprobiert werden müssen. MMT2 bietet eine Unterstützung für diesen Prozess, die bereits erfolgreich in einer der experimentellen Arbeiten [Wahl, 2005] eingesetzt wurde.

Da die Algorithmen zur automatischen Suche im Raum der Modellvarianten erst gegen Ende der Arbeit entwickelt wurden, konnten sie noch nicht in Modellbildungsprozessen eingesetzt werden. Bisher wurde die Modelldiskriminierung so durchgeführt, dass der Modellierer die n Modellvarianten aufgezählt hat, die innerhalb des Raumes von insgesamt m Modellen als sinnvoll erscheinen. Für diese Modelle wurden dann jeweils p Mal eine Parameteranpassung durchgeführt und die Ergebnisse nach dem Qualitätskriterium sortiert. Abschließend kann der Modellierer im Postprocessing mit Matlab die besten Parametersätze und Modellvarianten visualisieren um darauf basierend weitere Entscheidungen im Prozess der Modellbildung zu treffen.

16.4. Resultate und Ausblick

.....

Das Endresultat dieser Arbeit, in Form des Softwarewerkzeugs MMT2, das während der Erarbeitung und Erprobung von unterstützenden Vorgehensweisen bei der Modellbildung für Stimulus-Response-Experimente gewachsen ist, sind rund 7500 Zeilen C++ Code für die Implementierung des Source-Code Generators und rund 20000 Zeilen Code für den Simulator (ohne die Bestandteile, die durch den Source-Code Generator erzeugt werden). Der Source-Code Generator ist nur unter Linux lauffähig, während der erzeugte Simulator im C-Quelltext auf verschiedenen Hardwareplattformen (Linux, Windows, AIX und Solaris) lauffähig ist und von verschiedenen Compilern (gcc, mingw, icc und XLC) übersetzt werden kann.

MMT2 kam auf den Hochleistungsrechnern Rubens-Cluster der Universität Siegen und dem JUMP des Jülicher Forschungszentrums zum Einsatz und konnte damit bisher zu drei experimentellen Arbeiten unterstützend beitragen. Die neuen Vorgehensweisen und Implementierungen in MMT2 wurden in vier Zeitschriften- und Tagungspaper veröffentlicht und flossen weiterhin in mehrere Vorträge ein.

Mögliche zukünftige Arbeiten und Erweiterungen an MMT2 wären:

- Der MMT2 Grid-Service (Kap. 15.3) ist derzeit auf eine Zentrale Instanz angewiesen, die für die Auswahl der verfügbaren Rechner im Grid zuständig ist. Eine Weiterentwicklung dieser Vorgehensweise ist das Ad-Hoc Grid-Computing [Friese et al., 2004], mit dem der MMT2 Grid-Service realisiert werden könnte, um die Handhabung zu vereinfachen.
- Der Optimierungsalgorithmus für die Modellselektion (Kap. 13) ist derzeit mit der Java-RMI-Technik realisiert, die mit Job-Queuing-Systemen zusammenarbeiten kann, die üblicherweise auf Hochleistungsrechnern (Kap. 15.1.2, 15.1.3) zum

Einsatz kommt. Diese Middleware könnte gegen Grid-Middleware ausgetauscht werden, oder sogar gegen die Middleware für Ad-Hoc Grid-Computing.

- Erweiterung des Optimierungsalgorithmus zur Suche im Modellraum. Im Bereich der kontinuierlichen Optimierung könnte der Manager überwachen, ob die Parametersätze von mehreren parallel laufenden Parameteranpassungen in einem Konfidenzintervall liegen. In einem solchen Fall ist anzunehmen, dass diese Parameteranpassungen auf dasselbe lokale Minimum zusteuern. Von diesen Parameteranpassungen könnten dann alle, bis auf eine, abgebrochen werden um Ressourcen für die weitere Modellsuche frei zu machen.
- Portierung von MMT2 auf Windows, was aufgrund der umfangreichen und unüblichen Installation eines Compilers auf einem Windowssystem eine anspruchsvolle Aufgabe ist, aber durch Software wie Cygwin oder MinGW wesentlich erleichtert werden kann.
- Die Unterstützung für globale Parameter (Kap. 7.5) wurde der Implementierung der Sensitivitätsberechnung (Kap. 9.5) mit möglichst wenig Aufwand nachträglich hinzugefügt. Dabei wurde in Kauf genommen, dass die Berechnung zugunsten einer schnellen Implementierungsphase weniger effizient mit Rechenzeit und Speicher umgeht. Diese Implementierung hat erhebliches Einsparungspotenzial am Speicherplatzbedarf.
- Schließlich gibt es noch viele Ideen für die Verbesserung der Implementierung, die einerseits die Ergonomie der Software von Fall zu Fall mehr oder weniger steigern würden, andererseits aber, gemessen an den grundsätzlichen Vorgehensweisen und Methoden, die in dieser Arbeit diskutiert wurden, einen sehr geringen Stellenwert einnehmen. Ein Beispiel dafür wurde in Kapitel 14.4.2 erwähnt.

Literaturverzeichnis

- [Alberts et al., 2004] Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J. D. (2004). *Molecular Biology of the Cell*. Garland Publishing.
- [Arkin, 2001] Arkin, A. P. (2001). Synthetic cell biology. *Current Opinion in Biotechnology*, 12:638–644.
- [Berman et al., 2003] Berman, F., Fox, G., and Hey, A. J. G., editors (2003). *Grid Computing - Making the Global Infrastructure a Reality*. John Wiley and Sons Ltd.
- [Blattner et al., 1997] Blattner, F., Plunkett, G., Bloch, C., Perna, N., and Burland, V. e. a. (1997). The complete genome sequence of Escherichia coli K-12. *Science*, 277:1453.
- [Bleeken, 1990] Bleeken, S. (1990). Welches sind die existentiellen Grundlagen lebender Systeme? *Naturwissenschaften*, 77:277–282.
- [Borowiak, 1990] Borowiak, D. (1990). *Model discrimination for nonlinear regression models*. Marcel Dekker, New York.
- [Box and Hill, 1967] Box, G. E. P. and Hill, W. (1967). Discrimination among mechanistic models. *Technometrics*, 9:57–71.
- [Buchholz et al., 2001] Buchholz, A., Takors, R., and Christian, W. (2001). Quantification of Intracellular Metabolites in Escherichia coli K12 Using Liquid Chromatographic-Electrospray Ionization Tandem Mass Spectrometric Techniques. *Analytical Biochemistry*, 295(2):129–137.
- [Burnham and Anderson, 1998] Burnham, K. P. and Anderson, D. R. (1998). *Model Selection and Interference*. Springer.
- [Cohen and Hindmarsh, 1996] Cohen, S. and Hindmarsh, A. (1996). CVODE, a Stiff/Nonstiff ODE Solver in C. *Computers in Physics*, 10(2):138–43.
- [Cornish-Bowden, 1996] Cornish-Bowden, A. (1996). *Fundamentals of Enzyme Kinetics*. Portland Press, London.
- [Csete and Doyle, 2002] Csete, M. E. and Doyle, J. C. (2002). Reverse engineering of biological complexity. *Science*, 295:1664–1669.
- [de Koning and van Dam, 1992] de Koning, W. and van Dam, K. (1992). A method for the determination of changes of glycolytic metabolites in yeast on a subsecond time scale using extraction at neutral pH. *Analytical biochemistry*, 204:118–123.
- [Degenring, 2004] Degenring, D. (2004). *Erstellung und Validierung mechanistischer Modelle für den mikrobiellen Stoffwechsel zur Auswertung von Substrat-Puls-Experimenten*. Dissertation, Universität Rostock.
- [Dumke, 2003] Dumke, R. (2003). *Software Engineering*. Vieweg.

- [Ewings and Doelle, 1980] Ewings, K. N. and Doelle, H. W. (1980). Further kinetic characterization of the non-allosteric phosphofructokinase from *Escherichia coli* K-12. *Biochimica et biophysica acta*, 615(1):103–112.
- [Fenyó, 1999] Fenyó, D. (1999). The Biopolymer Markup Language. *Bioinformatics*, 15:339–340.
- [Foster and Kesselman, 1997] Foster, I. and Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, pages 115–128.
- [Foster and Kesselman, 2003] Foster, I. and Kesselman, C. (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc.
- [Friese et al., 2004] Friese, T., Smith, M., and Freisleben, B. (2004). Hot service deployment in an ad hoc grid environment. In *Proceedings of the 2nd Int. Conference on Service-Oriented Computing (ICSOC'04), New York, USA*, pages 75–83. ACM Press.
- [Gancedo and Gancedo, 1973] Gancedo, J. and Gancedo, C. (1973). Concentration of intermediary metabolites in yeast. *Biochemie*, 55:205–211.
- [Gombert and Nielsen, 2000] Gombert, A. K. and Nielsen, J. (2000). Mathematical modelling of metabolism. *Current Opinion in Biotechnology*, 11(2):180–186.
- [Goryanin et al., 1999] Goryanin, I., Hodgman, T. C., and Selkov, E. (1999). Mathematical simulation and analysis of cellular metabolism and regulation. *Bioinformatics*, 15:749–758.
- [Griewank et al., 1996] Griewank, A., Juedes, D., Mitev, H., Utke, J., Vogel, O., and Walther, A. (1996). Automatic differentiation by overloading in C++. *ACM TOMS*, 22(2):131–167. Algor. 755.
- [Haunschild, 2001] Haunschild, M. D. (2001). Ein Werkzeug zur automatischen Modellgenerierung und verteilten Simulation metabolischer Netzwerke. Diplomarbeit, University of Siegen, Dept. of Electrical Engineering and Computer Science (FB 12), University Siegen, Germany.
- [Haunschild, 2005] Haunschild, M. D. (2005). MMT2 reference manual and tutorial. http://www.simtec.mb.uni-siegen.de/software_mmt2.0.html.
- [Haunschild et al., 2005a] Haunschild, M. D., Freisleben, B., Takors, R., and Wiechert, W. (2005a). Investigating the dynamic behavior of biochemical networks using model families. *Bioinformatics*, 21:1617–1625.
- [Haunschild et al., 2002] Haunschild, M. D., Freisleben, B., Wiechert, W., and Takors, R. (2002). 16th European Simulation Multiconference: Modelling and Simulation 2002, June 3-5, 2002, Fachhochschule Darmstadt, Darmstadt, Germany. In Amborski, K. and Meuth, H., editors, *ESM*, pages 436 – 440. SCS Europe.
- [Haunschild et al., 2004] Haunschild, M. D., von Lieres, E., Takors, R., Wahl, A., Qeli, E., Freisleben, B., and Wiechert, W. (2004). MMT 2: Supporting the Modeling Process for Rapid Sampling Experiments. *Metabolic Engineering V: Genome to Product (Lake Tahoe)*, September 19-23. Poster presentation.

- [Haunschild et al., 2005b] Haunschild, M. D., Wahl, S. A., Freisleben, B., and Wiechert, W. (2005b). A general framework for large scale model selection. *eingereicht*.
- [Haunschild and Wiechert, 2003] Haunschild, M. D. and Wiechert, W. (2003). Sensitivity analysis of metabolic network models. In *Proceedings of ASIM 2003, 17. Symposium Simulationstechnik, 16.09. - 19.09.2003 in Magdeburg*, pages 415–420.
- [Haunschild and Wiechert, 2004] Haunschild, M. D. and Wiechert, W. (2004). Model selection in a family of metabolic networks. In *PARAOPE, International Workshop on Parameter Estimation and Optimal Design of Experiments, 30.6. - 2.7 in Heidelberg*.
- [Haunschild et al., 2005c] Haunschild, M. D., Wiechert, W., Wahl, A., Qeli, E., Freisleben, B., and Oldiges, M. (2005c). Model families as a tool for identifying regulatory mechanisms in intracellular biochemical networks. In *Proceedings on 6th International Conference on Systems Biology, 19-22 Oktober in Boston, Massachusetts, USA*.
- [Haunschild et al., 2005d] Haunschild, M. D., Wiechert, W., Wahl, A., Qeli, E., Freisleben, B., and Oldiges, M. (2005d). Stimulus response experiments for model driven discovery. In *Proceedings on 4th Workshop on Computation of Biochemical Pathways and Genetic Networks, 12-13 September in Heidelberg*.
- [Heinrich and Schuster, 1996] Heinrich, R. and Schuster, S. (1996). *The Regulation of Cellular Systems*. Chapman and Hall.
- [Herold, 2003] Herold, H. (2003). *lex & yacc . Die Profitools zur lexikalischen und syntaktischen Textanalyse*. Addison-Wesley.
- [Hindmarsh, 1983] Hindmarsh, A. C. (1983). ODEPACK, A Systematized Collection of ODE Solvers. In R. S. Stepleman et al., editor, *Scientific Computing*, volume 1 of *IMACS Transactions on Scientific Computation*, pages 55–64, North-Holland, Amsterdam.
- [Hoffmann et al., 2004] Hoffmann, J., Ellingwood, C., Bonsu, O., and Bentil, D. (2004). Ecological model selection via evolutionary computation and information theory. *Journal of Genetic Programming and Evolvable Machines*, 5(2):229–241.
- [Hofmann and Kopperschläger, 1982] Hofmann, E. and Kopperschläger, G. (1982). Phosphofructokinase in yeast. In Wood, W. A., editor, *Methods in Enzymology*, pages 49–60. Academic Press.
- [Hucka and et. al, 2003] Hucka, M. and et. al (2003). The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531.
- [Hucka et al., 2001] Hucka, M., Finney, A., Sauro, H., Bolouri, H., Doyle, J., and Kitano, H. (2001). The ERATO Systems Biology Workbench: An Integrated Environment for Multiscale and Multitheoretic Simulations in Systems Biology. *Foundations in System Biology*, MIT Press.
- [Hurlebaus, 2001] Hurlebaus, J. (2001). *A pathway modeling tool for metabolic engineering*. Dissertation, Universität Bonn.

- [Hurlebaus et al., 2002] Hurlebaus, J., Buchholz, A., Takors, R., Alt, W., and Wiechert, W. (2002). MMT - A pathway modeling tool applied to data from rapid sampling experiments. *In Silico Biology*, 2:1–18.
- [King et al., 2002] King, R., Leifheit, J., and Freyer, S. (2002). Automatic identification of mathematical models of chemical and biochemical reaction systems. In *proc. CHISA 2002, 25-29 August, Prag*.
- [Kitano, 2002] Kitano, H. (2002). Systems biology: a brief overview. *Science*, 295(5560):1662-4.
- [Klamt and Gilles, 2004] Klamt, S. and Gilles, E. (2004). Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20:226–234.
- [Lange et al., 2001] Lange, H. C., Emam, M., Zuijlen, G. v., Visser, D., van Dam, J. C., Frank, J., de Mattos, M. J. T., and Heijnen, J. J. (2001). Improved rapid sampling for in vivo kinetics of intracellular metabolites in *saccharomyces cerevisiae*. *Biotechnology and Bioengineering*, 75:334–344.
- [Lee and Hovland, 2001] Lee, S. L. and Hovland, P. D. (2001). Sensitivity Analysis Using Parallel ODE Solvers and Automatic Differentiation in C: SensPVODE and ADIC. In *Proc. Automatic Differentiation: From Simulation to Optimization*, page p. 217. Springer.
- [Leifheit and King, 2004] Leifheit, J. and King, R. (2004). (Semi-)automatic modeling of biological reaction systems with TAM-B. In *In: 9th Int. Conference Computer Application in Biotechnology CAB9, Nancy, France, IFAC*.
- [Linhart and Zucchini, 1986] Linhart, H. and Zucchini, W. (1986). *Model Selection*. Wiley, New York.
- [Lloyd et al., 2004] Lloyd, C. M., Halstead, M. D. B., and Nielsen, P. F. (2004). CellML: its future, present and past. *Progress in Biophysics and Molecular Biology*, 85(2-3):433–450.
- [Loew and Schaff, 2001] Loew, L. M. and Schaff, J. C. (2001). The Virtual Cell: A software environment for computational cell biology. *Trends in Biotechnology*, 19:401–406.
- [Maehly, 1960] Maehly, H. J. (1960). Methods for Fitting Rational Approximations, Part I: Telescoping Procedures for Continued Fractions. *Journal of the ACM*, 7(2):150 – 162.
- [Magnus, 2005] Magnus, J. (2005). *Modelling and Analysis of the valine biosynthesis pathway in Corynebacterium glutamicum*. Dissertation, Institut für Bioverfahrenstechnik, Universität Stuttgart.
- [Mendes, 1997] Mendes, P. (1997). Biochemistry by numbers: simulation of biochemical pathways with Gepasi 3. *Trends in biochemical sciences*, 22(9):361–363.
- [Morton-Firth and Bray, 1998] Morton-Firth, C. J. and Bray, D. (1998). Predicting temporal fluctuations in an intracellular signalling pathway. *Journal of theoretical biology*, 192:117–128.
- [Murray-Rust and Rzepa, 2003] Murray-Rust, P. and Rzepa, H. S. (2003). Chemical Markup, XML and the Worldwide Web. Part 4. CML Schema. *Journal of chemical information and computer sciences*, 43:757 – 772.

- [Oldiges, 2005] Oldiges, M. (2005). *Metabolomanalyse zur Untersuchung der Dynamik im Aromatenbiosyntheseweg in L-Phenylalanin Produzenten von Escherichia coli*. Dissertation, Universität Bonn.
- [Pennington and Berzins, 1994] Pennington, S. V. and Berzins, M. (1994). New NAG library software for first-order partial differential equations. *ACM Transactions on Mathematical Software (TOMS)*, 20(1):63 – 99.
- [Petersen et al., 2003] Petersen, S., von Lieres, E., de Graaf, A. A., Sahm, H., and Wiechert, W. (2003). Chapter 10, A Multi-scale Approach for the Predictive Modeling of Metabolic Regulation. In B.N. Kholodenko, H. W., editor, *Metabolic Engineering in the Post Genomic Era*. Horizon Scientific Press.
- [Pettinen et al., 2005] Pettinen, A., Aho, T., Smolander, O.-P., Manninen, T., Saarinen, A., Taatola, K.-L., Yli-Harja, O., and Linne, M.-L. (2005). Simulation tools for biochemical networks: evaluation of performance and usability. *Bioinformatics*, 21(3):357–363.
- [Qeli, 2005] Qeli, E. (2005). *Information Visualization Techniques for Metabolic Engineering*. Dissertation in Vorbereitung, Universität Marburg.
- [Qeli et al., 2003] Qeli, E., Wahl, A., Degenring, D., Wiechert, W., and Freisleben, B. (2003). Met-Vis: A Tool for Designing and Animating Metabolic Networks. In *Proceedings of the 2003 European Simulation and Modelling Conference, Naples, Italy*, pages 333–338. Eurosis Press.
- [Qeli et al., 2005] Qeli, E., Wiechert, W., and Freisleben, B. (2005). Visual Exploration of Time-Varying Matrices. In *Proceedings of the 2005 International Conference on Information Visualization (IV2005), London, UK*, page to appear. IEEE Press.
- [Rizzi et al., 1997] Rizzi, M., Baltes, M., Theobald, U., and Reuß, M. (1997). In vivo analysis of metabolic dynamics in *Saccharomyces cerevisiae*: II. Mathematical model. *Biotechnology and Bioengineering*, 55(4):592–608.
- [Rowan, 1990] Rowan, T. H. (1990). *Functional Stability Analysis of Numerical Algorithms*. Ph.D. thesis, Department of Computer Sciences, University of Texas at Austin.
- [Sauro, 1993] Sauro, H. (1993). SCAMP: a general-purpose simulator and metabolic control analysis program. *Biotechnology and applied biochemistry*, 9.
- [Sauro, 2000] Sauro, H. M. (2000). Jarnac: An Interactive Metabolic Systems Language in Computation in Cells. In Hamid & Paton, R. C., editor, *Proceedings of an EPSRC Emerging Computing Paradigms Workshop Bolouri*, Technical Report No. 345, University of Hertfordshire, UK. Dept. of Computer Science.
- [Sauro, 2001] Sauro, H. M. (2001). JDesigner: a simple biochemical network designer. Technical report.
- [Schaefer et al., 1999] Schaefer, U., Boos, W., Takors, R., , and Weuster-Botz, D. (1999). Automated sampling device for monitoring intracellular metabolite dynamics. *Analytical biochemistry*, 270:88–96.

- [Schilling et al., 1999] Schilling, C. H., Schuster, S., Palsson, B., and Heinrich, R. (1999). Metabolic pathway analysis: basic concepts and scientific applications in the post-genomic era. *Biotechnology Progress*, 15:296–303.
- [Schuster et al., 2000] Schuster, S., Dandekar, T., and Fell, D. (2000). Description of the algorithm for computing elementary flux modes. <http://bms-mudshark.brookes.ac.uk/algorithm.pdf>.
- [Schuster et al., 1999] Schuster, S., Fell, D., and Dandekar, T. (1999). A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks. *Nature Biotechnology*, 18:326–332.
- [Seber and Wild, 1989] Seber, G. A. F. and Wild, C. J. (1989). *Nonlinear Regression*. Wiley.
- [Segel, 1975] Segel, I. H. (1975). *Enzyme Kinetics: Behaviour and Analysis of Rapid Equilibrium and Steady-State*. Wiley, New York.
- [Stephanopoulos et al., 1998] Stephanopoulos, G., Aristidou, A., and Nielsen, J. (1998). *Metabolic Engineering: Principles and Methodologies*. Academic Press, San Diego.
- [Sterling, 2001] Sterling, T. L., editor (2001). *Beowulf Cluster Computing with Linux*. MIT Press.
- [Sugimoto et al., 2005] Sugimoto, M., Takahashi, K., Kitayama, T., Ito, D., and Tomita, M. (2005). Distributed cell biology simulations with E-Cell system. *Lecture Notes in Computer Science*, 3370:20–31.
- [Takahashi et al., 2003] Takahashi, K., Ishikawa, N., Sadamoto, Y., Sasamoto, H., Ohta, S., Shiozawa, A., Miyoshi, F., Naito, Y., Nakayama, Y., and Tomita, M. (2003). E-Cell 2: Multiplatform E-Cell simulation system. *Bioinformatics*, 19:1727–1729.
- [Takors et al., 2001] Takors, R., Weuster-Botz, D., Wiechert, W., and Wandrey, C. (2001). A Model Discrimination Approach for Data Analysis and Experimental Design in Engineering and Manufacturing for Biotechnology, (Hofman, M. and Thonart, P., eds). *Kluwer Academic Publisher*, pages 111–128.
- [Takors et al., 1997] Takors, R., Wiechert, W., and Weuster-Botz, D. (1997). Experimental design for the identification of macrokinetic models and model discrimination. *Biotechnology and Bioengineering*, 56:564–567.
- [Tomita, 2001] Tomita, M. (2001). Whole-cell simulation: a grand challenge of the 21st century. *Trends in Biotechnology*, 19:6.
- [Tomita et al., 2001] Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter, J., and Hutchison, C. (2001). E-CELL: software environment for whole-cell simulation. *Bioinformatics*, 15(1):72–84.
- [Tränkle et al., 2000] Tränkle, F., Zeitz, M., Ginkel, M., and Gilles, E. (2000). ProMoT: a modeling tool for chemical processes. *Mathematical and Computer Modelling of Dynamical Systems*, 6:283–307.
- [Visser and Heijnen, 2003] Visser, D. and Heijnen, J. (2003). Dynamic simulation and metabolic re-design of a branched pathway using linlog kinetics. *Metabolic Engineering*, 5:164–176.

- [Voit and Savageau, 1982] Voit, E. O. and Savageau, M. A. (1982). Power-law approach to modeling biological systems; III. Methods of analysis. *Journal of fermentation technology*, 60:233–241.
- [Wahl, 2005] Wahl, S. A. (2005). *Methoden zur Analyse metabolischer Netzwerke im stationären und instationären Zustand*. Dissertation, Universität Siegen.
- [Wastney et al., 1999] Wastney, M. E., Patterson, B. H., Linares, O. A., Greif, P. C., and Boston, R. C. (1999). *Investigating biological Systems using modeling*. Academic Press, San Diego, USA.
- [Weitzel, 2003] Weitzel, M. (2003). Eine Trend-String-basierte Methode zur Parameteranpassung bei nichtlinearen zeitabhängigen Funktionen. Diplomarbeit, Universität Siegen.
- [Werner, 2002] Werner, E. (2002). Bioinformatics and the Systems Biology Hierarchy: An Overview. *New Drugs Magazine*.
- [Wiechert, 2002] Wiechert, W. (2002). Modelling and simulation: Tools for metabolic engineering. *Journal of Biotechnology*, 94(1):37–63.
- [Wiechert and Takors, 2004] Wiechert, W. and Takors, R. (2004). Chapter 11: Validation of metabolic models: concepts, tools, and problems. In Kholodenko, B. N. and Westerhoff, H. V., editors, *Metabolic Engineering in the Post Genomic Era*. Horizon Bioscience, Wymondham, England.
- [Wolkenhauer, 2001] Wolkenhauer (2001). Systems Biology: The reincarnation of Systems Theory Applied in Biology? *Briefings in Bioinformatics*, 2(3):258–270.
- [Yen and Lee, 1996] Yen, J. and Lee, B. (1996). Using a hybrid genetic algorithm and fuzzy logic for metabolic modeling. In *FUZZ-IEEE 96: Proceedings of the 5th IEEE, International Conference on Fuzzy Systems, IEEE*.
- [Zhu et al., 1997] Zhu, C., Byrd, R. H., and Nocedal, J. (1997). Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550 – 560.