Analytical methods for genome-scale metabolic networks applied to *Streptococcus agalactiae*

Albert Gevorgyan

A thesis submitted in partial fulfilment of the requirements of
Oxford Brookes University
for the award of the degree Doctor of Philosophy

Cell Systems Modelling Group
School of Life Sciences

July 2009

In loving memory of my father,
who taught me to read and to think.

Acknowledgements

Over the last four years, it has been a great luck and pleasure for me to be supervised by Professor David Fell, Dr Mark Poolman and Dr Mark Anthony. I am enormously grateful to them for sharing their knowledge, ideas and passion for science with me. Their support and advice made this work possible, hugely contributed to its quality and enabled me to start fulfilling myself as a researcher.

I thank the members of the Cell Systems Modelling Group: Dr Bhushan Bonde, Dr Frances Brightman, Dr Harshil Patel, Achuthanunni Chokkathukalam, Avijit Guha Roy, Chiara Ferrazzi, Sudip Kundu and Amar Ghaisas, for being excellent teammates and travelmates, and for the wonderful working atmosphere they created in the group.

I thank Professor Michael Pidcock for the valuable discussions and help with the mathematical verification and presentation of my article. I thank Professor Stefan Schuster for sharing his knowledge about the conversion cone, which served as the theoretical basis for a considerable part of the work presented. I thank Professor Amanda Jones for the *S. agalactiae* gene expression data she kindly made available to our group. I thank Dr Jean-Marc Schwartz, Karin Radrich and Dr Andreas Hoppe for helping me to test the method of stoichiometric consistency check on their models. I thank Dr Maurice Scheer for sharing his knowledge of microarray analysis. My very special thanks go to my former teacher, Professor Ina Koch, who unveiled the beautiful world of bioinformatics for me and recommended me for this project.

I thank the Oxford Brookes University for funding this work and the staff of the School of Life Sciences for the administrative support. In particular, I would like to acknowledge Farida Ben Ghorab, Dr David Evans, Dr John Runions, Jill Organ and Catherine Hutchinson.

I am very grateful to my brother-in-law for supporting my visa extension. Finally, my deepest thanks go to my fiancee, my mother and my sister, for all their love, warmth and patience. They helped me whenever I needed help and gave me the confidence to overcome all the challenges I faced during this work.

Abstract

Streptococcus agalactiae is a leading cause of neonatal infectious mortality, causing pneumonia, septicaemia and meningitis. The recently reported genome sequencing results for several strains provide an opportunity for the computational metabolic reconstruction of the organism. The present work involves construction and analysis of genome-scale metabolic models of *S. agalactiae*, the characterisation of physiological properties and the identification of potential drug targets.

The models of four strains of the organism were constructed, using functional genome annotations as the primary input data. Two alternative data resources were used: existing annotations from the KEGG†[1] [75] database and *de novo* annotations generated using the tools RPS-BLAST† [67] and PRIAM† [12]. The lists of biochemical reactions were obtained from the KEGG LIGAND† [35] database. The feasible reaction directions were defined according to the literature data. A number of hypothetical reactions were included, such as transporters and biosynthesis reactions for major biomass components.

Algorithms were developed for the detection and correction of errors in the input data, such as inconsistent naming of metabolites. Quantitative indicators were introduced for the evaluation of model quality, based on the conformity with physical and biochemical constraints. Methods for the detection of stoichiometric inconsistencies (a common type of modelling error) were introduced and published [33].

Several reannotations were made using two alternative approaches: manual inspection of the 'failed' pathways and stochastic optimisation of genome annotations. The latter approach involves a simulated annealing algorithm [53] which optimises the quality of a model by randomly reassigning the enzymatic functions to the genes.

Apart from the existing methods of metabolic network analysis (such as elementary mode analysis [37]), original methods were developed, characterising the interactions of a metabolic network with its environment. The concepts of elementary substrate and product compositions were introduced and highly efficient algorithms for their detection were proposed. These algorithms enabled the enumeration of fermentation substrates, of minimal compositions of fermentation products, and of minimal combinations of amino acids sufficient for protein biosynthesis.

For drug target prediction, a computationally feasible algorithm for the detection of minimal cut sets (essential groups of reactions [55]) was developed. Lists of minimal cut sets for biomass synthesis and energy production were calculated.

A comparison with the available literature data demonstrated that the accuracy of the modelling predictions was comparable with that of experimental results. This observation confirms the informativeness and reliability of genome-scale metabolic reconstructions.

[1] Here and further, the symbol † directs the reader to the List of URLs.

Contents

List of Abbreviations

Metabolite abbreviations

-P	-phosphate
2PG	2-Phospho-glycerate
3PG	3-Phospho-glycerate
Ala	Alanine
Arg	Arginine
Asn	Asparagine
Asp	Aspartate
BPG	3-Phospho-glyceroyl phosphate
DAHP	2-Dehydro-3-deoxy-arabino-heptonate 7-phosphate
F6P	Fructose 6-phosphate
G3P	(2R)-2-Hydroxy-3-(phosphonooxy)-propanal
G6P	Glucose 6-phosphate
GSH	Glutathione
Gln	Glutamine
Glu	Glutamate
Gly	Glycine
Lys	Lysine
PEP	Phosphoenolpyruvate
PPi	Pyrophosphate
Pi	Orthophosphate
Pyr	Pyruvate
SAH	S-Adenosylhomocysteine
SAM	S-Adenosylmethionine
Ser	Serine
THF	Tetrahydrofolate

Other abbreviations

ABC	ATP-binding cassette
API	application programming interface
BLAST	Basic Local Alignment Search Tool
BRENDA	BRaunschweig ENzyme DAtabase
E-value	expectation value
EC	Enzyme Commission
FTP	File Transfer Protocol
GLPK	GNU Linear Programming Kit
GMP	GNU multiple precision arithmetic library
GUI	graphical user interface
ID	identifier
IUPAC	International Union of Pure and Applied Chemistry
InChi	International Chemical Identifier
KEGG	Kyoto Encyclopedia of Genes and Genomes
LP	linear programming
MCS	minimal cut set
MILP	mixed integer linear programming
NCBI	National Center for Biotechnology Information
OOP	object-oriented programming
ORF	open reading frame
PGDB	pathway-genome database
PRIAM	profils pour l'identification automatisee du metabolisme
PSI-BLAST	Position-Specific Iterative BLAST
PSSM	Position-Specific Scoring Matrix
RPS-BLAST	Reverse Position Specific BLAST
SBML	Systems Biology Markup Language
SMILES	Simplified Molecular Input Line Entry Specification
SWIG	Simplified Wrapper and Interface Generator
TIGR	The Institute for Genomic Research
XML	eXtensible Markup Language
XSPY	extended ScrumPy
csv	comma-separated values
iff	if and only if
yacc	Yet Another Compiler Compiler

List of URLs

BRENDA	www.brenda-enzymes.org
BioCyc	biocyc.org
BioPython	biopython.org
ENZYME	www.expasy.ch/enzyme
Enzyme Commision	www.chem.qmul.ac.uk/iubmb/enzyme
GLPK	www.gnu.org/software/glpk
GMP	www.swox.com/gmp
GenBank	ftp://ftp.ncbi.nih.gov/genbank/genomes/Bacteria
Graphviz	graphviz.org
IUPAC	www.iupac.org
InChi	www.iupac.org/inchi
Institute for Systems Biology	www.systemsbiology.org
KEGG	www.genome.jp/kegg
KEGG API	www.genome.jp/kegg/soap
KEGG FTP	ftp://ftp.genome.jp/pub/kegg/ligand
KEGG LIGAND	www.genome.jp/kegg/ligand.html
Latex	www.latex-project.org
Lindo	www.lindo.com
MatLab	www.mathworks.com/products/matlab
Mathematica	www.wolfram.com/products/mathematica
NCBI	www.ncbi.nlm.nih.gov
NJplot	pbil.univ-lyon1.fr/software/njplot.html
Newick	evolution.genetics.washington.edu/phylip/newicktree.html
NumPy	scipy.org/NumPy
PRIAM	www.priam.prabi.fr
PUBCHEM	pubchem.ncbi.nlm.nih.gov
Ply	www.dabeaz.com/ply
Pydot	www.dkbza.org/pydot.html
Python	www.python.org
RPS-BLAST	www.ncbi.nlm.nih.gov/Structure/cdd/cdd_help.shtml
SBML	sbml.org
SMILES	www.daylight.com/smiles
SWIG	www.swig.org
SagaList	genolist.pasteur.fr/SagaList/index.html
SciPy	www.scipy.org
ScrumPy	mudshark.brookes.ac.uk/index.php/Software/ScrumPy
TIGR	www.tigr.org
cPickle	docs.python.org/library/pickle.html
lex and yacc	dinosaur.compilertools.net

Chapter 1

Introduction

1.1 Rationale

The original objective of the present work was the genome-scale metabolic reconstruction of the pathogenic bacterium *Streptococcus agalactiae*, the investigation of its biochemical network and the detection of potential drug targets. The established theoretical and practical framework of metabolic modelling provided the methodological basis for the work. However, as the work progressed, effective solutions of a range of problems required the development of original concepts, methods and software. Hence, the scope of the thesis was extended to the development of a methodology for computational drug target prediction in pathogenic bacteria in general.

The genome-scale metabolic models of four strains of the organism were expected to reflect the current knowledge of its physiology and to enable further useful predictions. The reconstruction methods were intended to be possibly automatable, reproducible and universal, i.e. applicable to various bacterial genomes. The achievement of these criteria is strongly hindered by the presence of missannotations and ambiguous gene function predictions (multiple enzymatic functions predicted for the same gene with comparable expectation values). Original solutions had to be sought to solve these problems as well as many other technical issues associated with the automatic data import from metabolic databases [84, 33].

A number of methods have been proposed and are widely used for the comprehensive characterisation of metabolic networks, such as the calculation of elementary flux modes [37] and extreme pathways [103]. Unfortunately, these methods are poorly scalable to genome-scale models because of their computational complexity. Functional stoichiometric analysis [118] is a computationally tractable alternative which focuses on the interactions of a metabolic network with its environment, rather than on its internal structure. The development of original methods of functional analysis was expected to provide highly efficient tools for the characterisation of such important physiological properties as

nutritional requirements and possible fermentation routes.

The potential of metabolic modelling for drug target prediction is based on the central role of metabolic enzymes in physiology, their high conservation among pathogens and suitability for high-throughput identification [6]. Although cases of prediction and validation of drug targets have been reported [5, 127], effective methods of computational drug target identification are still to be established. An attempt to advance the existing solutions proposed for this problem was made in the scope of the present work.

1.2 Overview of *Streptococcus agalactiae*

Streptococcus agalactiae(Group B streptococcus; GBS) is a leading cause of neonatal infectious morbidity and mortality in the United States and Western Europe, causing pneumonia, septicaemia and meningitis. It is also an increasing cause of mortality in immunocompromised adults [107] and a major cause of intramammary infection in dairy cattle [52]. The organism is found in vaginal and anorectal flora of approximately 10-30% of pregnant women in UK and US [40], more frequently in lower socioeconomic groups [8]. Nine serotypes of GBS cause disease in human: Ia, Ib and II-VIII. The early-onset form of GBS ($<$ 7 days of age) occurs with a reported incidence of 1-2 cases per 1000 live births and has a mortality rate of nearly 20% of affected neonates [1]. This form is associated with prematurity, prolonged labor, preterm or prolonged rupture of membranes and chorioamnionitis. The less frequent late onset form (mostly in the first 3 months [106]) is mostly caused by the serotype III, which is also responsible for a significant proportion of the early onset desease, and for nearly all cases of meningitis, regardless of age of the onset [48, 40].

The early onset form is transmitted vertically through ruptured or intact membranes, or, less typically, during passage through a colonised birth canal. Fetal aspiration of contaminated amniotic fluid initiates pneumonia, leading further to bacteremia, septic shock and meningitis [8]. The rapid growth of *S. agalactiae* in amniotic fluid (up to 100-fold of *E. coli* growth [38]) is an important factor in the pathogenesis of infection. The lung is a probable portal of entry to bloodstream, due to the ability to adhere to and invade alveolar and endothelial cells [34]. The late onset form is acquired postpartum from maternal and other sites and leads to meningitis, bacteremia and ostheoarthritis [8]. Breast milk has been experimentally proven to be the source of late-onset GBS infection in a number of works [30, 123, 60].

The genomes of several *S. agalactiae* strains have been published recently [34, 115]. The genome of the strain NEM316 (serotype III) contains 2118 protein coding genes, 50% of which have orthologs in the closely related *S. pyogenes* genome. Putative functions could be assigned to 62% of the genes; 29% were similar to unknown proteins and 9% were unique. The analysis of the genes having no orthologs in *S. pyogenes* detected their clustering in 200 regions ranging in size from 1-77 genes. Fourteen large islands (11-77 genes) contain all genes related to mobile elements except one and the majority of known or putative

virulence genes [34]. A study comparing the strains NEM316 and 2603VR (serotype V) confirmed the correspondence of four islands to the criteria of a true pathogeneicity island; in general, the genomes of two strains were found to be extremely similar [39]. A comparison of six strains representing five serotypes revealed a core genome accounting for $\approx 80\%$ of any single genome and containing most of the 'housekeeping' genes. It has been demonstrated that the serotype classification does not reflect the actual evolutionary relationships [115].

Two surface polysaccharides are produced by *S. agalactiae*: the group B carbohydrate, which is common to all strains and the serotype specific capsular polysaccharide. Genes encoding 30 surface and 71 secreted proteins have been predicted [34]. These proteins play an important role in the virulence and pathogenesis of the infection. This group includes adhesins, such as fibronectin-binding protein and laminin-binding protein; toxins (haemolysins, CAMP factor and exfoliative toxin A); extracellular digestive enzymes, such as hyaluronate lyase and C5a peptidase. All the serotypes, but especially III, are poorly antigenic [40].

The diversity of environments colonised by *S. agalactiae* is reflected in the high proportion of genes responsible for regulation and stress adaptation. Transcriptional regulators comprise 5% of the predicted genes [40]. Stress adaptation proteins include superoxide dismutase and a large set of heat shock proteins. Inability of *S. agalactiae* to synthesise many amino acids, vitamins and co-factors and its dependence upon host-derived nutrients is associated with the high development of transport systems: 255 genes encoding transporters have been predicted in serotype III, most of which are ABC transporters. A genome analysis has demonstrated the capacity to import a wide range of carbon resources, including glucose, fructose, lactose, mannose, cellobiose, trehalose, mannitol, β-glucoside and N-acetylgalactosamine [34]. The same study has detected enzymes necessary for glycolysis, whereas pentose phosphate pathway is only involved in pentose and gluconate utilisation, but does not serve to by-pass glycolysis. In contrast, the citric acid cycle is reported to be completely missing, leading to inability to synthesise most of the amino acids. Their acquisition is supported by eight ABC transporters, eight permeases and a large number of peptidases [34]. The human strains of *S. agalactiae* in general do not ferment lactose, but the ability to do so can be achieved after 8 to 10 passages in substrate containing lactose [46].

S. agalactiae oxidises glucose to the following fermentation products: lactate, pyruvate, acetate, formate, ethanol, acetoin and CO_2, in different proportions depending on the availability of oxygen [71]. Although Streptococcaceae are generally considered as non-respiring organisms, some evidence of oxidative phosphorylation in *S. agalactiae* has been reported [71]. This finding has been confirmed by the identification of genes encoding cytochrome *bd* terminal quinol oxidase [34]. It has been demonstrated that *S. agalactiae* is able to respire in the presence of two essential factors in the environment: quinone and haem [126].

S. agalactiae is a commensal constituent of microflora, but the systems developed for nutrient acquisition may play a key role in pathogenesis when the organism encounters normally sterile sites, such as neonatal lung and amniotic

fluid [40]. Therefore, investigation of metabolic networks may lead to deeper insights into virulence and aid the development of novel prophylactic and therapeutic means.

At present, Penicillin G is the main choice for treatment and prophylaxis [1, 8]. Routine screening and chemoprophylaxis in the last years led to a decline in the incidence of early onset infection [1, 107]. However, intrapartum antibiotic prophylaxis is an imperfect solution, the disease still occurs, and infection and death from *S. agalactiae* are still a major public health issue. Antibiotic treatment of newborns may lead to a range of side effects, including development of resistance in other organisms. Therefore, effective narrow-spectrum antibiotics are required (*Mark Anthony*, personal communication). Identification of potential drug targets would be a crucial step towards the achievement of this goal.

Chapter 2

Theoretical Foundations

2.1 Introduction

In the past two centuries, the advances of biological sciences enabled the discovery of the most basic building blocks of life, such as genes, enzymes and metabolic species. Investigation of these components in the fields of genetics, biochemistry and molecular biology and the advent of 'omics' techniques made it possible to collect immense amounts of data. Although a more complete understanding of the structures and functions of biomolecules requires further observation and analysis using more sophisticated experimental methods, the reverse problem arises: how do the interactions of biomolecules produce the behaviour of entire living systems, such as cells and organisms? Depending on the specific problems, these interactions can be considered in the context of gene regulatory, signal transduction, metabolic or other networks; however, each of these networks is only an aspect of an integral functional unit. The structure, behaviour and regulation of such a network cannot be adequately described using qualitative, verbal principles. Instead, a formalised theory is required to integrate, quantify and generalise the existing knowledge about biomolecular networks.

The need of such a theory led to the origin of Systems Biology – 'the study of an organism, viewed as an integrated and interacting network of genes, proteins and biochemical reactions which give rise to life' (Institute for Systems Biology[†]). In general, a system can be defined as a set of interacting elements, e.g. enzymes and reactants in metabolic systems. The interactions result in the emergent properties of the system that cannot be attributed to the isolated components. The emergent properties of living systems include metabolism, growth and reproduction. Further, any living organism has the ability to maintain a relatively constant chemical composition, often called homeostasis. Metabolic networks tend to a dynamic equilibrium (steady state), when the input and output metabolites are exchanged with the environment, and the concentrations of internal metabolites are balanced by consuming and producing reactions [24].

Another important systemic property of living systems is robustness – the ability to function in various conditions, due to the regulation and adaptation on the level of enzymatic activity, gene expression and even evolutionary changes [58, 54]. The robustness of systems may be also increased due to their modularity – the ability of subsystems to work independently [86], as well as redundancy – the presence of multiple subsystems with similar functions [78]. The subsystems and elements, on the other hand, have various degrees of essentiality – relative importance for the survival and functioning of the whole system [61]. Finally, any living system not only adapts to the environment but also influences its state; it changes the chemical composition of the surrounding media by consuming external substrates and releasing end products.

Mathematical models are widely used in systems biology and in particular, in the studies of metabolic networks. A metabolic model is a mathematical representation of a collection of molecular conversions (reactions) in a living organism; its elements are metabolites and reactions. The first published metabolic models usually covered separate pathways, such as glycolysis [32, 96], electron transport [59] and photosynthesis [85]. The sizes of models, however, tend to grow, exceeding several hundreds of reactions in genome-scale models, which describe the metabolic networks of whole organisms [104, 27].

Models are usually developed for certain pragmatic purposes and their usefulness is determined by the compromise between adequacy and simplicity [37]. Most real world systems are too complex to consider every detail, given the lack of data and computational limitations. Moreover, many factors may be ignored as not appreciably affecting the outcomes. Therefore, any mathematical model involves simplifying assumptions, such as the distinctions between internal and external metabolites and between reversible and irreversible reactions (see below).

Metabolic systems are characterised by two groups of data: variables (such as concentrations and fluxes) and parameters (such as stoichiometric coefficients and kinetic constants) [37]. Depending on the parameters employed, the metabolic models are usually subdivided into two major groups: kinetic and structural [84]. Both types of models include reaction stoichiometries, which define the time-invariant logical topology of the metabolic networks, i.e. the connections of their elements - metabolites and reactions. Kinetic models also include rate equations and kinetic parameters, which provide the possibility of simulating the dynamic evolution of a network from a particular initial state, by calculating the time-depenedent changes of the flux rates and metabolite concentrations. Unfortunately, the rate laws and kinetic parameters are mostly unavailable; further, the analysis of kinetic models on the genome scale is computationally hard and the results are difficult to interpret [83]. Therefore, in the scope of the current thesis, only structural models were generated and analysed. The analysis of such models is usually based on the assumption that the system is at a steady state.

The starting point of the modelling process is the initial hypothesis, which, in the case of a metabolic model, implies that the behaviour of the investigated system can be adequately described as a function of the input list of reac-

tions [83]. The methods of model construction are based on two complementary approaches, which we define as data-driven and hypothesis-driven. The data driven approach involves the selection, collection, curation and processing of input data, such as the list of reactions. These data can be obtained from literature, from experimental data or from databases, such as KEGG [49], BioCyc† [50] and BRENDA† [105]. The use of databases enables automatic generation of models, which is, however, associated with a range of technical issues [84]. The hypothesis-driven approach includes the formulation of additional hypotheses about the probable behaviour of the investigated system. Although their inclusion increases the amount of uncertainty in the model, it often leads to more precise and practically useful results. Simplifying assumptions are an example of this approach, but some hypotheses may increase the complexity of a model, e.g. hypothetical reactions and metabolites.

Once a model is generated, it may be interrogated. Depending on the objectives, the interrogation of metabolic models can be subdivided into two phases: constructive and analytic [83]. The purpose of constructive interrogation is improvement of the model quality; this can be done by using specific methods for the detection of errors [84, 33] and for the prediction of missing data [93]. The results are used for further correction, extension and refinement of the models; the process can be repeated iteratively, until a sufficient level of accuracy is achieved (Figure 2.1). Analytical interrogation, in contrast, aims at the prediction of relevant biological properties of the underlying networks, such as reaction pathways [110, 103], net conversions of external metabolites [118], nutritional requirements [36], essential enzymes [61], functional modules [29, 86] and phylogenetic relations to other networks [15]. Although these results can also give rise to changes in the models, the ultimate goal is to make a contribution to biological knowledge.

The published genome-scale metabolic reconstructions include those of *E.coli* [20], *H. influenzae* [104], *H. pylori* [102] and many others. The metabolic reconstruction of *S. cerevisiae* [27] was claimed to produce the first comprehensive network of a eucariotic organism. The metabolic reconstruction of *P. falciparum* has been used for the identification of novel antimalarial drug targets [127]. Some genome-scale models have been constructed recently in our group, including a model of *E. coli* [9], *S. erythrea* (H. Patel, M. G. Poolman, unpublished) and plant metabolism (A. Chokkathukalam, work in progress).

2.2 Structural modelling of metabolism

Structural modelling refers to the construction and analysis of metabolic models including solely information about reaction stoichiometries and (optionally) their allowed directions. Stoichiometry describes the proportions of changes in metabolite concentrations in a particular reaction. By defining the 'is interconverting' relation between the sets of reactions and metabolites, stoichiometries form the topological structure of a network, i.e. the logical connections between its elements; this network structure defines the limits of the possible behaviour

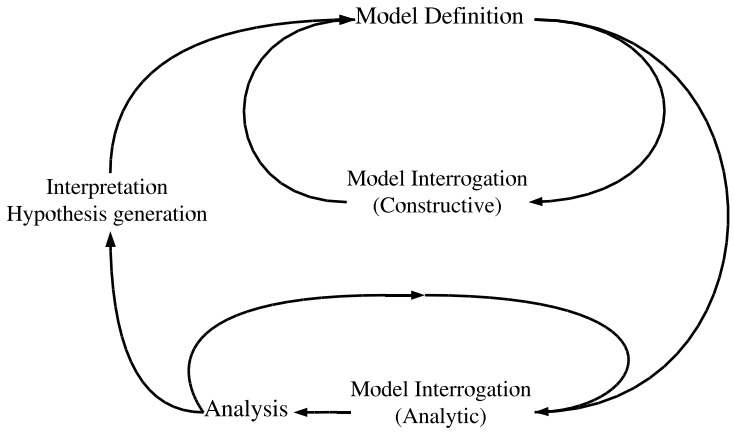


Figure 2.1: Typical work flow during a modelling investigation. Scientific value is only gained during the interpretation/hypothesis generation phase, which takes place outside of the computer. Picture by *Poolman* et al. [83].

of the system [55]. In contrast to kinetic properties, which are subject to dynamic changes due to activation and inhibition of enzymes, stoichiometries are invariant in time, unless evolutionary scales are considered [37]. Stoichiometries are often better known than kinetic parameters, they are expressed in discrete numbers and in most cases uniquely defined for each reaction in a model (unless variable polymerisation degrees are involved). These advantages make the use of structural models indispensable for the characterisation of genome-scale metabolic networks.

Metabolites Each metabolite in a model is a logical entity, referring to some chemical species. The level of abstraction can vary depending on the context, e.g. the same species can be defined as 'α-D-glucose', 'D-glucose' or 'glucose'. Some metabolites may refer to even broader groups of substances, such as 'protein' or 'primary alcohol'. However, for a range of technical reasons it is preferable to restrict the scope of metabolite definitions to possibly unique formulae [84, 33]. In conceptual models, created for theoretical or didactic purposes, the actual substances often do not need to be specified (see Figure 2.2).

The set of metabolites in a metabolic model is subdivided to two subsets: *internal* and *external* metabolites, depending on their relation to the model's boundary. Each internal metabolite is used solely by the reactions included into the model; hence, its concentration in the system is a function of the rates of these reactions. External metabolites, in contrast, are subject to mass flow

beyond the system. The following groups of substances are often defined as external:

- Nutrients and excretory products, e.g. glucose and lactate.
- Metabolites, which are present in a large excess or buffered, e.g. water, proton and carbon dioxide.
- Biomass components, such as protein, nucleic acids and cell wall.
- Other polymers, such as starch, where variable polymerisation degrees make it impossible to derive the concentrations from the flux rates.
- Highly connected metabolites (i.e. used by many reactions), such as ATP and NAD^+ can be also made external in order to limit the extent of the model.

The definition of the set of external metabolites may depend on the problem investigated. In particular, by changing the sets of sources and sinks, the system's behaviour in different environmental conditions can be analysed.

Reactions A reaction in a model indicates a conversion of some set of metabolites (substrates) into another set (products), taken in amounts specified by the reaction stoichiometry. In biochemical systems, most reactions are enzymatic; however, spontaneous reactions are also possible. Similarly to metabolites, reactions can be specified with various levels of detail, e.g. several enzymatic steps can be represented by one net reaction. Such generalised reactions may stand for complex cellular processes, such as protein biosynthesis from amino acids.

Transport reactions, instead of interconverting metabolites, change their location with regard to the boundary of the whole system or some of its subsystems (e.g. compartments). In structural models, such reactions are often represented as interconversions of distinct metabolites which refer to the same chemical species. For instance, the import of glucose can be represented as follows: x_glucose $\rightarrow$ glucose, where the substrate and the product denote the external and internal 'versions' of glucose, respectively. The reactions which consume or produce external metabolites are referred as *exchange reactions*.

The set of reactions can be subdivided into the subsets of *reversible* (bi-directional) and *irreversible* (uni-directional) reactions. This distinction is an example of a simplifying assumption, since there are no absolutely irreversible reactions in a real metabolism and the feasibility of a given direction depends on the rest of the system. However, thermodynamic gradients between externals may impose directionality on routes connecting them. Therefore, defining certain reactions as irreversible sometimes improves the accuracy of predictions. In particular, generalised biosynthetic reactions may be defined as irreversible, to reflect the growth of the organism. For each irreversible reaction, the proper direction must be specified.

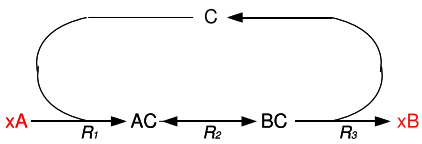
Mathematical representation A range of formalisms exist for the representation of metabolic models. Diagrams (Figure 2.2a) are widely used for illustration, whereas lists of reactions (Figure 2.2b) are convenient for data input and storage. For most of the quantitative analysis methods, a metabolic model is represented as a *stoichiometry matrix* [37] (Figure 2.2c, d). In such a matrix, each element defines the number of molecules of the metabolite, specified by the row, used in the reaction, specified by the column, whereby the sign indicates the direction of mass change (negative for consumption and positive for production).

The complete structure of a network is represented by its *external stoichiometry matrix* $\hat{\mathbf{N}}$ (Figure 2.2c). This matrix includes the rows describing both internal and external metabolites and hence represents a *closed* system [23]. By removing the rows describing external metabolites, we obtain the *internal stoichiometry matrix* $\mathbf{N}$, where the stoichiometries of exchange reactions are incomplete (Figure 2.2d). This matrix represents an *open* system, which exchanges mass with the 'environment', defined by the pool of external metabolites.

A structural model is completely defined by its external stoichiometry matrix and two sets: those of external metabolites and reversible reactions. Further in the thesis, we denote by m and n the numbers of metabolites and reactions in a model, respectively. Without loss of generality, we assume that the rows and columns in stoichiometry matrices are reordered (unless otherwise stated) so that the first k rows and the first r columns describe the internal metabolites and reversible reactions, respectively. Hence, a network can be represented as a triple $(\hat{\mathbf{N}}, k, r)$. The sets of reactions and metabolites are denoted $\mathcal{R}$ and $\mathcal{M}$, respectively.

We refer to a vector $\mathbf{d}$ as *positive* (denoted $\mathbf{d} > \mathbf{0}$) if it contains positive components only, *semipositive* ($\mathbf{d} \geq \mathbf{0}$) if it contains at least one positive and no negative components and *seminegative* ($\mathbf{d} \leq \mathbf{0}$) if it contains at least one negative and no positive components.

A number of physiologically important properties of metabolic systems can be determined by interrogation of external and internal stoichiometry matrices. The analysis methods described in this section are based on two mathematical fields: linear algebra and graph theory. Linear algebraic methods of the analysis of metabolic models are referred as *stoichiometric analysis* and are classified here into *mass-balance* and *flux-balance* analysis methods, depending on the fundamental constraints they apply to the networks. Mass-balance analysis describes relationships between metabolite concentrations and compositions subject to the mass conservation constraint; these relationships characterise the network at any point in time and do not depend on the feasibility of reaction directions. Flux-balance analysis concerns possible distributions of flux rates and concentration changes at a steady state; irreversibility constraints can be taken into account.



a

$$\begin{array}{ll} R_1: & \text{xA} + \text{C} \rightarrow \text{AC} \\ R_2: & \text{AC} \leftrightarrow \text{BC} \\ R_3: & \text{BC} \rightarrow \text{C} + \text{xB} \end{array}$$

b

$$\begin{array}{c} \\ \text{C} \\ \text{AC} \\ \text{BC} \\ \text{xA} \\ \text{xB} \end{array} \begin{array}{c} \begin{array}{ccc} R_1 & R_2 & R_3 \end{array} \\ \begin{pmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

c

$$\begin{array}{c} \\ \text{C} \\ \text{AC} \\ \text{BC} \end{array} \begin{array}{c} \begin{array}{ccc} R_1 & R_2 & R_3 \end{array} \\ \begin{pmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix} \end{array}$$

d

$$\begin{array}{c} \\ \text{C} \\ \text{AC} \\ \text{BC} \\ \text{xA} \\ \text{xB} \end{array} \begin{pmatrix} -1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

e

$$\begin{array}{c} \\ \text{C} \\ \text{AC} \\ \text{BC} \\ \text{xA} \\ \text{xB} \end{array} \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

f

Figure 2.2: A network represented as a diagram (a), a list of reactions (b), external (c) and internal (d) stoichiometry matrices; left null space matrix (e); matrix of extreme non-negative conservation relationships(f).

2.2.1 Mass-balance analysis

Although the actual metabolite concentrations depend on the initial values and flux rates, some important restrictions are implied by reaction stoichiometries and the law of mass conservation. Consider the network shown in Figure 2.2. While the network interconverts the external metabolites xA and xB, the internal metabolite C acts as a cofactor, by catalysing this conversion. Its atoms are not exchanged with the environment and comprise a *conserved moiety* – a chemical entity participating in the system without loss of integrity [37]. The total concentration of a conserved moiety is constant in the system and so is the total concentration of all metabolites containing it:

$$C + AC + BC = const. \tag{2.1}$$

These concentrations enter a *conservation relationship*: a linear combination of concentrations which is constant in time. In terms of linear algebra, this relationship can be expressed as follows:

$$(1\ 1\ 1\ 0\ 0) \cdot \begin{pmatrix} C \\ AC \\ BC \\ xA \\ xB \end{pmatrix} = const. \tag{2.2}$$

or, for a general case:

$$\mathbf{g}^T \hat{\mathbf{c}} = const. \tag{2.3}$$

where $\hat{\mathbf{c}}$ is the vector of concentrations and $\mathbf{g}$ is the conservation vector, whose non-zero components are the conservation coefficients of the metabolites contributing to the relationship. A typical example of a conservation relationship is the ATP, ADP and AMP pool, which preserves the adenylate moiety. The conservation coefficients are not necessarily equal to unity; e.g. in the phosphate-conserving relationship, ATP and ADP occur with the coefficients 3 and 2, respectively. In these examples, the conservation coefficients indicate how many times the conserved moiety occurs in the molecule of the given metabolite.

How can a conservation relationship be found in the stoichiometry matrix? Let us return to the network shown in Figure 2.2. Since the concentration of the conserved moiety C in each reaction remains unchanged; the sum of conservation coefficients weighted by the stoichiometric coefficients is equal to zero; e.g. in R_1 we have: $(-1 * 0) + (-1 * 1) + (1 * 1) = 0$. To generalise, we write:

$$\sum_{i=1}^{m} g_i \hat{\mathbf{N}}_{ij} = \mathbf{0} : \ 1 \leq j \leq n \tag{2.4}$$

This equation can be written in a simpler manner:

$$\hat{\mathbf{N}}^T \mathbf{g} = \mathbf{0} \tag{2.5}$$

Any solution $\mathbf{g}$ of this equation is a conservation vector and vice versa. The solution space (often referred as the *left null space* [23]) is spanned by a non-unique set of $m - rank(\hat{\mathbf{N}})$ linearly independent conservation vectors, which can be arranged into a matrix $\mathbf{G}$:

$$\mathbf{G}\hat{\mathbf{N}} = \mathbf{0} \tag{2.6}$$

An example is shown in Figure 2.2e. The first column in this matrix contains a negative coefficient for C and does not correspond to any conserved moiety.

In general, only semipositive conservation relationships represent the conservation of chemical units [37]. The set of semipositive solutions of Equation 2.5 is a pointed polyhedral convex cone [97]. Each vector in this set can be represented as a non-negative linear combination of a finite number of generating vectors, which are unique up to positive multiples. These vectors correspond to the *extreme non-negative conservation relationships* [37]. An algorithm for the detection of the complete set of generating vectors has been proposed by Schuster and Höfer [111]. The generating vectors for our example are the columns of the matrix shown in Figure 2.2f. Here, the first and second columns correspond to the conservation of the atoms exchanged with environment and of those remaining within the systems boundary, respectively. Only the second relationship persists in the open system and can be found in the internal stoichiometry matrix.

The most encompassing conservation relationship in a network represents the total conservation of mass [111]:

$$\hat{\mathbf{N}}^T\mathbf{m} = \mathbf{0} \tag{2.7}$$

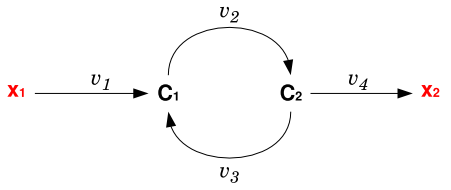
The components of the conservation vector $\mathbf{m}$ can be considered as feasible molecular masses of the metabolites; in a correctly defined network, all components must be positive. The *unconserved* metabolites (those not involved in any semipositive conservation relationships) indicate modelling errors [73]. This issue is elaborated in Section 5.2 and in a recent paper of our group [33].

2.2.2 Flux-balance analysis

To reveal the essential features of metabolic systems, one often restricts the analysis to asymptotic time behaviour (i.e. after a sufficiently long time span). Although this behaviour may be oscillatory or chaotic, many systems reach a steady state. The concept of a steady state is a mathematical idealisation, which implies that concentrations and fluxes inside the system do not change within a tolerable accuracy over a certain time span [37]. The assumption that a system is at a steady state plays an important role in metabolic modelling, since it enables a relatively simple approach to the analysis of fluxes based on linear constraints.

In general, a concentration changes in time as a function of flux rates. E.g. the rate of change of concentration of c_1 in Figure 2.3 can be calculated as follows:

$$\frac{dc_1}{dt} = v_1 - v_2 + v_3$$



a

$$\begin{array}{c} \\ c_1 \\ c_2 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 1 & -1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{pmatrix} \end{array}$$

b

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cc} & \end{array} \\ \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$$

c

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cc} & \end{array} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \end{array}$$

d

	v_1	v_2	v_3	v_4
v_1	1.0	0.4	0.4	1.0
v_2		1.0	0.67	0.4
v_3			1.0	0.4
v_4				1.0

e

Figure 2.3: A network represented as a diagram (a) and internal stoichiometry matrix (b); two suitable null space matrices (c, d); table of reaction correlation coefficients (e).

This can be written in the form of vectors:

$$\frac{dc_1}{dt} = (1 \ -1 \ 1 \ 0) \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}$$

where the left multiplier is the corresponding row in the stoichiometry matrix. This equation can be summarised for all concentrations:

$$\frac{d\mathbf{c}}{dt} = \mathbf{N}\mathbf{v} \tag{2.8}$$

where $\mathbf{v}$ is the vector of flux rates and $\mathbf{c}$ is the vector of concentrations of internal metabolites. At a steady state, these concentrations are constant:

$$\mathbf{N}\mathbf{v} = \mathbf{0} \tag{2.9}$$

Assuming that all reactions are reversible, any solution $\mathbf{v}$ of this system is a possible distribution of flux rates at a steady state.

Null space analysis The solution space of Equation 2.9 is spanned by its null space basis of dimension $n - rank(\mathbf{N})$, which can be represented as a non-unique matrix $\mathbf{K}$:

$$\mathbf{N}\mathbf{K} = \mathbf{0} \tag{2.10}$$

The columns of $\mathbf{K}$ are linearly independent solutions of Equation 2.9 and any steady state flux vector can be written as a linear combination of these columns.

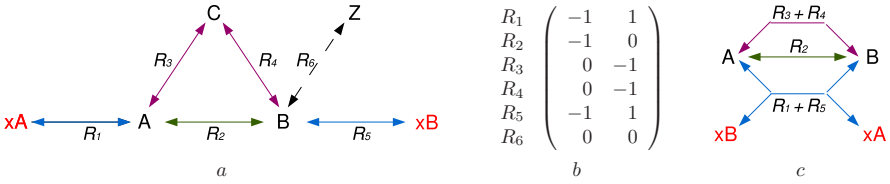


Figure 2.4: a) Reaction subsets in a network. The colouring of arrows indicate the inclusion of reactions into different subsets; the dead reaction is represented by a dashed arrow. b) Null space matrix: the subsets $\{R_1, R_5\}$ and $\{R_3, R_4\}$ are represented by proportional (equal) rows; the dead reaction R_6 is represented by a zero row. c) The 'condensed' version of the same network, where each subset is replaced by a net reaction. The dead reaction R_6 is removed. R_2 and $R_3 + R_4$ have the same stoichiometry, so one of them should be also removed before analysis.

An example is shown in Figure 2.3c; the first column of this matrix represents an internal cycle of length two and the second column is a 3-reaction pathway converting x_1 into x_2.

The analysis of the null space reveals some important relationships between steady-state flux vectors. Proportional rows in a null space matrix indicate *reaction subsets* (also known as enzyme subsets [79]): groups of reactions whose flux rates always remain in a fixed non-zero ratio at a steady state. E.g. in Figure 2.3, the fluxes $\mathbf{v}_1$ and $\mathbf{v}_4$ always operate simultaneously and the corresponding rows in the right null space matrix are identical. Another example is shown in Figure 2.4a. In this network, three subsets are present: $\{R_1, R_5\}, \{R_3, R_4\}$ and the singleton subset $\{R_3\}$. The flux rate of R_6 is zero at any steady state; such reactions are often called *dead* or strictly detailed balanced reactions [113]. In reversible networks, dead reactions are always represented by zero rows in the null space matrix.

Since the reactions in a subset operate simultaneously at any steady state, they are essential for each other, i.e. blocking one of them leads to a block in the others. A number of studies indicate co-expression of enzymes catalysing reactions involved in the same subsets [94, 112, 9].

The calculation of reaction subsets provides a possibility of simplifying a model by replacing the reactions of each subset with one net reaction; dead reactions can be removed, since they do not contribute anything to the steady state fluxes (see Figure 2.4b).

While reaction subsets represent a boolean relation on the set of reactions (two reactions are either in a subset or not), more subtle relationships may be detected by the analysis of an orthogonal null space matrix, in which all columns are perpendicular to each other [86]. The cosines of the angles between the rows of this matrix are defined as *reaction correlation coefficients* and are equal to Pearson's correlation coefficients between the sets of fluxes carried by

the corresponding reactions in all steady states. Reaction correlation coefficients are real numbers in the interval (-1, 1), which reveal the level of 'cooperation' between reactions and can be used for hierarchical modular decomposition of metabolic networks. In particular, reactions in disconnected subnetworks have a correlation of zero and those in a subset have a correlation of 1 or -1 (see Figure 2.3e).

Null space analysis has two important limitations: firstly, the null space matrix is not uniquely defined for a given network; secondly, it ignores the irreversibility constraint. An example is shown in Figure 2.3d: in this matrix, the second column represents a pathway, which involves the irreversible flux v_3 in the backward direction. One can overcome these limitations by using the methods of convex analysis and linear programming, whose applications to metabolic modelling are reviewed in the following two sections.

Pathway analysis If a network contains irreversible reactions, the flux vector can be decomposed into subvectors $\mathbf{v}^{rev}$ and $\mathbf{v}^{irr}$, where $\mathbf{v}^{rev} \in \mathbb{R}^r$. Provided that the directions of irreversible reactions are appropriately defined, their flux rates are always positive:

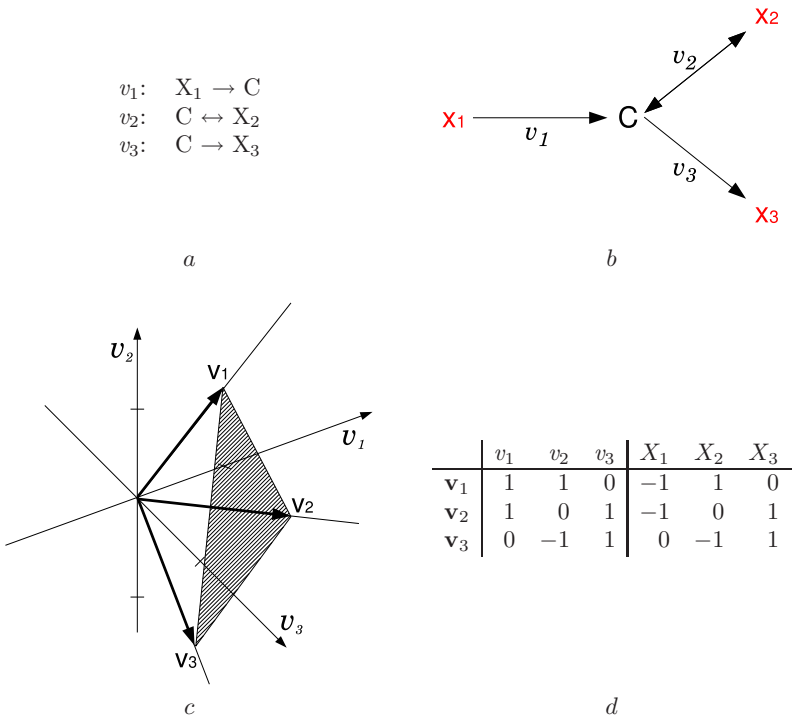
$$\mathbf{v}^{irr} \geq \mathbf{0} \tag{2.11}$$

The set of flux vectors satisfying Equations 2.9 and 2.11 is a polyhedral convex cone, referred as the *flux cone* [37]. This cone is located in a high-dimensional space, where each dimension corresponds to a reaction flux. If all reactions are reversible, the flux cone occupies the whole null space of the system; if all are irreversible, the flux cone is the intersection of the null space and the non-negative orthant.

Consider the network shown in Figure 2.5. Each feasible flux distribution in this network is a point inside the trihedral flux cone, located in a three-dimensional reaction space. The definition of a cone implies that it contains all non-negative multiples of any of its points; hence, multiplication of a feasible flux vector by any positive real number gives another feasible flux vector. The set of positive multiples of some non-zero vector in a flux cone is called a *flux mode*. Geometrically, each flux mode is a half-line inside the flux cone, commencing from the origin (but not including it). Hence, it can be sufficiently characterised by any of its points and is often denoted as a single vector (e.g. $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ in Figure 2.5). A flux mode is reversible if the opposite half-line is also within the flux cone; this implies that all reactions entering it with non-zero coeffiecients are reversible.

Biochemically, each flux mode is a feasible combination of flux rates at a steady state, defined uniquely up to positive scaling. It can be also considered as a reaction pathway in a network, starting and ending at external metabolites. The set of reactions participating in a flux mode is represented by its support, defined as the set of the indices of non-zero components [97]:

$$P(\mathbf{v}) = \{i : v_i \neq 0\} \tag{2.12}$$



	v_1	v_2	v_3	X_1	X_2	X_3
$\mathbf{v}_1$	1	1	0	−1	1	0
$\mathbf{v}_2$	1	0	1	−1	0	1
$\mathbf{v}_3$	0	−1	1	0	−1	1

Figure 2.5: a, b) A network of three reactions, where X_1, X_2 and X_3 are external. c) Three-dimensional flux space: the axes correspond to reaction fluxes; the marks indicate unit values. $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ are feasible flux vectors representing the elementary flux modes, which are shown as half-lines. These half-lines are the generating vectors (edges) of the trihedral flux cone; its cross-section is hatched. d) Table of elementary flux modes (left part) with the corresponding net conversions (right part).

Since the flux cone is convex, any non-negative linear combination of any pair of feasible flux vectors is also feasible; hence, the number of flux modes in a general case is infinite. However, some modes can be represented as superpositions of simpler ones; e.g. the flux mode $(-2 \quad 1 \quad 1)^T$ in Figure 2.5 is simply a sum of $\mathbf{v}_1$ and $\mathbf{v}_2$. Moreover, these two modes taken in any positive proportions would produce different flux modes representing the same pathway (defined as a set of reactions). Therefore, pathway analysis of metabolic networks often aims to detect possibly simple, irreducible pathways, which can operate independently from the rest of the system. Two closely related concepts addressing this problem are widely accepted: those of elementary flux modes [37] and extreme pathways [103].

A flux mode $\mathbf{v}$ is *elementary*, if it cannot be represented as a non-negative linear combination of any other two flux modes $\mathbf{v}'$ and $\mathbf{v}''$, such that $P(\mathbf{v}') \subset P(\mathbf{v})$ and $P(\mathbf{v}'') \subset P(\mathbf{v})$. In other words, an elementary flux mode cannot be decomposed as a sum of other flux modes without cancellation occuring in at least one component. The set of elementary modes is uniquely defined for a given network and has some important properties [110]:

- An elementary flux mode is uniquely defined by its support (hence, the proportions of flux rates are fixed for the set of participating reactions).
- A flux mode is elementary iff it is minimal, i.e. its support does not contain the support of any other flux mode as a proper subset.
- Any flux vector is a positive linear combination of vectors representing some elementary flux modes.

From the first property it follows that an elementary mode can be sufficiently represented by a set of reactions; the second property implies that this set is irreducible. Hence, each elementary mode is a minimal functional unit of a metabolic network, able to operate independently at a steady state; if only the reactions participating in the elementary mode are active, removing one of them would cause a cessation of steady state fluxes in the rest. According to the third property, the set of elementary modes characterises all possible steady state flux distributions. This set can be calculated using the algorithms given by Schuster and Hilgetag [109] and by Urbanczik and Wagner [90].

A feasible flux vector represents an *extreme pathway* if it cannot be decomposed as a non-negative linear combination of any two feasible flux vectors. The definition of an extreme pathway is stricter than the definition of an elementary mode, since it does not imply that components may not be cancelled out during addition. Hence, the set of extreme pathways is a subset of the set of elementary modes and is also uniquely defined for a given network. In fact, it is the minimal set of flux modes which is sufficient to represent all possible steady state flux distributions as superpositions of its elements. The calculation of extreme pathways requires a reconfiguration of the network, so that each reversible reaction is replaced by a pair of mutually opposite irreversible ones and each metabolite is used by at most one exchange reaction.

Geometrically, extreme pathways are the generating vectors of the flux cone. This is also true for elementary modes, if all exchange fluxes in the network are irreversible [57]. An example is shown in Figure 2.5c, where the sets of elementary modes and extreme pathways coincide. Figure 2.6 shows a situation, where these sets are different. In this network, some elementary modes can be decomposed as positive linear combinations of others:

$$\begin{aligned}\mathbf{v}_3 &= \mathbf{v}_2 + (-\mathbf{v}_1)\\ \mathbf{v}_5 &= \mathbf{v}_4 + \mathbf{v}_1\\ \mathbf{v}_7 &= \mathbf{v}_6 + (-\mathbf{v}_1)\end{aligned}$$

(note that $-\mathbf{v}_1$ is an elementary mode). Since some components are cancelled out in each case, these decompositions do not violate the definition of an elementary mode. However, they do violate the definition of an extreme pathway.

Each flux vector is characterised by its net conversion (net stoichiometry), which is defined by the concentration changes occuring as the result of the flux. The net conversion can be determined by means of the external stoichiometry matrix:

$$\dot{\mathbf{c}} = \hat{\mathbf{N}}\mathbf{v} \tag{2.13}$$

where $\dot{\mathbf{c}}$ is the vector of time derivatives of concentrations. Clearly, if $\mathbf{v}$ is feasible at a steady state, all non-zero components of $\dot{\mathbf{c}}$ correspond to external metabolites[1]. Examples are shown in Figures 2.5 and 2.6. Biochemically, each net conversion describes changes in the composition of the environment of a system which take place when a given pathway is operated. In particular, if the pathway is an internal cycle, its net conversion is equal to zero (e.g. $\mathbf{v}_1$ in Figure 2.6). The set of all possible net conversions of a network at a steady state is a polyhedral convex cone, referred as *conversion cone* [118]. This topic is elaborated in Section 7.

Elementary modes and extreme pathways are essential structural invariants of metabolic networks. Their number indicates the functional richness and flexibility of biochemical systems. The number of independent pathways interconverting the same external metabolites is a measure of pathway redundancy, which characterises the robustness of the systems [78]. However, using the two approaches may lead to different results [57, 77]; e.g. in Figure 2.6 each non-zero net conversion is performed by two elementary modes, but only one extreme pathway.

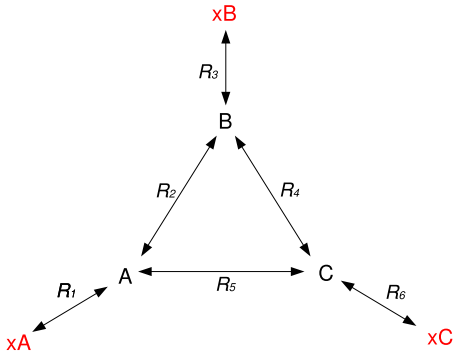
Pathway analysis provides a possibility of predicting essential enzymes and genes, whose deletion prevents particular metabolic functions [108]. The calculation of elementary modes enables the detection of *minimal cut sets* - irreducible sets of reactions in a network whose inactivation results in the failure in a certain function of the system [55]. The potential applications of this concept include drug target identification.

The calculation of elementary modes and extreme pathways is an NP-hard computational problem, which often leads to a combinatorial explosion in large

[1]Exceptions from this rule are possible in incorrectly defined networks, see Section 5.2

R_1: xA $\leftrightarrow$ A
R_2: A $\leftrightarrow$ B
R_3: xB $\leftrightarrow$ B
R_4: B $\leftrightarrow$ C
R_5: A $\leftrightarrow$ C
R_6: xC $\leftrightarrow$ C

a



b

	R_1	R_2	R_3	R_4	R_5	R_6	xA	xB	xC
$\mathbf{v}_1$	0	1	0	1	−1	0	0	0	0
$\mathbf{v}_2$	1	1	−1	0	0	0	−1	1	0
$\mathbf{v}_3$	1	0	−1	−1	1	0	−1	1	0
$\mathbf{v}_4$	1	0	0	0	1	−1	−1	0	1
$\mathbf{v}_5$	1	1	0	1	0	−1	−1	0	1
$\mathbf{v}_6$	0	0	1	1	0	−1	0	−1	1
$\mathbf{v}_7$	0	−1	1	0	1	−1	0	−1	1

c

Figure 2.6: a, b) A network of six reactions. c) Table of elementary flux modes (left part) and the corresponding net conversions (right part); only one direction of each mode is shown. $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_4$ and $\mathbf{v}_6$ are the extreme pathways.

models. The efficiency of the algorithms can be improved by applying them to 'condensed' models (see Figure 2.4b); however, the calculation of a complete set of elementary modes in genome-scale models remains a serious computational challenge. An alternative approach to pathway analysis is described in the next section.

Linear programming methods Despite the enormous number of possible flux distributions in metabolic systems (the number of elementary modes in genome-scale models may exceed a million), in many cases only a few of them are physiologically meaningful and represent a practical interest. Such optimal states of the system can be predicted using the mathematical technique called linear programming.

A function f of variables $x_1, ..., x_n$ is called a *linear form* if it can be written as $c_1x_1 + ... + c_nx_n$ (or $\mathbf{cx}$ in a vector form), where the coefficients c_i are constant real numbers. f is called an *affine function* if it is the sum of a linear form and a constant: $f(\mathbf{x}) = \mathbf{cx} + d$. A *linear* constraint is a linear equality or inequality, e.g. $\mathbf{a}^T\mathbf{x} = 0$ or $\mathbf{a}^T\mathbf{x} \leq 0$. A set of linear constraints can be written in a matrix form, e.g. $\mathbf{Ax} = \mathbf{b}$ or $\mathbf{Ax} \leq \mathbf{b}$, where each row of the matrix $\mathbf{A}$ corresponds to one linear constraint.

A *linear program* (LP) is an optimisation problem where an affine function is required to be optimised (maximised or minimised) subject to a finite set of linear constraints [119]. A linear program can be written in the canonical form:

$$\begin{array}{rl} \text{Minimise} & \mathbf{cx} + d \\ \text{Subject to} & \mathbf{Ax} \leq \mathbf{b} \\ \text{Where} & \mathbf{x} \geq \mathbf{0} \end{array} \tag{2.14}$$

where $\mathbf{x}$ represents the vector of variables to be determined, while the vectors $\mathbf{c}$ and $\mathbf{b}$, the real number d and the matrix of coefficients $\mathbf{A}$ are known (note that an equality $\mathbf{a}^T\mathbf{x} = 0$ can be written as a pair of inequalities $\mathbf{a}^T\mathbf{x} \leq 0,\ \mathbf{a}^T\mathbf{x} \geq 0$). The affine function to be minimised is called *objective function*. The set of solutions satisfying the system of linear constraints is a convex polyhedron, termed the *feasible region*. Hence, the aim of linear programming is to optimise the objective function within the feasible region. The methods developed for this purpose include the well known simplex algorithm, introduced by G. Dantzig in 1947. These methods are implemented by a number of commercial and open-source LP solvers, such as GLPK[†] and Lindo[†].

In metabolic modelling, the linear constraints circumscribe the limits of the possible systems behaviour, within which any actual metabolic phenotype of the cell must lie [21]. In the previous sections, we introduced three important sets of linear constraints which govern metabolic systems: mass conservation (Equations 2.5 and 2.7), flux balance at a steady state (Equation 2.9) and positivity of fluxes in irreversible reactions (Equation 2.11). A special case of a feasible region is the flux cone, defined by Equations 2.9 and 2.11. The region of feasible flux vectors can be reduced by applying additional constraints, e.g. upper bounds of the flux rates in some reactions, based on a knowledge of

enzyme capacities [89]. For instance, in Figure 2.5, the maximal rates in v_1, v_2 and v_3 could be set to 1:

$$\begin{array}{rl} \text{Subject to} & \mathbf{Nv} = \mathbf{0} \\ \text{Where} & 0 \leq v_i \leq 1,\ 1 \leq i \leq n \end{array} \tag{2.15}$$

This would limit the feasible region to the flux cone closed by its cross-section, which is shown as the hatched triangle. In addition, the rates in some reactions can be fixed to constant values. In particular, setting a flux rate to zero implies the removal of a given flux and thereby can simulate an enzyme or gene deletion. The objective function may involve a single variable (e.g. maximising the flux in a given reaction) or a group of variables (e.g. minimising the total flux in a pathway). One of the typical objectives is the optimisation of growth rates, measured as the flux in biomass synthesis reactions [19, 43].

One of the potential issues associated with linear programming is that only a single solution can be calculated, whereas multiple solutions with equal values of the objective function may exist. A set of alternative solutions can be calculated using the methods of mixed-integer linear programming (MILP), which assume that some variables may be assigned only integer values [80]. However, the computational complexity of MILP problems is higher than that of LP problems.

2.2.3 Graph-theoretical methods

Graph theory is the study of graphs: mathematical structures used to model pairwise relations between objects from a certain collection. A *graph* is defined as an ordered pair (V, E), where V is a non-empty set of *vertices* and E is a set of pairs of vertices, called *edges*. If the edges are ordered pairs, then the graph is *directed* (digraph). The definition of a graph may include *weights*: numerical values assigned to edges.

Vertices usually represent objects and edges represent relations or connections between them. An edge connects two vertices, which are said to be *incident* to it and *adjacent* to each other. The *degree* of a vertex is the number of edges incident to it. A *path* is a sequence of vertices and edges, where consecutive elements are incident; the path length is defined as the number of edges. Two vertices are *connected* if there exists a path between them. A graph is connected if all of its vertices are connected and disconnected otherwise; the *connected components* of a graph are its connected subgraphs.

A number of approaches exist to graph representation of metabolic networks. In a *substrate graph*, the metabolites are represented as vertices and connected by edges, if they occur in the same reaction. In a *reaction graph*, the vertices represent reactions, which are connected if they use at least one common metabolite [122]. In a *bipartite* metabolic graph, both reactions and metabolites are represented as disjoint sets of vertices, where the edges connect the reactions to the metabolites they interconvert. A bipartite graph completely preserves the information present in a stoichiometry matrix, including the stoichiometric coefficients, which can be encoded as edge weights. Graphs are often depicted as diagrams, where vertices and edges are represented by points (circles) and lines

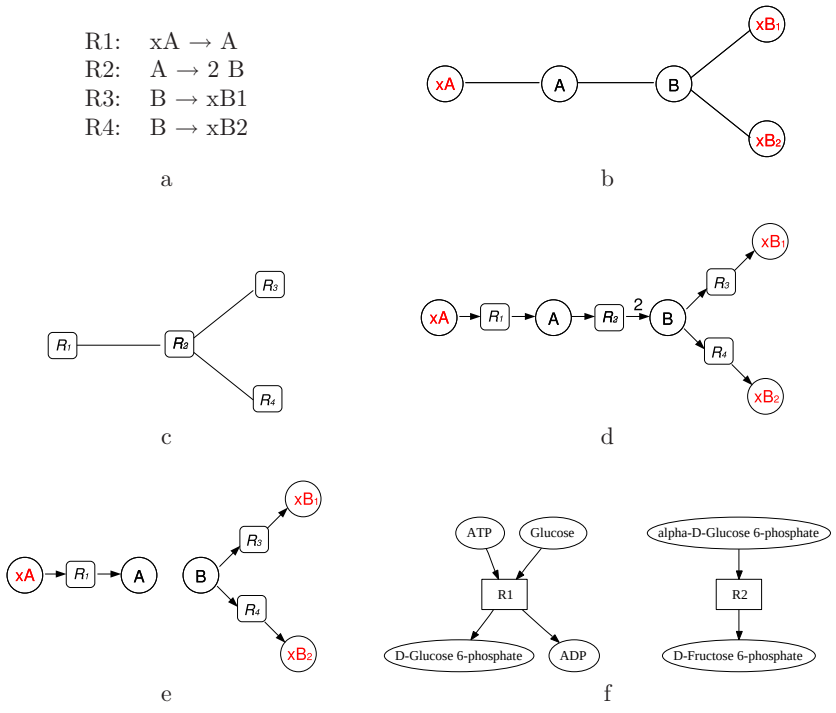


Figure 2.7: The same network (a) represented as a substrate (b), reaction (c) and bipartite (d) graph. In the bipartite graph, the unit edge weights are not shown. Deleting R_2 results in a disconnected network (e). Inconsistent naming of the same metabolite (D-glucose 6-phosphate/α-D-gucose 6-phosphate) gives rise to another disconnected network (f).

between adjacent vertices, respectively; the edge directions in a digraph can be indicated by arrows (Figure 2.7).

Wagner and Fell, 2001 analysed the metabolic graph of *E. coli* and concluded that it shared the characteristics of a small world network, where any node can be achieved from any other by a relatively small number of steps [122]. Arita, 2004 proposed to connect metabolites in a graph only if carbon atoms are transferred between them [2]. The Petri net approach, based on the use of directed bipartite graphs, may be used to detect some important system properties and structural patterns [121, 128]. Efforts have been made to develop methods of hierarchical decomposition of metabolic graphs [29] and to detect relatively independent functional modules [92, 66]. Damage analysis [61] is a graph-theoretical method investigating the influence of the knock-out of individual enzymes on the function of the network, by counting the number of blocked reactions. The network expansion method [18] analyses network topology to detect metabolites that can be produced from a given set of substrates. A

modification of this method predicts nutritional requirements of organisms [36].

Graph-theoretical analysis can be used for the detection of modelling errors [84]. Complete metabolic networks of organisms are usually assumed to be connected. This assumption is obvious if an organism is able to grow on a simple medium, containing single carbon, nitrogen and phosphate sources. The situation is less clear if an organism requires essential amino acids and cofactors. However, even then it may be assumed that all metabolites participate in the paths leading to protein, nucleic acids and other biomass constituents. Hence, a disconnected network can be considered as an error. The typical causes of disconnectedness include missing reactions (Figure 2.7e) and inconsistent naming of metabolites (Figure 2.7f). In terms of graph theory, disconnected subnetworks can be identified as multiple connected components, e.g. {xA, R_1, A} and {B, R_3, xB_1, R_4, xB_2} in Figure 2.7e).

Orphan metabolites are the internal metabolites with a degree of one (i.e. used by only one reaction), e.g. Z in Figure 2.4. Since the concentrations of such metabolites are not balanced and the reactions involving them are dead, their presence also indicates modelling errors. *Dead-end* metabolites are those only consumed or only produced; the concentration of such a metabolite can be balanced only if it is not an orphan and at least one reaction using it is reversible.

The 'Core' algorithm (Poolman, unpublished) removes the orphan metabolites, extracting the stoichiometrically flux-balanceable, 'core' part of a metabolic network. The algorithm is iterative: in each iteration, those reactions involving orphans are removed from a network (thus new metabolites may become orphans); the algorithm terminates, when the network contains no more orphans.

2.3 Construction of genome-scale models

The recent success of genome sequencing projects and the development of functional annotation tools provide an opportunity to considerably enhance the knowledge of physiology and biochemistry of organisms, by reconstructing their metabolic networks computationally from genome data. Such reconstructions are called 'genome-scale', since all reactions that can be identified in a genome annotation are included in the model. The possible steps of the construction of a genome-scale model are implied by the standard pathway of information flow [84]:

$$DNA \rightarrow RNA \rightarrow enzymes \rightarrow reactions \rightarrow metabolites \tag{2.16}$$

These steps may include: functional annotation of the genome; selection of the set of relevant enzymes; its translation into a set of catalysed reactions and further into a set of interconverted metabolites; definition of the feasible reaction directions and of external metabolites.

2.3.1 Genome annotation

Many sequenced genomes are available in public databases, such as NCBI†, KEGG and TIGR†. Functional annotation refers to the identification of open reading frames in a sequenced genome and to the assignment of functions to them. The most common approach to gene function prediction is based on searching for sequence homologies between the putative genes and molecules of known functions in other organisms. It is anticipated that orthologous genes are likely to carry similar functions in most cases, although it is known that the enzyme specificity may be changed by a single change in the sequence. Further, the functions of enzymes used as references may be themselves assigned as a result of a homology search, thereby increasing the probability of missannotations [25]. On the other hand, the description of the complete functionality of a protein is impossible beyond the context of the metabolic network [28]. A range of computational tools have been developed to aid the automated annotations, such as metaSHARK [81], BioMiner [65] and Pathway Tools [51]. Many annotations can be obtained together with the genomes from public databases, such as KEGG. The enzymatic functions of proteins are represented by EC (Enzyme Commission†) numbers [4].

PRIAM [12] is a database of automatically generated position-specic scoring matrices (PSSM, also called profiles) associated with entries in the ENZYME† database [4]. Each profile in PRIAM is generated by applying a PSI-BLAST search to a set of enzymatic sequences sharing an EC number. Hence, it can be considered as a descriptor of the given EC number; some EC numbers are described by multiple profiles. The PRIAM July 2006 release (corresponding to ENZYME release 39) was used in the current work. Unfortunately, the database does not cover most of the currently known EC numbers (see Table 4.6e).

RPS-BLAST [67] uses a BLAST-like algorithm to search a query sequence against a database of profiles (in contrast to PSI-BLAST, which searches a profile against a database of sequences). The database must be provided in a separate file; in our work, the PRIAM database was used. As the input, the program uses a nucleotide sequence file in the FASTA format, which may contain multiple sequences (e.g. predicted ORFs in a genome). For each sequence, the program reports the profiles (hits), whose alignment against the query sequence produces an E-value (expectation value: the number of matches as good as the observed one that would be expected to appear by chance [62]) lower than a certain threshold value, which is supplied as an additional parameter. The result is sent to the text output, which represents a sequence of records associated with ORFs.

An example of RPS-BLAST output record is shown in Figure 2.8a. It consists of two parts: the upper part contains meta-information, while the lower part (starting from the word 'Fields') represents a table of predictions. The table contains three rows, each of them corresponding to a single hit. The first entry is identical in all rows and shows the query ID (in this case, the gene identifier sag:SAG0030). The second entry (subject ID) shows the profile name, which consists of a number, followed by the letter 'p' and the EC number de-

```
# RPSTBLASTN 2.2.13 [Nov-27-2005]
# Query: sag:SAG0030  purH; bifunctional phosphoribosylaminoimidazolecarboxamide
formyltransferase/IMP cyclohydrolase (EC:3.5.4.10 2.1.2.3);
K00602 phosphoribosylaminoimidazolecarboxamide formyltransferase [EC:2.1.2.3];
K01492 IMP cyclohydrolase [EC:3.5.4
# Database: /usr/local/share/bio/Priam/DATA/RPSBLAST_MAT_EZ/profile_EZ
# Fields: Query id, Subject id, % identity, alignment length, mismatches,
gap openings, q. start, q. end, s. start, s. end, e-value, bit score
sag:SAG0030  1p3.5.4.10   60.34  532  192  6   7    1545  2     533   0.0    885
sag:SAG0030  1p2.1.2.3    60.34  532  192  6   7    1545  2     533   0.0    885
sag:SAG0030  1p3.4.24.13  18.40  326  255  10  598  1542  1169  1435  7e-07  48.9
```

Figure 2.8: A record in an RPS-BLAST/PRIAM output containing an annotation of the gene SAG0030.

scribed by the profile. The following entries show the alighnment parameters, namely percentage of identical letter pairs, the overall length of the alignment, the number of mismatched letter pairs, the number of gap openings, the starting and ending coordinates of the query sequence (q. start, q. end) and subject sequence (s. start, s. end), the E-value and the bit score. The rows of the table are ordered by an ascending E-value and descending bit score; hence, the higher rows represent the more significant hits. In the example shown, the EC numbers 3.5.4.10 and 2.1.2.3 are predicted with the lowest possible E-value of 0.0, while 3.4.24.13 produces a relatively high E-value of $7 \cdot 10^{-7}$.

2.3.2 Definition of a reaction set

The list of EC numbers encoded in a genome can be translated into a list of metabolic reactions with the aid of biochemical databases, such as KEGG LIGAND, BioCyc and BRENDA. BioCyc contains pathway-genome databases (PGDBs) for completely sequenced species, each of them describing the genome and the predicted metabolic network for a certain organism. The main problems at this stage include the lack of information about the reactions catalysed, incorrectly defined reactions (e.g. some metabolites may be skipped in the stoichiometries) and the inconsistent naming of metabolites. Apart from the reactions catalysed by the known enzymes, the reaction list may include hypothetical reactions, which often belong to the following types:

- spontaneous reactions, such as hydrolysis of carbonic acid;
- generalised reactions representing complex biosynthesis processes, such as protein synthesis;
- transport reactions, whose inclusion is often based on the knowledge of nutrients and end products, rather than the annotation;
- expenditure reactions, which generalise the consumption of currency metabolites, such as ATP and NADH.

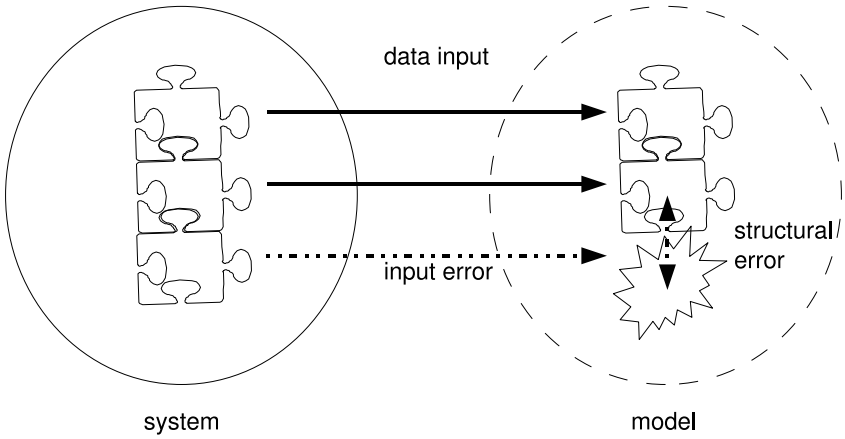


Figure 2.9: Input and structural errors in metabolic reconstruction. The elements of the model represent the elements of the underlying system. An input error is an incorrect representation of an element. It may further lead to a structual error: an internal inconsistency in the model.

2.3.3 Detection and correction of errors

Constructive interrogation of large, automatically generated models is associated with specific problems. Such models typically contain large numbers of errors, whose detection is especially difficult, given that the elements of a model and their identifiers (names) are determined by the database, not the modeller [84]. In the current thesis, we make a distinction between two classes of errors: *input errors* and *structural errors* (see Figure 2.9). Some common types of problems in input data are listed below:

- missannotations;
- atomically unbalanced reactions: some metabolites, such as protons, are often skipped in reaction definitions in databases; more considerable violations are also possible, e.g. in the following reaction[2]:

  4-Methylene-L-glutamate ↔ 4-Methylene-2-oxoglutarate

  nitrogen, oxygen and hydrogen are unbalanced.
- generic metabolites, e.g:

  Primary alcohol + Acceptor ↔ Aldehyde + Reduced acceptor

[2] All examples of errors in this section are found in the KEGG database, release 29.

- polymers with variable degrees and units of polymerisation; e.g. the following pair of reactions:

$$\begin{aligned} \text{ATP} + \text{tRNA} &\leftrightarrow \text{Pyrophosphate} + \text{tRNA} \\ \text{CTP} + \text{tRNA} &\leftrightarrow \text{Pyrophosphate} + \text{tRNA} \end{aligned}$$

  implies that ATP and CTP are isomers.

- inconsistent naming of metabolites (see Figure 2.7f).

Some of these problems can be detected before or during model construction (see Section 4.1.1); unfortunately, many of the input errors (especially the missannotations) remain unrevealed. Structural errors result from input errors and violate certain physical and biochemical constraints inside the model, such as mass conservation and ability to achieve a steady state. Some types of the structural errors can be detected by the analysis of the whole model; they include orphan metabolites, dead reactions, disconnected subnetworks [84] and stoichiometric inconsistencies [33]. Isostoichiometric reactions (i.e. those with identical or proportional stoichiometries) can be also considered as structural errors, since they represent redundant information and form spurious internal cycles, e.g. v_2 and v_3 in Figure 2.3a.

The identification of missing reactions in metabolic models is commonly referred to as gap analysis or gap filling; in genome-scale models, this process may result in annotation or reannotation proposals [74]. Gap filling may be based on literature search or homology search in sequence databases. A number of methods have been developed for the automatisation of the gap filling process, using linear optimisation. The programs GapFind and GapFill [99] resolve the dead-end metabolites in a model (referred by the authors as root no-production and root no-consumption metabolites) by a minimal combination of editing operations, including reversing reaction directionalities, adding new reactions from the MetaCyc database and adding transporters. A method has been proposed for restoring the ability of a model to grow in experimentally observed conditions by adding a minimal number of reactions from a database and transporters for excretion [93]. The GIMME algorithm [7] aims to reconcile a metabolic reconstruction with gene expresssion data.

2.4 Metabolic modelling software

Software development is an important part of the present-day research in systems biology. Only very simple, conceptual systems can be handled manually; practical applications of the existing methods require intensive use of computational tools.

2.4.1 Overview of existing tools and formats

Common scientific packages, such as Mathematica† and MatLab†, are widely used in metabolic modelling. However, the solution of specific problems often

requires development of specialised software. A variety of tools have been created for kinetic modelling, such as Gepasi [69], SCAMP [101], Jarnac [100] and PySCeS [76].

METATOOL [79] was one of the pioneering tools designed particularly for stoichiometric analysis. Being operated from a command line, it calculates the convex basis of a flux cone, elementary flux modes and reaction subsets (termed by the authors as 'enzyme subsets'). A graphical user interface to METATOOL is provided by YANA [114], which also integrates structural models with proteomics and gene expression data. The toolbox SNA [117] calculates, in addition to elementary flux modes, the elementary vectors of a conversion cone (see Section 7). Stoichiometric analysis methods are provided by CellNetAnalyzer [56] and Copasi [41] (successor of Gepasi).

Various formats have been introduced for the description of metabolic models. Gepasi, METATOOL and PySCeS use their own formats which are intended to be readable and editable by humans. Jarnac defines own programming language for the description and interrogation of models.

SBML† (Systems Biology Markup Language) [42] is an open-source, software-independent format for representing biochemical reaction networks, which extends the standard data-description language XML. SBML is increasingly popular in the metabolic modelling community and supported by a wide range of software tools. Other XML-based formats used in systems biology include CellML [63], KGML (used in KEGG database) and the native format of Copasi.

2.4.2 Software design: concepts and patterns

The process of software production may involve several major phases [10]:

- Requirements analysis: identifying what is needed from the software.
- Specification: defining what the software is to do.
- Design: describing how the system is to perform its tasks.
- Implementation: translating the design into a form usable by a computer (e.g. a programming language).
- Testing: validation of the implementation.

In the current thesis, we mainly focus on the first three phases. Although the solutions developed at these steps can be formulated in a human language, their understanding requires the knowledge of some concepts concerning the objectives and methods of software design. Apart from adequately addressing the specified needs, the desirable qualities of a software system often include the following attributes [47]:

- Efficiency: processing time, memory and disk space should not be wasted.
- Reusability: a solution should be applicable to a whole class of problems, rather than address a single problem in a single context.

- Extensibility: it must be possible to add new capabilities to existing software without major changes in its implementation.
- Usability: the user interface must be comprehensible and learnable.

The reusability and extensibility of software systems is often achieved by means of modular design: decomposition of complex systems into sets of relatively simple modules, which must be highly cohesive but loosely coupled [47]. The implementation of design solutions is largely defined by *programming paradigms*: the fundamental styles of problem solving underlying the programming languages. In the *procedural* (imperative) paradigm, a program is a sequence of statements, which can be structured into *procedures* (functions) performing specific tasks. A procedure receives a list of arguments and returns a result; the same procedure can be executed with different arguments in different contexts. In the *object-oriented programming* (OOP) paradigm, a program is formulated in terms of *objects* and their interactions. Table 2.1 introduces some of the basic concepts of OOP, mentioned further in the chapter.

The productivity of software development process can be improved by using *design patterns*, which capture general reusable solutions to commonly occurring problems. These solutions have typically developed and evolved over time and do not depend on the programming language, nor on the specific field of application. Design patterns gained popularity in computer science due to the book 'Design Patterns: Elements of Reusable Object-Oriented Software' [31].

2.4.3 Python programming language

Python† [64] is an open-source, high-level programming language. One of the main advantages of Python is the productivity of work with it: the programming time is about half as long in comparison to lower-level languages, such as C, C++ and Java [87]. The high productivity of Python is provided by its clear, well readable syntax, the absence of a compilation step and the use of efficient, universal built-in data types, such as lists and dictionaries (see Table 2.2). Further, the readability of Python and such features as automatic memory management make it relatively error-safe.

Python is a multi-paradigm language, supporting procedural, object-oriented and functional programming [116]. In addition to classes, the elements of modular design in Python include modules (source code files) and packages (folders) which provide interfaces of their own.

Due to the availability of an interactive console, Python can be used as an environment for both writing and executing programs, thus facilitating incremental development and testing. This feature is utilised by the metabolic modelling tools ScrumPy and PySCeS, which provide access to a Python console.

Although Python programs are less efficient than those written in C or C++, this disadvantage can be circumvented by encoding time and memory-critical components in C and embedding them into Python programs using the SWIG† technology. A variety of Python libraries and packages are available for scientific

Table 2.1: Concepts of object-oriented programming [22, 47, 116, 31]

Concept	*Description*
Encapsulation	Grouping of logically related data and functionalities into a new entity.
Object	Primary unit of decomposition in OOP. Encapsulates data and procedures; has a unique identity and state.
Class	A set of objects with a similar structure and behaviour. Each object is an *instance* of its class.
Built-in types	Basic data types defined by a programming language, such as numbers and characters.
Attributes	Data encapsulated in an object.
Methods	Procedures encapsulated in an object.
Initialisation	The process of creating an object.
Constructor	A special procedure initialising an instance of the class in which it is defined.
Interface	A set of methods and attrubutes by which an object can be accessed; typically hides the details of implementation.
Abstraction	A mechanism by which an interface can be specified independently from the details of implementation.
High-level	Providing strong abstraction from details.
Client	A user or a piece of software which uses a given object
Defined statically	Defined in the source code.
Defined dynamically	Defined during the execution of the program.
Inheritance	A mechanism by which the attributes and methods of one class (superclass) can be extended in another class (subclass); defined statically.
Overriding	Defining a new implementation for an inherited method in a subclass.
Abstract method	Method with no implementation, designed as a part of an interface; to be overridden in subclasses.
Composition	A mechanism by which new functionalities can be created by using objects in ensemble; defined dynamically through object acquiring references to other objects.
Aggregation	A mechanism of composition, by which one object (aggregatee) acts as a component of another object (aggregator).
Delegation	A mechanism of composition, by which one object receives a request from a client and sends it to another object.

tasks, including SciPy†, NumPy† and BioPython†. Python is portable on all major operating systems.

2.4.4 ScrumPy: metabolic modelling in Python

ScrumPy† is an open-source metabolic modelling package, developed in our group. The program is implemented in Python (although certain time-critical algorithms are coded in C). The user interface of ScrumPy is the Python console and the input of requests is performed in the form of Python statements (see Figure 2.10a). In addition, pieces of code can be written in separate source files and executed from the console. The close integration of modelling and programming processes results in the following advantages of ScrumPy over command line and GUI-based modelling tools:

- Flexibility: the range of operations applicable to a model is restricted solely by the user's programming skills and the capabilities of Python.
- Transparency: the user has an almost full control over the behaviour of a model (except for the low-level computations).
- Extensibility: new data structures, methods and algorithms can be added to ScrumPy in the form of modules and packages.
- Reusability: any sequence of operations once applied to a model can be stored in a source file and re-executed. The same solutions can be used in different contexts.
- Efficiency: no system resources are expended for the support of complex graphical user interfaces.
- Usability: the syntax of Python is simple and intuitive; a comprehensive system of nested menus and widgets is unnecessary.

A metabolic model is encapsulated in an object of the class `Model`, which contains references to the internal and external stoichiometry matrices. A stoichiometry matrix is represented as an object of the class `StoMat`, which associates the rows and columns with unique names; it also defines the lists of external metabolites and irreversible reactions as attributes. A model can be represented and stored in a special plain-text based file format (see Figure 2.10b).

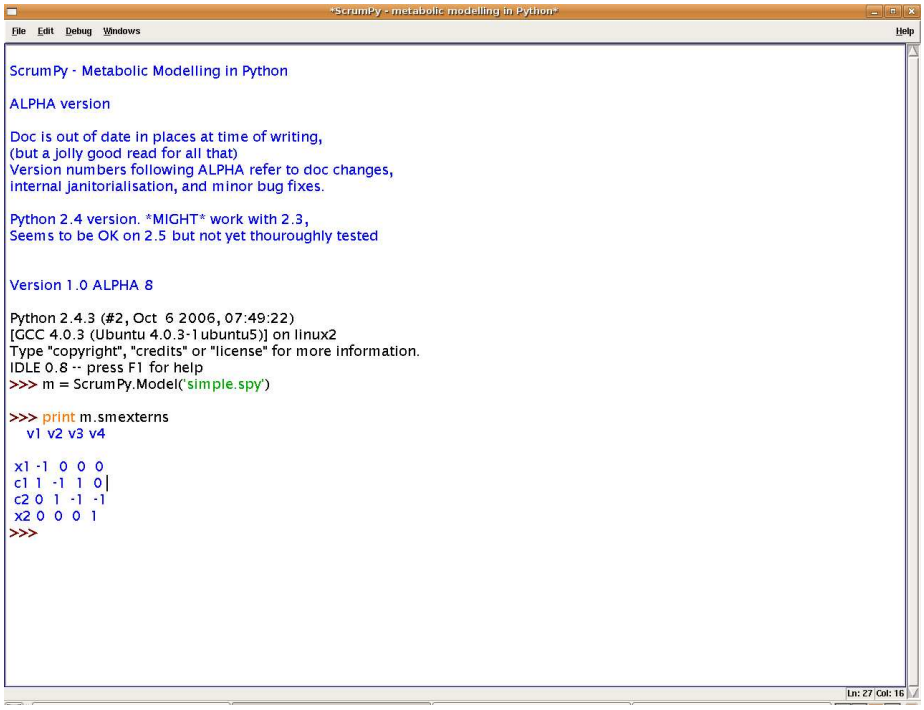
ScrumPy provides methods of both kinetic and structural modelling, including the calculation of elementary modes (implemented in C), reaction subsets, conservation relationships, reaction correlation coefficients and some graph-theoretical methods. Numerical errors can be avoided due to the use of arbitrary-rational numerical types provided by the package GMP†(GNU multiple precision arithmetic library).

A number of extensions to ScrumPy are available in the form of separate packages, such as `Kegg2` and `PyoCyc`, which are intended for the interrogation of the KEGG and BioCyc databases, respectively. The package `glpk` provides

Table 2.2: Built-in types and classes with examples of usage in the current work. The original classes listed in the table are described in chapter 3.

Type, class	*Description*	*Can represent*
Built-in Python types		
`str`	string	metabolite name: `"water"`
`bool`	boolean	liveness of a reaction: `True` or `False`
`int`	integer number	stoichiometric coefficient: `1`
`float`	floating point number	correlation coefficient: `1.0`
`tuple`	immutable ordered collection	correlation coefficient with the p-value: `(1.0, 0.0)`
`list`	mutable ordered collection	genes in a genome: `[SAG0001, SAG0002]`
`dict`	dictionary associating unique keys to values	reaction stoichiometry: {`a:-1, b:1`} means 'a → b'
ScrumPy classes		
`matrix`	matrix in which rows and columns are associated with unique names	arbitrary matrix
`StoMat`	subclass of `matrix`; provides some methods of stoichiometric analysis	stoichiometry matrix
`Model`	encapsulates a metabolic model; contains an external and internal stoichiometry matrices as attributes	metabolic model
Original classes		
`Set`	unordered collection	enzymes encoded by a gene
`Relation`	dictionary associating keys with sets	relation between genes and encoded enzymes
`Record`	dictionary with a unique ID	description of an enzyme in a metabolic database
`Database`	dictionary associating IDs with records	metabolic database
`MtxLP`	encapsulates an LP	LP based on a given coefficient matrix
`System`	dictionary of genes, enzymes, reactions and metabolites	a layer in an integrated metabolic reconstruction

an object oriented Python interface to the linear programming solver GLPK[†]. A range of ScrumPy extensions have been developed in the scope of the current work; these are described in the following chapter.



a

```
Structural()
External(x1, x2)

v1: x1 -> c1  ~
v2: c1 -> c2  ~
v3: c2 -> c1  ~
v4: c2 -> x2  ~
```

b

Figure 2.10: a) The ScrumPy user interface. The first statement in the Python console initialises the metabolic model stored in the file 'simple.spy' (shown in Figure 2.3a). The second statement shows the text representation of the external stoichiometry matrix. b) The same model represented in ScrumPy format: the first line indicates that the model is structural; the second line defines the list of external metabolites; the following lines define reaction names and stoichiometries (the arrow symbol '`->`' indicates that a reaction is irreversible).

Chapter 3

Software Development

Software development on its own right was not an objective of the current project; our in-house tool ScrumPy [82] was used for metabolic modelling at all stages of the work. Nevertheless, the solution of various problems arising during the construction and analysis of models led to the necessity of new software development. This necessity was determined by two major reasons: firstly, original analysis methods were developed and needed implementation; secondly, a range of external data resources and tools was employed and interfaces were required between these and ScrumPy.

The software was implemented in the Python programming language in the form of extensions to ScrumPy. Although these extensions were developed with particular current problems in mind, they were aimed to represent possibly accomplished and reusable components rather than *ad hoc* solutions. The achievement of this objective was facilitated by the employment of existing design patterns [31]. The resulting software was organised into several packages representing relatively independent functional units.

In this chapter, we describe the packages `Utils, LP, Source` and `Bio` (see the CD attached), focusing on the basic design solutions and the implementation of the existing analysis methods. The implementation of original methods is described in the following chapters.

3.1 Utility functions and classes

The package `Utils` provides a variety of basic, general-purpose functinalities, which are intended for three major purposes: implementation of mathematical routines, providing universal data structures for information storage, and connecting ScrumPy to external software.

Table 3.1: Design patterns [31] and their applications in the current work.

Pattern name	*Intent*	*Applications*
Factory method	Define an interface for creating objects, without specifying their exact class.	Initialisation of databases and integrated metabolic reconstructions.
Singleton	Ensure that a class has only one instance.	Initialisation of a KEGG database object.
Composite	Compose objects into tree structures to represent part-whole hierarchies.	Structure of databases and integrated metabolic reconstructions.
Decorator (Wrapper)	Attach additional responsibilities to an object dynamically; alternative to inheritance.	Multilayered structure of an integrated metabolic reconstruction.
Facade	Provide a unified interface to a set of lower-level interfaces.	Unified interface to the subdatabases of KEGG LIGAND. Interface to databases and analysis methods provided by an integrated metabolic reconstruction.
Proxy	Provide a surrogate or placeholder for another object; create expensive objects on demand.	Initialisation of element objects in an integrated metabolic reconstruction.
Strategy	Define a family of algorithms, encapsulate each one, and make them interchangeable.	Lexical analysers, parsers and field editors for KEGG database.

3.1.1 Mathematical routines

Although ScrumPy provides built-in methods for the work with matrices, sequences and sets, additional functionalities were required. Since these functionalities apply to already existing instances of data structures, they were implemented procedurally (except for the class `Annealer`) and organised into several modules[1], described below.

Module `Round` This module is designed for handling round-off errors: a frequent issue occuring in computations involving floating-point numbers. Such numbers are represented in computers with a final precision, termed as *machine accuracy*[2]. In typical 32-bit computers, the machine accuracy is around 3×10^{-8} [88]. The module defines the constant `ZEROVAL` $= 10^{-8}$ and simple procedures for comparing numbers with this constant: any number with a smaller absolute value is assumed to be equal to zero. This comparison is used in matrix and LP calculations to provide a tolerable accuracy of results.

Module `Sto` Reaction stoichiometries can be conveniently represented in the form of Python dictionaries, with keys and values corresponding to metabolite names and coefficients, respectively, as shown in Table 2.2. This module defines procedures operating on these dictionaries, such as comparision, multiplication, normalisation, extraction of substrate and product sets.

Module `Mtx` Defines a range of procedures operating on matrices represented as instances of the ScrumPy class `StoMat`. The functionalities provided by this module include the following:

- Comparing matrices, their rows, columns and elements with zero, in respect of the tolerable floating point accuracy.
- Comparing a pair of matrices, detection of equal columns regardless of the order of rows.
- Detection of proportional rows.
- Null-space analysis methods, including detection of dead reactions and reaction subsets in a null-space matrix containing floating point numbers (the corresponding ScrumPy methods are applicable to matrices of arbitrary precision numbers only).
- Reaction correlation analysis of stoichiometry matrices.
- Editing operations on stoichiometry matrices, such as creation of a column representing a given stoichiometry.

[1] Note that the word *module* here and further in the chapter refers to Python source files, rather than to the general meaning of modules in engineering and systems theory.

[2] More precisely, machine accuracy is the smallest floating point number which, when added to the floating point number 1.0, produces a floating point number different from 1.0.

- Operations on boolean matrices representing binary relations, such as finding the union of two relations.

Module `Connect` Implements graph-theoretical analysis of stoichiometry matrices, including detection of connected components, orphan metabolites and the 'Core' algorithm.

Module `String` Defines procedures for parsing, editing and generating strings, including those representing reaction equations and EC numbers.

Module `Comp` Implements the calculation of elementary substrate and product compositions (see Chapter 7).

Module `StoiCons` Implements the detection of minimal inconsistent net stoichiometries and elementary leakage modes (see Chapter 7).

Module `Annealing` Defines the class `Annealer`, which provides a generic implementation of the simulated annealing algorithm (see Chapter 6).

3.1.2 Data structures

The classes described below are intended to provide a collection of universal structures for parsing, editing, storing and interrogating data. To achieve this objective, the functionalities of built-in Python types (namely lists and dictionaries) were extended with new capabilities including serialisation (i.e. storing objects in files) and certain features of relational databases. Two mechanisms of serialisation are supported: 1) the standard Python mechanism called cPickle[†] and 2) storing and parsing text files, for which special methods are provided. The inheritance hierarchy and composition of classes is shown in the upper part of Figure 3.1.

Class `IOContainer` Provides an interface to cPickle. Methods for generating and parsing text representation are declared.

Class `IOList` Subclass of `IOContainer` and Python `list`.

Class `IODict` Subclass of `IOContainer` and Python `dict`.

Class `Set` Implements basic set-theoretical operations and parsing/storing methods for the text representation of sets in the form of comma-separated values. The module also provides functions for set distance calculation.

Class `Relation` A dictionary associating each key with a collection of objects, which is internally represented as a `Set` object. In the text representation, in each line a key is followed by a tab and the text representation of the corresponding set. An example is shown below:

```
SAG0850
SAG0884     1.1.1.1, 6.3.2.7
SAG1391     4.2.1.58, 6.3.2.7
None     1.3.1.10, 1.5.1.12
```

The first three lines show gene identifiers SAG0850, SAG0884 and SAG1391 followed by the lists of EC numbers of the enzymes encoded by the corresponding genes, whereby SAG0850 encodes nothing. Note that each element in the sets can be associated with more than one key; e.g. 6.3.2.7 is encoded by two genes. The last line starts with the default key `None`, followed by those enzymes encoded by none of the genes. Hence, the class `Relation` represents a many-to-many relation between two sets (in this example - sets of genes and enzymes)[3]. Although such relations are typically represented as binary matrices, the dictionary representation has some advantages, including low memory consumption (no memory is occupied by empty elements of the matrix) and availability of the highly efficient search by key. By calling the `GetReverse()` method, the reverse representation of the same relation can be obtained:

```
1.1.1.1         sag:SAG0884
1.3.1.10
1.5.1.12
4.2.1.58        sag:SAG1391
6.3.2.7      sag:SAG0884, sag:SAG1391
None
```

The class provides a range of methods for editing and selecting keys, elements or subrelations according to various criteria.

Class `Record` Associates each key with a list or a dictionary representing a data field; in addition, contains the string attribute `ID` and implements text representation, e.g:

```
Water:
        Formula: ['H2O']
        Reaction: ['R00001', 'R00002', 'R00004']
        Canonical SMILES: ['O']
        Mass: ['18.0106']
```

This is an example of a record describing Water; the top line contains the ID, each of the remaining lines shows a key followed by a colon and the content of the corresponding field.

[3]The text format described here (except the `None` key) is used in the KEGG database for storing annotated genomes.

Class `Database` Associates each key with a unique record, whose ID is identical to the key. In addition, contains a `Relation` object which associates the values of certain fields to the IDs of the corresponding records, thus implementing database indexing. E.g. in a database of metabolites, the IDs could represent the names (as in the example above) and the indexing relation could associate the reactions with the metabolites used.

Implements a range of selection methods. Each selection method returns a group of records satisfying a given query, which is organised as a new database, e.g:

```
>>> subDB = DB.Select(key = 'REACTION', value = 'R00001')
```

In this example, the method `Select` returns a subdatabase of all metabolites used by the reaction R00001.

The design of this class uses the pattern Composite, i.e. some methods of the class are delegated to the components, namely the records. For instance, the text representation of a database is simply a sequence of text representations of its records. A database can be parsed from an external resource (the parsing method must be implemented in a subclass) and stored in a cPickle or text file.

3.1.3 Interfaces to external software

Module `Latex` Represents data in the Latex† format, which is used by the typesetting system of the same name.

Module `NJ` Represents hierarchical data in and parses them from the Newick† tree format, which is used by the NJplot† tree viewer.

Module `Layout` Uses the package Pydot† for representing metabolic networks in the form of bipartite graphs, which can be visualised and stored in various image file formats via the graph visualization software Graphviz†.

3.2 Interfaces to data resources

Integration of metabolic models with external data resources was one of the main objectives of software development in the current project. Such resources as biochemical and other databases, annotation tools and microarrays were used for the construction of models, their analysis and the interpretation of results; therefore, the development of interfaces connecting them to ScrumPy was an absolute necessity. Although the databases and tools strongly differ in the formats used and functionalities provided, abstraction was used to design generalised interfaces, providing access to the data required for modelling tasks and ignoring other fields. Thus, by extending the classes `Database` and `Record`, generic data structures were developed for biochemical and annotative databases. The abstract methods were implemented in the classes linked to specific resources, namely KEGG and PRIAM.

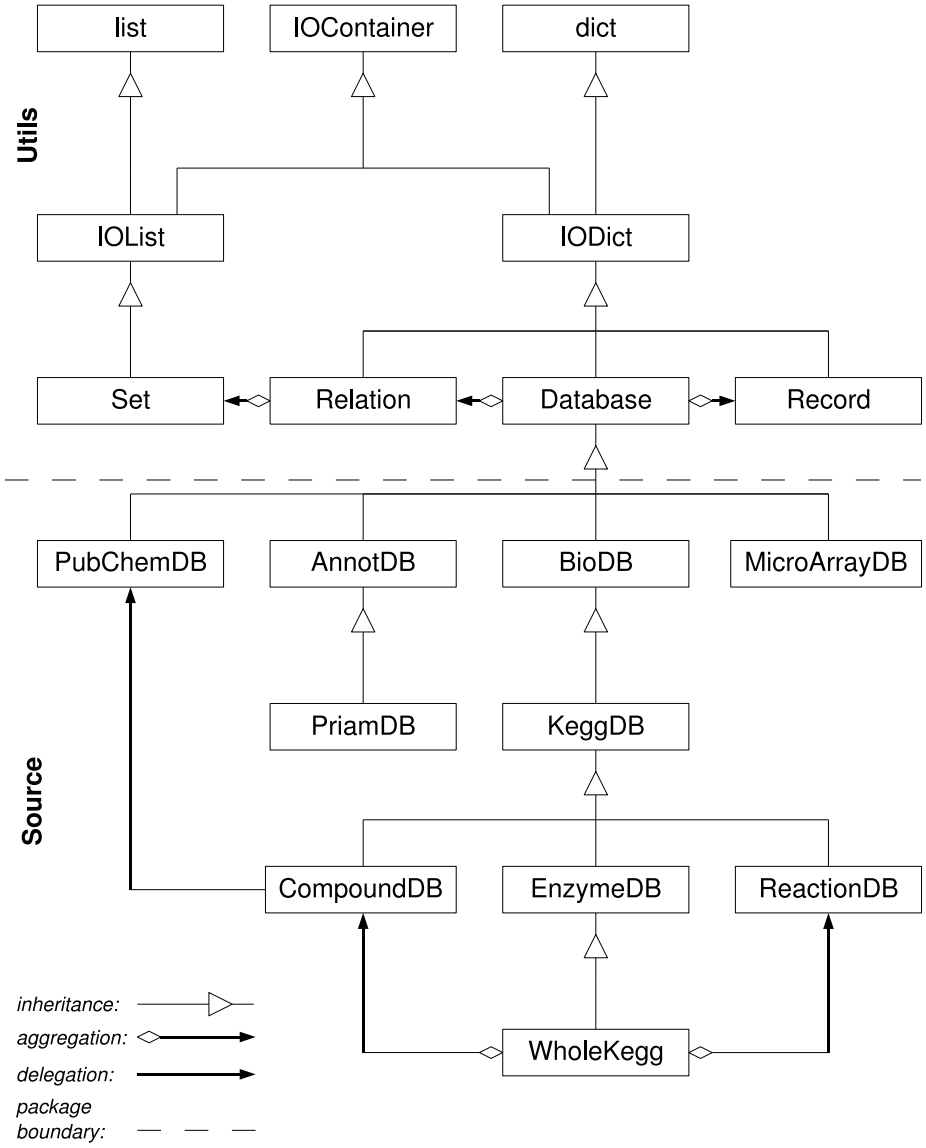


Figure 3.1: Inheritance and composition of the data structure classes in the packages `Utils` and `Source`. The subclasses of `Record` are not shown; their inheritance hierarchy mirrors that of database classes.

The classes described below are collected in the package `Source`; their inheritance hierarchy and composition is shown in the lower part of Figure 3.1.

3.2.1 Biochemical databases

By biochemical databases we mean those describing compounds, reactions and/or enzymes, such as KEGG LIGAND, BioCyc and BRENDA.

Class `BioRecord` Declares a range of abstract access methods returning synonyms, web links, comments and other data.

Classes `CompoundRecord, ReactionRecord` **and** `EnzymeRecord` Declare specific abstract methods, e.g. returning the formula of a given metabolite, the equation of a given reaction and the list of reactions catalysed by a given enzyme.

Class `BioDB` Contains records of the three classes mentioned above. The methods `GetCompoundDB, GetReactionDB, GetEnzymeDB` return subdatabases containing records of the given class only. Declares abstract selection methods by biochemical criteria (e.g. isomers) and methods returning general biochemical information, e.g. the identifiers of amino acids and polymers in the given data resource.

3.2.2 KEGG interface

KEGG LIGAND database was used in the current work for automatic model generation and for retrieving additional information (not included into the models) about reactions, metabolites and enzymes. The database consists of several parts, including COMPOUND (information about chemical compounds), REACTION (biochemical reactions), ENZYME (enzyme nomenclature) and GLYCAN (experimentally determined glycan structures).

Although the database provides an on-line application programming interface (KEGG API[†]), it includes only a limited set of methods. Further, the access through a network considerably slows down the work with massive datasets. Finally, using on-line data as an input for model construction makes the resulting models potentially irreproducible, since the content of the database is not constant.

By the beginning of the work, an interface to KEGG LIGAND had been already developed by M.G.Poolman. This interface included classes for accessing the COMPOUND and REACTION databases, which were downloaded in the form of flat files from the KEGG FTP[†] server and stored locally. However, it was decided to develop a new implementation using the technologies lex/yacc and cPickle for parsing and serialisation of data, respectively. A unified interface to the databases COMPOUND, REACTION, ENZYME, GLYCAN and PUBCHEM was developed using the design patterns `Facade, Strategy` and `Singleton`.

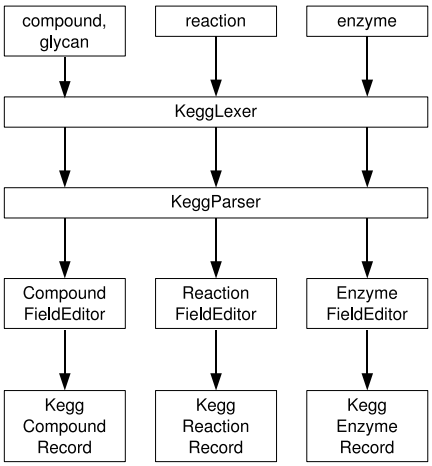


Figure 3.2: Information flow during the parsing of the KEGG local copy. The fourth row shows the subclasses of `KeggFieldEditor`.

Parsing The local copy of KEGG API includes flat files named `compound, reaction, enzyme` and `glycan`, containing the data from the corresponding parts of the database, represented as sequences of text records. Unfortunately, no documentation about the formats of these files was found, so the formatting rules were inferred as a result of trial-and-error-based reverse engineering process. The parsing of the files uses the external package Ply[†] which is a Python implementation of the widely used parsing tools lex and yacc[†]. lex generates a lexical analyzer, which scans an input stream (e.g. a file) and picks up the basic items, called tokens, according to definitions and rules supplied by the user. yacc generates a parser, which recognises the tokens and performs some user-defined actions. The KEGG parsing rules for lex and yacc are defined in the modules `KeggLexer` and `KeggParser`, respectively. The parser recognises the start and end of a record in the text, the ID and the entire fields. The internal structures of the fields are analysed by the class `KeggFieldEditor` and its subclasses, which store the fields, represented as lists or dictionaries, in the record objects (see Figure 3.2).

The format of the KEGG flat files is not constant; we found a number of changes in different releases of the database. This issue is addressed by changing the rules defined in `KeggLexer` and `KeggParser` modules and/or the methods defined in `KeggFieldEditor` and its subclasses. Note that the changes in the database format do not affect the class `KeggDB` and its subclasses, since the structure of the database classes is decoupled from the parsing process. Moreover, the same objects can be loaded from different files using different parsers (design pattern Strategy).

Class `KeggRecord` Implements access to common fields, such as names, comments, links and pathways.

Classes `KeggCompoundRecord`, `KeggReactionRecord` **and** `KeggEnzymeRecord` Each of these classes has two parents in the hierarchy: `KeggRecord` and the corresponding generic BioRecord class. Implement the specific access methods declared in their generic parents.

Class `KeggDB` Contains the locations of input files and directories and implements high-level methods for parsing, saving and loading files. During the initialisation, an object of this class searches the special directory, in which its content can be serialised in a cPickle file. If it finds the file, it loads the contents from it. Otherwise, it parses the flat file and immediately serialises itself in a cPickle file. The reason for dual storage of data is that loading a database from cPickle is about ten times less time-consuming then parsing the KEGG flat files.

Class `PubChemDB` The PUBCHEM† database contains information about molecular structures of chemical compounds. This class parses the web pages of the database and extracts the following data from each record: canonical and isomeric SMILES† strings (unambiguous description of the molecular structure), IUPAC† name and International Chemical Identifier (InChI†).

Class `CompoundDB` Contains the data from the files `compound` and `glycan`. In addition, during the initialisation an object of this class initialises a `PubChemDB` object and updates itself with the content of the latter. Hence, the information from KEGG COMPOUND and PUBCHEM databases is integrated in the same records.

Classes `ReactionDB` **and** `EnzymeDB` Implement methods for selecting sets of reactions by the catalysing enzymes and vice versa, as well as by genes and organisms (as specified in KEGG LIGAND).

Class `WholeKegg` Is a subclass of `EnzymeDB` but also contains refences to `CompoundDB` and `ReactionDB` objects, which are loaded during its initialisation (design pattern Facade). Implements a range of methods used for model construction, some of which are described in Chapter 4.

3.2.3 Annotative databases

The purpose of annotative databases is the storage of information about genes and predicted enzymatic functions. Each object of the class `AnnotRecord` describes a single gene or ORF, whose identifier is the record ID. The fields contain information about the predicted proteins and prediction parameters, such as alignment length, bit score and e-value. The class `AnnotDB` implements selection methods returning subdatabases of records satisfying certain criteria.

Classes `PriamRecord` and `PriamDB` Implement the methods for parsing the output of the annotation tool RPS-BLAST executed in combination with the database of enzyme signature PRIAM.

3.2.4 Gene expression database

The class `MicroArrayDB` provides an interface for the analysis of gene expression data. The data can be parsed from a comma-separated spreadsheet file ('.csv' format). Each record (class `MARecord`) stores the data from one spreadsheet row; the field names are defined by the column names in the spreadsheet; the values in one of the columns (defined by the user) are used as record IDs (see Figure 3.3a). The content of a database can be represented as a gene expression table (Figure 3.3b). The rows in this table are used to calculate the correlations between gene expression patterns; the result can be represented as a correlation matrix (Figure 3.3c). The correlation method can be supplied by the user as an argument; by default, Pearson's correlation coefficient is used.

3.2.5 Initialisation of databases

Many applications depend on objects of certain families of classes, whereby the specific classes themselves are determined dynamically. For instance, an integrated metabolic reconstruction object may contain references to biochemical, annotative and gene expression databases, but the names of the resources are specified in the input file or defined by the user. The package `Source` implements the method `GetDB`, which receives the name of a resource and returns the corresponding database (design pattern Factory Method). For instance, the following two statements initialise objects of the classes `WholeKegg` and `PriamDB`, respectively:

```
>>> Kegg = Source.GetDB('KEGG')
>>> Priam = Source.GetDB('PRIAM')
```

The initialisation of a `WholeKegg` object is time-consuming and the object itself requires a massive amount of memory. Initialising it more than once leads to a waste of resources, since the content of the database is unique (unless different versions are used). Therefore, after the initialisation of a `WholeKegg` object, the package keeps a reference to it. If the function `GetDB` is called again with the argument 'KEGG', this reference is returned (design pattern Singleton).

3.3 Linear programming interface

Linear programming methods have been widely used in the current work. In addition to established methods, such as flux optimisation and coupling analysis, original methods have been developed involving LP and MILP techniques. The package LP includes a hierarchy of classes, where each class encapsulates a set

```
ORF01584:
        2.5_hr: [0.025999999999999999]
        8.0_hr: [0.94299999999999995]
        4.5_hr: [-2.3959999999999999]
        6.0_hr: [-0.86299999999999999]
        0.5_hr: [0.5]
        Locus: ['SAG1422']
        2.0_hr: [-0.012]
        1.0_hr: [0.65900000000000003]
        4.0_hr: [-0.45300000000000001]
        0_hr: [0.161]
        3.25_hr: [-0.437]
        Common name: ['"glycosyl transferase, group 2"']
        Organism: ['2603']
        5.0_hr: [-0.93300000000000005]
ORF00928:
        2.5_hr: [0.246]
        8.0_hr: [-1.464]
        4.5_hr: [-2.7869999999999999]
        6.0_hr: [-3.9980000000000002]
        0.5_hr: [-0.76000000000000001]
        Locus: ['SAG0818']
        2.0_hr: [0.13600000000000001]
        1.0_hr: [-0.16400000000000001]
        4.0_hr: [-0.621]
        0_hr: [-0.69099999999999995]
        3.25_hr: [0.01]
        Common name: ['ribonucleoside-diphosphate red']
        Organism: ['2603']
        5.0_hr: [-1.891]
```

a

	0.5_hr	0_hr	1.0_hr	2.0_hr	2.5_hr	3.25_hr	4.0_hr	4.5_hr	5.0_hr	6.0_hr	8.0_hr
SAG0818	-0.76	-0.691	-0.164	0.136	0.246	0.01	-0.621	-2.787	-1.891	-3.998	-1.464
SAG1422	0.5	0.161	0.659	-0.012	0.026	-0.437	-0.453	-2.396	-0.933	-0.863	0.943

b

	SAG0818	SAG1422
SAG0818	1.0	0.573097909064
SAG1422	0.573097909064	1.0

c

Figure 3.3: a) The text representation of a gene expression database containing two records. The record IDs are set to the ORF identifier; the fields correspond to the gene identifiers ('Locus'), gene product names ('Common name'), organism names and expression levels at different time points. b) Expression table: gene IDs are used as row labels, the columns correspond to expression levels. c) Correlation matrix with Pearson's correlation coefficients.

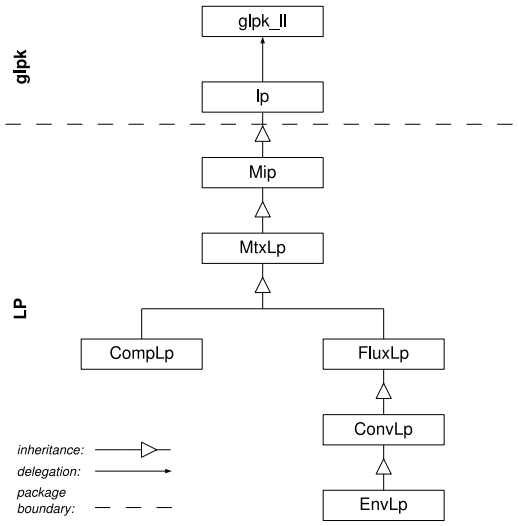


Figure 3.4: Inheritance hierarchy of classes in the packages glpk and LP.

of linear programming methods based on one of the approaches, such as mass balance analysis, flux balance analysis and conversion analysis (see Figure 3.4).

Low-level classes GNU Linear Programming Kit[†] (GLPK) is an open-source software package intended for solving LP, MILP, and other related problems. The package is implemented in the C programming language. A low-level Python interface to GLPK was created by M.G.Poolman and encapsulated in the package `glpk`; it includes the automatically generated module `glpk_ll` and the class `lp`. Each object of this class represents a linear program; it can be edited after the initialisation (e.g. the objective function can be changed and new constraints can be included) and solved repeatedly. The class implements basic methods for defining the objective function and linear constraints and the method `Solve`, which delegates the solution to the GLPK solver.

The subclass `Mip` was developed in the scope of the current work; it introduces the possibility of defining integer variables and solving mixed-integer linear programs.

Class `MtxLp` Since it is convenient to represent linear constraints in the form of a stoichiometry matrix, the constructor of this class accepts a `StoMat` object as an argument and uses it as the matrix of linear constraints. Additional arguments define the direction of optimisation (minimisation or maximisation), the class of the problem (LP or MILP) and the lower and upper bounds of the variables, e.g:

```
>>> lp = LP.MtxLp(mtx = A, direc = 'Min', klass = 'LP',
```

```
lowBound = 0, upBound = 1)
```

The created object corresponds to the following LP with an empty objective function:

$$\begin{array}{rl} \text{Minimise} & \\ \text{Subject to} & \mathbf{Ax} = \mathbf{0} \\ \text{Where} & 0 \leq x_i \leq 1,\ 0 \leq i \leq a \end{array} \tag{3.1}$$

where a is the dimension of $\mathbf{A}$. By replacing the argument 'LP' by 'MIP', an MILP can be created, where each real-number variable x_i is associated with an additional integer variable k_i in the interval $[0, 1]$, e.g.

```
>>> milp = LP.MtxLp(mtx = A, direc = 'Max', klass = 'MIP')
```

creates the following MILP:

$$\begin{array}{rl} \text{Maximise} & \\ \text{Subject to} & \mathbf{Ax} = \mathbf{0} \\ \text{Where} & x_i \geq 0,\ 0 \leq i \leq a, \\ & k_i \in \{0, 1\},\ 0 \leq i \leq a \end{array} \tag{3.2}$$

(the default lower and upper bounds for all x_i are 0 and ∞, respectively).

The method `SetObjective` defines the stoichiometry of the objective function, which can be passed in as a dictionary, e.g.

```
>>> lp.SetObjective({'x1' : 1, 'x2' : 2})
```

results in the following LP:

$$\begin{array}{rl} \text{Minimise} & x_1 + 2x_2 \\ \text{Subject to} & \mathbf{Ax} = \mathbf{0} \\ \text{Where} & 0 \leq x_i \leq 1,\ 0 \leq i \leq a, \end{array} \tag{3.3}$$

Two methods define special objective functions, widely used for a range of tasks. `SetSumObjective` defines the objective function as the simple sum of variables x_i, i.e. $\sum_{i=1}^{a} x_i$ (or $\mathbf{1}^T\mathbf{x}$ in the vector form).

The method `SetLenObjective` is only applicable to mixed-integer problems. It defines the objective function as the sum of integer variables k_i and includes additional inequality constraints of the form $x_i \leq k_i$ if the objective is minimisation and $k_i \leq x_i$ if it is maximisation, e.g:

```
>>> milp.SetLenObjective()
```

results in the following MILP:

$$\begin{array}{rl} \text{Maximise} & \sum_{i=1}^{a} k_i \\ \text{Subject to} & \mathbf{Ax} = \mathbf{0} \\ \text{Where} & 0 \leq k_i \leq x_i,\ k_i \in \{0, 1\},\ 0 \leq i \leq a \end{array} \tag{3.4}$$

The solution $\mathbf{k}$ contains a maximal number of unit components. However, the third line ensures that a component k_i can be set to a unit only if the corresponding x_i is positive. Hence, the program effectively maximises the number of

positive components in $\mathbf{x}$. Similarly, if the objective is minimisation, the MILP has the following form:

$$\begin{array}{rl} \text{Minimise} & \sum_{i=1}^{a} k_i \\ \text{Subject to} & \mathbf{Ax} = \mathbf{0} \\ \text{Where} & 0 \leq x_i \leq k_i,\ k_i \in \{0,1\},\ 0 \leq i \leq a \end{array} \tag{3.5}$$

Each k_i can be set to zero only if the corresponding x_i is zero. Hence, the program minimises the number of positive components in $\mathbf{x}$. Equations 3.4 and 3.5 are widely used for various tasks, e.g. minimising the number of reactions involved in a pathway.

The class implements a number of methods for complementing a linear program with new constraints. The method `MakePositive` forces a given list of variables to be greater then or equal to a predefined small constant `EPSILON = ZEROVAL`$^{\frac{1}{2}}$, e.g.

```
>>> lp.SetSumObjective()
>>> lp.MakePositive(['x1', 'x2'])
```

results in the following LP:

$$\begin{array}{rl} \text{Minimise} & \sum_{i=1}^{a} x_i \\ \text{Subject to} & \mathbf{Ax} = \mathbf{0} \\ & x_1, x_2 \geq \epsilon \\ \text{Where} & 0 \leq x_i \leq 1,\ 0 \leq i \leq a, \end{array} \tag{3.6}$$

where $\epsilon =$ `EPSILON`. This program calculates the solution $\mathbf{x}$ with the minimal sum of components, such that x_1 and x_2 are positive. The reason for writing $x_i \geq \epsilon$ instead of $x_i > 0$ is the impossibility of using strict inequalities in linear programs.

As mentioned in Chapter 2, mixed integer linear programming enables finding multiple solutions satisfying the same problem. This can be done by iteratively solving an MILP, whereby after each iteration the last solution must be excluded from the feasible region. The method `SetIntegerCut` provides this possibility by complementing the MILP with a constraint termed *integer cut* [73]:

$$\sum_{i \in P(\mathbf{k})} k_i \leq |P(\mathbf{k})| - 1 \tag{3.7}$$

where $\mathbf{k}$ is a solution previously found. The integer cut ensures that the unit components of $\mathbf{k}$ do not obtain unit values simultaneously in further solutions since their sum is set to be less then their number.

The method `GetSolution` returns the current solution in the form of a dictionary or a column vector:

```
>>> lp.Solve()
>>> print lp.GetSolution()
{'x2': 1.0, 'x1': 1.0, 'x4': 1.0}
```

```
>>> print lp.GetSolution(asmatrix = True)
    v

 x1 1.0
 x2 1.0
 x3 0.0
 x4 1.0
```

Other functionalities implemented in this class include general methods for coupling analysis [11, 73] and essentiality analysis (see Chapter 8).

Class `CompLp` The constructor of this class receives the external stoichiometry matrix $\hat{\mathbf{N}}^T$ of a network as the matrix argument. The class implements methods of mass-balance analysis and in particular, the verification of stoichiometric consistency and detection of unconserved metabolites (see Chapter 5).

Class `FluxLp` This class is intended for flux-balance analysis. In the LP problems defined by this class, reaction flux rates and stoichiometries are represented as variables and constraints, respectively. A simple example of such a representation is shown in Equation 2.15, where the internal stoichiometry matrix is used as the matrix of linear constraints. However, this equation ignores the possibility of a negative flux rate in the reversible flux v_2, since the lower bounds of all variables are set to zero. Defining no lower bounds for the flux rates of reversible reactions would possibly lead to the unboundedness of minimisation problems and to the impossibility to calculate minimal absolute rates.

This problem is often tackled by 'splitting up' the reversible reactions, i.e. by replacing each of them by a pair of mutually opposite irreversible ones (the same approach is used for the calculation of extreme pathways). In the current work, we apply a slightly modified approach: Firtsly, we 'split up' all columns in the internal stoichiometry matrix. The resulting matrix can be mathematically represented as the split matrix $\ddot{\mathbf{N}} = (\mathbf{N}| - \mathbf{N})$. We denote the distributions of forward and backward flux rates by ${}^{+}\mathbf{v}$ and ${}^{-}\mathbf{v}$, respectively, and the split flux vector $\ddot{\mathbf{v}} = ({}^{+}\mathbf{v}|{}^{-}\mathbf{v})^T$. The following equality is satisfied at a steady state:

$$\ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0} \tag{3.8}$$

Then we explicitly set the flux rates of the backward directions of irreversible reactions to zero, thus preventing the backward fluxes:

$${}^{-}\mathbf{v}^{irr} = \mathbf{0} \tag{3.9}$$

The constructor of `FluxLP` receives a `StoMat` object representing an internal stoichiometry as the matrix argument, e.g.

```
>>> flp = LP.FluxLp(N)
```

creates the following LP:

$$\begin{array}{rl} \text{Minimise} & \\ \text{Subject to} & \ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0}, \\ & {}^{-}\mathbf{v}^{irr} = \mathbf{0} \\ \text{Where} & \ddot{\mathbf{v}} \geq \mathbf{0} \end{array} \tag{3.10}$$

(note that the list of irreversible reactions is contained as an attribute in `StoMat` itself). Although any solution of this program has the form $\ddot{\mathbf{v}}$, an actual distribution of net flux rates is calculated as follows:

$$\mathbf{v} = \mathbf{v}^{+} - \mathbf{v}^{-} \tag{3.11}$$

This distribution is returned by default by the method `GetSolution`.

The class implements editing methods, such as including and deleting reactions and changing their reversibility. The method `IsLive` determines the 'liveness' of a given reaction, e.g. its ability to carry some flux at a steady state. This problem does not reduce itself to the calculation of a solution with a positive value in the given component, since any reversible flux is involved in a spurious cycle with its opposite direction (we define these cycles as *trivial*). To exclude the fluxes in trivial cycles, we block the opposite direction:

$$\begin{array}{rl} \text{Minimise} & \\ \text{Subject to} & \ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0}, \\ & {}^{-}\mathbf{v}^{irr} = \mathbf{0}, \\ & \ddot{v}_j \geq \epsilon, \\ & \ddot{v}_{opp(j)} = 0 \\ \text{Where} & \ddot{\mathbf{v}} \geq \mathbf{0} \end{array} \tag{3.12}$$

where ϵ is a small positive number and $opp(j)$ is the opposite direction of the j-th flux:

$$opp(j) = (j + n) \ \% \ 2n \tag{3.13}$$

Note that the LP shown in Equation 3.12 has an empty objective function; it only determines the feasibility of the problem. The j-th reaction is live iff at least one feasible solution exists. The LP is solved twice, with forward and backward fluxes taken as $\ddot{v}_j$; the method returns `True` iff at least one of the directions is feasible. In contrast to the null space analysis method for detection of strictly detailed balanced reactions, the LP-based method takes into account the irreversibility constraint and is applicable to individual reactions.

A number of methods is provided for the calculation of flux modes satisfying given constraints. The method `MinSumMode` minimises the sum of all flux rates in a mode, while retaining a non-zero flux in a given reaction, using the following LP:

$$\begin{array}{rl} \text{Minimise} & \sum_{i=1}^{2n} \ddot{\mathbf{v}}_i \\ \text{Subject to} & \ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0}, \\ & {}^{-}\mathbf{v}^{irr} = \mathbf{0}, \\ & \ddot{v}_j \geq \epsilon, \\ & \ddot{v}_{opp(j)} = 0 \\ \text{Where} & \ddot{\mathbf{v}} \geq \mathbf{0} \end{array} \tag{3.14}$$

The method `ShortestMode` calculates the shortest (i.e. involving the minimal number of reactions) flux mode in which a given reaction is participating in either direction. The following MILP is invoked for each direction:

$$\begin{array}{rl} \text{Minimise} & \sum_{i=1}^{2n} k_i \\ \text{Subject to} & \ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0}, \\ & {}^{-}\mathbf{v}^{irr} = \mathbf{0}, \\ & \ddot{v}_j \geq \epsilon, \\ & \ddot{v}_{opp(j)} = 0 \\ \text{Where} & 0 \leq \ddot{v}_i \leq k_i,\ 1 \leq i \leq 2n \end{array} \tag{3.15}$$

By minimising the support of the flux mode, Equation 3.15 ensures that it is elementary. Equations 3.14 and 3.15 must be solved for the forward and backward fluxes of the query reaction; the more optimal solution is returned by the method.

Other functionalities implemented in this class include calculation of a flux mode performing a given net conversion (see Chapter 7), flux essentiality analysis (see Chapter 8) and flux coupling analysis [11].

Classes `ConvLp` and `EnvLp` These classes are used for the calculation of minimal net conversions and minimal substrate and product compositions (see Chapter 7).

3.4 Integrated metabolic reconstruction

The package `Bio` is intended to provide a unified interface to a metabolic model, the data resources used for its construction, and the analysis methods used for its interrogation; we further refer to this interface as an integrated metabolic reconstruction (reconstruction for short).

3.4.1 Design

A possible way to attach additional functionalities to an object is inheritance: a subclass of `Model` could be created, implementing analysis methods and access to databases. This would be not a flexible structure, however, because the choice of functionalities and their implementation would be predefined statically. Instead, we used the design pattern Decorator (Wrapper), which enables adding new functionalities to an existing object dynamically, by 'enclosing' it into another object, called a *wrapper*. For instance, a metabolic model can be enclosed in an object which implements LP tasks and delegates other requests to the model itself. The model and its wrapper provide a single unified interface; on the other hand, the model can be still accessed directly. Moreover, a wrapper can be 'dismounted' and another wrapper can be attached instead; hence, the set of functionalities provided by the interface can be changed dynamically. A wrapper, in turn, can be enclosed in another wrapper, which

enriches the interface with a new flavour of functionalities. The package `Bio` includes a number of wrapper classes for ScrumPy metabolic models, intended for construction, editing, analysis, database integration, storage and optimisation of metabolic reconstructions. In an integrated metabolic reconstruction, these wrappers form a multilayered structure, where each layer handles the requests for which it is responsible and delegates the rest to the underlying layer. Some of the requests are delegated further to other objects, such as databases and linear programs; thus, the whole reconstruction provides a unified interface to a set of lower-level interfaces (design pattern Facade). Since the selection of layers in a reconstruction can be defined dynamically, its initialisation is decoupled from the interface and implemented in separate methods (design pattern Factory method).

Another feature of integrated metabolic reconstructions is their composite structure. Although a metabolic model is typically considered as a set of biochemical reactions interconverting metabolites, in genome-scale reconstructions, the information about enzymes and genes is often crucial. Therefore, we developed separate classes representing genes, enzymes, reactions and metabolites. Although these classes are designed as aggregatee elements of a reconstruction, their objects do not exist during the whole lifetime of the latter. Instead, they are created on demand, when the information about a given element is requested by its unique identifier (design pattern Proxy). On the other hand, a reconstruction delegates some of the methods to the element classes (design pattern Composite).

3.4.2 Generic classes

Class `Wrapper` Implements the basic functionality of a generic wrapper, which provides a unified interface with an underlying object of an arbitrary class. The reference to this object is stored in the attribute `Sublevel`, which is by default initialised as a dictionary. The method `__getattr__` is called when a given attribute or method (denoted `attr`) is requested, but not found in the wrapper; it looks up for `attr` in the sublevel:

```
def __getattr__(self, attr):
    return getattr(self.Sublevel, attr)
```

Similarly, the method `__setattr__` assigns a new value `val` to the attribute `attr` found in the sublevel, if it cannot be found in the wrapper:

```
def __setattr__(self, attr, val):
    if not attr in self.__dict__:
        setattr(self.Sublevel, attr, val)
    else:
        self.__dict__[attr] = val
```

Due to these methods, a client can access and edit the content of the sublevel, as if it belonged to the wrapper itself. Since the default sublevel is a dictionary,

the latter serves as the 'core' of the multilayered systems of wrappers, described below.

Class `System` A generic wrapper for a metabolic model. Implements basic methods for creating and accessing elements, copying its content and extracting a subsystem by a list of element IDs.

Class `Element` A generic element of a reconstruction. Contains a reference to its aggregator `System` object, in which it has a unique ID.

Each of the modules described below defines own subclasses of the classes `System` and `Element` under the same names, and subclasses `Gene`, `Enzyme`, `Reaction` and `Metabolite` of the own class `Element`. While a `System` object `system1` encloses the `System` object `system2`, each element of `system1` encloses the homonymous (e.g. having the same ID) element of `system2`. Hence, a reconstruction is designed as a multilayered composite structure (see Figure 3.5).

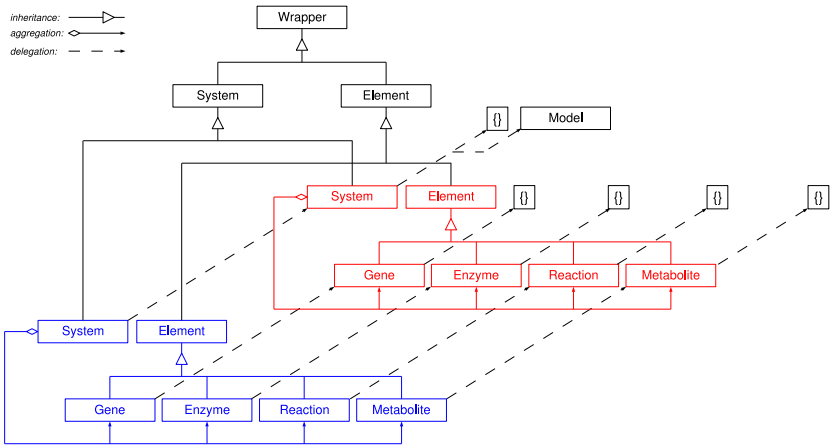
3.4.3 Internal structure and access methods

Module `Access` Defines the innermost layer of a reconstruction, stores its internal structure (relations between elements) and implements access methods. The `System` object encloses two sublevels: a dictionary and a `Model` object, which by default contains empty stoichiometry matrices. In addition, the `System` object itself contains two logical binary matrices: the *catalysis matrix* $\mathbf{C}$, representing the relation between reactions and the catalysing enzymes, and the *annotation matrix* $\mathbf{A}$, relating enzymes to the encoding genes (see Figure 3.6b, c). The advantage of representing relations in the form of matrices is the possibility of applying matrix operations, including multiplication. For instance, the product $\mathbf{C} \cdot \mathbf{A}$ relates reactions to genes, while in the product $\hat{\mathbf{N}} \cdot \mathbf{C}$, each column represents the net stoichiometry of a given enzyme (see Figure 3.6e, f).

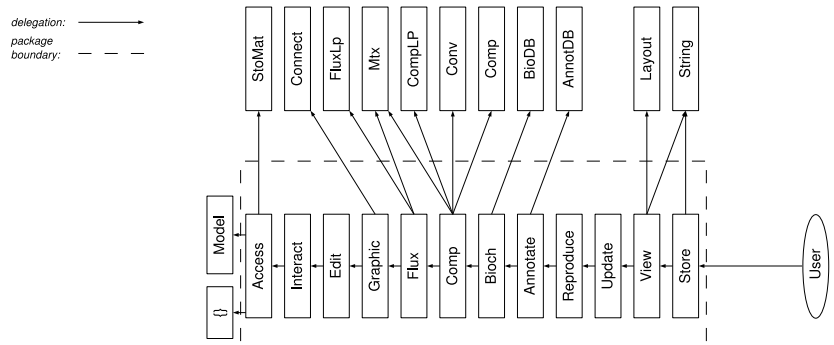
The `__getattr__` and `__setattr__` methods in the class `System` are overridden, so that any attribute or method is firstly looked up in the object itself, then in the underlying model and finally, in the underlying dictionary. The latter stores the element IDs as keys, enabling each element in a reconstruction to be accessed by its ID using the dictionary operator `[]`, e.g:

```
>>> gene = system['g1']
```

(in fact, the element object is created once this operator is applied). Each element has a reference to its 'parent's' matrix (where the elements of its class are associated with the rows) and 'children's' matrix (where they are associated with the columns). E.g. for an enzyme these are the annotation matrix and the catalysis matrix, respectively; the 'parent's' matrices of genes and the 'children's' matrices of metabolites are empty. By looking up in these matrices, the elements present the information about their hierarchical relationships. For instance, the method `GetChildren` returns the list of lower-level elements in the



a



b

Figure 3.5: The multilayered composite structure of an integrated metabolic reconstruction.
a) Inheritance and composition in two layers. **Inheritance:** The generic classes `Wrapper, System` and `Element` are on the top of the inheritance hierarchy. Each module defines subclasses of `System` and `Element` under the same names, and four subclasses of the own class `Element`. **Aggregation:** Each layer is composed of objects of classes defined in one of the modules. The `System` object is aggregated with `Metabolite, Reaction, Enzyme` and `Gene` objects, which are created on demand. **Delegation:** The objects of the outer layer (blue) enclose the homonymous objects of the inner layer, which in turn enclose dictionaries (denoted {}). The inner `System` object, in addition, encloses a `Model` object.
b) Default layers and delegation of requests. The layers are shown in the bottom row; each layer performs the tasks it is responsible for and delegates the rest to the inner layer (leftward arrow) or to other classes and modules (upward arrows).

hierarchy, i.e. enzymes encoded by a gene, reactions catalysed by an enzyme or metabolites used by a reaction. The names of the methods `GetParents`, `GetSiblings`, `GetPartners` and `GetExclusiveChildren` are self-explaining:

```
>>> print system['g1'].GetChildren()
['e1', 'e2']
>>> print system['e2'].GetParents()
['g1', 'g2']
>>> print system['e1'].GetSiblings()
['e2']
>>> print system['g1'].GetPartners()
['g2']
>>> print system['g1'].GetExclusiveChildren()
['e1']
```

The data describing an element object (e.g. the set of 'children') are collected from the matrices and other sources during its initialisation and stored in its underlying dictionary. They can be accessed by field names, similarly to fields in a database record, e.g:

```
>>> print gene['children']
['e1', 'e2']
```

Module `Interact` Provides the interactivity of a reconstruction, by sending messages to users and receiving requests from them. The `System` object defines the address to which the messages are sent. By default, the output goes to the Python console, but it can be redirected to files or shown in message boxes in a graphical user interface. The method `ErrorMsg` is called when an error occurs in an outer layer; it prints a report about the error in the output. A user can send requests to the reconstruction via the console or dialogue boxes. File dialogue boxes are used for opening and saving data in files.

3.4.4 Editing methods

Module `Edit` Serves for editing an existing reconstruction. The element classes in this module are accessible directly from the interface of the package `Bio` and can be initialised on their own, outside of a reconstruction. Then they can be included into a reconstruction under given IDs, e.g:

```
>>> system['m'] = Bio.Metabolite(external = False)
```

In this example, a metabolite object is created and included into the reconstruction (named `system`) under the ID 'm' (an empty row is included into the external stoichiometry matrix). The existing elements can be edited, e.g a metabolite can be declared external:

```
>>> system['m'].SetExternal()
```

```
>>> system = Bio.System()
>>> system['g1'] = Bio.Gene(['e1', 'e2'])
>>> system['g2'] = Bio.Gene(['e2', 'e3'])
>>> system['e1'].SetChildren(['r1', 'r2'])
>>> system['r1'].SetStoich({'a' : -1, 'b': 1})
>>> system['r2'].SetStoich({'b' : -1, 'c': 1})
>>> system['r3'] = Bio.Reaction({'c' : -1, 'd' : 1}, irrev = True)
>>> system['e2'].SetChildren(['r2'])
>>> system['e3'].SetChildren(['r3'])
>>> system['tx_A'] = Bio.Transporter('a', 'x_a', source = True)
>>> system['tx_D'] = Bio.Transporter('d', 'x_d', source = False)
>>> system.Save('example.xspy')
```

a

$$\mathbf{A} = \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \end{array} \begin{array}{c} \begin{array}{cc} g_1 & g_2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \end{array}$$

b

$$\mathbf{C} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ tx\_A \\ tx\_D \end{array} \begin{array}{c} \begin{array}{ccc} e_1 & e_2 & e_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

c

$$\hat{\mathbf{N}} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ x\_a \\ x\_d \end{array} \begin{array}{c} \begin{array}{ccccc} r_1 & r_2 & r_3 & tx\_A & tx\_D \end{array} \\ \begin{pmatrix} -1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

d

$$\mathbf{C} \cdot \mathbf{A} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ tx\_A \\ tx\_D \end{array} \begin{array}{c} \begin{array}{cc} g_1 & g_2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array}$$

e

$$\hat{\mathbf{N}} \cdot \mathbf{C} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ x\_a \\ x\_d \end{array} \begin{array}{c} \begin{array}{ccc} e_1 & e_2 & e_3 \end{array} \\ \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

f

Figure 3.6: A sequence of commands (a) for initialisation, editing and storing of an integrated metabolic reconstruction; its annotation (b), catalysis (c) and external stoichiometry (d) matrices; the relation between reactions and genes (d) (note that g_1 encodes two enzymes catalysing r_2); net stoichiometries of enzymes (e).

Further examples of creating and editing elements are shown in Figure 3.6a. Note that a reaction can be initialised with a given stoichiometry (represented as a dictionary), while a gene or an enzyme can be initialised with a given list of children. The class `Transporter` is a subclass of the class `Reaction`, which is declared only in this module and intended for including transport reactions for given metabolites.

By default, an element is deleted with its exclusive children. E.g. in the following example, the reaction tx_A is deleted with the metabolite x_a, which is not used by any other reaction.

```
>>> del system['tx_A']
```

An element can be also copied from one reconstruction to another with all of its descendents, using the method `Paste` (by analogy with copying and pasting in various programs), e.g:

```
>>> system['g1'].Paste(other_system)
```

Here the gene g_1 is copied into the reconstruction `other_system` with its descendant enzymes, reactions and metabolites; whereby the hierarchical relations and stoichiometries are retained.

Two elements in the same reconstruction can be 'merged', by replacing the corresponding pairs of rows and columns in the matrices by their sums. E.g. in the following example, the reactions r_1 and r_3 are replaced by one reaction with their net stoichiometry, named r_3 and catalysed by the enzymes which had previously catalysed any of them:

```
>>> system['r1'].MergeWith('r3')
>>> print system['r3'].GetEquation()
 a +  c ->  b +  d
>>> print system['r3'].GetParents()
['e1', 'e3']
```

This operation is used for handling redundancies in networks, such as synonyms and isostoichiometric reactions, i.e. those with identical stoichiometries.

Some editing operations are applicable to a whole reconstruction, such as updating the current stoichiometry, catalysis or annotation matrix with the content of another matrix and updating the reconstruction with the content of another reconstruction.

Module `Update` Ensures that the internal structure of a reconstruction is correctly updated after the completion of an editing request. For instance, if reaction stoichiometry is changed in the external stoichiometry matrix, the same change must be applied to the internal stoichiometry matrix. The reason for decoupling the editing and updating processes is that the latter is relatively time-consuming, while editing operations are often applied in a sequence or a loop. Hence, instead of updating the reconstruction after each operation

executed in lower levels, it is updated only once, when the whole editing task requested by the client is completed.

The updating routine includes the detection of the pairs of isostoichiometric reactions, to which the method `MergeWith` is applied. Hence, the software ensures that no isostoichiometric reactions can exist simultaneously in a model[4].

3.4.5 Analysis methods

The modules described below provide interfaces tor the analysis methods implemented in the packages `Utils` and `LP`, and implement some higher-level methods.

Module `Graphic` Provides an interface to graph-theoretical methods. Implements a method for the detection of isostoichiometric reactions.

Module `Flux` Provides an interface to methods of flux balance analysis and essentiality analysis (see Chapter 8).

Module `Comp` Provides an interface to methods of mass balance analysis and a range of original methods, including the detection of stoichiometric inconsistencies (see Chaper 5), elementary net conversions and elementary substrate and product compositions (see Chapter 7).

Most of the methods implemented in these classes are time-critical with the computational time depending on the size of the network. As an optional preprocessing step, these methods apply the 'Core' algorithm to the corresponding stoichiometry matrix in order to reduce its size.

3.4.6 Integration with databases

Apart from providing interfaces to databases, the modules described below enable automatic generation of reconstructions, by importing data from external resources.

Module `Integrate` Defines generic classes for integrating a reconstruction with a database. The `System` object contains a reference to a unique database with which it is associated. The design of the whole reconstruction implies that it contains one layer associated with a reconstruction of a given type, which is defined by a special key (e.g. 'Bioch' for biochemical and 'Anno' for annotative). The layers use these keys to recognise the requests addressing their databases.

Each element has a reference to a unique record in the database. The method `LoadRecords` receives a list of IDs and imports the data from the corresponding records into the reconstruction, creating new elements. The method `LoadSource` imports all records, creating a whole-database reconstruction.

[4]Here we mean the reactions with identical, rather than any proportional stoichiometries.

Module `Bioch` Interface to a biochemical database, whose records can be associated with enzyme, reaction and metabolite elements. In the following example, two elements are imported from the KEGG database:

```
>>> system.LoadRecords(['1.1.1.1', 'R00001'], 'Bioch')
```

Note that the IDs belong to records of different types: R00001 is a reaction and 1.1.1.1 is an enzyme. The latter is imported with the reactions which it catalyses according to KEGG, while all reactions are imported with the corresponding metabolites, retaining the stoichiometries.

Module `Annotate` Interface to an annotative database, whose records are associated with the genes. Genes can be imported, e.g:

```
>>> system.LoadRecords(['SAG0001', 'SAG0002'], 'Anno')
```

Besides the genes, the enzymes encoded are imported from the annotative database, and the reactions catalysed and the metabolites interconverted are imported from the biochemical database. Hence, the following command being applied to an empty reconstruction creates a complete metabolic model based on a given annotation:

```
>>> system.LoadSource('Anno')
```

3.4.7 Visualisation and storage

Module `View` Implements methods for representing a reconstruction or its parts in the form of a text string, a ScrumPy file or a bipartite graph image. The text representations of an element includes the ID followed by a name and then by dictionary fields in separate lines, e.g:

```
>>> print system['C00001']
C00001: H2O
        external: False
```

For a reaction, the name is replaced with an equation written in a human-readable form:

```
>>> print system['R01600']
R01600:  ATP +  Glucose <>  D-Glucose 6-phosphate +  ADP
        irrev: False
        equation:  C00002 +  C00293 <>  C00092 +  C00008
```

A reaction can be also shown in the ScrumPy format:

```
>>> print system['R01600'].ToSpy()
R01600:
        C00002 +  C00293 <>  C00092 +  C00008
        ~
```

The text and ScrumPy representations of a reconstruction are assembled from the corresponding representations of its elements.

The generation of graph images is delegated to the module `Layout`. An example is shown in Figure 3.7b.

Module `Store` Since the original ScrumPy is not sufficient for a complete description of a metabolic reconstruction, an extended ScrumPy format (XSPY) was developed (see Figure 3.7a). A file in this format consists of four parts: The first part represents a dictionary of databases; once a file is loaded, these databases are initialised and linked to the reconstruction. The second and third parts describe the relations between genes and enzymes and between enzymes and reactions, respectively. Note that each line in the first three parts starts with the '#' symbol and is therefore ignored by the ScrumPy parser. The fourth part is the model representation in the original ScrumPy format.

The module provides reading and writing methods for the XSPY format, as well as for comma-separated text files, which may contain lists of external metabolites, irreversible reactions and other supplementary data.

3.4.8 Optimised annotation

The following modules implement the stochastic optimisation algorithms used for optimised genome annotation (see Chapter 6).

Module `Reproduce` Defines and implements the mutation operators.

Module `Optim` Defines an optional layer, which invokes the objective function. The latter can be provided by the user as an argument or reimplemented in subclasses. The module also defines classes and interface methods for genetic algorithms and simulated annealing.

3.4.9 Initialisation

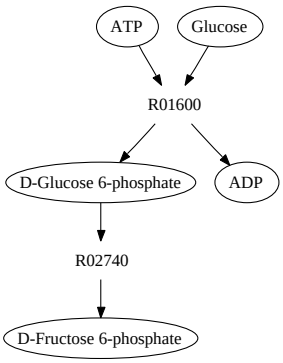
Since a metabolic reconstruction is an ensemble of objects of variable classes, it is preferable to decouple its initialisation from its structure and to implement it in factory methods. The default factory method creates the layers and encloses them into each other in the order shown in Figure 3.5b. The method is accessible in the interface of the package `Bio`; optional arguments define the input XSPY file, the output address and the associated databases, e.g:

```
>>> system = Bio.System('example.xspy', output = 'term',
                Bioch = 'KEGG', Anno = 'PRIAM')
```

This command initialises the reconstruction described in the file 'example.xspy', directs the output to the console, intialises the KEGG database (if not already existing) and an empty PRIAM-based annotative database and associates them with the reconstruction. If the database names are not supplied as arguments,

```
########################################
#SOURCES:

#Anno: PRIAM
#Bioch: KEGG

########################################
#GENES/ENZYMES:

#SAG0040 2.7.1.2
#SAG0402 5.3.1.9
#None
########################################
#ENZYMES/REACTIONS:

#2.7.1.2 R01600
#5.3.1.9 R02740
#None
########################################
#REACTIONS/METABOLITES:

Structural()

R01600:
        C00002 +  C00293 <>  C00092 +  C00008
        ~
R02740:
        C00092 <>  C00085
        ~
```

a



b

Figure 3.7: A reconstruction containing two genes, two enzymes and two reactions, represented in the extended ScrumPy format (a) and in a graph image (b).

they are set to the definitions in the input file. The selection and order of layers can be easily modified by overriding the factory method.

3.5 Discussion and conclusions

In the current chapter, the development of the software packages `Utils`, `Source`, `LP` and `Bio` was described. Although these packages were designed and can be used separately, their entirety represents a relatively complete and self-contained framework for the construction, validation and analysis of genome-scale metabolic reconstructions.

The package `Bio` provides the top-level interface of this framework, which is referred in this chapter as integrated metabolic reconstruction. This interface encapsulates the structure of a metabolic reconstruction, most of the methods required for the work with it and references to related data resources. Note that none of the underlying objects can be considered as a top-level or 'master' object. In fact, the user directly accesses the outermost layer of a reconstruction, which has no preferential control over the rest of the system. The information flow between the layers (see Figure 3.5b) is to an extent similar to the mass flow in a biochemical pathway, where the control over the system behaviour is shared between all steps. Such 'egalitarian', bottom-up approach to modular design provides strong flexibility and extensibility: the number of layers providing different functionalities is unrestricted and the mechanism of their creation and inclusion is relatively simple. However, from the user's perspective, the whole reconstruction appears and acts as a single, integral object.

The reusability of the software was achieved by the encapsulation of lower-level functionalities in packages, modules and classes. In particular, the class `Database` provides universal functionalities for data storage, while the package `LP` and the KEGG interface can be used as standalone tools for linear programming and biochemical data interrogation, respectively.

Efficiency was not among the primary objectives of software development, since most of the low-level, time-critical computations were delegated to ScrumPy and GLPK. Nevertheless, the high-level design solutions proved to be useful for the improvement of efficiency. For instance, the application of the 'Core' algorithm as a preprocessing step significantly accelerated the computational time in analysis methods. Further, the utilisation of the design patterns Singleton and Proxy strongly reduced the consumption of memory.

One of the most considerable outcomes of the development of high-level functionalities was the increase in software usability and modeller's productivity. As an example, we demonstrate the automatic generation of a genome-scale reconstruction based on a given annotation, using two Python statements:

```
>>> system = Bio.System(Bioch = 'KEGG', Anno = 'PRIAM')
>>> system.LoadAnnotation('annotation.out')
```

The time required for the execution of these statements does not exceed several minutes. Clearly, the development of a valid, ready-to-work model requires the

application of a range of further methods; most of which are also automatised. These methods are described in the following chapters.

Chapter 4

Model Construction

As shown in Figure 2.1, the construction of a model is an iterative process, and its definition typically depends on the intermediate analysis results. In this chapter, we describe the part of the construction process which can be completed at the first iteration, assuming that the methods used at this stage do not require the knowledge of any systemic properties of a model.

4.1 Methods

Although the methods described below were developed particularly for the metabolic reconstruction of *S. agalactiae*, they are intended to be applicable (possibly after minor modifications) to the construction of a genome-scale model of an arbitrary bacterium using a sequenced genome as the primary source of data. The genome annotation can be obtained from a database or generated *de novo*; biochemical data are obtained from the KEGG LIGAND database. The import of annotative and biochemical data has been completely automatised.

4.1.1 Curation of biochemical data

Methods have been developed for the detection of certain types of errors during this process, thus preventing potential input errors in the model.

Empirical formulae By parsing the content of the field 'FORMULA' in a KEGG COMPOUND record, the empirical formula of the corresponding compound is identified and represented as a dictionary mapping elements to stoichiometric coefficients. However, in some records this field is missing, so the resulting dictionary is empty. Further, some empirical formulae cannot be parsed unambiguously; e.g. '(1C12H20O10)n' for starch (where 'n' denotes the polymerisation degree) and 'HOR' for the generic compound 'Alcohol' (where 'R' denotes a radical). In such cases, the parser returns an empty dictionary, and

the compound is qualified as generic, if the field is missing or contains the substring 'R'[1] and as a polymer, otherwise (see Table 4.1a).

Atomic balance The net balances of elements in a reaction can be calculated using the empirical formulae of reactants and represented similarly to a reaction stoichiometry. For instance, the atomic balance of the reaction R02713: $C_6H_9NO_4 \leftrightarrow C_6H_6O_5$ is:

$$\{H : -3,\ C : 0,\ O : 1,\ N : -1\}$$

This result implies that 3 atoms of hydrogen and one atom of nitrogen are lost in the reaction, while one atom of oxygen is produced in excess. We subdivide all reactions into three distinct categories:

- balanced: the net balances of all atoms are equal to zero;
- unbalanced: at least one net balance is non-zero;
- undeterminable: the atomic balances cannot be calculated because the reaction involves one or more metabolites with missing or unparseable empirical formulae.

Table 4.1b demonstrates that reactions with undeterminable atomic balance comprise a considerable proportion (more than the quarter) in the KEGG database. Table 4.1c shows that hydrogen is the most frequently unbalanced element in KEGG reactions. The violations are often caused by skipping H^+ in the reactions involving such metabolites as NAD^+, $NADP^+$ and HCO_3^-. The empirically developed Algorithm 1 attempts to correct these violations.

Standardisation of metabolite names Inconsistencies in the naming of metabolites in KEGG LIGAND are largely caused by the following two factors:

- the same metabolites specified in different reactions as a compound or a glycan (e.g. C00369 and G10545, respectively for starch);
- the same metabolites specified with different levels of abstraction (e.g. 'α-D-glucose', 'D-glucose' or 'glucose');

The records in the COMPOUND and GLYCAN subdatabases contain fields named 'GLYCAN' and 'COMPOUND', respectively, which enable the detection of correspondences between them. The inconsistencies of the second type are resolved as follows: the metabolites with identical Canonical SMILES strings are detected and grouped; hence, each group includes optical isomers (see Table 4.2 a, b). These groups are used as an input for an algorithm generating a semantic

[1]Note that the character 'R' is not contained in the names of any elements involved in KEGG (see Table 4.1b). The metabolites C00342 (thioredoxin), C00343 (oxidised thioredoxin), C04261 (protein N(pi)-phospho-L-histidine) and C00615 (protein histidine) are not qualified as generic, altough their empirical formulae contain the substring 'R'. The glycans with missing formulae are not qualified as generic.

Table 4.1: Categories of empirical formulae (a), reactions (b) and elements (c) in the KEGG LIGAND database (release 29). For each category, the number and percentage is shown. For each element, the number of reactions violating its balance and of all reactions involving is shown.

	num.	%
parseable	11058	77.71
generic	926	6.5
polymers	181	1.27
missing	2064	14.5

a

category	num.	%
balanced	4002	58.77
unbalanced	1059	15.55
undeterminable	1749	25.69
involving polymers	281	4.13
involving generics	216	3.17

b

el.	unbalanced	involved
H	941	4988
O	333	4953
C	157	4968
N	63	3789
P	34	2830
S	13	986
Cl	11	164
Mn	0	1
Ni	0	1
Hg	0	1
As	0	2
F	0	4
I	0	7
Mg	0	19
Se	0	22
Br	0	23
Fe	0	26
Co	0	28

b

Algorithm 1 Correct the coefficient of H^+ in the stoichiometry *sto*.

```
balance := balance_hydrogen(sto)
if balance ≠ 0 then
  coef := 0
  if sto[H2O] = 0 then
    if sto[NAD+] = −[NADH] then
      coef := sto[NADH]
    else if sto[NADP+] = −[NADPH] then
      coef := sto[NADPH]
    end if
  end if
  if sto[HCO3−] ≠ 0 then
    coef := sto[HCO3−]
  end if
  //save the correction if it reduces the absolute net balance:
  if coef ≠ 0 then
    test_sto := copy(sto)
    test_sto[H+] := coef
    test_balance := balance_hydrogen(test_sto)
    if abs(test_balance) < abs(balance) then
      sto[H+] := coef
    end if
  end if
end if
```

Table 4.2: Ilustration of the methods used for the standardisation of metabolite names: a) Canonical SMILES strings enable distinguishing optical isomers (α-D-glucose and β-D-glucose) in a group of metabolites with an identical atomic composition. b) A relation associating canonical SMILES strings with groups of isomers. c) A relation associating the most generic metabolite definitions to the other definitions in the groups (D-glutamine is excluded).

	FORMULA	Canonical SMILES
α-D-glucose	$C_6H_{12}O_6$	C(C1C(C(C(C(O1)O)O)O)O)O
β-D-glucose	$C_6H_{12}O_6$	C(C1C(C(C(C(O1)O)O)O)O)O
fructose	$C_6H_{12}O_6$	C(C1C(C(C(O1)(CO)O)O)O)O

a

C(C1C(C(C(C(O1)O)O)O)O)O	glucose, D-glucose, α-D-glucose, β-D-glucose
C(CC(=O)N)C(C(=O)O)N	glutamine, D-glutamine, L-glutamine

b

glucose	D-glucose, α-D-glucose, β-D-glucose
glutamine	L-glutamine

c

hierarchy of metabolites (Table 4.2c). The algorithm compares the names of the members of each group, assuming that a more generic name (e.g. 'glucose') is contained in the tail of a more specific name (e.g. 'D-glucose'). The algorithm is apparently not absolutely sensitive, but it appears to be fully specific (no false-positives have been found by manual check of the output). The resulting hierarchy is represented as a relation associating each metabolite with its most generic definition. D-forms of the amino acids are not included in the hierarchy, since they are metabolically distinct from the L-forms.

To avoid naming inconsistencies, the following rules are applied to all metabolites imported from the KEGG LIGAND database:

- if a metabolite is defined as a glycan (i.e. its identifier starts with 'G') but can be alternatively defined as a compound, then the identifier of the corresponding compound is used, e.g. C00369 instead of G10545.
- if a metabolite is included in the semantic hierarchy, then the identifier of the most generic definition is used, e.g. C00293 (glucose) instead of C00267 (α-D-glucose).

The identifiers defined by these rules are further referred as standardised identifiers.

4.1.2 Irreversible reactions

The following books have been used as a data source for the definition of reaction irreversibilities: 'Biochemical pathways: an atlas of biochemistry and molecular biology' (Gerhard Michal, 1999 [70]) and 'Biochemistry' (Berg *et al.*, 2002 [44]). The enzymatic reactions indicated in the diagrams by unilateral arrows were defined as irreversible (see Table 4.3).

In addition, the following groups of reactions are defined as irreversible based on hypothetical assumptions:

- reactions catalysed by the enzymes of the class 'hydrolases',
- reactions consuming nucleoside triphosphates, except for the following subgroups:
  - the known ATP producers, catalysed by phosphoglycerate kinase, acetate kinase and aspartate kinase;
  - reactions transferring the phosphate groups between different nucleoside triphosphates, e.g. ATP + GDP $\leftrightarrow$ ADP + GTP

It has been found that a number of irreversible reactions in KEGG LIGAND are not defined according to their actual direction; these reactions need to be inverted (see Table 4.4).

4.1.3 Hypothetical reactions

Since the automatically imported biochemical data describe solely the enzymatic reactions involved in small molecule metabolism, a range of important metabolic functions need to be represented by manually defined hypothetical reactions. Here we describe a possible set of hypothetical reactions to be included in a genome-scale model based on the KEGG database.

Biosynthetic pathways Protein, RNA, DNA, plasma membrane and cell wall are defined as the biomass components to be covered by a model. The synthesis of these products is represented in the form of generic reactions, which are assumed to approximate the actual net stoichiometries of biosynthetic pathways. These reactions are defined as irreversible, thus imposing the growth condition (i.e. non-negative net production of biomass). All reactions are listed in table 4.5. The reactions `syn_PROTEIN`, `syn_RNA` and `syn_DNA` produce the hypothetical generic metabolites representing protein, RNA and DNA, respectively from their natural precursors. The stoichiometry of the reaction `syn_FATTY_ACID` corresponds to the net stoichiometry of palmitate synthesis [44]; hence, the hypothetical metabolite `FATTY_ACID` has the composition of palmitate. The further reactions represent the synthesis of diacylglycerol, phosphatidate, phosphatidylserine, phosphatydilethanolamine, phosphatydilglycerol and cardiolipin. The reaction `syn_MEMBRANE` represents the synthesis of plasma membrane from its major components. The synthesis of peptidoglycan is represented by two successive reactions. The KEGG identifier C05894 denotes the

Table 4.3: The reactions catalysed by the enzymes shown below were defined as irreversible based on a search in [70] (upper part) and [44] (lower part).

EC number	name
4.1.3.27:	anthranilate synthase
2.1.1.45:	thymidylate synthase
2.3.3.10:	hydroxymethylglutaryl-CoA synthase
2.7.6.1:	ribose-phosphate diphosphokinase
2.7.1.11:	6-phosphofructokinase
2.4.2.14:	amidophosphoribosyltransferase
2.7.1.15:	ribokinase
2.7.1.30:	glycerol kinase
2.4.1.1:	phosphorylase
1.1.1.133:	dTDP-4-dehydrorhamnose reductase
2.3.1.54:	formate C-acetyltransferase
2.3.1.30:	serine O-acetyltransferase
2.4.1.18:	1,4-alpha-glucan branching enzyme
3.2.1.1:	alpha-amylase
3.1.3.11:	fructose-bisphosphatase
2.5.1.6:	methionine adenosyltransferase
1.2.1.10:	acetaldehyde dehydrogenase (acetylating)
4.1.1.5:	acetolactate decarboxylase
2.7.1.40:	pyruvate kinase
2.1.3.2:	aspartate carbamoyltransferase
2.5.1.9:	riboflavin synthase
4.2.3.5:	chorismate synthase
4.2.3.1:	threonine synthase
2.2.1.6:	acetolactate synthase
4.1.2.5:	threonine aldolase
1.5.1.2:	pyrroline-5-carboxylate reductase
4.1.1.31:	phosphoenolpyruvate carboxylase
2.7.1.2:	glucokinase
2.7.1.11:	6-phosphofructokinase
6.3.4.5:	argininosuccinate synthase
2.7.1.30:	glycerol kinase
6.3.1.2:	glutamate-ammonia ligase
6.4.1.2:	acetyl-CoA carboxylase
2.7.1.40:	pyruvate kinase
3.1.3.11:	fructose-bisphosphatase
2.1.3.3:	ornithine carbamoyltransferase

Table 4.4: The reactions catalysed by the enzymes shown in the upper part and the reactions shown in the lower part were iverted in the model, in accordance with their proper directions.

EC number	name
2.2.1.6:	acetolactate synthase
1.5.1.2:	pyrroline-5-carboxylate reductase
4.1.1.31:	phosphoenolpyruvate carboxylase
1.2.1.10:	acetaldehyde dehydrogenase (acetylating)
3.5.2.3:	dihydroorotase
2.7.1.40:	pyruvate kinase
2.3.1.54:	formate C-acetyltransferase
2.4.2.14:	amidophosphoribosyltransferase
3.5.4.10:	IMP cyclohydrolase
R00177:	ATP + H_2O + methionine → pyrophosphate + orthophosphate + S-adenosyl-L-methionine
R02750:	deoxyribose + ATP → 2-deoxy-D-ribose 5-phosphate + ADP

compound Undecaprenyl-diphospho-N-acetylmuramoyl-(N-acetylglucosamine)-L-alanyl-D-isoglutaminyl-L-lysyl-D-alanyl-D-alanine. The full biomass is represented by a single external hypothetical metabolite.

Expenditure reactions The following reactions are intended to represent the consumption of currency metabolites in the catabolic processes not covered by the model:

ATPase:	ATP + H_2O → ADP + Phosphate
NADH_Oxidase:	NADH → NAD^+ + H^+
NADPH_Oxidase:	NADPH → $NADP^+$ + H^+

Spontaneous reactions The reaction `Carboxyanhydrase`: $HCO3^-$ + H^+ ↔ CO_2 + H_2O is included.

Transporters Reversible transporters are included for some metabolites which can freely pass through the cell membrane, namely water, ammonia, CO_2 and oxygen.

4.2 Application to genome-scale models

The methods described above were used to generate models of *S. agalactiae* strains `sag`, `sak`, `san` and `coh`[2]. Two lines of models were constructed using different data sources for the determination of the sets of metabolic enzymes. The

[2]These identifiers denote the strain names and are used further in the thesis, see Table 4.6a.

Table 4.5: Hypothetical biosynthesis reactions. Hypothetical metabolites are denoted by uppercase names. Non-hypothetical metabolites are represented in the actual model by their KEGG identifiers.

protein, nucleic acids:
syn_PROTEIN:
Glycine + Proline + Leucine + Tyrosine + Asparagine + Tryptophan + Aspartate + Phenylalanine + Threonine + Isoleucine + Alanine + Methionine + Histidine + Cysteine + Lysine + Glutamate + Glutamine + Arginine + Serine + Valine → PROTEIN
syn_RNA:
GTP + CTP + ATP + UTP → 4 Pyrophosphate + xRNA
syn_DNA:
dCTP + dTTP + dGTP + dATP → 4 Pyrophosphate + xDNA
lipids, membrane:
syn_FATTY_ACID:
7 Malonyl-CoA + Acetyl-CoA + 20 H^+ + 14 NADPH → 8 CoA + 7 CO_2 + 14 $NADP^+$ + 6 H_2O + FATTY_ACID
syn_PHOSPHATIDATE:
2 FATTY_ACID + sn-Glycerol 3-phosphate → PHOSPHATIDATE
syn_CDP_DIACYLGLYCEROL:
PHOSPHATIDATE + CTP → Pyrophosphate + CDP_DIACYLGLYCEROL
syn_PHOSPHATIDYL_SERINE:
Serine + CDP_DIACYLGLYCEROL → CMP + PHOSPHATIDYL_SERINE
syn_PHOSPHATIDYL_ETHANOLAMINE:
PHOSPHATIDYL_SERINE → CO_2 + PHOSPHATIDYL_ETHANOLAMINE
syn_PHOSPHATIDYL_GLYCEROL:
sn-Glycerol 3-phosphate + CDP_DIACYLGLYCEROL → CMP + Orthophosphate + PHOSPHATIDYL_GLYCEROL
syn_CARDIOLIPIN:
2 PHOSPHATIDYL_GLYCEROL → Glycerol + CARDIOLIPIN
syn_MEMBRANE:
PHOSPHATIDATE + PHOSPHATIDYL_SERINE + PHOSPHATIDYL_ETHANOLAMINE + CARDIOLIPIN → MEMBRANE
peptidoglycan
syn_UDC_P:
FATTY_ACID + Orthophosphate → Undecaprenyl phosphate
syn_PG:
Serine + Alanine + 2 C05894 → D-Alanine + PEPTIDOGLYCAN
biomass
syn_BIOMASS:
PROTEIN + xRNA + xDNA + MEMBRANE + PEPTIDOGLYCAN → BIOMASS

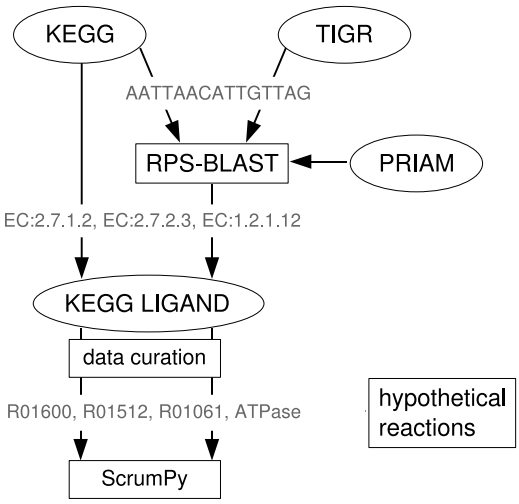


Figure 4.1: Construction of two lines of models. First line (left): the enzyme sets were obtained from the annotations available in KEGG. Second line (right): the genome sequences with predicted ORFs were obtained from KEGG and TIGR; the enzyme sets were predicted using RPS-BLAST and PRIAM.

first line was based on genome annotations obtained from the KEGG database and included models of the strains `sag`, `sak` and `san`. The second line was based on in-house annotations of all four strains. The whole process is outlined in Figure 4.1.

4.2.1 Data selection and collection

Genome sequences The complete genomes of the strains `sag`, `sak` and `san` are available in the GenBank† and KEGG databases. Other sources of complete genomes include the TIGR database for `sag` and `sak` and the organism-specific SagaList† database for `san`. The genome files were obtained in the FASTA format, with predicted open reading frames. Since the KEGG genome files were found to cover more ORFs (see Table 4.6b), they were selected as the primary data source for the second-line models of `sag`, `sak` and `san` (the files were released on January 27, 2009). In addition, the incomplete genome of the strain `coh` was retrieved from the TIGR database (released on January 18, 2006) and used as a data source for a second-line model.

Annotations Genome annotations of the strains `sag`, `sak` and `san` were obtained from the KEGG database and used as input data for the first-line models. The annotations are represented in the form of flat files containing gene identifiers followed by comma-separated EC numbers (the text representation of the class `Relation` conforms with this format, see Chapter 3). Table 4.6 shows

Table 4.6: a) Organism identifiers for the modelled strains of *S. agalactiae*(the first three are the KEGG identifiers). b) The numbers of ORFs in the genome files obtained from different data sources. c) The numbers of ORFs annotated and EC numbers predicted in the functional annotations obtained from KEGG. d) Numbers of records in the KEGG LIGAND releases 29 (used in the current work) and 49 (January 1, 2009). e) The content of the PRIAM database; the third line indicates the EC numbers in KEGG LIGAND (release 49) without profiles in PRIAM.

ID	Strain	Serotype
sag	2603	V
sak	A909	Ia
san	NEM316	III
coh	COH1	III

a

Database	sag	san	sak	coh
GenBank	2124	1996	2134	
KEGG	2276	2136	2235	
TIGR	2169	2034		2475
SagaList			2134	

b

	sag	sak	san
ORFs total	566	595	576
encoding multiple EC numbers	27	49	27
EC numbers total	412	446	418
encoded by multiple genes	87	94	90
incompletely qualified	43	57	43

c

Release	29	49
Total records	36644	39172
Compounds and glycans	25180	26311
Reactions	6810	7819
Enzymes	4654	5042

d

EC numbers	2099
Profiles	2706
EC numbers missing	2943

e

that these annotations represent many-to-many relations between genes and enzymes. Some EC numbers in the annotations are not completely qualified and contain one or more missing fields denoted by hyphens, e.g. `1.1.1.-`.

Biochemical resources KEGG LIGAND database was also selected as the source of biochemical data for the translation of the enzyme sets into the sets of catalysed reactions. Release 29.0 (January 1, 2004) was used for the current work. The local interface to KEGG LIGAND is described in Chapter 3.

4.2.2 Implementation and results

Figure 4.2 shows an example Python code producing the first and second-line models of a single strain. The first two lines demonstrate the generation of the *de novo* annotation. The method `Run` of the class `PriamDB` executes RPS-BLAST with the arguments indicating the input genome file, the threshold E-value (set to 10^{-3}) and the output file, respectively.

The further operations are applied to integrated metabolic reconstructions. The methods `OpenAnnoRel` and `LoadAnnotation` are used to import annotations from a file obtained from KEGG and from an RPS-BLAST output file, respectively; the biochemical data from KEGG are imported in parallel. The argument `rule` of the method `LoadAnnotation` specifies the rule defining which EC numbers are selected from the lists of hits and assigned to the ORFs. The rule 'best' implies that the most significant hit is assigned to each ORF. If multiple best hits are predicted with equal E-values (such as the first two hits in Figure 2.8a), all of them are assigned, thus resulting in the presence of genes associated with multiple enzymes in the model.

By default, Algorithm 1 is applied to all reactions imported from KEGG LIGAND. The reactions violating the balance of any element other than hydrogen and those involving polymers or generic metabolites are not included in the models. The treatment of isostoichiometric reactions is described in Chapter 3.

The lists of EC numbers catalysing reactions to be defined as irreversible, reversible or to be inverted and the list of the metabolites for which transporters need to be included are defined in the supplementary text files, in a comma-separated format. These lists are imported by the methods `LoadIrrevs`, `LoadToInvert` and `LoadEnv`, respectively. The ScrumPy function `Complement` calculates the complement of the sets of reactions consuming and producing nucleoside triphosphates, e.g. those reactions only consuming, but not producing NTPs. These reactions and the ones catalysed by hydrolases are made irreversible using the method `SetIrrevs`. The same method called with the second optional argument set to False, makes the given reactions reversible. The biosynthetic, spontaneous and expenditure reactions are defined in separate ScrumPy model files and included into the reconstructions using the method `AddModel`.

The implementation described above was used for the construction of the models of all strains. Table 4.7 shows the quantitative characteristics of the resulting models.

```
##annotation
Priam = Source.GetDB('PRIAM')
Priam.Run('genome.seq', 1e-3, 'genome.out')

##first model:
system1 = Bio.System(Bioch = 'KEGG')
system1.OpenAnnoRel('sag_enzyme.list')

##second model:
system2 = Bio.System(Bioch = 'KEGG', Anno = 'PRIAM')
system2.LoadAnnotation('genome.out', rule = 'best')

##both models in a loop:
for system in [system1, system2]:
    ##irreversibles:
    system.LoadIrrevs('irrevs.irr', True)

    ##hypothetical irreversibles
    NTPs = system.GetNTPs()
    NTP_consumers = system.GetConsumers(NTPs)
    NTP_producers = system.GetProducers(NTPs)
    NTP_irrevs = Complement(NTP_consumers, NTP_producers)
    hydrolases = system.SelectByClass('Hydrolases')
    system.SetIrrevs(NTP_irrevs + hydrolases)

    ##reversibles (phosphoglycerate, acetate and aspartate kinases):
    system.LoadIrrevs('revers_NTP_consumers.rev', False)

    ##define proper directions:
    system.LoadToInvert('invert.inv')

    ##biosynthesis reactions:
    system.AddModel('growth.spy')

    ##expenditure and spontaneous reactions:
    system.AddModel('hypo.spy')

    ##transporters:
    system.LoadEnv('transmembrane.env')

##saving the models:
system1.Save('1.xspy')
system2.Save('2.xspy')
```

Figure 4.2: Construction of the first and second-line models of a single strain.

Table 4.7: The numbers of the elements in the models of the first and second line. The ∩ symbol denotes the intersections of the sets (the intersections cannot be calculated for the genes, since their identifiers are strain-dependent).

	1-st line				2-nd line				
Strain	sag	sak	san	∩	sag	sak	san	coh	∩
Genes	566	595	576	NA	1355	1345	1419	1489	NA
Enzymes	369	389	375	366	608	604	619	599	559
Reactions	498	521	502	494	851	854	841	845	787
Metabolites	488	514	492	484	759	760	735	752	687

4.3 Discussion

The decisions made at the first stage of model construction, such as the selection of data sources, the treatment of inconsistent data and the inclusion of hypotheses are crucial for the outcome of the modelling process, since minor inaccuracies at this stage tend to give rise to serious errors in the analysis results. Therefore, dubious cases were handled conservatively and potentially problematic data were excluded from the input of the models. The motivations of some specific solutions are explained below.

The data curation methods described in this chapter are based on a trade-off between the completeness and specificity of data on the one hand and their consistency on the other hand. Potentially relevant information is lost because of the exclusion of unbalanced reactions and polymers. The usage of standardised metabolite identifiers leads to the loss of distinction between optical isomers. However, due to these solutions, the risk of a range of serious errors is reduced, including stoichiometric inconsistencies and network gaps.

The irreversibilities and proper directions of reactions are the most deficient type of input data. On the model construction stage, we were not aware of any comprehensive and publicly available source of data describing thermodynamic constraints in biochemical reactions[3]. Therefore, literature search was performed as the only method of data collection. Since the visual inspection of biochemical diagrams is laborious, the search was limited to the sections describing carbohydrate metabolism, fermentation and synthesis of amino acid and nucleotides.

The hypothetical reactions included into the models are intended for the the solution of certain practical problems, rather than for the complete description of the metabolic networks. Only five major biosynthetic functions are covered, aiming for the prediction of potential drug targets. The transporters for most of the substrates are not included at this stage, assuming that different sets of transporters could represent variable environmental conditions.

[3]While writing the thesis, we learned about the group contribution method for thermodynamic analysis [68, 45]. Unfortunately, because of the lack of time, this method could not be used for the definition of irreversible reactions.

The KEGG annotations of *S. agalactiae* strains contain some incomplete EC numbers, such as 5.-.-.- and 1.6.4.-. Records corresponding to some of these numbers are available in KEGG LIGAND (e.g. for 25 out of 43 in the `sag` annotation). However, none of these records contained a name and it was not clear, whether they represented single enzymes or broader groups. Therefore, it was decided not to include the incomplete EC numbers into the first-line models.

Since RPS-BLAST reports multiple hits with different levels of significance for most of the ORFs, the resulting annotations do not deliver an ultimate solution to the problem of assigning enzymatic functions to genes. The authors of PRIAM recommend to detect the best non-overlapping hits in order to detect potential multienzymes. We used a simpler approach, assigning the most significant hits. These gene annotations, however, were assumed to be subject to further revisions, depending on the results of model analysis.

The other optional decision was the choice of the threshold E-value for RPS-BLAST, for which the value of 10^{-10} was recommended by the authors of PRIAM. However, we applied the less restrictive threshold of 10^{-3} in order to generate more encompassing models and to reduce them later, using the results of constructive interrogation.

The databases KEGG LIGAND, BioCyc and BRENDA were considered as potential primary sources of biochemical data. The selection of the former was motivated by the following reasons:

- Availability in the form of flat files.
- Usage of uniform identifiers, which facilitates the data management and reduces the susceptibility to typographic errors.
- A previous analysis made in our group demonstrated that the models of six organisms based on KEGG LIGAND covered more reactions than those based on BioCyc, while the numbers of errors were comparable [84].

The relatively old release 29.0 (January 1, 2004) was used for the current work. Despite the bigger sizes of the more up-to-date releases (see Table 4.6d), they were not used because of the following technical problems:

- Absence of empirical formulae for some important metabolites, such as glucose (C00293).
- Impossibility of tracing mutual correspondences between glycan and compound records (the corresponding database fields were removed).

It must be noted, however, that the completeness and correctness of the underlying databases PRIAM and KEGG LIGAND is one of the major limiting factors of the quality of the models constructed. This issue is discussed in detail in the following chapter.

Chapter 5

Constructive Interrogation

This chapter describes the detection and correction of structural errors, which arise in models as a result of input errors included at the construction stage (see Figure 2.9). In addition to the definitions of the classes of errors, we introduce the concept of *quality indicators*, used to evaluate the general structural quality of a model, i.e. its consistency with physical and biochemical constraints. Following the classification of structural analysis methods into three major branches (graph-theoretical, mass-balance and flux-balance analysis), we distinguish topological, flux and stoichiometric consistency of a model, concerning its ability to comply with the three fundamental constraints: network connectedness, mass conservation and steady state.

5.1 Topological consistency

The methods described below assume that a network is represented as a bipartite graph. The sets of reactions and metabolites are the disjoint sets of nodes; a reaction node and a metabolite node are connected if the given reaction uses the given metabolite. The graph is not directed (i.e. all reactions are assumed to be reversible).

Connected components are identified by a graph traversal algorithm based on a depth-first search [14] (see Algorithms 2 and 3). The number of connected components can be used to measure the degree of network disconnectedness. However, in our experience, genome-scale reconstructions typically comprise one large component (containing above 90% of all metabolites) and multiple small ones [84]. Therefore, we consider the percentage of metabolites in the biggest component as a more informative indicator of model quality. For instance, in the network shown in Figure 2.7e, the number of components is 2 and the percentage is $\frac{2}{3} \approx 66.6\%$.

Orphan metaboles are detected by a simple inspection of the rows of the internal stoichiometry matrix. A model can be evaluated more precisely using the 'Core' algorithm, which iteratively detects and removes the orphan metabolites

Algorithm 2 Identify the set C of connected components in a network with an external stoichiometry matrix $\hat{\mathbf{N}}$.

$C := \emptyset$
$U := metabolites(\hat{\mathbf{N}}) \cup reactions(\hat{\mathbf{N}})$ //initial set of nodes
while $U \neq \emptyset$ **do**
  $a := select(U)$ //any element
  $S := \emptyset$ //the current component
  $Visit(a, S, \hat{\mathbf{N}})$ //Alg. 3
  $U := U \setminus S$
  $C := C \cup \{S\}$
end while

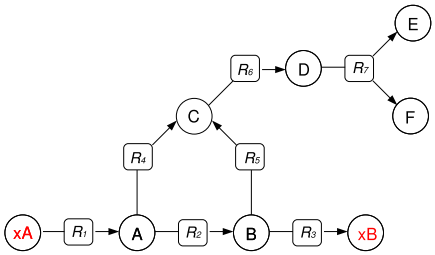
Algorithm 3 Recursive subroutine used for the detection of connected components. $\hat{\mathbf{N}}$ is an external stoichiometry matrix, S is a set, a is a reaction or metabolite; the function $InvolvedWith$ returns the list of metabolites used by a if it is a reaction or the list of reactions using a if it is a metabolite.

def $Visit(a, S, \hat{\mathbf{N}})$:
  $S := S \cup \{a\}$
  for all $b \in InvolvedWith(\hat{\mathbf{N}}, a)$ **do**
    $Visit(b, S)$
  end for

from the internal stoichiometry matrix, thus extracting the *core matrix* (see Algorithm 4 and Figure 5.1). Note that none of the reactions removed in the course of the algorithm is able to carry a steady state flux. The algorithm can be used in order to reduce a stoichiometry matrix before applying more time-consuming methods of linear programming or null space analysis; it can be also applied to an external stoichiometry matrix for mass-balance analysis. For these purposes, the algorithm is used as an optional preprocessing step in the analysis methods provided by the integrated metabolic reconstruction classes (see Chapter 3).

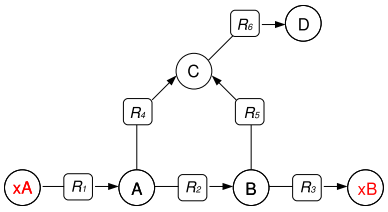
A dead-end metabolite can be considered as a structural error only if it is and orphan or if it is used by irreversible reactions only (e.g. C in Figure 5.1c). Otherwise, its concentration can be balanced if one of the reversible reactions using it can operate in the backward direction. However, this may be impossible, if this reaction is involved in a reaction subset with an irreversible reaction (see Section 5.3 for more details).

We consider the percentages of non-orphan and non-dead-end metabolites in the total set of metabolites and the percentage of the reactions in the core matrix (further referred to as *core reactions*) in the total set of reactions as indicators of model quality.



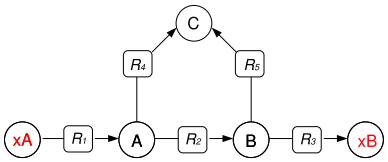
orphans: {E, F}
remove: $\{R_7\}$

a



orphans: {D}
remove: $\{R_6\}$

b



orphans: $\emptyset$
remove: $\emptyset$

c

Figure 5.1: Iterations of the 'core' algorithm. Initially, the network contains two orphan metabolites (a). In the second iteration, one more orphan is detected and removed (b). In the third iteration, no orphans are detected and the algorithm terminates. Note that the external metabolites (coloured red) are not considered as orphans

Algorithm 4 Given an internal stoichiometry matrix $\mathbf{N}$, identify the core matrix $\acute{\mathbf{N}}$.

$\acute{\mathbf{N}} := copy(\mathbf{N})$
$O := orphans(\acute{\mathbf{N}})$
while $O \neq \emptyset$ **do**
  for all $m \in O$ **do**
    for all $r \in InvolvedWith(\acute{\mathbf{N}}, m)$ **do**
      $del\_col(\acute{\mathbf{N}}, r)$ //delete the column
    end for
  end for
  $del\_zero\_rows(\acute{\mathbf{N}})$ //delete zero rows
  $O := orphans(\acute{\mathbf{N}})$
end while

5.2 Stoichiometric consistency

The concept of stoichiometric consistency was introduced in the recent publication of our group: 'Detection of stoichiometric inconsistencies in biomolecular models' [33] (see Appendix A). Any natural metabolic system fulfills two fundamental physical constraints: positivity of molecular masses of all metabolites and mass conservation in all reactions. The conflicts between these constraints in metabolic models must be considered as structural errors; an example is shown below:

$$\begin{array}{l} R_1\text{: A} \leftrightarrow \text{B} \\ R_2\text{: A} \leftrightarrow \text{B + C} \end{array} \tag{5.1}$$

Under the assumption of mass conservation it is impossible to assign any positive molecular mass to C; the only admissible mass is zero. The presence of the error does not depend on the actual chemical species, denoted A, B and C; it only depends on the reaction stoichiometries. Therefore, we term such errors *stoichiometric inconsistencies*. A metabolic network is *stoichiometrically consistent* if all metabolites can be assigned some positive molecular masses without violating mass conservation, and *stoichiometrically inconsistent* otherwise.

Although in this work we consider only structural models, the concept of stoichiometric consistency is also applicable to kinetic models of metabolism and, more generally, to any systems of mass-conserving stoichiometric interconversions. The typical causes of stoichiometric inconsistencies are summarised below:

- atomically unbalanced reactions,
- generic metabolites referring to multiple substances (e.g. primary alcohol),
- polymers with variable molecular compositions,
- metabolites not referring to actual substances (in conceptual models) .

In a model where each metabolite represents a unique chemical substance, stoichiometric inconsistencies are always caused by the presence of atomically unbalanced reactions. Unfortunately, the atomic balances are often undeterminable (see Table 4.1a, b). Here we propose a more reliable and universal solution of the problem.

Verification of consistency We associate each metabolite in a network with a numerical value specifying its molecular mass; these values comprise a column vector $\mathbf{m}$ of dimension m. Molecular masses of natural substances are positive:

$$\mathbf{m} > \mathbf{0} \tag{5.2}$$

The total conservation of mass is described by the equation $\hat{\mathbf{N}}^T\mathbf{m} = \mathbf{0}$ (Eq. 2.7). We refer to any solution of Equation 2.7 as a *mass conservation vector*. A network is called *consistent* if it has at least one positive mass conservation vector and *inconsistent*, otherwise. Equations 5.2 and 2.7 define the following LP problem:

$$\begin{array}{rl} \text{Minimise} & \sum_{i=1}^{m} m_i \\ \text{Subject to} & \hat{\mathbf{N}}^T\mathbf{m} = \mathbf{0} \\ \text{Where} & m_i \geq 1:\ 1 \leq i \leq m \end{array} \tag{5.3}$$

The program is solvable if the system is consistent. The third line ensures that the molecular masses are positive (since strict inequalities are not valid in linear programming, the expression > 0 must be replaced by $\geq \alpha$ where α is an arbitrary positive number).

5.2.1 Inconsistent net stoichiometries

The net stoichiometry of any linear combination of reactions is in the column space of $\hat{\mathbf{N}}$, according to Equation 2.13. By combining this equation with Equation 2.7, we obtain the following equality, which is true for an any net stoichiometry $\dot{\mathbf{c}}$ and any mass conservation vector $\mathbf{m}$:

$$\dot{\mathbf{c}}^T\mathbf{m} = 0 \tag{5.4}$$

The following proposition and corollary state an important property of inconsistent networks:

Proposition 1 ([16]) *An equation $\dot{\mathbf{c}}^T\mathbf{m} = 0$ has no positive solution $\mathbf{m}$ iff $\dot{\mathbf{c}}$ is either semipositive or seminegative.*

Proof *Necessary condition*: Let us assume that $\dot{\mathbf{c}}$ is neither semipositive nor seminegative. If $\dot{\mathbf{c}} = \mathbf{0}$ then the equation is solvable with arbitrary $\mathbf{m}$. Otherwise, let us denote the positive components of $\dot{\mathbf{c}}$ by $\dot{c}_i,\ i = i_1, ..., i_P$ and the negative ones by $\dot{c}_j,\ j = j_1, ..., j_N;\ P + N = m$. Then the given equation can be written in the form:

$$\sum_{i=i_1}^{i_P} \dot{c}_i m_i = -\sum_{j=j_1}^{j_N} \dot{c}_j m_j$$

and a positive solution is obtained by taking

$$m_i = -\sum_{j=j_1}^{j_N} \dot{c}_j : \quad i = i_1, ..., i_P$$
$$m_j = \sum_{i=i_1}^{i_P} \dot{c}_i : \quad j = j_1, ..., j_N \tag{5.5}$$

Sufficient condition: Let us assume that $\mathbf{m} > \mathbf{0}$. If $\dot{\mathbf{c}} \geq \mathbf{0}$ then $\dot{\mathbf{c}}^T\mathbf{m} > 0$ and if $\dot{\mathbf{c}} \leq \mathbf{0}$ then $\dot{\mathbf{c}}^T\mathbf{m} < 0$ $\square$

We term a net stoichiometry *inconsistent*, if it is either semipositive or seminegative.

Corollary 1.1 *A network is inconsistent, iff it has at least one inconsistent net stoichiometry.*

A metabolite set is *conservable*, if its elements can be assigned positive molecular masses simultaneously in some mass conservation vector and *unconservable* otherwise (e.g. $\{A, C\}$ in Equation 5.1). Since only the terms with zero coefficients can be excluded from an equation $\dot{\mathbf{c}}^T\mathbf{m} = 0$ without changing its solutions set, according to Proposition 1, the metabolites with non-zero coefficients in an inconsistent net stoichiometry comprise an unconservable set. The full metabolite set of an inconsistent network is unconservable, but it may contain conservable subsets (e.g. $\{A, B\}$ in Equation 5.1).

An unconservable set is *minimal* if all of its proper subsets are conservable (e.g. $\{C\}$ in Equation 5.1 and $\{A', B', C'\}$ in Figure 5.2). An inconsistent net stoichiometry is *minimal* if its support is a minimal unconservable set. The following corollary is implied by the criterion of solvability of the equation $\dot{\mathbf{c}}^T\mathbf{m} = 0$ with a positive $\dot{\mathbf{c}}$ according to Proposition 1:

Corollary 1.2 *If $\mathcal{M}$ is a minimal unconserved metabolite set, and $|\mathcal{M}| = 1$, then the metabolite has zero mass. If $|\mathcal{M}| > 1$, then all metabolites in the mass conservation vector have zero mass, or if any have a positive mass, then at least one metabolite has a negative mass.*

Let us consider a semipositive mass conservation vector. Corollary 1.2 implies that in this vector, the components corresponding to the elements of minimal unconservable sets can be only equal to zero. Since each of the remaining metabolites can be assigned a positive molecular mass in at least one mass conservation vector, there must exist one in which all metabolites not contained in minimal unconservable sets are assigned positive molecular masses. We refer to such a vector as a *maximal conservation vector*. The fact that it is non-negative helps to prove the following proposition and corollary.

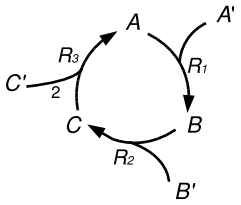
Proposition 2 *A metabolite $met \in \mathcal{M}$ is unconserved iff it is an element of at least one minimal unconservable set $\mathcal{M}' \subseteq \mathcal{M}$.*

$R_1 : A + A' \leftrightarrow B$
$R_2 : B + B' \leftrightarrow C$
$R_2 : C + 2\ C' \leftrightarrow A$

a

A	A'	B	B'	C	C'
0	1	0	1	0	2

b



c

Figure 5.2: a) A system with a minimal unconservable set $\{A', B',\ C'\}$. b) An inconsistent minimal net stoichiometry c) Graph representation: A', B' and C' enter the reaction cycle, but nothing leaves it. If these metabolites would have positive molecular masses and the cycle would carry an internal flux, it would create mass from nothing or convert it into nothing, depending on the reaction directions. Hence, the total mass of the closed system would change by the masses of the metabolites weighted by the coefficients specified in (b).

Proof *Necessary condition:* If *met* is not an element of any minimal unconservable set, then the corresponding component in the maximal conservation vector of $\mathcal{N}$ is positive and *met* is not unconserved. *Sufficient condition:* According to Corollary 1.2, $met \in \mathcal{M}'$ cannot obtain a positive value in any non-negative conservation vector, therefore, *met* is unconserved. □

Corollary 2.1 *A metabolite is unconserved iff its mass is not positive in a maximal conservation vector.*

Hence, a network is inconsistent if it contains some inconsistent net stoichiometries, and a net stoichiometry is inconsistent iff it involves unconserved metabolites. We consider the numbers of unconserved metabolites and minimal inconsistent net stoichiometries as indicators of model quality.

5.2.2 Leakage modes

The localisation of stoichiometric inconsistencies requires the detection of inconsistent subnetworks. In the simplest case, an inconsistent network consists of only one reaction; such a reaction is also called *inconsistent*. An example is shown below:

$$A \leftrightarrow A + B \tag{5.6}$$

Proposition 1 implies that the reaction is inconsistent if it has either an empty set of substrates ($\dot{\mathbf{c}} \geq \mathbf{0}$) or an empty set of products ($\dot{\mathbf{c}} \leq \mathbf{0}$), but not both ($\dot{\mathbf{c}} = \mathbf{0}$). Metabolites occuring on both sides with equal coefficients are for most purposes represented by zeroes in a stoichiometry matrix, since their net concentration change is zero. Therefore, the reaction shown in Equation 5.6 is equivalent to $\emptyset \leftrightarrow$ B and hence inconsistent.

The more complex network shown below contains no inconsistent reactions. However, by subtracting the stoichiometry of R_2 from that of R_1 we obtain the inconsistent net stoichiometry $\emptyset \leftrightarrow C$. Further, the linear combinations $(R_1, -R_3)^T$ and $(R_1, -R_2, -R_3)^T$ result in the net stoichiometries $\emptyset \leftrightarrow B$ and $A \leftrightarrow \emptyset$, respectively.

$$\begin{aligned} R_1 &: A \leftrightarrow B + C \\ R_2 &: A \leftrightarrow B \\ R_3 &: A \leftrightarrow C \end{aligned} \tag{5.7}$$

Likewise, in the network shown in Figure 5.2a, the sum of all three reaction stoichiometries is $A' + B' + 2\ C' \leftrightarrow \emptyset$.

The solutions of the system $\hat{\mathbf{N}}\mathbf{v} = \dot{\mathbf{c}}$ for an inconsistent $\dot{\mathbf{c}}$ are termed *leakage modes* and those with minimal supports are termed *elementary*, e.g the aforementioned linear combinations in Equation 5.7 and Figure 5.2. In contrast to the closely related concept of flux modes, the reaction directions and reversibilities in leakage modes are irrelevant. But similarly to elementary flux modes, elementary leakage modes represent minimal subnetworks able to 'leak' (i.e. to produce in a steady state without consuming anything or vice versa) certain metabolite compositions. Therefore, they can be defined as precise locations of stoichiometric inconsistencies and their identification is helpful for the detection of input errors.

The ability of leakage modes to carry (invalid) steady state fluxes in a closed network implies that the reactions comprising them are organised in cycles. These cycles have either of the properties described below:

a) Some metabolites enter the cycle, but nothing leaves it, or vice versa (see Equations 5.1 and 5.7 and Figure 5.2). So, after each turnover, the total mass of the system increases or decreases, assuming that the entering or leaving metabolites have positive molecular masses. Therefore, these metabolites comprise a minimal unconservable set. Elementary leakage modes of this type can be detected using elementary modes analysis, given that the entering or leaving metabolites are externalised. In contrast, the algorithms proposed in this section do not depend on the selection of external metabolites.

b) Inconsistently defined stoichiometric coefficients. So, considering these coefficients as the edge weights in a graph, the product of all edge weights in a cycle is greater or less than one, thus increasing or decreasing the total mass of the system. A simple example is shown below:

$$\begin{aligned} R_1 &: A \leftrightarrow B \\ R_2 &: A \leftrightarrow 2B \end{aligned} \tag{5.8}$$

Interestingly enough, the reactions involved in such a cycle may comprise more than one elementary leakage mode with different net stoichiometries, depending

on the coefficients. So, in Equation 5.8, the leakage modes $(R_2, -R_1)^T$ and $(R_2, -2R_1)^T$ have the net stoichiometries $\emptyset \leftrightarrow B$ and $\emptyset \leftrightarrow A$.

5.2.3 Detection of stoichiometric inconsistencies

Detection of unbalanced metabolites According to Corollary 2.1, conserved and unconserved metabolites can be distinguished in a maximal conservation vector, which can be calculated by means of the following MILP:

$$\begin{array}{rl} \text{Maximise} & \sum_{i=1}^{m} k_i \\ \text{Subject to} & \hat{\mathbf{N}}^T \mathbf{m} = \mathbf{0} \\ \text{Where} & 0 \leq k_i \leq m_i,\ k_i \in \{0,1\} :\ 1 \leq i \leq m \end{array} \tag{5.9}$$

The number of positive components of $\mathbf{m}$ is maximised. The third line ensures that each component k_i of the vector of integers $\mathbf{k}$ can be set to 1 if $m_i \geq 1$ and is 0 otherwise. Since no upper bound is defined for the components of $\mathbf{m}$, the non-zero components are increased until all of them become greater than or equal to one. At this point, the number of unit components of $\mathbf{k}$ reaches the possible maximum, thus satisfying the termination criterion.

Detection of minimal inconsistent net stoichiometries Since any solution of Equation 2.7 can be found in the left null space, Equation 5.4 has the same solution set as the system shown below:

$$\mathbf{K}^T \dot{\mathbf{c}} = \mathbf{0} \tag{5.10}$$

where $\mathbf{K}$ is a left null space matrix. Each solution of this system is some vector in the column space of $\hat{\mathbf{N}}$, i.e. a net stoichiometry. If the left null space is empty, then the system has the trivial solution only. Hence, the only conservable molecular mass of each metabolite is zero and the network is inconsistent. Otherwise, each inconsistent minimal net stoichiometry is an undecomposable semipositive or seminegative solution of Equation 5.10. As a special case, a singleton minimal unconservable set is always represented by an all-zero row in the left nullspace matrix. Further, a minimal inconsistent net stoichiometry involving a given unconserved metabolite with a non-zero coefficient can be found using the following MILP:

$$\begin{array}{rl} \text{Minimise} & \sum_{i=1}^{m} k_i \\ \text{Subject to} & \mathbf{K}^T \dot{\mathbf{c}} = \mathbf{0}, \\ & \dot{c}_j \geq \epsilon \\ \text{Where} & 0 \leq \dot{c}_i \leq k_i,\ k_i \in \{0,1\} :\ 1 \leq i \leq m \end{array} \tag{5.11}$$

The program minimises the number of positive components in the vector $\dot{\mathbf{c}}$, thus ensuring that its support is minimal. The third line states that the mass of the metabolite corresponding to the j-th row is positive in the solution. However, the same metabolite can be an element of more than one minimal unconservable set (given that none of them is singleton). In order to find the next solution,

the one already found is excluded from the further iterations using an integer cut. The program must be invoked iteratively until no more solutions can be found (see Algorithm 5).

Algorithm 5 Identify the set Y of inconsistent minimal net stoichiometries in a stoichiometry matrix $\hat{\mathbf{N}}$

```
Y := {}
U := unconserved_metabolites(N̂^T)    /* Eq. 5.9 */
K := nullspace_mtx(N̂^T)
for all met ∈ U : do
  if K_met = 0 then
    ċ := [0, ..., 0]
    ċ_met := 1    /* all-zero row → singleton set*/
    Y := Y ∪ {ċ}
  else
    prog := mixed_integer_program(K)
    ċ, k := minimal_solution(prog, met)    /* Eq. 5.11 */
    while is_feasible(prog) do
      Y := Y ∪ {ċ}
      set_integer_cut(prog, k)    /* Eq. 3.7 */
      ċ, k := minimal_solution(prog, met)
    end while
  end if
end for
```

Detection of elementary leakage modes A given inconsistent minimal net stoichiometry $\dot{\mathbf{c}}$ can be augmented at the right side of the stoichiometry matrix. This is equivalent to including an additional inconsistent reaction into the network; this reaction may be also considered as a transport reaction in an open metabolic system and will be further referred as the transporter (Figure 5.3, b). In terms of metabolic modelling, the transporter delivers the molecules which are then lost in the leakage modes, thus satisfying the steady state condition and enabling the calculation of elementary flux modes in the system $(\hat{\mathbf{N}}|-\dot{\mathbf{c}})(\mathbf{v}|1) = \mathbf{0}$, considering all reactions as reversible. The elementary flux modes involving the transporter have an inconsistent net stoichiometry and the corresponding leakage modes in the original network can be obtained from them by removing the transporter (Figure 5.3, c). The procedure described above must be repeated for each inconsistent minimal net stoichiometry.

As an alternative solution for large models, we propose instead of a complete set of elementary modes to calculate only a spanning set, which comprises the columns of a nullspace matrix of the system $(\hat{\mathbf{N}}| - \dot{\mathbf{c}})(\mathbf{v}|1) = \mathbf{0}$, which can be obtained by Gauss-Jordan elimination. The inspection of this matrix enables the detection of at least some, if not all of the input errors causing the leakage of a given metabolite set (Figure 5.3, d). After the error correction, the matrix

$R_1 : A \leftrightarrow B$
$R_2 : A \leftrightarrow B + X$
$R_3 : P \leftrightarrow Q$
$R_4 : P \leftrightarrow Q + X$

a

	R_1	R_2	R_3	R_4	T_X
A	-1	-1	0	0	0
B	1	1	0	0	0
P	0	0	-1	-1	0
Q	0	0	1	1	0
X	0	1	0	1	1

b

	em_1	em_2	em_3
R_1	-1	-1	0
R_2	1	1	0
R_3	1	0	-1
R_4	-1	0	1
T_X	0	-1	-1

c

	em_1	em_2
R_1	-1	-1
R_2	1	1
R_3	1	0
R_4	-1	0
T_X	0	-1

d

Figure 5.3: a) A system with a minimal unconservable set $\{X\}$. b) The stoichiometry matrix with the augmented inconsistent minimal net stoichiometry under the column name T_X, which represents the transport reaction $\emptyset \leftrightarrow X$. c) Elementary flux modes: em_1 is an internal cycle, whereas em_2 and em_3 involve the transporter T_X and correspond to the elementary leakage modes $(-R_1, R_2)^T$ and $(-R_3, R_4)^T$. d) The right nullspace matrix of the augmented matrix: one out of two elementary leakage modes appears here.

can be recalculated to detect the remaining errors.

Individual elementary leakage modes with a given net stoichiometry can be obtained using linear programming methods (see Chapter 7).

5.2.4 Practical applications

We proposed an algorithm detecting the complete set of unconserved metabolites in a network using a MILP. Although the complexity of the latter is much higher than the complexity of the linear program used in the algorithm proposed by [73], the MILP needs to be invoked only once for the whole model. Thus the full computation takes a significantly shorter time. The detection of unconserved metabolites enables a simple method for resolving all inconsistencies in a network, namely by removing the corresponding rows from the stoichiometry matrices. This solution may be acceptable if the concentrations and conversions of unconserved metabolites are not relevant for the investigated problem, e.g. for water. However, in most cases, more sophisticated solutions are needed, for which the identification of unconserved metabolites is a necessary step.

A more precise characterisation of inconsistent networks is possible by detection of minimal inconsistent net stoichiometries. Although Algorithm 5 solves the non-polynomial problem of finding a complete set of minimal solutions in a system of linear equations, it appears to be feasible even in large genome-scale models due to a) the relatively small size of the left nullspace matrix, compared to a whole network and b) usually small numbers of minimal inconsistent net stoichiometries involving a given metabolite.

In contrast, the calculation of a complete set of elementary leakage modes is subject to combinatorial explosion and is currently not feasible in genome scale models. On the other hand, the total number of elementary leakage modes can be very large. So, in a model of a potato tuber (constructed manually in our group) containing 30 reactions and 31 metabolites, 7 out of them unconserved, we found 123 elementary leakage modes, all caused by a single input error. Such amounts of data are hardly analysable and apparently superfluous. Therefore, we propose the calculation and analysis of a limited population of elementary leakage modes using Gauss-Jordan elimination. This result can be used in two ways: inspection of individual modes and of those reactions which are involved in large percentages of the calculated modes.

The smallest possible leakage mode with a given net stoichiometry can be identified using MILP. The advantage of this method is the maximally precise localisation of input errors, the disadvantage is the high computational cost of using integer constraints and especially integer cuts in large models. Hence, the next mode should be calculated after the error in an already found one is corrected.

Notably the stoichiometric consistency of a network does not depend on the quality of the underlying annotation, neither on the thermodynamic constraints and is usually defined by the quality of the biochemical data used for the model construction. The algorithms presented in this section are applicable to complete biochemical databases and can be used to detect errors in them in order

to prevent their propagation into the models.

5.3 Flux consistency

Live reactions We assume that a realistic model must be able to operate at a steady state. A reaction is termed *live*, if it can carry a steady state flux and *dead*, otherwise. The concept of dead reactions is often used interchangeably with that of strictly detailed balanced reactions, which are represented by zero rows in a null space matrix. Under this assumption, the definition of dead reactions does not depend on the irreversibilities and directions. Therefore, we term such reactions *stoichiometrically dead*, to distinguish them from *thermodynamically* dead reactions, which cannot carry a steady state flux with respect to the irreversibility constraint. In a general case, the stoichiometrically dead reactions comprise a subset of the thermodynamically dead ones; in a reversible network, both sets are equal. Further in the thesis, the terms dead and live refer to thermodynamic deadness and liveness, unless otherwise stated.

The liveness of an individual reaction can be identified by solving the linear program shown in Equation 3.12 for its forward and backward directions. However, solving this LP for each reaction in a network may be costly. In most cases, the computational time can be considerably reduced by calculating the core matrix of the network and identifying the live reactions within it, since the remaining reactions are definitely dead.

The detection of live reactions can be further accelerated by maximising the number of reactions carrying a positive flux in a solution, using an MILP approach, similarly to the detection of unconserved metabolites. However, this approach does not guarantee the detection of all live reactions in one go, since some reaction flux rates may be negative (unlike metabolite masses). For instance, in the network shown in Figure 5.4, the maximal positive solution involves the reactions R_1, R_2 and R_4, although R_2 is also live. Clearly, after identifying the liveness of as many reactions as possible by means of MILP, the remaining ones can be tested using Equation 3.12. This can be done by iteratively invoking the following MILP:

$$\begin{array}{rl} \text{Maximise} & k_i, i \in T \\ \text{Subject to} & \ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0}, \\ & {}^{-}\mathbf{v}^{irr} = \mathbf{0}, \\ \text{Where} & 0 \leq k_i \leq \ddot{v}_i,\ 1 \leq i \leq 2n \end{array} \tag{5.12}$$

where T is the set of reactions whose fluxe rates are to be maximised. In the initial iteration, T is the complete set of reactions; in each further iteration, it is reduced to those reactions whose liveness is still uknown. Once the MILP returns an empty solution (i.e. no more positive solutions can be found), the loop terminates and the remaining reactions are tested individually.

Although we consider a dead reaction as a structural error, its presence does not necessarily indicate an input error in the annotative or biochemical data, since it depends on the definition of external metabolites. For instance, the

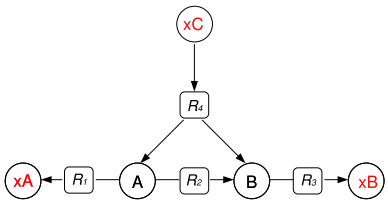


Figure 5.4:

reactions R_4, R_5, R_6 and R_7 in Figure 5.1a are dead, but would become live after declaring E and F external. Another typical condition is the inclusion of an enzyme catalysing multiple reactions, some of which are live in the given network while others are dead. We define an enzyme as *working* if it catalyses at least one live reaction and *unemployed*, otherwise. Similarly, a gene is working if it encodes at least one working enzyme and unemployed, otherwise. We assume that in a correct reconstruction, each metabolic enzyme must be able to catalyse at least one steady state flux (given that all external sources and sinks are present). Therefore, the percentages of working enzymes and genes are more informative indicators of model quality than the percentage of dead reactions.

Directional consistency In Figure 5.1c, the dead reactions R_4 and R_6 are represented by non-zero rows in the null space matrix. These reactions comprise a reaction subset, but since both of them irreversibly produce the dead-end metabolite C, none of them carries a steady state flux. Since all reactions in a reaction subset operate symultaneously at any flux mode, if any of them is dead, the others are dead as well. Reaction subsets consisting of dead reactions are termed *directionally inconsistent* (Poolman, unpublished), since the inability to carry a steady state flux is stipulated by the irreversibility constraint. We define a model as *directionally consistent* if it does not contain inconsistent reaction subsets and *directionally inconsistent* otherwise. We hypothesise that directional inconsistencies always coincide with the presence of dead-end metabolites, although these metabolites are not necessarily used by the reactions of the inconsistent subsets (e.g. in Figure 5.1a, the inconsistency of the subset $\{R_4, R_6\}$ is associated with the dead-end metabolites E and F). The presence of an innconsistent reaction subset can be caused by one of the following conditions:

- Incorrectly defined reaction directions (e.g. in Figure 5.1, inverting R_6 would resolve the inconsistency)
- Missing reaction (e.g. a reaction consuming C in Figure 5.1c).
- Defining dead-end metabolites as internal (e.g E and F in Figure 5.1a).

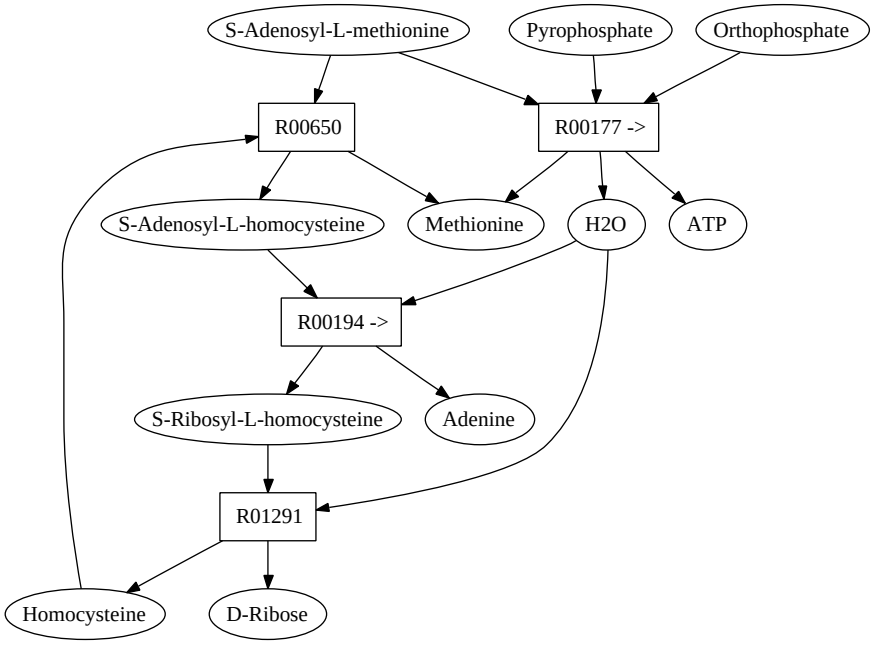
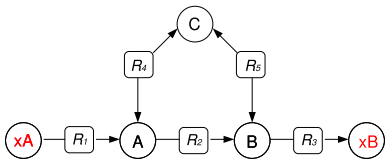


Figure 5.5: A directionally inconsistent reaction subset in the second-line model of `sag`. The irreversible reactions are marked with the symbol $->$. Adenine and S-adenosyl-L-methionine are dead-end metabolites.

Note that inconsistent reaction subsets may involve reversible reactions (see Figure 5.5).

Internal cycles Some flux modes may represent internal cycles with zero net stoichiometries (e.g. $\mathbf{v}_1$ in Figure 2.6). Such flux modes are not thermodynamically feasible, since they are not driven by a difference of free energies. If all the flux modes involving a given reaction are internal cycles, both LP and null space analysis based methods would qualify it as live, although it does not operate at any feasible steady state. Such a reaction can be detected using the reaction correlation method, since its correlation with any transporter is zero [86]. Unfortunately, this method ignores the irreversibility constraint.

The presence of internal cycles often complicates the interpretation of FBA results. An example is shown in Figure 5.6. Let us try to predict, whether the presence of R_1 in this network is necessary for the ability of R_2 to carry a steady-state flux. After removing R_1, R_2 would be still live according to Equation 3.12. However, this 'liveness' is only due to the involvement of R_2 in the internal cycle with R_4 and R_5. In fact, the removal of R_1 would lead to the block of any thermodynamically feasible steady-state flux in R_2. This error could be avoided, if the internal flux were blocked, e.g. by defining R_4 and/or R_5 as irreversible. However, in large networks, the detection of all internal cycles



a

$$\begin{array}{c} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \end{array} \begin{pmatrix} 1e-16 \\ -0.577 \\ 1e-17 \\ 0.577 \\ -0.577 \end{pmatrix}$$

b

Figure 5.6: a) A network with an internal cycle. R_4 and R_5 reversibly produce C. b) The orthogonal null space matrix of the external stoichiometry matrix. The only column represents the internal flux, the rows corresponding to R_1 and R_3 contain values smaller than the computational rounding error and can be treated as zero rows; hence, these reactions are not involved in internal fluxes. The reactions correlation coefficients (absolute cosines of the angles) between R_2 and R_4 and between R_2 and R_5 are equal to 1.

involving a given reaction may be computationally infeasible.

Algorithm 6 presents a method enabling the prevention of internal fluxes through a given reaction, by disallowing the backward fluxes in some minimal set of reversible reactions. The algorithm is applied to an external stoichiometry matrix $\hat{\mathbf{N}}$, which represents a closed system; hence, any flux mode detected in this matrix is internal. The algorithm assumes that the target reaction t is involved in at least one internal flux. Firstly, the orthogonal null space matrix $\mathbf{K}$ of $\hat{\mathbf{N}}$ is calculated. Then the zero rows are deleted from this matrix (with respect to the rounding error), since they represent reactions not involved in internal fluxes (see Figure 5.6b). The list R of candidate reactions is constructed by removing the irreversible reactions and the target reaction from the list of remaining row names of $\mathbf{K}$. The function $corr(i)$ is defined as the absolute value of the cosine of the angle $\theta^K_{i,t}$ between the rows $\mathbf{K}_i$ and $\mathbf{K}_t$; this cosine is the reaction correlation coefficient between the reactions i and t [86]. Using the comparison function *cmp* as an argument for the sorting function, the list R is ordered by descending absolute values of the correlation coefficients of the candidate reactions with the target reaction. Hence, the candidates with stronger correlations with the target are tested first. In the first while-loop, the candidate reactions are iteratively made irreversible and the target reaction is tested for liveness, until it becomes dead (i.e. all internal cycles involving it are blocked). If the loop terminates while the target is still live, the algorithm returns an empty list and the user is notified that the problem is unsolvable. Otherwise, the candidate list is reduced by removing the reactions not tested yet and the second while-loop begins, which iterates over the candidate list in a reverse order. In each iteration, the current reaction is made reversible and the target is tested for liveness. If it becomes live, the current reaction is made irreversible again. Otherwise, it is removed from the candidate list. Hence, the algorithm ensures that the resulting candidate list is irreducible, although it does not guarantee that the list has the minimal possible length. The

candidate reactions suggested by this algorithm may be defined as irreversible in the original network, thus preventing the internal fluxes through the target reaction. However, the result does not depend on the actual irreversibilities and directions of the candidate reactions. As a preprocessing step, the candidate list can be limited using the available thermodynamic data.

Algorithm 6 Given a network with the external stoichiometry matrix $\hat{\mathbf{N}}$, the set of irreversible reactions I and a reaction t involved in some internal cycles, detect a minimal list R of reactions, defining which as irreversible would lead to the block of all internal fluxes involving t.

$\mathbf{K} := orthogonal\_null\_space(\hat{\mathbf{N}})$
$del\_zero\_rows(\mathbf{K})$
$R := row\_names(\mathbf{K}) - I - \langle t \rangle$
$corr(i) := abs(cos(\theta^K_{i,t}))$
$cmp(i,j) := cmp(corr(i), corr(j)))$
$sort(R, cmp)$
$i := 0$
$l := True$
while $l = True$ *and* $i < length(R)$ **do**
  $I := I \cup \{i\}$
  $l := is\_live(\hat{\mathbf{N}}, I, t)$ //Equation 3.12
  $i := i + 1$
end while
if $l = True$ **then**
  $R := \langle\rangle$ //the problem is unsolvable
else
  $R := R[:i]$
  while $i > 0$ **do**
    $i := i - 1$
    $I := I \setminus \{i\}$
    if $is\_live(\hat{\mathbf{N}}, I, t)$ **then**
      $I := I \cup \{i\}$
    else
      $delete(R, i)$
    end if
  end while
end if

5.4 Application to genome-scale models

The methods described above were applied to the first and second-line models of *S. agalactiae*, and the analysis results were used to correct the models.

Table 5.1: Structural indicators of model quality and their dependency of the input data.

Depend on reaction stoichiometries only:
num. of connected components
% of metabolites in the biggest component
stoichiometric consistency
num. of unconserved metabolites
num. of minimal inconsistent net stoichiometries
Depend on reaction stoichiometries and external metabolites:
% of non-orphan metabolites
% of reactions in the core matrix
Depend on reaction stoichiometries, external metabolites and reaction directions:
% of non-dead-end metabolites
Depend on reaction stoichiometries, external metabolites, reaction directions and irreversibilities:
% of live reactions
% of working enzymes
% of working genes
num. of inconsistent reaction subsets

5.4.1 Unconserved metabolites

All models were found to be stoichiometrically inconsistent, with proton (C00080) being the only unconserved metabolite. In order to resolve the inconsistencies, proton was removed from all models.

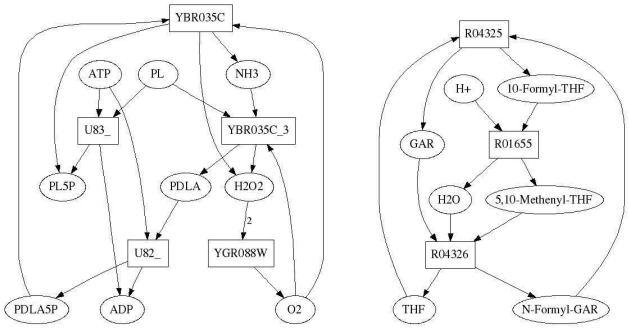
Application to other models In order to illustrate the algorithms detecting minimal inconsistent net stoichiometries and elementary leakage modes, here we present the results of their application to genome-scale models of *E. coli* iJR904 GSM/GPR [95], *S. cerevisiae* [27] and `sag` (constructed on the basis of the KEGG annotation, but without any correction of input data and inclusion of hypothetical reactions).

The model of *E. coli*, (931 reactions, 625 metabolites) is stoichiometrically consistent. The model of *S. cerevisiae* (1172 reactions, 809 metabolites), in contrast, contains 359 unconserved metabolites (very similar results describing these models have been published by [73]). Further, in the model of *S. cerevisiae*, 4030 minimal unconservable sets have been found, out of which 144 are singleton, 2651 comprise two and 1235 comprise three metabolites. Nullspace calculation has been used for the detection of elementary leakage modes. An example is shown in Figure 5.2 (a, c): the same reaction set comprises two elementary leakage modes, which differ in the flux rate coefficients and produce $\{H_2O_2\}$ and $\{O_2\}$, respectively. The inconsistency is caused by the incorrect definition

Table 5.2: Elementary leakage modes in the models of *S. cerevisiae* (a, c) and *S. agalactiae* (b, d). In the right hand side columns, the flux rate coefficients and the produced minimal unconservable sets are shown. In (a), the same reaction set comprises two different modes, depending on the coefficients. The names of unconserved metabolites and incorrectly defined reactions are highlighted bold.

a)		$\{H_2O_2\}$	$\{O_2\}$
U82_:	ATP + PDLA ↔ PDLA5P + ADP	-1/2	-1
U83_:	ATP + PL ↔ PL5P + ADP	1/2	1
YGR088W:	2 $\mathbf{H_2O_2}$ ↔ $\mathbf{O_2}$	-1	-1
YBR035C:	PL + $\mathbf{O_2}$ + NH_3 ↔ PDLA + $\mathbf{H_2O_2}$	-1/2	-1
YBR035C:	PDLA5P + $\mathbf{O_2}$ ↔ PL5P + NH_3 + $\mathbf{H_2O_2}$	-1/2	-1

b)		$\{H^+\}$
R04325:	GAR + 10-Formyl-THF ↔ N-Formyl-GAR + THF	1
R04326:	GAR + 5,10-Methenyl-THF + H_2O ↔ N-Formyl-GAR + THF	-1
R01655:	5,10-Methenyl-THF + H_2O ↔ 10-Formyl-THF + $\mathbf{H^+}$	1



c d

of the reaction YGR088W, where two molecules of water are missing on the right hand side. After correcting this reaction, the subnetwork shown becomes consistent.

The model of *S. agalactiae* (718 reactions, 904 metabolites) contains 215 unconserved metabolites and 214 minimal unconservable subsets; 212 out of them are singleton and 2 comprise 2 metabolites. For each minimal unconservable set, one elementary leakage mode has been detected by nullspace calculation; the numbers of involved reactions vary between 4 and 96. The analysis of de-

tected modes revealed that most of them involve reactions, in whose definitions the same polymeric molecules appear on both sides and are cancelled out in the stoichiometry matrix. Two examples are shown below:

R07284: ATP ↔ Pyrophosphate
R07283: CTP ↔ Pyrophosphate

According to these reactions, ATP and CTP are isomers of pyrophosphate and of each other (in KEGG, they are defined as follows: ATP + tRNA ↔ Pyrophosphate + tRNA and ATP + tRNA ↔ Pyrophosphate + tRNA). These reactions are involved in 204 and 158 elementary leakage modes, respectively, out of the detected 214 ones. The whole model contains 18 reactions involving a polymer on both sides; after removing all of them, only two metabolites remain unconserved: water and proton.

An elementary leakage mode producing a proton is shown in Figure 5.2 (b, d); this mode has been obtained using mixed-integer programming. The inconsistency is caused by the incorrect definition of the reaction R04326, where one hydrogen atom is missing on the right hand side. The model contains 71 reactions with incorrect atomic balances; however, even after removing all of them, proton still remains unconserved due to the 170 reactions with undeterminable atomic balances.

5.4.2 Environment definition

To enable flux-balance analysis, the exchange of metabolites between the model and its environment had to be represented in the form of transporter reactions. Reversible transporters were included for the metabolites listed in Table 5.3. Note that the transporters for water, NH_3, CO_2, and O_2 had been included at the model construction stage; they are considered as permanent parts of the models. The sets of cofactors required and fermentation products were found in the literature, see [125] and [71], respectively. In addition, a transporter was included for fumarate, which is released as a by-product in purine biosynthesis.

The biomass components representing protein, RNA, DNA, membrane and peptidoglycan were declared external; thus, the reactions producing them effectively became transporters.

5.4.3 Model reduction

In Chapter 4 it was mentioned that the second-line models were constructed applying the relatively relaxed threshold E-value of 10^{-3} to the in-house annotations, in order to reduce them later, depending on the results of constructive interrogation. In order to investigate the dependency of the size and quality of a reconstruction on the annotation threshold, sub-models with different thresholds were extracted from all second-line models and analysed (the integer powers of ten in the range between 10^{-30} and 10^{-3}, inclusively were used as threshold values). Table 5.4 demonstrates that the numbers of metabolites, reactions,

Table 5.3: Input and output metabolites in the models of *S. agalactiae*

Small molecules:	water, ammonia, carbon dioxide, oxygen, phosphate
Sugars:	glucose, fructose, lactose, mannose, ribose, galactose, sucrose
Amino acids:	alanine, glycine, valine, lysine, leucine, isoleucine, tyrosine, tryptophan, glutamate, glutamine, aspartate, asparagine, serine, cysteine, methionine, threonine, phenylalanine, histidine, proline, arginine
Nucleobases:	adenine, guanine, cytosine, uracil, xanthine, thymine
Cofactors:	nicotinic acid, pantothenate, pyridoxal, thiamine, biotin, folate, riboflavin
Fermentation products:	lactate, acetate, formate, acetoin, ethanol, fumarate

enzymes and genes in the models grow monotonously with increasing threshold values. To evaluate the quality of the models, the quality indicators described above were used; the results are shown in Table 5.5. Although the distributions of quality indicators are not monotonous, all of them (except for the percentages of metabolites in the biggest connected components) tend to deteriorate, while relaxing the threshold. Thus, the selection of the threshold E-value involves a tradeoff between the size of a model and its quality. The table also shows that all quality indicators in all models abruptly decrease in the region of 10^{-12} (percentages of non-orphan and non-dead-end metabolites, core and live reactions) or 10^{-13} (other indicators). The more conservative threshold E-value of 10^{-13} was selected, and the models generated using this threshold were further considered as the second-line models.

5.4.4 Internal fluxes

Algorithm 6 was used to prevent internal cycles involving the ATPase reaction. The algorithm was modified, so that certain predefined reactions could be excluded from the candidate list; the reactions catalysed by the known ATP producers, namely phosphoglycerate kinase, acetate kinase and aspartate kinase, were excluded. In all models, the algorithm suggested the same pair of candidate reactions, shown below and catalysed by N-acetylneuraminate lyase and $NADP^+$-glyceraldehyde phosphate dehydrogenase, respectively:

R01811: N-acetylneuraminate $\leftrightarrow$ N-acetyl-D-mannosamine + pyruvate
R01058: $NADP^+$ + G3P + H_2O $\leftrightarrow$ H^+ + NADPH + 3PG

Cycles involving these reactions and ATPase are shown in Figure 5.7. These reactions were defined as irreversible, thus preventing the internal fluxes through ATPase.

Table 5.4: Dependency of the numbers of elements in the second-line models on the $\log_{10}$ threshold E-value (see the legends).

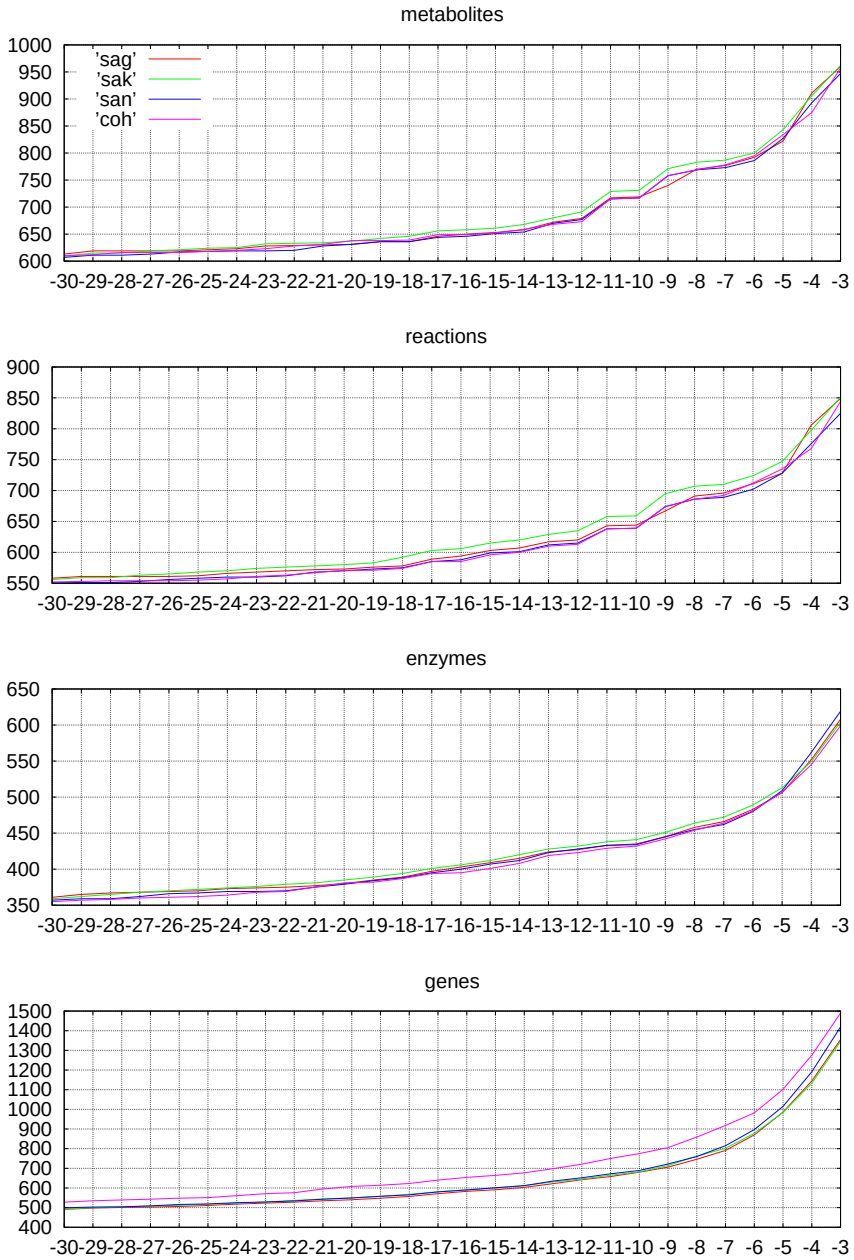
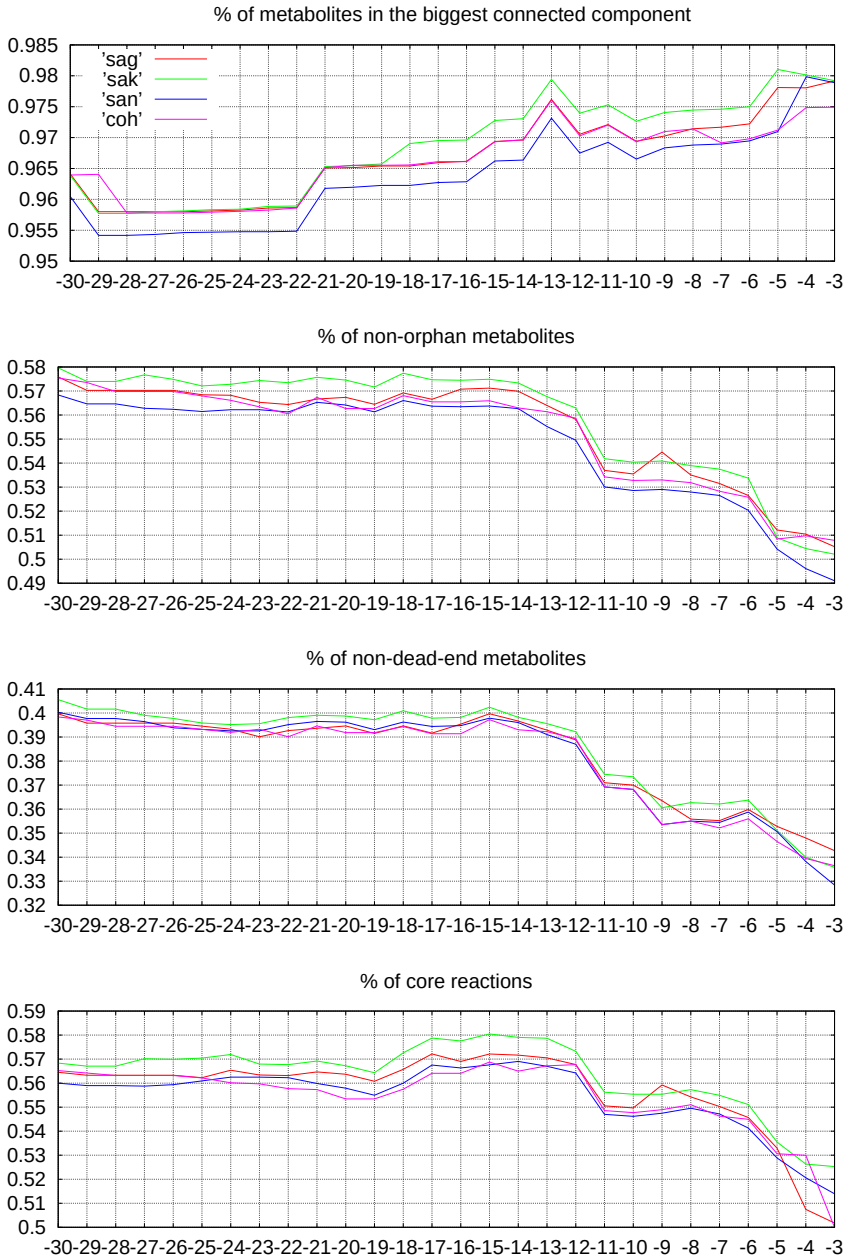
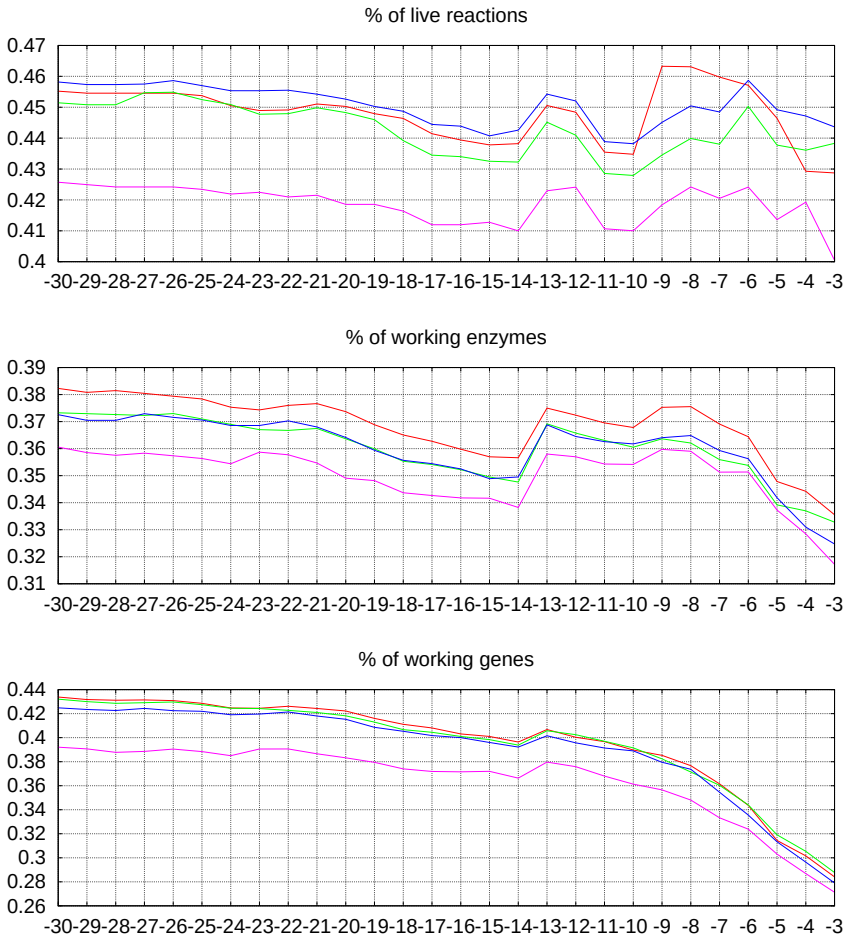
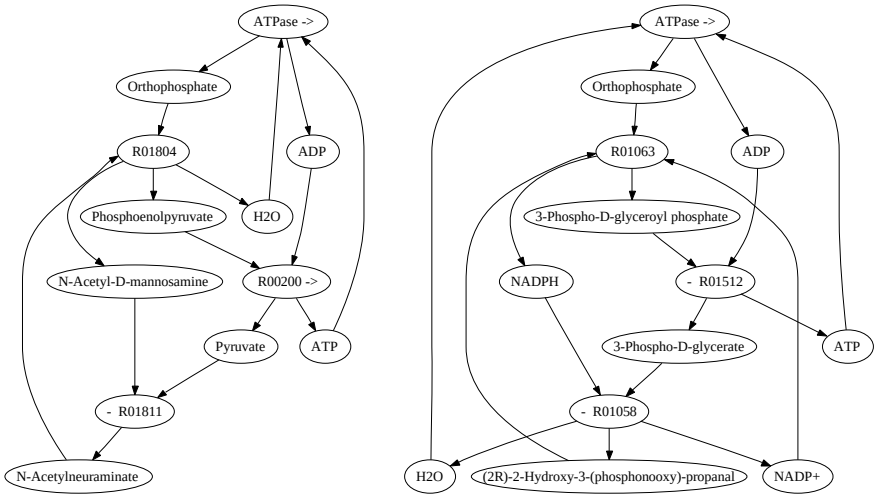


Table 5.5: Dependency of quality indicators in second-line models on the $\log_{10}$ threshold E-value (see the legends).




% of live reactions
0.47
0.46
0.45
0.44
0.43
0.42
0.41
0.4
-30 -29 -28 -27 -26 -25 -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3
% of working enzymes
0.39
0.38
0.37
0.36
0.35
0.34
0.33
0.32
0.31
-30 -29 -28 -27 -26 -25 -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3
% of working genes
0.44
0.42
0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.26
-30 -29 -28 -27 -26 -25 -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3



a b

Figure 5.7: Cycles involving ATPase and the reactions suggested by the Algorithm 6.

5.4.5 Implementation

Figure 5.8 shows an example Python code used for the constructive interrogation, correction and subsequent quality evaluation of a second-line model of `sag`, constructed using the code shown in Figure 4.2. The method `UnconservedMets` returns the list of unconserved metabolites, which are removed using the method `Delete`. Then the model is associated with the PRIAM database and reduced with the threshold E-value of 10^{-13}.

The method `LoadEnv` complements the model with transporters for a given set of metabolites, defined in a comma-separated text file. The amino acid transporters are included by the method `FeedWithAminoAcids`; then the biomass components are made external.

The method `LoadNames` reads the list of ATP-producing enzymes from a text file. The reactions catalysed by these enzymes are excluded from the candidate list of reactions in Algorithm 6, which is invoked by the method `CycleKillers`. The reactions of the suggested list are then made irreversible to block the cycles involving ATPase.

The same set of operations was applied to the first-line models, except for model reduction.

5.4.6 Results and discussion

Table 5.6 shows the numbers of elements and the quality indicators of the resulting models. Each second-line model contains approximately 10% more genes,

```
system = Bio.System('2.xspy')

##unconserved metabolites
UM = system.UnconservedMets()
system.Delete(UM)

##attaching the annotative database and reducing the model
P = Source.GetDB('PRIAM')
P.Open('genome.out')
system.SetSource('Anno', P)
system = Bio.ExtractByQual(system, 1e-13)

##defining the transporters
system.LoadEnv('Rich.env')
system.FeedWithAminoAcids()
system.Externalise(['PROTEIN', 'xRNA', 'xDNA', 'MEMBRANE',
'PEPTIDOGLYCAN'])

##blocking the cycles involving ATPase
names = system.LoadNames('include/NTP.rev')
k = system.CycleKillers('ATPase', system.GetChildren(names))
system.SetIrrevs(k)

##results
unbal = system.UnbalancedReacs()
orphans = system.Orphans()
live_reacs = system.LiveReacs()
working_enzymes = system.GetParents(live_reacs)
working_genes = system.GetParents(working_enzymes)
incons = system.InconsSubsets()

system.Save('2.xspy')
```

Figure 5.8: Constructive interrogation, correction and evaluation of a second-line model.

Table 5.6: Numbers of elements and quality indicators in the models after the corrections. The symbol ∩ denotes intersections.

line	1-st				2-nd					∩
strain	sag	sak	san	∩	sag	sak	san	coh	∩	
Genes	566	595	576	NA	622	631	635	698	NA	NA
% of working genes	36.0	35.8	37.8	NA	44.2	44.2	45.2	42.3	NA	NA
Enzymes	369	389	375	366	424	428	423	419	406	313
% of working enzymes	39.8	39.1	41.1	39.9	40.8	40.2	41.1	39.4	39.4	39
Reactions	560	586	564	556	621	634	615	614	600	508
% of unbal. reacs.	16.1	15.2	15.8	15.5	20.5	20.2	19.5	20.2	18.8	15.6
% of live reacs.	53.6	52.0	55.3	53.6	48.3	47.6	50.2	45.8	46.2	48.0
Metabolites	568	587	571	564	662	671	658	658	644	552
% of orphans	37.7	37.5	37.1	34.2	42	41.7	42.4	42.2	40.1	29.7
Incons subsets	5	7	5	5	7	9	5	6	4	4

enzymes and reactions than the first-line model of the same strain, as well as higher percentages of working genes. On the other hand, the percentages of live reactions in the second-line models are slightly lower and those of unbalanced reactions and orphan metabolites are higher. In general, we did not find any considerable and consistent difference in the quality between the first and second-line models.

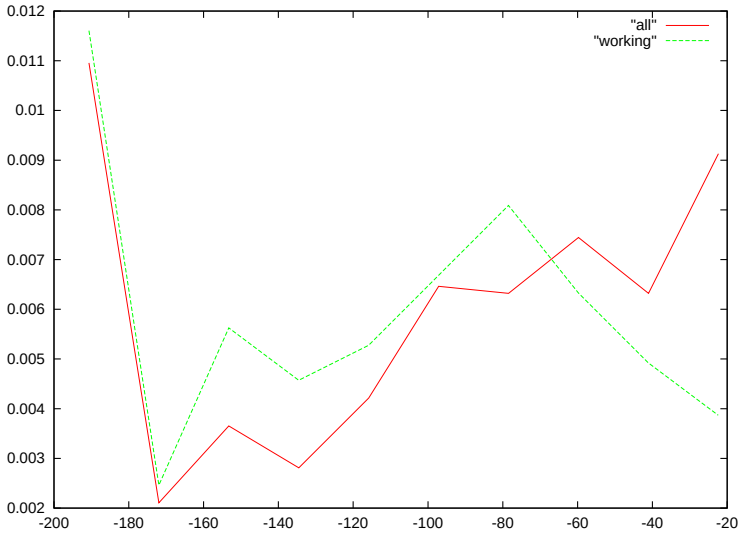
In all models, hydrogen is the only atom with violated balance. The high percentages of unbalanced reactions are caused by the removal of proton from the models; thus, the balance of hydrogen was violated in some originally correct reactions.

A number of model properties (including some quality indicators) depend on the definition of external metabolites. In order to minimise the effect of missing sources or sinks, a relatively 'rich' environment was simulated by including the corresponding transporter reactions. In principle, any reaction in a model can be rendered live by externalising some internal metabolites. The same is true also for inconsistent reaction subsets (an example of such a subset is shown in Figure 5.5).

Algorithm 6 provides a simple method for the prevention of internal fluxes. The advantage of this method is the possibility to detect feasible fluxes involving a target reaction. The payoff is the possibility that the reactions suggested by the algorithm are physiologically reversible, so making them irreversible in the model affects its accuracy (especially if they are defined with wrong directions). Therefore, we applied the model to only one target reaction, namely ATPase, which is highly important for the investigation of metabolic capabilities of the network.

The selection of the threshold E-value for model reduction was based on the hypothesis that the quality indicators introduced in this chapter correlate with the accuracy of the model, e.g. the percentage of elements (enzymes, reactions and metabolites) in the model which are actually present in the underlying sys-

Figure 5.9: Normalised distributions of $\log_{10}$ E-values in the sets of all and working enzymes, in the second-line model of `sag`.



tem. This hypothesis is supported by Figure 5.9, which shows two distributions of $\log_{10}$ E-values. These distributions were constructed as follows: for each EC number in the second-line model of `sag`, the best E-value in the annotation was included into the first set (the logarithm value for 0.0 was defined as -200). The second set comprised the E-values corresponding to the working enzymes only. Both distributions are bimodal and the left peaks (containing the low E-values) are almost equal, while the right peak of the second distribution is strongly shifted to the left. A Kolmogorov-Smirnov test demonstrated a significant difference between the distributions with a p-value of 0.02. Hence, the proportion of low-quality predictions with high E-values in the set of working enzymes is smaller than that in the complete set of enzymes.

Taking the above mentioned into consideration, the relatively low percentages of working enzymes in the resulting models might be considered as an evidence of their poor accuracy. In the following chapters, we will propose methods for a further improvement of the accuracy of the models. It will be also demonstrated that despite the presence of possible errors, the models demonstrate a good agreement with the available experimental data.

Chapter 6

Reannotations

6.1 Introduction

In chapter 4 it was mentioned that the in-house annotations based on the best hits predicted using RPS-BLAST and PRIAM are subject to further revisions. We assume that holistic analysis of metabolic models may enable the generation of hypotheses about the underlying genome annotations. This process is closely related to the correction of errors in a model itself, since missannotations are among the major causes of errors. For instance, if a particular enzyme is missing in the model but has been predicted for one of the genes as a suboptimal hit, it may be associated with the gene instead of the best hit. Furthermore, even an enzyme which is absent in the list of predictions (e.g. if it is missing in the PRIAM database) may be assigned to one of the genes. If the presence of an enzyme is hypothesised but the gene encoding it is not known, it may be included into the reannotated model as an 'orphan enzyme', e.g. represented by an empty row in the annotation matrix.

Clearly, each hypothetical annotation needs to be verified experimentally. Nevertheless, the plausibility of hypotheses may be supported by one or more of the conditions listed below:

- The missing enzyme is essential for a vital cellular function, e.g. biosynthesis of a biomass component.
- The reannotation and the subsequent regeneration of the model improves its quality (an example is shown in Figure 6.1).
- The model based on the reannotated genome demonstrates a better agreement with the available experimental data.
- The reannotation agrees with a publicly available annotation (e.g. from KEGG).
- The reannotation is based on a homology detected by more than one independent method (e.g. RPS-BLAST and BLASTn).

- The reannotated gene has a high co-expression with the genes encoding other enzymes in the pathway (e.g. elementary mode) restored due to the reannotation.

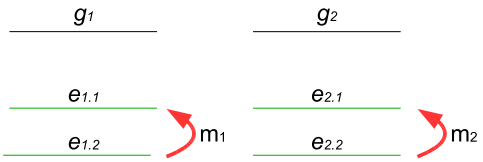
Here is an example of the latter condition: assume that the genes g_1 and g_2 in Figure 6.1 highly correlate in a set of microarray experiments. The reannotations associate these genes with the reactions $R_{1.2}$ and $R_{2.2}$, which are involved in the same reaction subset and terefore have the maximal possible correlation coefficient of 1.0. We assume that a coincidence of a high reaction correlation coefficient and a strong gene co-expression can be considered as an experimental support for a given pair of gene annotations.

A missannotation can be detected by means of general analysis methods; e.g. if it causes a failure in some expected metabolic function. Reannotations can be based on a search in the literature or databases. Although such a 'manual' approach has been occasionally employed in the present work, its weekness is demonstrated by Figure 6.1, where the flux between xA and xB can be restored only through a pair of reannotations. Assuming that each gene is associated with a single enzyme, the number of possible annotations of a given genome is $\prod_i n_i$, where n_i is the number of predictions for the i-th gene. Clearly, in a genome-scale model, testing all possible combinations of predictions would be computationally infeasible. In this chapter, we describe methods for automatic generation of hypotheses about potential reannotations, using two complementary methods: search space reduction and stochastic optimisation. These methods aim to improve the quality of the metabolic model by finding an optimal genome annotation.

6.2 Search space reduction

This method is based on the fact that if a reaction is dead in a metabolic network then it would remain dead in any of its subnetworks obtained by reducing the reaction set; in other words, it is impossible to render a reaction live by removing other reactions. To reduce the annotation, we construct a *supermodel* containing all reactions catalysed by all enzymes predicted in the annotation, e.g. it covers all predictions for each gene. Than we detect the unemployed enzymes in the supermodel and remove them from the lists of predictions, since they would remain unemployed in any submodel associating the genes with single predictions or subsets of predictions.

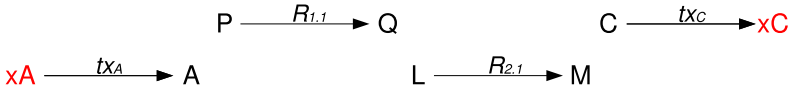
An example is shown in Figure 6.1, where the enzymes $e_{1.1}$ and $e_{2.1}$ are unemployed in the supermodel and can be removed from the annotation, thus resulting in the correct final model. In a general case, this method does not necessarily lead to an unambiguous annotation, since some genes may catalyse multiple working enzymes in the supermodel. However, the method reduces the number of predictions and thereby diminishes the search space in which an optimal annotation is to be found.



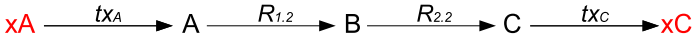
a

$e_{1.1} \Rightarrow \quad R_{1.1.}$: P → Q
$e_{1.2} \Rightarrow \quad R_{1.2.}$: A → B
$e_{2.1} \Rightarrow \quad R_{2.1.}$: L → M
$e_{2.2} \Rightarrow \quad R_{2.2.}$: B → C

b



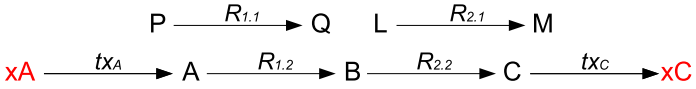
c



d

indicator	before	after
connected components	4	1
orphan metabolites	8	0
dead reactions	4	0
unemployed enzymes	2	0
unemployed genes	2	0

e



f

Figure 6.1: The relationship between a genome annotation and the quality of the corresponding metabolic model. a) For each gene (black line), two possible enzymes are predicted (green lines), which are shown in the order of increasing E-values. Hence, in the initial annotation, g_1 and g_2 are assigned the enzymes $e_{1.1}$ and $e_{2.1}$, respectively. The reannotations m_1 and m_2 associate the genes with the second best predictions. b) Each enzyme catalyses one reaction. c) The model based on the initial annotation and including predefined transporters for A and C is disconnected. d) After the reannotations, the gaps in the model are filled. e) The values of the quality indicators before and after the reannotation. f) In the supermodel, $R_{1.1.}$ and $R_{2.1.}$ are dead; the corresponding enzymes are unemployed and can be removed from the annotation.

6.3 Stochastic search

The problem of finding an optimal annotation has the following important properties:

- The set of feasible solutions is discrete: each solution can be described in terms of discrete elements (genes and enzymes) and a transition between two neighbour solutions is possible by a single reannotation.
- The search space may contain local optima: annotations which are better than any of their neighbours, yet worse than the 'true' global optimum.

The first property implies that the problem belongs to the domain of combinatorial optimisation [13]: a domain of applied mathematics whose subject is the development of algorithms for finding minimum or maximum values of a function of very many independent variables. In this particular case, the number of variables is equal to the number of genes in the genome, and the number of possible values of a variable is equal to the number of predicted enzymatic functions. The objective function of the optimisation is the model quality, which can be measured using one of the indicators described in Chapter 5.

From the second property it follows that the problem cannot be solved by simple iterative improvement of the annotation (i.e. looking for the best possible reannotations until no further improvement can be found), since the search would possibly converge to a local optimum and fail to progress further. To overcome this problem, a group of stochastic search algorithms has been developed which incorporate probabilistic (random) choices. In particular, the neighbour solutions are selected randomly (although the probabilities of different choices may be weighted), thus enabling an escape from a local optimum.

Simulated annealing In the present work, we used the stochastic search algorithm called simulated annealing [53]. This algorithm is intended to model the process of annealing in metallurgy, involving heating and controlled cooling of a material into a crystalline structure. The process begins at a high temperature, which causes the atoms to wander randomly away from their initial positions. Then the temperature is gradually reduced, thus decreasing the mobility of atoms and increasing the probability of moving 'downhill' to the state with a lower energy. If temperature is reduced slowly enough, the system is likely to attain the minimum energy state.

This principle can be applied to more general combinatorial optimisation problems. The system is initialised to some state $s \in S$, where S is the search space. The internal energy $E(s)$ may represent any objective function. The global variable T (temperature) is initialised with a high enough value and then iteratively reduced. At each iteration, a neighbour state s' is generated by making some user-defined changes in the current state s. If $E(s')$ is lower than $E(s)$, than s is replaced by s'. Otherwise, s is replaced by s' with the probability $P = exp((E(s) - E(s'))/T)$. Thus, at large values of T, the probability of

accepting a higher-energy state is relatively high, while at small values, this probability tends to zero and the procedure reduces to iterative improvement.

Algorithm 7 shows the modification of simulated annealing used in the present work. It uses an exponential cooling scheme, i.e. at each external iteration, T is multiplied by a constant α, which is smaller than, but close to 1. Within each external iteration, n neighbour states are generated and either accepted or rejected. Note that in addition to the probability function P, the acceptance of a new state is determined by its feasibility (a user-defined boolean function). The variable eb is used to memorise the best energy value achieved so far; the state with the best energy is saved. The algorithm terminates when T achieves the final value T_{min}.

Algorithm 7 Simulated annealing in a system with an initial state s and the initial temperature T. n is the number of iterations at each temperature; α is a constant multiplier; T_{min} is the final temperature.

$e := E(s)$
$eb := e$
while $T > T_{min}$ **do**
  for $i = 0$ to n **do**
    $s' := neighbour(s)$
    $e' := E(s')$
    if $random\_number(0,1) < P(T,e,e')$ and $is\_feasible(s')$ **then**
      $s := s'$
      $e := e'$
      if $e < eb$ **then**
        $eb := e$
        $save(s)$
      end if
    end if
  end for
  $T := \alpha T$
end while

Mutation operator For a neighbour state generation, a mutation operator is applied, which selects a random gene in the genome and associates it with an enzyme selected from the list of predictions. The probability of the selection of a given enzyme is proportional to the significance of the prediction, which is measured by the bit score of the alignment. This is achieved by using roulette wheel selection (a technique widely used in genetic algorithms [26]): a new list is composed by including q copies of each enzyme, where q is the bit score. From this list, a single enzyme is selected randomly.

Feasibility test For being accepted, each neighbour state has to pass a feasibility test, which evaluates the viability of the corresponding metabolic network.

We consider a metabolism as viable if it is capable of synthesising biomass and ATP.

Objective function Although different quality indicators were used as objective functions for simulated annealing (such as the percentages of orphan metabolites, dead reactions and unemployed enzymes), using the number of unemployed genes was found to lead to optimised models of better quality. This may be due to the fact that the total number of genes in a reconstruction is constant, while the numbers of enzymes, reactions and metabolites vary depending on the reannotations.

Practical applications The ultimate objective of an optimisation procedure is typically the detection of a global optimum. However, since in the case of genome annotations, it is not clear whether such an optimum can be detected and if yes, how it should be interpreted, we propose the following practical steps:

1. Run the algorithm n times on the same original model.
2. Find the intersection of the sets of working enzymes in all n optimised models.
3. Find the set difference of this intersection and the set of all enzymes in the original model.

The resulting difference comprises the enzymes which are missing in the original annotation but tend to be included in the optimised ones, regardless of the variability created by random choices. We assume that these enzymes (further termed *consensus enzymes*) and the corresponding genes can be considered as candidates for reannotations.

6.4 Application to genome-scale models

The methods described above were applied to the models constructed at the previous stages. The transporter sets were defined as in Table 5.3.

6.4.1 Manual reannotations

It was found that neither the first, nor the second-line models were able to synthesise peptidoglycan. One of the causes of this inability was the auxotrophy for the amino acid meso-diaminopimelate (C00680), which is consumed by the following reaction producing one of the precursors of peptidoglycan:

R02788: UDP-N-acetylmuramoyl-L-alanyl-D-glutamate + meso-diaminopimelate + ATP → orthophosphate + ADP + UDP-N-acetylmuramoyl-L-alanyl-D-gamma-glutamyl-meso-2,6-diaminopimelate

Table 6.1: Reannotations in the second-line models. The columns from left to right contain the strain names, the gene names, the KEGG annotations, the PRIAM best hits and the reannotations. The PRIAM predictions are shown with their E-values; the '+' symbol in the rightmost bottom column means that the new enzyme is assigned in addition to the existing one.

strain	gene	KEGG	PRIAM best	reannotated
sag	SAG1391	6.3.2.13	6.3.2.13: 3e-148	6.3.2.7
	SAG0884	6.3.2.13	6.3.2.13: 2e-18	6.3.2.7
sak	SAK_1424	6.3.2.13	6.3.2.13: 3e-148	6.3.2.7
	SAK_1007	6.3.2.13	6.3.2.13: 2e-18	6.3.2.7
san	gbs0901	6.3.2.13	6.3.2.13: 2e-18	6.3.2.7
	gbs1461	6.3.2.13	6.3.2.13: 1e-148	6.3.2.7
coh	SAN_1499	NA	6.3.2.13: 5e-149	6.3.2.7
	SAN_0988	NA	6.3.2.13: 1e-17	6.3.2.7

a

sag	SAG1538	2.3.1.157, 2.7.7.23	2.3.1.157: 4e-146	+ 2.7.7.23: 1e-74
sak	SAK_1561	2.3.1.157, 2.7.7.23	2.3.1.157: 4e-146	+ 2.7.7.23: 1e-74
san	gbs1594	2.3.1.157, 2.7.7.23	2.3.1.157: 1e-145	+ 2.7.7.23: 1e-74
coh	SAN_1645	NA	2.3.1.157: 4e-146	+ 2.7.7.23: 4e-74

b

The reaction is catalysed by the enzyme UDP-N-acetylmuramoyl-L-alanyl-D-glutamate-2,6-diaminopimelateligase (6.3.2.13), which was assigned in both public and in-house annotations of each strain to the same pair of genes (see Table 6.1a). However, meso-diaminopimelate is known to be used as a component of peptidoglycan mostly in Gram-negative species, whereas most Gram-positive species use lysine instead [120]. Therefore, these genes in the second-line models were reannotated with the enzyme UDP-N-acetylmuramoyl-L-alanyl-D-glutamate-L-lysine ligase (6.3.2.7, also known as MurE synthetase), which consumes lysine:

R02786: UDP-N-acetylmuramoyl-L-alanyl-D-glutamate + L-Lysine + ATP → orthophosphate + ADP + UDP-N-acetylmuramoyl-L-alanyl-gamma-D-glutamyl-L-lysine

Another enzymatic step was found to be missing in the second-line models only, namely UDP-N-acetylglucosamine diphosphorylase (2.7.7.23), which catalyses the following reaction:

R00415: N-Acetyl-D-glucosamine 1-phosphate + UTP → UDP-N-acetyl-D-glucosamine + pyrophosphate

Table 6.1 shows that this enzyme is predicted for a single gene in all available

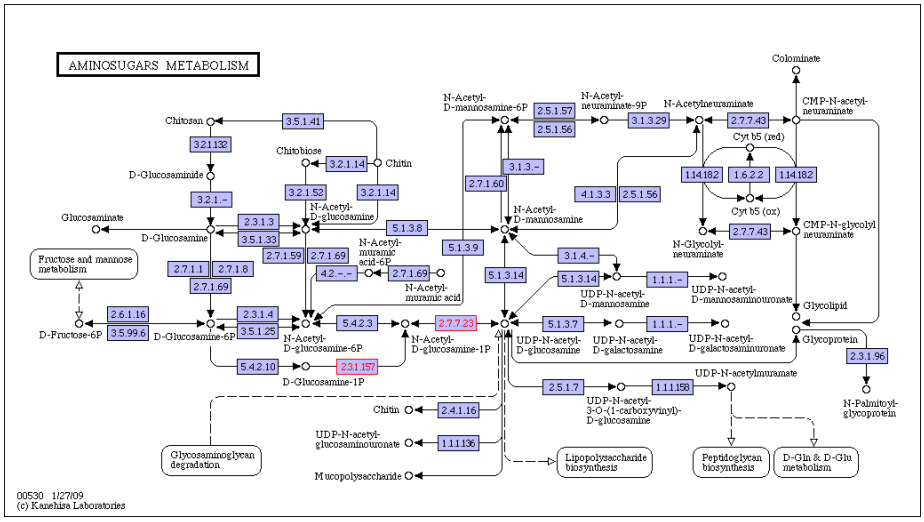


Figure 6.2: KEGG pathway diagram showing the biological functions of the enzymes 2.3.1.157 and 2.7.7.23 (highlighted red).

Table 6.2: The increase in the values of quality indicators after the reannotations presented in Table 6.1

strain	non-orphan mets.	live reactions	working enzymes	working genes
sag	2	15	12	14
sak	2	15	12	14
san	2	15	12	14
coh	2	17	13	18

KEGG annotations and each PRIAM annotation with an E-value of $\approx 10^{-74}$. This is the second-best hit for each of these genes; while the best hit is 2.3.1.157, which catalyses the previous step in the pathway:

R05332: Acetyl-CoA + α-D-Glucosamine 1-phosphate $\leftrightarrow$ CoA + N-Acetyl-D-glucosamine 1-phosphate

(both steps are highlighted in Figure 6.2). The KEGG annotations, in contrast, associates the same genes with both enzymes. In addition, the BRENDA database qualifies 2.3.1.157 as a bifunctional enzyme also possessing the activity of EC 2.7.7.23 in *E. coli* and *S. pneumoniae*. Therefore, the second enzyme was assigned also to the genes in the second-line models.

As a result of two reannotations, the second-line models became capable of

Table 6.3: The magnitude of the annotation search space of `sag` before and after the reduction. The columns '1' to '8' show the numbers of genes with a given number of predictions or more. The last column shows the numbers of possible annotations.

	enzymes	1	2	3	4	5	6	7	8	comb.
initial	730	622	320	212	163	135	127	123	114	10^{220}
reduced	149	72	72	39	24	16	15	8	6	10^{36}

producing peptidoglycan. In addition, some of the quality indicators were found to be improved (see Table 6.2).

6.4.2 Optimised annotation

The methods of optimised annotation were applied to the second-line model of `sag`. The biomass components were declared internal; hence, external biomass was represented solely by the hypothetical metabolite of the same name.

Search space reduction A supermodel was constructed and the unemployed enzymes were detected in it; these enzymes were removed from the search space, as well as 20 enzymes catalysing the reaction R00086, which is isostoichiometric to the hypothetical reaction ATPase. The results of the reduction are shown in Table 6.3, which demonstrates that the number of solutions was reduced by a factor of 10^{184}.

Simulated annealing Before the annealing, the biomass components were made internal, so biomass was represented by the single external metabolite of the same name. The viability test was defined as the liveness of the reactions syn_BIOMASS and ATPase.

The algorithm was run six times, using the number of unemployed genes as the objective function and the following parameters:

initial temperature	100
final temperature	0.001
decrement factor	0.95
number of iterations at each temperature	10

The initial value of the objective function was 333 and the final values varied between 303 and 305. Figure 6.3a shows that after the drastic improvement at the beginning, the value of the objective function decreases almost linearly, so one could expect a further improvement at lower temperatures. The resulting optimised models are described in Table 6.4a. The comparison shows relatively large differences in the compositions of the models. However, the Table 6.4a demonstrates that the the sets of working enzymes are more similar among the optimised models, than between the optimised ones and the original one.

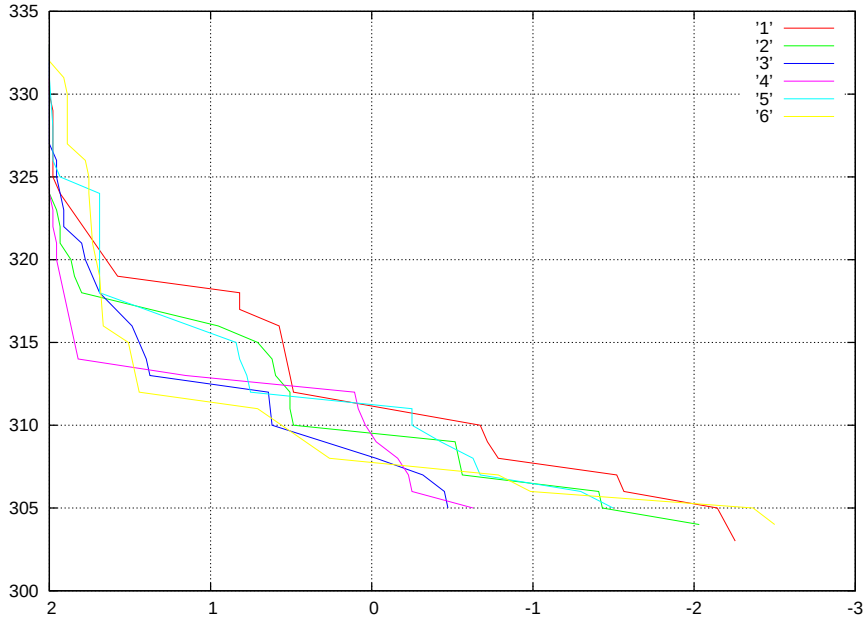


Figure 6.3: The dynamics of model quality improvement in the 6 runs of simulated annealing: the graph shows the dependency of the number of unemployed genes (y-axis) against the $\log_{10}$ of the temperature.

The distance tree (Table 6.4b) makes evident that the original model is a clear outlier.

Applying the procedure described in Section 6.3, we identified 5 consenus enzymes. Some of them were included into a copy of the second-line model of `sag`, which is further referred to as the third-line model; the enzymes and the corresponding changes are presented in the Table 6.5 and described below:

2.7.1.60: N-acylmannosamine kinase catalyses the following reaction:

R02705: N-Acetyl-D-mannosamine + ATP ↔ N-Acetyl-D-mannosamine 6-phosphate + ADP

Table 6.5a shows the genes for which it is predicted among the other hits. All of these genes already encode working enzymes in the second line model, but reannotating any of them leads to an improvement of model quality.

For each of the candidate genes, a BLASTn homology search was performed using the NCBI web service, attempting to detect homologies with sequences encoding 2.7.1.60. For the gene SAG0040, a number of homologies was found in

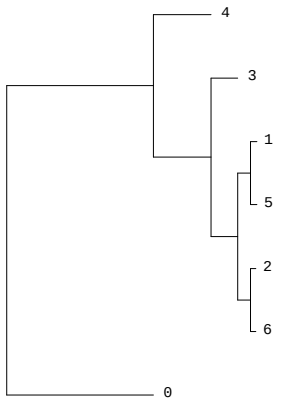
Table 6.4: a) Quantitative characteristics of the optimised models. The last column shows the number of the identically annotated genes in the two top rows and the cardinalities of the set intersections in the other rows. b) Cardinalities of the intersections of the sets of working enzymes in the original (denoted by 0) and optimised models. c) Tree of the set distances.

	1	2	3	4	5	6	∩
Genes	622	622	622	622	622	622	566
Working genes	319	318	317	317	317	318	316
Enzymes	430	432	432	428	435	428	388
Working enzymes	211	212	211	207	214	208	168
Reactions	623	629	668	645	632	624	534
Unbalanced reactions	141	129	165	153	155	141	90
Live reactions	359	361	377	369	354	372	287
Metabolites	656	655	699	679	668	640	601
Orphans	270	267	293	280	276	243	208
Incons. subsets	4	4	4	4	4	3	3

a

	0	1	2	3	4	5
1	62					
2	57	35				
3	52	44	41			
4	58	42	51	42		
5	65	39	46	39	53	
6	67	45	34	41	49	36

b



c

Table 6.5: Candidate genes for the consensus enzymes identified using simulated annealing. The columns from left to right show the identifiers, KEGG annotations, PRIAM best hits with the E-values, E-values for the identified enzymes and the increase in the number of working genes as a result of a reannotation in an otherwise unchanged model.

ID	KEGG	best	new	+
2.7.1.60				
SAG0471	glk; glucokinase; K00845 glucokinase [EC:2.7.1.2]	2.7.1.2: 1e-95	1e-52	3
SAG1689	scrK; fructokinase; K00847 fructokinase [EC:2.7.1.4]	2.7.1.4: 2e-129	8e-16	3
SAG0040	ROK family protein	2.7.1.2: 3e-37	3e-33	3
3.2.1.85				
SAG1103	arb; 6-phospho-beta-glucosidase (EC:3.2.1.86); K01223 6-phospho-beta-glucosidase [EC:3.2.1.86]	3.2.1.86: 0.0	2e-79	5
SAG0791	bglA; 6-phospho-beta-glucosidase; K01223 6-phospho-beta-glucosidase [EC:3.2.1.86]	3.2.1.86: 0.0	2e-81	5
1.1.1.36				
SAG0703	D-mannonate oxidoreductase (EC:1.1.1.131)	1.1.1.100: 8e-44	1e-29	2
SAG1091	short chain dehydrogenase/reductase family oxidoreductase	1.1.1.276: 3e-61	4e-15	2
SAG0664	cylG; CylG protein; K11049 CylG protein	1.1.1.100: 6e-79	6e-60	2
SAG1544	fabG; 3-ketoacyl-(acyl-carrier-protein) reductase (EC:1.1.1.100); K00059 3-oxoacyl-[acyl-carrier protein] reductase [EC:1.1.1.100]	1.1.1.100: 2e-34	8e-25	2
SAG0348	fabG; 3-ketoacyl-(acyl-carrier-protein) reductase (EC:1.1.1.100); K00059 3-oxoacyl-[acyl-carrier protein] reductase [EC:1.1.1.100]	1.1.1.100: 1e-96	6e-61	2
SAG1209	short chain dehydrogenase/reductase family oxidoreductase; K07124	1.1.1.100: 8e-32	1e-23	2
1.5.1.12				
SAG0823	gapN; glyceraldehyde-3-phosphate dehydrogenase, NADP-dependent; K00131 glyceraldehyde-3-phosphate dehydrogenase (NADP) [EC:1.2.1.9]	1.2.1.9: 00	5e-77	4
SAG1124	aldehyde dehydrogenase family protein; K00135 succinate-semialdehyde dehydrogenase ($NADP^+$) [EC:1.2.1.16]	1.2.1.16: 1e-122	8e-58	4
4.3.1.19				
SAG0334	cysK; cysteine synthase A; K01738 cysteine synthase [EC:2.5.1.47]	2.5.1.47: 2e-125	8e-22	1

the genomes of various strains of *Streptococcus pyogenes, Streptococcus equi* and *Haemophilus somnus*; the BLAST output for the most significant homologies is shown below:

```
>gb|CP000260.1| Streptococcus pyogenes MGAS10270, complete genome
Length=1928252

 Features in this part of subject sequence:
   N-acetylmannosamine kinase / Transcriptional regulator

 Score =  298 bits (330),  Expect = 2e-77
 Identities = 498/703 (70%), Gaps = 24/703 (3%)
 Strand=Plus/Plus
...

>gb|CP000003.1| Streptococcus pyogenes MGAS10394, complete genome
Length=1899877

 Features in this part of subject sequence:
   N-acetylmannosamine kinase

 Score =  298 bits (330),  Expect = 2e-77
 Identities = 552/791 (69%), Gaps = 28/791 (3%)
 Strand=Plus/Plus
```

Since no homologies were found for the other two genes, the enzyme 2.7.1.60 in the third-line model was assigned to the gene SAG0040.

3.2.1.85: 6-phospho-beta-galactosidase catalyses the following reaction:

R06110: Lactose 6-phosphate + H_2O $\leftrightarrow$ D-Galactose 6-phosphate + Glucose

Table 6.5 shows that both candidate genes for this enzyme are associated with the enzyme 3.2.1.86 in both KEGG and PRIAM annotations. However, this enzyme is unemployed in the second line model and reannotating any of the candidate genes leads to a considerable improvement of model quality. The BLASTn search revealed homological sequences encoding 3.2.1.85 for the gene SAG1103; a part of the output is shown below:

```
>dbj|AB003927.1| Lactobacillus gasseri DNA for
phospho-beta-galactosidase 1, complete cds
Length=1449

 Score =  168 bits (186),  Expect = 5e-38
 Identities = 274/391 (70%), Gaps = 4/391 (1%)
 Strand=Plus/Plus
```

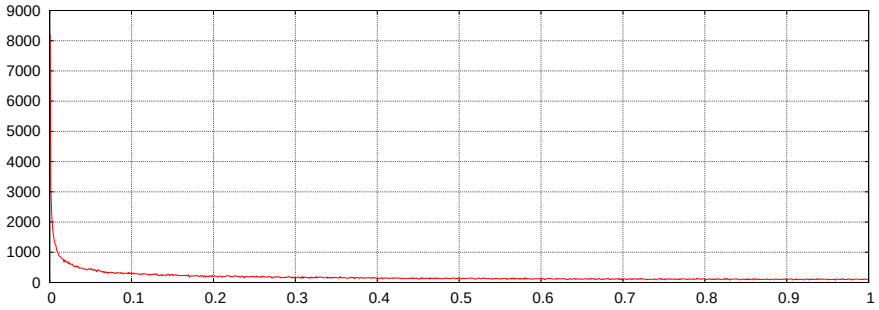


Figure 6.4: Distribution of the p-values of the Pearson's correlation coefficients of the genes of the 3-rd-line model in the gene expression data.

Table 6.6: P-values of the Pearson's correlation coefficients between the candidate genes for 3.2.1.85 and the genes encoding 5.3.1.26.

EC	3.2.1.85		5.3.1.26	
gene	SAG1103	SAG0791	SAG1931	SAG1930
SAG1103	0	0.147	0.011	0.186
SAG0791		0	0.04	0.069
SAG1931			0	0.089
SAG1930				0

For SAG0791, no significant homologies were found. Therefore, in the third-line model, the enzyme 3.2.1.85 was assigned to the gene SAG1103. Additional support for this reannotation was found in gene expression data characterising the strain `sak` (courtesy of Professor Aamanda Jones, University of Washington). The data describe the growth of the wild-type strain in tryptic soy broth in 5% CO_2. The expression was measured at 11 time points: 0, 0.5, 1, 2, 2.5, 3.25, 4, 4.5, 5, 6 and 8 hours. The microarray covers 2373 genes, 622 out of which are present in the 3-rd line model. For each pair of these genes, the Pearson's correlation coefficient was measured between the expression values. The distribution of all p-values is shown in Figure 6.4. Table 6.6 shows the p-values of the Pearson's correlation coefficients between the hypothetical genes encoding 3.2.1.85 and the genes encoding the enzyme galactose-6-phosphate isomerase (5.3.1.26). The latter acts as the next step in the lactose consumption pathway (see Figure 6.5); the reactions catalysed by both enzymes are elements of the same reaction subset in the third-line model. The genes SAG1103 and SAG1931 have a significant correlation with a p-value of 0.011, which is lower than 90% of all values.

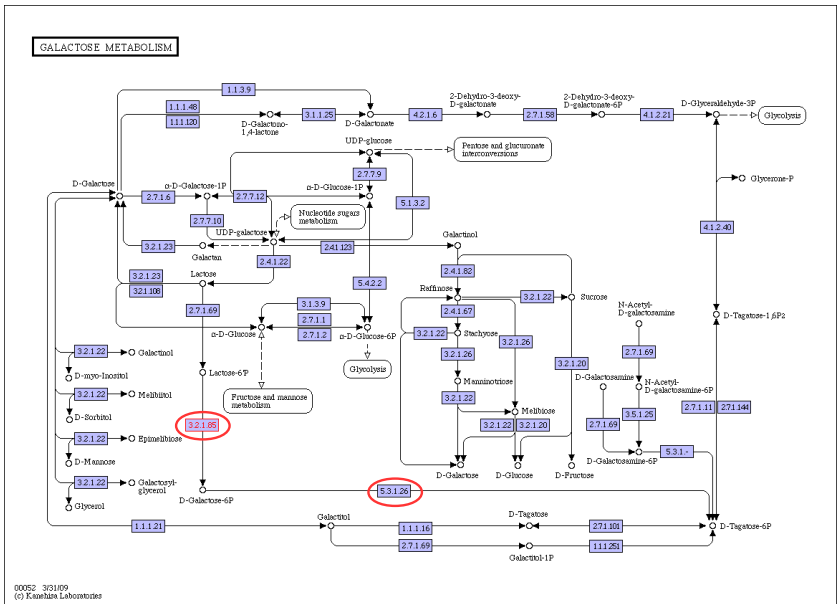


Figure 6.5: KEGG pathway diagram showing the biological functions of the enzyme 3.2.1.85 and 5.3.1.26 (circled).

1.1.1.36: acetoacetyl-CoA reductase catalyses the following reaction:

R01977: 3-Hydroxybutanoyl-CoA + $NADP^+$ ↔ NADPH + Acetoacetyl-CoA

Table 6.5 shows a number of candidate genes for this enzyme; reannotating any of these enzymes leads to some improvement of model quality. The BLASTn search revealed no significant matches for the candidate genes. Therefore, 1.1.1.36 was included into the third-line model as an orphan enzyme.

1.5.1.12: 1-pyrroline-5-carboxylate dehydrogenase catalyses the following reactions:

R04444: L-1-Pyrroline-3-hydroxy-5-carboxylate + NAD^+ + 2 H_2O ↔ NADH + L-erythro-4-Hydroxyglutamate
R04445: L-1-Pyrroline-3-hydroxy-5-carboxylate + $NADP^+$ + 2 H_2O ↔ NADPH + L-erythro-4-Hydroxyglutamate
R00245: L-Glutamate 5-semialdehyde + NAD^+ + H_2O ↔ NADH + Glutamate
R03294: 4-Hydroxy-L-proline + FAD ↔ L-1-Pyrroline-3-hydroxy-5-carboxylate + FADH2
R00708: $NADP^+$ + 2 H_2O + (S)-1-Pyrroline-5-carboxylate ↔ NADPH + Glutamate
R00707: (S)-1-Pyrroline-5-carboxylate + NAD^+ + 2 H_2O ↔ NADH + Glutamate
R05051: NAD^+ + L-erythro-4-Hydroxyglutamate ↔ L-4-Hydroxyglutamate semialdehyde + NADH + H_2O

Both candidate genes for this enzyme (see Table 6.5) encode working enzymes in the second line model; however, reannotating any of them leads to a considerable improvement of model quality. Since BLASTn search revealed no homologies, 1.5.1.12 was included into the third-line model as an orphan enzyme.

4.3.1.19: threonine ammonia-lyase catalyses the following reactions:

R00996: L-Threonine ↔ NH_3 + 2-Oxobutanoate
R00220: Serine ↔ NH_3 + Pyruvate

Table 6.5 shows that the KEGG and PRIAM annotations associate the only candidate gene with the enzyme 2.5.1.47, which is predicted with the E-value of 2e-125. Since the enzyme 4.3.1.19 is predicted with a much higher E-value of 8e-22, the BLASTn search revealed no matches and the reannotation increases the number of working genes by only one, the enzyme was not included into the third-line model.

Table 6.7: Numbers of elements and quality indicators in the models after the reannotations. The symbol ∩ denotes intersections.

line	2-nd				3-rd	
strain	`sag`	`sak`	`san`	`coh`	`sag`	∩
Genes	622	631	635	698	622	NA
% of working genes	46.5	46.4	47.4	44.8	48.7	NA
Enzymes	425	429	424	420	429	406
% of working enzymes	43.5	42.9	43.9	42.4	46.6	42.6
Reactions	624	637	618	617	631	599
% of unbalanced reactions	20.4	20.1	19.4	20.1	21.1	18.7
% of live reactions	50.5	49.8	52.4	48.3	53.7	49.1
Metabolites	662	671	658	658	660	640
% of orphans	41.7	41.4	42.1	41.9	41.2	38.9
Incons	7	9	5	6	6	3

6.4.3 Results

Table 6.7 shows the quantitative characteristics of the second-line models after the manual reannotations and those of the third-line model of `sag`. A comparison with Table 5.6 shows an improvement of the quality indicators as a result of manual reannotations and a further improvement in the third-line model. The models of all three lines are presented on the CD attached.

6.5 Discussion

In this chapter, a semi-automated method for reannotations and gap-filling in genome-scale metabolic models was presented. In contrast to other optimisation-based gap-filling methods [99, 93], which use a 'universal' metabolic database as a source of reactions to be included into the model, our method employs a more specific and reliable data source: a set of prediction lists for genes, where each prediction is based on a homology search and weighted with an alignment score. Hence, instead of attempting to identify missing reactions in a model, the method directly optimises the underlying annotation. Since the search space in this case involves two hierarchical relationships (genes to enzymes and enzymes to reactions) and cannot be represented as a single matrix, the problem is not solvable by linear optimisation methods. Instead, we propose a combination of two complementary approaches: deterministic reduction of the search space and stochastic optimisation.

The choice of simulated annealing as the stochastic optimisation method was motivated primarily by the relative ease of implementation. We hypothesise that better results can be achieved using genetic algorithms [124], which

simulate the natural processes of variability, selection and adaptation. However, the simultaneous representation of a sufficiently large population of metabolic reconstructions in a computer memory is technically problematic and was therefore not done in the scope of the present work. Nevertheless, we combined the general framework of simulated annealing with some approaches widely used in genetic algorithms, including genetic representation (representation of solutions in the form of arrays of discrete variables), mutation operators and roulette wheel selection.

Recurrent invocation of the simulated annealing algorithm and the subsequent identification of the consensus enzymes helps to generate hypotheses about network gaps and reannotations. The reannotations suggested can be further supported by alternative methods, such as BLAST or other homology search tools. Another source of evidence could be provided by gene expression data, since one could expect a co-expression of genes encoding cooperating metabolic enzymes; an example is shown in Table 6.6.

Another question raised, but not answered in the current chapter is the possibility of identifying a globally optimal annotation and its biological implication. It is not clear, whether a global optimum of any of the model quality indicators corresponds or is close to the global minimum of the number of annotation errors. A linear extrapolation of the quality curves shown in Figure 6.3 indicates that the results obtained are relatively far from the global optimum and a further improvement is possible. On the other hand, Table 6.4 demonstrates that the optimised models are more similar to each other than to the original model. Hence, the optimisation tends to move in the same 'direction', which is possibly the direction to the global optimum.

Although the methods presented in this chapter can facilitate and accelerate the process of reannotating a genome, any reannotation requires an experimental verification. In the next chapter, we describe the biological significance of the consensus enzymes 3.2.1.85 and 1.5.1.12 and demonstrate the agreement of the corresponding reannotations with the experimental data found in the literature.

Chapter 7

Functional Analysis

7.1 Introduction

One of the practically relevant problems to be solved in the framework of stoichiometric analysis is the identification of all minimal (irreducible) media able to sustain a microorganism. This problem can be reduced to the calculation of minimal sets of substrates required by the organism for the synthesis of some target products. Then, by specifying the complete biomass as the target product, the minimal media can be identified. The prediction of appropriate proportions of substrate concentrations would increase the informativeness of such a method.

Handorf *et al.* [36] describe the calculation of minimal sets of substrates for a given set of target products using a combination of a greedy algorithm with the network expansion method [17]. However, this method does not guarantee a complete enumeration of minimal sets of substrates and does not predict the appropriate proportions of concentrations. In addition, the expansion method does not take into account the steady state constraint.

A comprehensive description of a biochemical system, including the nutritional requirements, can be obtained using elementary modes analysis. Unfortunately, the calculation of the complete set of elementary flux modes is subject to combinatorial explosion and is therefore not computationally feasible in genome-scale models. Moreover, the number of elementary flux modes in such models can exceed several millions, so their analysis and classification becomes an important problem of its own.

Each flux mode is characterised by its net conversion, which represents its effect on the environment of the organism. Any net conversion can be considered as a particular metabolic function of an organism; hence, the term *functional analysis* was used by Urbanczik and Wagner [118] to describe the analysis of the interaction of a metabolism with its external chemical environment. The authors introduced the conversion cone - the set of all possible net conversions. Further, they proposed a method for the calculation of the elementary vectors of this cone. The analysis of the conversion cone provides a valuable information

about the functional capabilities and nutritional requirements of an organism. However, further we demonstrate that the calculation of elementary net conversions does not enable the enumeration of minimal sets of substrates.

In this chapter, we introduce the concept of feasible substrate compositions - vectors representing the amounts of substrates which can be converted by the network into the target set of products, specified with the desired proportions. We also define the inverse concept of feasible product compositions, representing the amounts of products which can be synthesised by the network from a given composition of substrates. The sets of substrate and product compositions are bounded convex polyhedrons in the space of metabolite amounts; we define the vertices of these polyhedrons as elementary substrate and product compositions, respectively. We propose algorithms for the calculation of complete sets of elementary net conversions and elementary substrate and product compositions in a reversible network. Being based on left nullspace analysis, these methods are highly computationally efficient and scalable to genome-scale models. Alternatively, we propose methods for calculation of minimal substrate and product sets (without the appropriate proportions) with respect to reaction irreversibilities.

To describe the possible practical applications of the methods introduced, we apply them to the genome-scale metabolic models constructed at the previous stages of the work.

7.2 Concepts and methods

In this section, we consider a metabolic network $\mathcal{N} = (\hat{\mathbf{N}}, k, r)$ of n reactions and m metabolites, as defined in Chapter 2.

7.2.1 Conversion cone

The equation $\dot{\mathbf{c}} = \hat{\mathbf{N}}\mathbf{v}$ (Eq. 2.13) describes the metabolite concentration changes as a result of any flux in a network. The flux vector $\mathbf{v}$ can be decomposed as $(\mathbf{v}^{rev}|\mathbf{v}^{irr})^T$, where $\mathbf{v}^{rev} \in \mathbb{R}^r$ and $\mathbf{v}^{irr} \in \mathbb{R}^{n-r}$ are the subvectors of reversible and irreversible reactions, respectively. Similarly, the conversion vector $\dot{\mathbf{c}}$ can be decomposed as $\dot{\mathbf{c}} = (\dot{\mathbf{c}}^{int}|\dot{\mathbf{c}}^{ext})^T$, where $\dot{\mathbf{c}}^{int} \in \mathbb{R}^k$ and $\dot{\mathbf{c}}^{ext} \in \mathbb{R}^{m-k}$ contain the concentration changes of internal and external metabolites, respectively. At a steady state, the concentrations of internal metabolites are constant:

$$\dot{\mathbf{c}}^{int} = \mathbf{0} \tag{7.1}$$

Note that the combination of Equations 2.13 and 7.1 is equivalent to Equation 2.9. Hence, any non-zero vector $\mathbf{v}$ satisfying Equations 2.13, 7.1 and 2.11 is a flux mode. We use the term *net stoichiometry* to refer to any vector $\dot{\mathbf{c}}$ satisfying Equation 2.13 and the term *net conversion* to refer to a net stoichiometry of a flux mode, e.g. a vector $\dot{\mathbf{c}}$ satisfying Equations 2.13, 7.1 and 2.11.

The set of net conversions is a polyhedral convex cone, denoted $\mathcal{C}$ and called the *conversion cone* [118]. Interestingly, the zero vector may or may not be a

net conversion, depending on whether the network contains internal cycles with zero net conversions, since the flux modes themselves are non-zero per definition. For the sake of consistency, we will assume that any conversion cone includes the origin:

$$\mathcal{C} = \hat{\mathbf{N}}\mathcal{F} \cup \{\mathbf{0}\} \tag{7.2}$$

We assume that any net conversion of a set of reactants into itself is equivalent to the zero conversion. We define such net conversions (including the zero conversion itself) as *trivial*.

The *elementary net conversions* [118] represent the simplest possible conversions of external substrates to products; we denote their set by $\mathcal{E}(C)$. This concept is illustrated in Figure 7.1. The network depicted contains nine elementary flux modes, but performs only three non-trivial elementary net conversions, which can be written as: xA $\leftrightarrow$ xB, xA $\leftrightarrow$ xC and xB $\leftrightarrow$ xC. EM_1 is an internal cycle; hence, its net conversion is trivial. The net conversions of EM_2-EM_8 are elementary, whereas the net conversion of EM_9 can be represented as a sum of two elementary ones. This example demonstrates that the mapping from the set of elementary modes into $\mathcal{E}(C)$ is neither injective nor surjective. On the other hand, each elementary net conversion in this network is an image of at least one elementary mode. However, in Section 7.2.6 we demonstrate that this property is not generalisable.

7.2.2 Substrate and product compositions

We define a *composition* as any non-negative vector in $\mathbb{R}^m$. We consider the components of a composition as numbers of units of amount of substance, such as molecules or moles. Similarly to net conversions, an external composition $\mathbf{s}$ can be decomposed as $(\mathbf{s}^{int}|\mathbf{s}^{ext})^T$, $\mathbf{s}^{int} \in \mathbb{R}^k$, $\mathbf{s}^{ext} \in \mathbb{R}^{m-k}$. We define $\mathbf{s}$ as *external*, iff $\mathbf{s}^{int} = \mathbf{0}$; hence, an external composition defines the chemical composition of some environment of the network. It is convenient to represent compositions as multisets, using Python dictionary notation and skipping zero components, e.g. {xA : 1} and {xB : 1} instead of $(1\ 0)^T$ and $(0\ 1)^T$ in the net conversion shown below:

$$\text{xA} \leftrightarrow \text{xB} \tag{7.3}$$

We define the *substrate composition* and the *product composition* of a net conversion $\dot{\mathbf{c}}$ as external compositions $\dot{\mathbf{c}}^-$ and $\dot{\mathbf{c}}^+$, which satisfy the following condition:

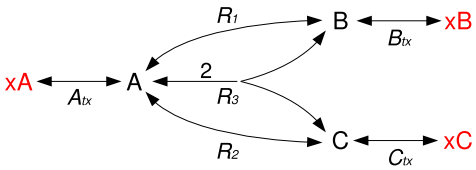
$$-\dot{\mathbf{c}}^- + \dot{\mathbf{c}}^+ = \dot{\mathbf{c}},\ P(\dot{\mathbf{c}}^-) \cap P(\dot{\mathbf{c}}^+) = \emptyset \tag{7.4}$$

Hence, $\dot{\mathbf{c}}^-$ and $\dot{\mathbf{c}}^+$ contain the absolute values of non-positive and non-negative components of $\dot{\mathbf{c}}$, respectively. In Equation 7.3, {xA : 1} is the substrate composition and {xB : 1} is the product composition.

Assume that this is the only non-zero net conversion in some network which is used for the synthesis of the target product denoted xB. Clearly, the ability of the network to produce the required amount of the product depends on the

R_1: B $\leftrightarrow$ A
R_2: A $\leftrightarrow$ C
R_3: 2 A $\leftrightarrow$ B + C
A_{tx}: xA $\leftrightarrow$ A
B_{tx}: B $\leftrightarrow$ xB
C_{tx}: C $\leftrightarrow$ xC

a



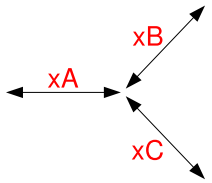
b

	R_1	R_2	R_3	A_{tx}	B_{tx}	C_{tx}	xA	xB	xC
EM_1	-1	1	-1	0	0	0	0	0	0
EM_2	2	0	1	0	-1	1	0	-1	1
EM_3	0	2	-1	0	-1	1	0	-1	1
EM_4	1	1	0	0	-1	1	0	-1	1
EM_5	-1	0	0	1	1	0	-1	1	0
EM_6	0	-1	1	1	1	0	-1	1	0
EM_7	1	0	1	1	0	1	-1	0	1
EM_8	0	1	0	1	0	1	-1	0	1
EM_9	0	0	1	2	1	1	-2	1	1

c

xA	xB	xC	A	B	C
-1	-1	-1	0	0	0

d



e

Figure 7.1: a, b) A network; c) the table of elementary flux modes (left part) and the corresponding net conversions (right part, internal metabolites are omitted); d) transposed external left nullspace matrix; e) its network representation: the external metabolites here are represented by pseudoreactions, 'interconverting' unspecified metabolites. The elementary modes of this network are the elementary net conversions of the original one.

availabily of substrates. We define a substrate composition $\mathbf{s}$ as *feasible* for a product composition $\mathbf{p}$, if the following condition is satisfied:

$$\exists \dot{\mathbf{c}} \in \mathcal{C},\ \mathbf{s} = \dot{\mathbf{c}}^{-},\ \dot{\mathbf{c}}^{+} = \mathbf{p} \tag{7.5}$$

Since the trivial conversion is equivalent to $\mathbf{p} \leftrightarrow \mathbf{p}$, which produces $\mathbf{p}$, the latter is feasible for itself; we define it as the *trivial* substrate composition. The set of feasible substrate compositions for $\mathbf{p}$ is denoted $\mathcal{S}_p$. In the example above, it is easy to find two feasible substrate compositions for xB, namely $\mathbf{s}_1 = \{\text{xA} : 1\}$ and $\mathbf{s}_2 = \{\text{xB} : 1\}$. It is less obvious that any convex combination of $\mathbf{s}_1$ and $\mathbf{s}_2$ is also feasible. We define $\mathbf{s} \in \mathcal{S}_p$ as *elementary*, if it cannot be represented as a convex combination of other elements of $\mathcal{S}_p$:

$$\mathbf{s} \neq \lambda \mathbf{s}' + (1-\lambda)\mathbf{s}'',\ \lambda \in (0,1),\ \mathbf{s}', \mathbf{s}'' \in \mathcal{S}_p \tag{7.6}$$

The only elementary substrate compositions in our example are $\mathbf{s}_1$ and $\mathbf{s}_2$. The set of elementary substrate compositions for $\mathbf{p}$ is denoted $\mathcal{E}(\mathcal{S}_p)$.

The concepts of feasible and elementary substrate compositions can be defined geometrically. Unlike flux and conversion cones, $\mathcal{S}_p$ is not closed under non-negative scalar multiplication, i.e. is not a cone. To explain the relation between $\mathcal{C}$ and $\mathcal{S}_p$, we re-write the equalities $\mathbf{s} = \dot{\mathbf{c}}^{-}$ and $\dot{\mathbf{c}}^{+} = \mathbf{p}$ as $\mathbf{s} = -\dot{\mathbf{c}} + \mathbf{p}$, which is a necessary condition for $\mathbf{s}$ to be an element of $\mathcal{S}_p$. Combining this with the requirement of non-negativity, we obtain a necessary and sufficient condition, which can be applied to the whole conversion cone:

$$\mathcal{S}_p = (-\mathcal{C} + \mathbf{p}) \cap \mathbb{R}^m_+ \tag{7.7}$$

where $\mathbb{R}^m_+$ is the non-negative orthant. The translate of $-\mathcal{C}$ is an unbounded convex polyhedron and its intersection with $\mathbb{R}^m_+$ is a convex polytope. Equation 7.6 is the definition of vertices of $\mathcal{S}_p$, which are thus the elementary substrate compositions. Hence, each feasible substrate composition is a convex combination of some elementary substrate compositions.

The geometry of the set $\mathcal{S}_p$ is shown in Figure 7.2b. Interestingly enough, none of the elementary substrate compositions in this example corresponds to an elementary net conversion. Hence, the mapping $\mathcal{E}(C) \rightarrow \mathcal{E}(Sp)$ is neither injective nor surjective.

Let us consider the inverse problem: which products can be synthesised from given substrates. For a substrate composition $\mathbf{s}$, we define a product composition $\mathbf{p}$ as *feasible* if it satisfies the following condition:

$$\exists \dot{\mathbf{c}} \in \mathcal{C},\ \mathbf{p} = \dot{\mathbf{c}}^{+},\ \dot{\mathbf{c}}^{-} = \mathbf{s} \tag{7.8}$$

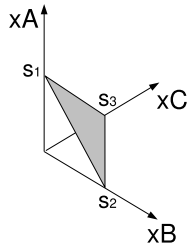
The set of *feasible product compositions* for $\mathbf{s}$ is denoted $\mathcal{P}_s$ and can be redefined as follows:

$$\mathcal{P}_s = (\mathcal{C} + \mathbf{s}) \cap \mathbb{R}^m_+ \tag{7.9}$$

Hence, $\mathcal{P}_s$ is also a convex polytope contained in the positive orthant. Any of its elements is a convex combination of its vertices, which are called *elementary product compositions*.

	xA	xB	xC
$\mathbf{s}_1$	2	0	0
$\mathbf{s}_2$	0	2	0
$\mathbf{s}_3$	0	0	2

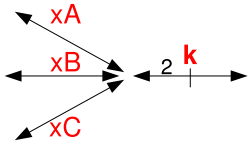
a



b

xA	xB	xC	$-\mathbf{k}$
1	1	1	-2

c



d

Figure 7.2: a) Elementary substrate compositions for {xB : 1, xC : 1} in the network shown in Figure 7.1; b) the same compositions in a 3-dimensional space; the shaded triangle is the set of feasible substrate compositions; c) external left nullspace matrix with the augmented column $-\mathbf{k} = -\mathbf{K_x}^T\mathbf{p}$; d) its network representation: the appropriately scaled elementary modes in this network contain the minimal substrate compositions in the original one as subvectors.

7.2.3 Functional analysis of reversible networks

For a network $\mathcal{N} = (\hat{\mathbf{N}}, k, r)$, we define its reversible version as $\mathcal{N}^0 = (\hat{\mathbf{N}}, k, n)$, i.e. the set of reversible reactions is equal to the complete set of reactions. The sets $\mathcal{C}, \mathcal{S}_p$ and $\mathcal{P}_s$ in this network are denoted as $\mathcal{C}^0, \mathcal{S}_p^0$ and $\mathcal{P}_s^0$, respectively.

Calculation of net conversions The equation $\mathbf{K}^T\dot{\mathbf{c}} = \mathbf{0}$ (Eq. 5.10) enables the calculation of an arbitrary net stoichiometry. In order to restrict the solution set of this equation to the net conversions, we set the rows of $\mathbf{K}$ describing internal metabolites to zero, thus obtaining the *external left nullspace matrix* $\mathbf{K_x}$. The set of net conversions in a reversible network satisfies the following equality:

$$\mathcal{C}^0 = \{\dot{\mathbf{c}} : \mathbf{K_x}^T\dot{\mathbf{c}} = \mathbf{0}\} \tag{7.10}$$

To calculate $\mathcal{E}(\mathcal{C}^0)$, we consider $\mathbf{K_x}^T$ as a stoichiometry matrix of a reversible network and $\dot{\mathbf{c}}$ as a flux vector (see Figure 7.1(d,e)). Since elementary modes and elementary net conversions satisfy the same udecomposability condition [37, 118], the set of elementary modes in this system is the set $\mathcal{E}(\mathcal{C}^0)$. This set can be enumerated by applying one of the existing algorithms for the calculation of elementary modes [110, 90] to Equation 7.10.

Calculation of elementary compositions Equation $\dot{\mathbf{c}}^T\mathbf{m} = 0$ (Eq. 5.4) can be re-written, so that it applies to substrate and product compositions of any $\dot{\mathbf{c}} = \hat{\mathbf{N}}\mathbf{v}$:

$$\mathbf{m}^T\dot{\mathbf{c}}^- = \mathbf{m}^T\dot{\mathbf{c}}^+ \tag{7.11}$$

This equation represents the obvious fact that the total masses of substrates and products in any chemical interconversion are equal. As explained in Chapter 5, the feasible values of $\mathbf{m}$ are spanned by $\mathbf{K}$; hence, $\mathbf{K_x}^T$ can replace $\mathbf{m}^T$ in Equation 7.11, resulting in the following equality:

$$\mathbf{K_x}^T\dot{\mathbf{c}}^- = \mathbf{K_x}^T\dot{\mathbf{c}}^+ \tag{7.12}$$

Combining this with the condition of non-negativity, we obtain the following definition of $\mathcal{S}_p^0$:

$$\mathcal{S}_p^0 = \{\mathbf{s} : \mathbf{K_x}^T\mathbf{s} = \mathbf{K_x}^T\mathbf{p},\ \mathbf{s} \geq \mathbf{0}\} \tag{7.13}$$

The product $\mathbf{K_x}^T\mathbf{p}$ is fixed and can be denoted $\mathbf{k}$:

$$\mathcal{S}_p^0 = \{\mathbf{s} : (\mathbf{K_x}^T| - \mathbf{k})(\mathbf{s}|1)^T = \mathbf{0},\ \mathbf{s} \geq \mathbf{0}\} \tag{7.14}$$

If we still consider $\mathbf{K_x}^T$ as a stoichiometry matrix of a network, then augmenting $-\mathbf{k}$ at its right side is equivalent to including a transporter reaction (see Figure 7.2(c, d)). The non-negativity condition implies that the network in this case must be considered as completely irreversible. Any flux mode in this network has a positive value in the last component; since a flux mode $(\mathbf{s}|0)^T$ would correspond to a semipositive solution of the system $\mathbf{K}^T\dot{\mathbf{c}} = \mathbf{0}$, i.e. to an

inconsistent net conversion. Hence, the set of solutions of form $(\mathbf{s}|1)^T$ is a cross-section of the flux cone and the set $\mathcal{S}_p$ is a projection of this cross-section. The vertices of $\mathcal{S}_p$ correspond to the vertices of the cross-section, which lie on the generating vectors of the flux cone. In an irreversible network, the generating vectors of the flux cone correspond to the elementary modes [37]. Hence, any elementary substrate composition $\mathbf{s}$ corresponds to an elementary mode $(\mathbf{s}|1)^T$ and vice versa. To calculate $\mathcal{E}(\mathcal{S}_p)$, one can detect the elementary modes in the system described above and scale them appropriately (see Algorithm 8).

Algorithm 8 Detect the elementary substrate compositions for a product composition $\mathbf{p}$ in a reversible network. Input: $\mathcal{N}^0 = (\hat{\mathbf{N}}, k, n)$, $\mathbf{p}$. Output: $E = \mathcal{E}(\mathcal{S}_p^0)$

$E := \{\}$
$\mathbf{K_x} := \text{nullspace}(\hat{\mathbf{N}}^T)$
for all $i \in (1, k)$: **do**
  $\mathbf{K}_{\mathbf{x}(i)} := \mathbf{0}^T$ //set the rows describing internal metabolites to zero.
end for
$\mathbf{k} = \mathbf{K_x}^T\mathbf{p}$
$F := \text{el.modes}((\mathbf{K_x}^T| - \mathbf{k}), irreversible)$
for all $(\mathbf{s}|\lambda)^T \in F$: **do**
  $E := E \cup \{\frac{1}{\lambda}\mathbf{s}\}$
end for

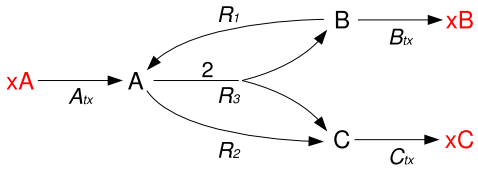
Net conversions, substrate and product compositions in a reversible network have the following properties:

1. *A linear combination of net conversions is a net conversion.*
2. *A net conversion, substrate or product composition is elementary iff it is minimal, i.e. its support is irreducible.*
3. *An elementary net conversion is uniquely defined by its support up to scalar multiples.*
4. *For any composition, the set of minimal substrate compositions is equal to the set of minimal product compositions.*

In particular, (1) follows from Equation 7.10; (2) is a property of elementary modes [110]; (3) is proved in Proposition 3 (see below); (4) follows from the reversibility of conversions. Hence, Algorithm 8 can be also used for the calculation of elementary product compositions.

Proposition 3 *In a reversible network, if* $\dot{\mathbf{c}}$ *and* $\dot{\mathbf{c}}'$ *are net conversions and* $\dot{\mathbf{c}}$ *is elementary, then* $P(\dot{\mathbf{c}}) = P(\dot{\mathbf{c}}')$ *implies* $\dot{\mathbf{c}} = \lambda\dot{\mathbf{c}}'$, $\lambda \in \mathbb{R}_+^m$.

Proof Let $\dot{\mathbf{c}}^* = \dot{\mathbf{c}} - \lambda\dot{\mathbf{c}}'$, such that $\dot{c}_i^* = 0$ for some $i \in P(\dot{\mathbf{c}})$. If $\dot{\mathbf{c}}^* = \mathbf{0}$, then $\dot{\mathbf{c}} = \lambda\dot{\mathbf{c}}'$. Otherwise, $P(\dot{\mathbf{c}}^*) \subset P(\dot{\mathbf{c}})$, hence $\dot{\mathbf{c}}$ is not minimal and therefore not elementary $\square$



a

	A_{tx}	B_{tx}	C_{tx}	R_1	R_2	R_3	xA	xB	xC
EM_1	1	0	1	1	0	1	-1	0	1
EM_2	0	1	0	1	0	1	-1	0	1
EM_3	0	0	1	2	1	1	-2	1	1

b

Figure 7.3: a) A completely irreversible network; b) elementary flux modes and the corresponding net conversions. All net conversions are irreversible; two of them are elementary, but only one is minimal.

7.2.4 Feasibility of net conversions

Irreversibility of fluxes may result in limitations on the possible interconversions in a metabolic network: some of them may become infeasible in either one or both directions. An example is shown in Figure 7.3, which is a completely irreversible version of the network depicted in Figure 7.1. The feasibility of a net conversion can be identified using the following equation, which is equivalent to the equation $\dot{\mathbf{c}} = \hat{\mathbf{N}}\mathbf{v}$:

$$(\hat{\mathbf{N}}|-\dot{\mathbf{c}})(\mathbf{v}|1)^T = \mathbf{0} \tag{7.15}$$

Augmenting a negated net conversion at the right side of $\hat{\mathbf{N}}$ is equivalent to including an exchange reaction, which consumes the products and returns the substrates, thus satisfying the steady state condition. So, $\dot{\mathbf{c}}$ is feasible, iff the exchange reaction is able to carry a steady state flux in the opposite direction. This ability can be tested in a linear program:

$$\begin{array}{rl} \text{Maximise} & \lambda \\ \text{Subject to} & (\hat{\mathbf{N}}|-\dot{\mathbf{c}})(\mathbf{v}|\lambda)^T = \mathbf{0}, \\ \text{Where} & -\mathbf{1} \leq \mathbf{v}_{rev} \leq \mathbf{1}, \\ & \mathbf{0} \leq \mathbf{v}_{irr} \leq \mathbf{1} \end{array} \tag{7.16}$$

$\dot{\mathbf{c}}$ is feasible, iff the objective value is positive.

The feasibility of a substrate composition $\mathbf{s}$ for $\mathbf{p}$ or vice versa can be tested by applying the linear program to $\dot{\mathbf{c}} = -\mathbf{s} + \mathbf{p}$. In Figure 7.3, the elementary substrate compositions for {xB : 1, xC : 1} are {xA : 2}, {xA : 1, xB : 1} and {xB : 1, xC : 1}.

7.2.5 Calculation of flux modes

A slight modification of Equation 7.15 enables the detection of a flux mode performing the net conversion $\dot{\mathbf{c}}$ in the split external stoichiometry matrix $\ddot{\mathbf{N}} = (\hat{\mathbf{N}}| - \hat{\mathbf{N}})$. Similarly to Equation 7.15, the negated net conversion can be augmented at the right side of the split matrix:

$$(\ddot{\mathbf{N}}| - \dot{\mathbf{c}})(\ddot{\mathbf{v}}|1)^T = \mathbf{0} \tag{7.17}$$

Then we use linear or mixed-integer programming to calculate a non-negative solution satisfying Equations 3.9 and 7.17. For instance, the sum of all flux rates can be minimised in the following LP:

$$\begin{array}{rl} \text{Minimise} & \sum_{i=1}^{2n} \ddot{\mathbf{v}}_i \\ \text{Subject to} & (\ddot{\mathbf{N}}| - \dot{\mathbf{c}})(\ddot{\mathbf{v}}|1)^T = \mathbf{0}, \\ & {}^{-}\mathbf{v}^{irr} = \mathbf{0}, \\ \text{Where} & \ddot{\mathbf{v}} \geq \mathbf{0} \end{array} \tag{7.18}$$

The MILP used in the method `ShortestMode` (calculates the shortest flux mode involving a given reaction, see Eq. 3.15) enables the minimisaion of the number of non-zero flux rates in the solution. The use of integer cuts enables the exclusion of solutions already found from the feasible solution set.

The objective function in the first line can be also redefined as the minimisation or maximisation of the flux rate in some target reaction. This enables, for instance, the calculation of a fermentation flux mode with a given net conversion and maximal yield of ATP, by maximising the flux rate of ATPase.

7.2.6 Functional analysis of standard networks

For practical purposes, it is often useful to know the sets of feasible substrates and products, even if the exact proportions are unknown. We define a set of external metabolites as a *minimal substrate set* or a *minimal product set* if it represents a support of a minimal substrate or product composition, respectively.

Clearly, the minimal substrate and product sets can be easily found in the sets of elementary substrate or product compositions, respectively (e.g. in Figure 7.3, {xA : 1} is the only non-trivial minimal substrate composition for {xB : 1, xC : 1} and the corresponding minimal substrate set is {xA}). In this section, we propose an alternative method, which enables a direct calculation of these sets, with respect to the irreversibility of reactions.

We define a network as *standard*, if for each $i > k$, $\hat{\mathbf{N}}_{(i)}$ contains exactly one nonzero component. We denote its index by $tx(i)$. In other words, each external metabolite is involved in exactly one reaction, which acts as its transporter. The transporter flux rates and the net conversions in a standard network are related as follows:

$$\dot{c}_i = \hat{\mathbf{N}}_{i,tx(i)} v_{tx(i)}, \quad k < i \leq m \tag{7.19}$$

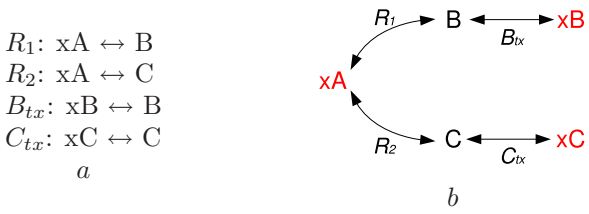


Figure 7.4: A non-standard network. The elementary net conversion xB ↔ xC corresponds only to the non-elementary flux mode involving all reactions.

This implies, in particular, that the support of a net conversion can be easily derived from the set of non-zero transporter flux rates and vice versa, thus enabling the proof of the following proposition, which states that in a standard reversible network (such as the network shown in Figure 7.1), each elementary net conversion is an image of at least one elementary flux mode:

Proposition 4 *In a standard reversible network with a stoichiometry matrix* $\hat{\mathbf{N}}$*, for each elementary net conversion* $\dot{\mathbf{c}}$ *there exists at least one elementary flux mode* $\mathbf{v}'$*, such that* $\dot{\mathbf{c}} = \hat{\mathbf{N}}\mathbf{v}'$*.*

Proof Let $\dot{\mathbf{c}} = \hat{\mathbf{N}}\mathbf{v}^*$, where $\mathbf{v}^*$ is not elementary. There exists an elementary mode $\mathbf{v}'$, such that $P(\mathbf{v}') \subset P(\mathbf{v}^*)$. We consider three possible relations between $\dot{\mathbf{c}}$ and $\dot{\mathbf{c}}' = \hat{\mathbf{N}}\mathbf{v}'$:

1. If $P(\dot{\mathbf{c}}') \subset P(\dot{\mathbf{c}})$, then $\dot{\mathbf{c}}$ is not minimal and therefore not elementary.
2. If $P(\dot{\mathbf{c}}) \subset P(\dot{\mathbf{c}}')$, then $\mathbf{v}^*_{tx(i)} = 0$ and $\mathbf{v}'_{tx(i)} \neq 0$, for each $i \subset P(\dot{\mathbf{c}}') \setminus P(\dot{\mathbf{c}})$; hence, $P(\mathbf{v}') \not\subseteq P(\mathbf{v}^*)$.
3. If $P(\dot{\mathbf{c}}) = P(\dot{\mathbf{c}}')$, then, according to Proposition 3, either $\dot{\mathbf{c}}$ is not elementary or $\dot{\mathbf{c}} = \dot{\mathbf{c}}' = \hat{\mathbf{N}}\mathbf{v}'$ □

Figure 7.4, in contrast, shows the violation of this property in a non-standard network.

Equation 7.19 provides the possibility of finding net conversions in distributions of transporter fluxes, which can be calculated using linear programming methods. For each $i > k$, we define the index of the import flux rate as follows:

$$in(i) = \begin{cases} tx(i), & \hat{\mathbf{N}}_{i,tx(i)} < 0 \\ tx(i) + n, & \hat{\mathbf{N}}_{i,tx(i)} > 0 \end{cases} \tag{7.20}$$

We also define the index of the export flux rate as $out(i) = (in(i) + n) \ \% \ 2n$. To find some feasible substrate composition $\mathbf{s}$ for $\mathbf{p}$, we assign the export flux rates as follows:

$$\ddot{\mathbf{v}}_{out(i)} = p_i, \ k < i \leq m \tag{7.21}$$

Note that only the flux rates of the products involved in $\mathbf{p}$ are fixed; this is to allow the synthesis of by-products. Then we calculate any non-negative solution $\ddot{\mathbf{v}}$ satisfying the equations $\ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0}$ (Eq. 3.8), ${}^{-}\mathbf{v}^{irr} = \mathbf{0}$ (Eq. 3.9) and Equation 7.21. The distribution of net flux rates is $\mathbf{v} =^{+} \mathbf{v} -^{-} \mathbf{v}$, the net conversion is $\dot{\mathbf{c}} = \hat{\mathbf{N}}\mathbf{v}$ and the substrate composition is $\mathbf{s} = -\dot{\mathbf{c}} + \mathbf{p}$. To ensure that $\mathbf{s}$ is minimal, we minimise the number of positive import flux rates using MILP:

$$\begin{array}{rl} \text{Minimise} & \sum_{i=k+1}^{m} q_{in(i)} \\ \text{Subject to} & \ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0}, \\ & {}^{-}\mathbf{v}^{irr} = \mathbf{0} \\ & \ddot{v}_{out(i)} = \lambda p_i,\ k < i \leq m \\ \text{Where} & 0 \leq \ddot{v}_i \leq q_i,\ q_i \in \{0,1\},\ 0 \leq i \leq 2n \end{array} \tag{7.22}$$

Here λ is a scaling factor. Since the upper bound of the components of $\ddot{\mathbf{v}}$ is 1, λ must be sufficiently small, so that all scaled flux rates remain in this range. Using integer cuts, it is possible to calculate all feasible solutions. The supports of the compositions detected represent all minimal substrate sets (see Algorithm 9).

Algorithm 9 Identify the set M of minimal substrate sets for a product composition $\mathbf{p}$. Input: $\mathcal{N} = (\hat{\mathbf{N}}, r, k)$, $\mathbf{p}$, λ. Output: M.

$M := \{\}$
$\mathbf{N} := \text{internal}(\hat{\mathbf{N}}, k)$
$\ddot{\mathbf{N}} = \text{split}(\hat{\mathbf{N}})$
$prog := \text{MILP}(\ddot{\mathbf{N}}, \text{r}, \epsilon)$ /* Eq. 7.22*/
$\mathbf{q}, ({}^{+}\mathbf{v}|{}^{-}\mathbf{v}) := \text{minimal\_solution}(prog)$
while is_feasible($prog$) **do**
  $\mathbf{v} := \frac{1}{\lambda}({}^{+}\mathbf{v} -^{-} \mathbf{v})$
  $\dot{\mathbf{c}} := \hat{\mathbf{N}}\mathbf{v}$
  $\mathbf{s} := -\dot{\mathbf{c}} + \mathbf{p}$
  $M := M \cup \{P(\mathbf{s})\}$
  set_integer_cut($prog, \mathbf{q}$) /* Eq. 3.7*/
  $\mathbf{q}, ({}^{+}\mathbf{v}|{}^{-}\mathbf{v}) := \text{minimal\_solution}(prog)$
end while

Similarly, a minimal feasible composition $\mathbf{p}$ for $\mathbf{s}$ can be calculated using the MILP shown in Equation 7.23, where $\mathbf{p} = \dot{\mathbf{c}} + \mathbf{s}$:

$$\begin{array}{rl} \text{Minimise} & \sum_{i=k+1}^{m} q_{out(i)} \\ \text{Subject to} & \ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0}, \\ & {}^{-}\mathbf{v}^{irr} = \mathbf{0} \\ & \ddot{v}_{in(i)} = \epsilon s_i,\ k < i \leq m \\ \text{Where} & 0 \leq \ddot{v}_i \leq q_i,\ q_i \in \{0,1\},\ 0 \leq i \leq 2n \end{array} \tag{7.23}$$

A slight modification of Algorithm 9 enables the detection of all minimal product sets for $\mathbf{s}$.

Table 7.1: Elementary substrate compositions for one unit of lactate. The composition shown in column 6 is only present in the models of `san` and the third-line model of `sag`.

	1	2	3	4	5	6
Glucose		1/2				
Fructose			1/2			
Lactose					1/4	
Mannose						1/2
Ribose				3/5		
Galactose						
Sucrose	1/4					
H_2O	1/4				1/4	

Inclusion of by-products It is often not possible or not desirable to precisely define the complete composition, since only one or a few target products are of practical interest. For instance, if the target product is protein, its synthesis may be combined with the release of water and ammonia, whose exact proportions vary in different compositions. To allow the export of by-products in arbitrary proportions, we define the export flux rates of the target products only:

$$\ddot{\mathbf{v}}_{out(i)} = p_i,\ i \in P(p) \tag{7.24}$$

The fourth line in Equation 7.22 is modified correspondingly. Then, in the resulting substrate composition $\mathbf{s} = -\dot{\mathbf{c}} + \mathbf{p}$, the by-products occur with negative coefficients.

7.3 Application to genome-scale models

The methods described above were applied to the first, second and third-line models of *S. agalactiae*, to investigate the processes of fermentation and protein biosynthesis. The set of transporters in each case was completely redefined (apart from a permanent transporter for water).

7.3.1 Fermentation substrates

Since the organism is known to perform lactic acid fermentation, an irreversible exporter was included for lactate. To identify the possible fermentation substrates, irreversible importers were included for the sugars shown in Table 5.3. The elementary substrate compositions were calculated for one unit of lactate, using the methods for reversible and standard networks; the results of both methods were identical for each model.

The results are shown in Table 7.1 and are summarised in the following conclusions:

Table 7.2: Genes in `sak` and `coh` for which the enzyme 3.2.1.85 was predicted as a suboptimal hit and the results of reannotations in second-line models. The columns from left to right show the gene or strain identifiers, KEGG annotations, PRIAM best hits with the E-values, E-values for 3.2.1.85 and the increase in the number of working genes as a result of a reannotation.

ID, strain	**KEGG**	**best**	**3.2.1.85**	+
sak				
SAK_1188	3.2.1.86	3.2.1.86: 0.0	7e-79	5
SAK_0916	3.2.1.86	3.2.1.86: 0.0	2e-81	5
coh				
SAN_0887	NA	3.2.1.86: 0.0	6e-82	5
SAN_1223	NA	3.2.1.86: 0.0	3e-79	5
SAN_1249	NA	3.2.1.86: 6e-101	5e-34	5

- All models ferment glucose, fructose, mannose, ribose, and sucrose.
- None of the models ferments galactose.
- Both models of `san` and the third-line model of `sag` ferment lactose.

The inability of the first and second-line models of `sag`, `san` and `coh` to ferment lactose is caused by the absence of the enzyme 3.2.1.85 (6-phospho-beta-galactosidase, see Figure 6.5), which is present in both annotations of `san` and was included into the third-line model (see Chapter 6). This enzyme is predicted as a suboptimal hit also in the other PRIAM annotations and a reannotation in each case leads to an improvement of model quality (see Table 7.2). Note that the best hit in each case is the 3.2.1.86 which has the same name but operates on different substrates.

7.3.2 Fermentation products

To identify the fermentation types present in the organism, transporters were included for glucose (irreversible importer) and for typical fermentation products, namely CO_2, lactate, acetate, formate, acetoin, ethanol, diacetyl, fumarate, propionate, H_2, butyrate, acetone, methane and succinate (irreversible exporters).

In order to reflect the dependency of fermentation modes upon the availability of oxygen and the possibility of oxidative phosphorylation, the hypothetical NADH and NADPH oxidases were redefined as follows:

$$\begin{aligned} &\text{NADH\_Oxidase: } O_2 + 2\ \text{NADH} \rightarrow 2\ \text{NAD}^+ + 2\ H_2O \\ &\text{NADPH\_Oxidase: } O_2 + 2\ \text{NADPH} \rightarrow 2\ \text{NADP}^+ + 2\ H_2O \end{aligned} \tag{7.25}$$

Hence, these reactions were effectively blocked in the absence of an oxygen transporter. Then, compositions of fermentation products were calculated in anaerobic and aerobic conditions.

Anaerobic conditions The substrate composition was defined as one unit of glucose and two units of water, to allow hydrolysis reactions.

Table 7.3a shows the union of the compositions calculated in all models using left null space analysis, under the assumption of reversibility. Each composition except {ethanol : 1.5, formate : 3} is directionally feasible in at least one of the models. The compositions involving diacetyl are not present in the first-line models of `sag` and `san` because of the absence of acetoin dehydrogenase.

Table 7.4a shows the results calculated using MILP (note that these compositions are not necessarily elementary, although definitely minimal). The compositions involve all experimentally known fermentation products of the organism [71], plus diacetyl and fumarate. For each composition, in each model an elementary flux mode was calculated with a maximised flux rate of ATPase. The composition {acetate : 3} is of particular interest, since to our knowledge, fermentation of glucose into three molecules of acetate has not been described in the biochemical literature. Figure 7.5 shows a possible pathway of this fermentation, which is strongly different from typical glycolytical pathways. Instead, it involves a number of enzymes used in the pentose-phosphate pathway and photosynthesis, such as transketolase, phosphoketolase, ribose-5-phosphate isomerase, ribulose-phosphate 3-epimerase and triose-phosphate isomerase. Aldolase is also used in a manner which is typical for photosynthesis, converting glycerone phosphate and erythrose 4-phosphate into sedoheptulose 1,7-bisphosphate. The only reactions consuming and producing ATP are those of glucokinase and acetate kinase, respectively; the net production of ATP is 2 per 1 mole of glucose. The pathway is not present in the first-line models of `sag` and `san` because of the absence of phosphoketolase. It is also not present in the second and third-line models, since fructose-bisphosphatase was not predicted for any genes by RPS-BLAST. Interestingly enough, analysis of elementary modes revealed that because of the absence of fructose-bisphosphatase these models are also not able to convert glucose into diacetyl and CO_2.

The maximal ATP yields shown in Table 7.4a varies between 0 and 2, except for the compositions {acetate : 2, formate : 1, ethanol : 0.5} and {acetate : 1.71, ethanol : 0.64, formate : 1.29}, which yield 3 molecules of ATP in most of the models. An elementary mode corresponding to the former composition is shown in Figure 7.6. It consists of two parallel routes: The route shown in the upper part of the figure involves glycolytical enzymes, yields ATP in the phosphoglycerate kinase reaction and ends up with ethanol and formate. The route in the lower part involves the enzymes of pentose-phosphate pathway, yields ATP in the acetate kinase reaction and ends up with acetate. The first-line models of `sag` and `san` cannot operate the latter route because of the absence of phosphoketolase.

Aerobic conditions A reversible transporter was included for oxygen and the substrate composition was defined as one unit of glucose, two units of water and six units of oxygen. The results are shown in Tables 7.3b and 7.4b; the rows named 'O_2 balance' show the balance of oxygen in the corresponding net

Table 7.3: Elementary compositions of fermentation products in anaerobic (a) and aerobic (b) conditions, calculated using left null space analysis. The lower part of each table shows the presence of the corresponding conversions in the reversible versions of the models (non-empty cells) and their directional feasibilities ('+').

	1	2	3	4	5	6	7	8	9	10	11
Acetate			3								
Acetoin						1	3/4	6/5			
CO_2				2				6/5			2/3
Diacetyl									6/5	1	4/3
Ethanol		1		2	3/2						
Formate					3	2			6/5		
Fumarate		1					3/4			1/2	
Lactate	2										
1-st line:											
sag	+	+	-	+	-	+	-	-			
sak	+	+	+	+	-	+	+	+	+	+	+
san	+	+	-	+	-	+	-	-			
2-nd line:											
sag	+	+	-	+	-	+	+	+	+	+	-
sak	+	+	-	+	-	+	+	+	+	+	-
san	+	+	-	+	-	+	+	+	+	+	-
coh	+	+	-	+	-	+	+	+	+	+	-
3-rd line:											
sag	+	+	-	+	-	+	+	+	+	+	-

a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Acetate									1			3		
Acetoin	1/2	3/2												
CO_2			6								2/3			
Diacetyl													1/3	3/2
Ethanol	2			5/2	2		2		2		8/3		7/3	
Formate							2			6				
Fumarate				1/4		3/2								
Lactate					2/3			2						
O_2 balance	5/2	3/2	-6	9/4	2	-3/2	1	0	2	-3	2	0	5/2	3/4
1-st line:														
sag	-	-	-	-	-	-	-	+	-	-	-	-		
sak	-	-	-	-	-	-	-	+	-	-	-	+	-	-
san	-	-	-	-	-	-	-	+	-	-	-	-		
2-nd line:														
sag	-	-	-	-	-	-	-	+	-	-	-	-	-	-
sak	-	-	-	-	-	-	-	+	-	-	-	-	-	-
san	-	-	-	-	-	-	-	+	-	-	-	-	-	-
coh	-	-	-	-	-	-	-	+	-	-	-	-	-	-
3-rd line:														
sag	-	-	-	-	-	-	-	+	-	-	-	-	-	-

b

Table 7.4: Minimal compositions of fermentation products in anaerobic (a) and aerobic (b) conditions, calculated using MILP. The lower part of each table shows the presence in the models (non-empty cells) and maximal yields of ATP per unit of glucose.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Acetate				2			3					2.52	0.80	1.71
Acetoin			1					0.75			1.2			
CO_2					2					0.67	1.2			
Diacetyl						1			1.2	1.33				
Ethanol		1		0.5	2							0.24	1.1	0.64
Formate			2	1					1.2			0.48	2.2	1.29
Fumarate		1				0.5		0.75						
Lactate	2													
1-st line:														
sag	2	0	2	2	2									2.29
sak	2	0	2	3	2	0	2	0	1.2	0.33	1.2	2.48		3
san	2	0	2	2	2									2.29
2-nd line:														
sag	2	1	2	3	2	0.5		0.75	1.2		1.2	0.78	1.2	3
sak	2	1	2	3	2	0.5		0.75	1.2		1.2	0.78	1.2	3
san	2	1	2	3	2	0.5		0.75	1.2		1.2	0.78	1.2	3
coh	2	1	2	3	2	0.5		0.75	1.2		1.2	0.78	1.2	3
3-rd line:														
sag	2	1	2	3	2	0.5		0.75	1.2		1.2	0.78	1.2	3

a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Acetate			2		2	1		3		2.71	1.1	2.08	1.95	2.68	1.89	2.59	2.73
Acetoin		1		1													
CO_2				2	2		2			0.58		1.84		0.63			0.53
Diacetyl									1								
Ethanol							2										
Formate		2	2						2				2.10		2.22	0.82	
Fumarate						1					0.95						
Lactate	2																
O_2 balance			-1	-1	-2	-1			-0.5	-0.58	-0.95	-1.84	-1.05	-0.63	-1.11	-0.41	-0.53
1-st line:																	
sag	2	2	4	2	4	1	2										
sak	2	2	4	2	4	1	2	2	2	2.87	1.1	3.92		2.95		3.24	2.80
san	2	2	4	2	4	1	2										
2-nd line:																	
sag	2	2	4	2	4	2	2		2	0.33	2.05	3.92	3.12	0.74	2.16	2.18	0
sak	2	2	4	2	4	2	2		2	0.33	2.05	3.92	3.12	0.74	2.16	2.18	0
san	2	2	4	2	4	2	2		2	0.33	2.05	3.92	3.12	0.74	2.16	2.18	0
coh	2	2	4	2	4	2	2		2	0.33	2.05	3.92	3.12	0.74	2.16	2.18	0
3-rd line:																	
sag	2	2	4	2	4	2	2		2	0.33	2.05	3.92	3.12	0.74	2.16	2.18	0

b

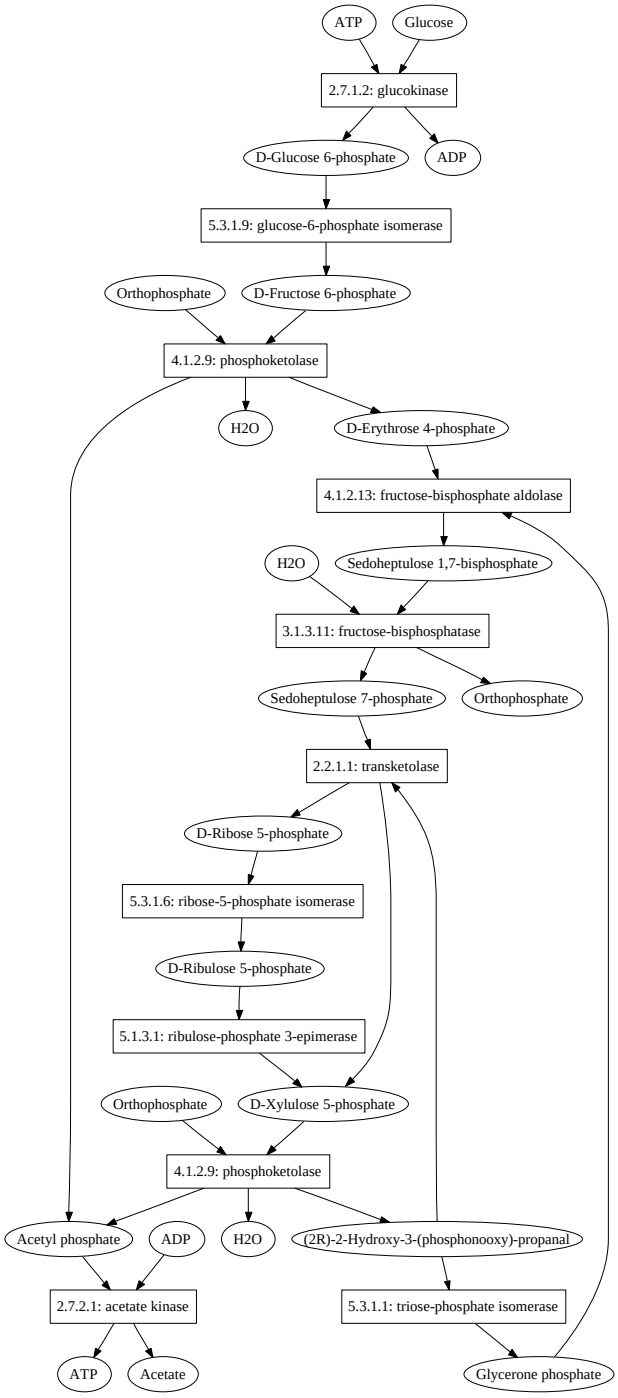


Figure 7.5: Elementary mode converting one mole of glucose into three moles of acetate, yielding two moles of ATP. Hypothetical reactions are not shown.

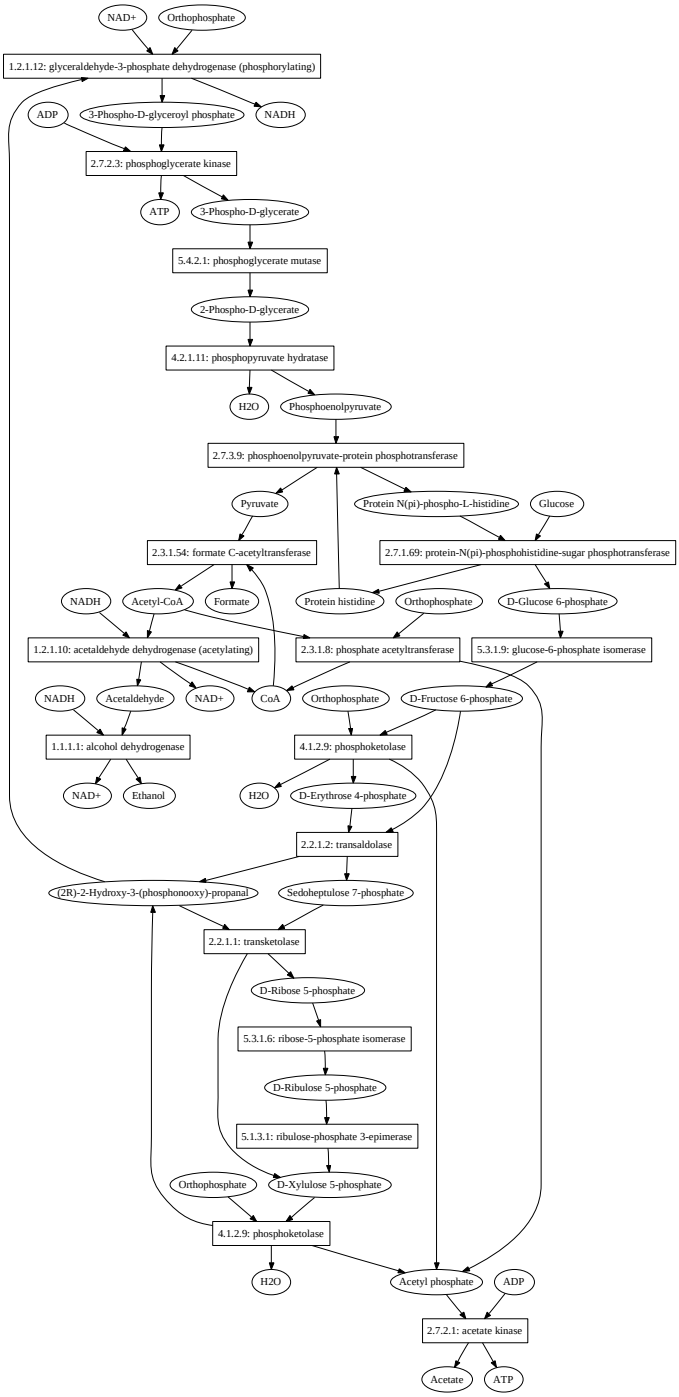


Figure 7.6: Elementary mode converting one mole of glucose into the composition {acetate : 2, formate : 1, ethanol : 0.5} and yielding three moles of ATP. Hypothetical reactions are not shown.

conversions.

Most of the elementary product compositions in Table 7.3a are infeasible in all models; the only exceptions correspond to the conversions of glucose into two molecules of lactate or three molecules of acetate with no net uptake or yield of oxygen. Note that in most of the other conversions, oxygen is produced.

Forty minimal product compositions were calculated using MILP. Because of the space limit, in Table 7.4b only the ones yielding more than two moles of ATP in at least one of the model are shown. The compositions involve the same products as in anaerobic conditions. However, no compositions involving acetate, ethanol and formate simultaneously are found. This decoupling of products is enabled by the influx of NAD^+ and $NADP^+$ due to the availability of oxygen. The most efficient conversion in terms of ATP output per input of glucose and oxygen corresponds to the composition {acetate : 2, formate : 2}. An elementary flux mode performing this conversion is shown in Figure 7.7.

7.3.3 Minimal amino acid compositions

To identify the minimal substrate compositions required for protein biosynthesis, the following irreversible transporters were included into the models: importers for amino acids and glucose and exporters for lactate, CO_2 and ammonia (glucose and lactate transporters were needed to enable fermentation, thus providing ATP required for amino acid interconversions). The target product composition was defined as 1 unit of protein; the other biomass components and biomass itself were declared internal.

Table 7.5 shows the results calculated using MILP. The table shows only the inclusion of the amino acids in the compositions, ignoring the coefficients and the other substrates. The compositions corresponding to the first-line models and those corresponding to the second and third-line models comprise two distinct sets; further, most of the compositions sufficient for the third-line model are not feasible for the second-line ones. Figire 7.9a shows the distances between the sets of amino acid sets which are sufficient for the different models.

The amino acids in Table 7.5 can be subdivided into three groups (delimited with vertical bars): the group of 10 amino acids which are essential in all models and two other groups, such that each composition includes at least one member of each of them. The composition '36' is the only one including exactly one amino acid out of each of these groups, namely glutamate and threonine; this composition is sufficient for the third-line model only. The numbers of amino acids involved in the other compositions vary between 13 and 18.

Table 7.7 compares the sets of essential amino acids found in the models with those reported by three independent literature sources, based on experimental results. Note that proline is essential in all first and second-line models but is not essential in the third-line one. This is due to the inclusion of the enzyme 1.5.1.12 (1-pyrroline-5-carboxylate dehydrogenase) which is involved in proline biosynthesis (see Figure 7.8). The experimental sources confirm the prototrophy for proline; hence, the inclusion of the enzyme improves not only the quality of the model (see Chapter 6) but also the consistency with experimental results.

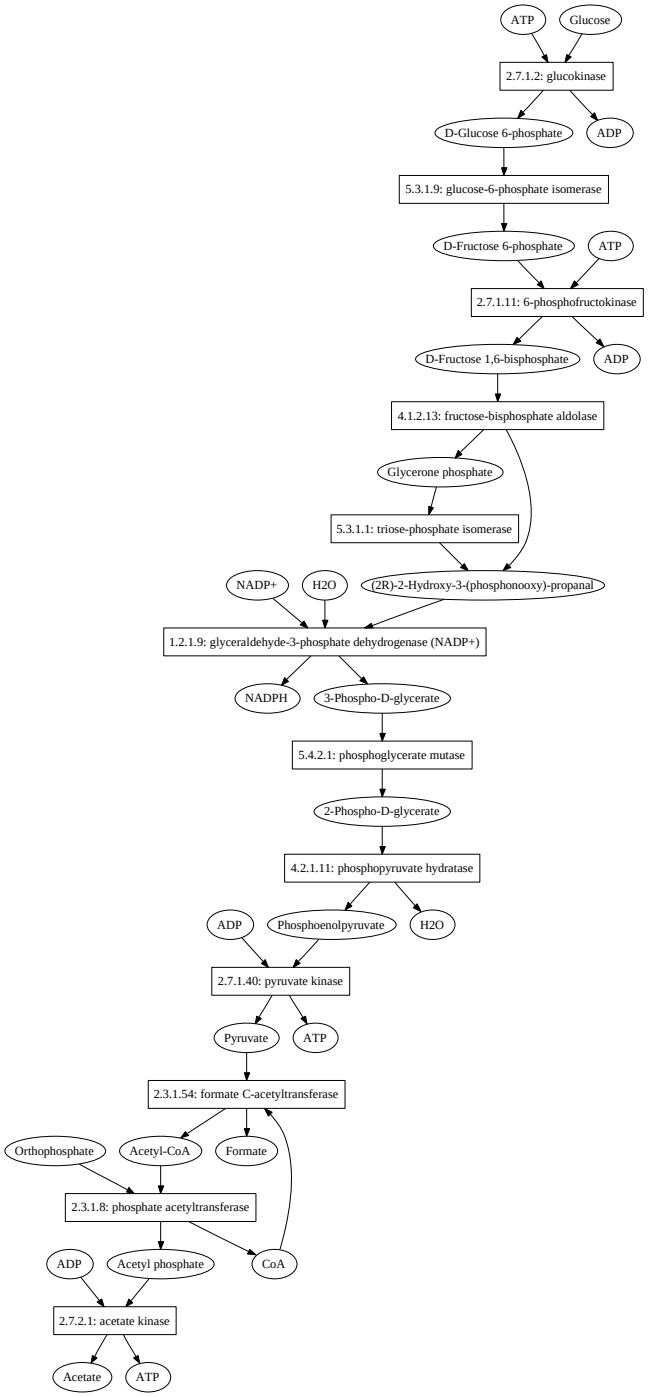


Figure 7.7: Elementary mode converting one mole of glucose into the composition {acetate : 2, formate : 2}, consuming 1 mole of O_2 and yielding 2 moles of ATP. Hypothetical reactions are not shown.

Table 7.5: Minimal amino acid sets required for protein biosynthesis. The right part of the table shows the feasibility of the compositions in the models.

	Amino acids																				**Models, lines**								
																					1-st			2-nd				3-rd
	Histidine	**Tryptophan**	**Isoleucine**	**Methionine**	**Tyrosine**	**Valine**	**Lysine**	**Leucine**	**Arginine**	**Cysteine**	**Proline**	**Glutamine**	**Glutamate**	**Threonine**	**Glycine**	**Phenylalanine**	**Asparagine**	**Alanine**	**Aspartate**	**Serine**	sag	sak	san	sag	sak	san	coh	sag
1	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+		+	+	+	+	+	+					
2	+	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+	+			+	+	+					
3	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+	+	+			+							
4	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+	+	+		+	+	+	+					
5	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+		+	+		+	+	+					
6	+	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+				+	+	+					
7	+	+	+	+	+	+	+	+	+	+	+	+		+	+	+		+	+	+	+	+	+					
8	+	+	+	+	+	+	+	+	+	+	+	+		+	+	+		+	+		+	+	+					
9	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+			+	+	+	+					
10	+	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+		+	+	+							
11	+	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+		+	+		+	+					
12	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+	+	+				+	+					
13	+	+	+	+	+	+	+	+	+	+	+		+							+				+	+	+	+	
14	+	+	+	+	+	+	+	+	+	+	+		+	+										+	+	+	+	
15	+	+	+	+	+	+	+	+	+	+	+		+	+		+	+							+		+		+
16	+	+	+	+	+	+	+	+	+	+	+		+					+						+				
17	+	+	+	+	+	+	+	+	+	+	+		+	+	+	+	+							+	+	+	+	+
18	+	+	+	+	+	+	+	+	+	+	+	+		+		+	+							+				+
19	+	+	+	+	+	+	+	+	+	+	+	+							+					+	+	+		
20	+	+	+	+	+	+	+	+	+	+	+		+		+									+	+	+	+	
21	+	+	+	+	+	+	+	+	+	+	+	+						+						+	+	+		
22	+	+	+	+	+	+	+	+	+	+	+	+			+									+	+	+	+	
23	+	+	+	+	+	+	+	+	+	+	+	+		+										+	+			
24	+	+	+	+	+	+	+	+	+	+	+	+								+				+	+	+		
25	+	+	+	+	+	+	+	+	+	+	+		+				+							+	+	+	+	
26	+	+	+	+	+	+	+	+	+	+	+		+						+					+	+		+	
27	+	+	+	+	+	+	+	+	+	+	+	+					+							+	+	+	+	
28	+	+	+	+	+	+	+	+	+	+	+		+					+							+	+	+	
29	+	+	+	+	+	+	+	+	+	+	+	+		+		+	+								+	+	+	
30	+	+	+	+	+	+	+	+	+	+	+		+	+		+	+								+		+	
31	+	+	+	+	+	+	+	+	+	+	+		+						+							+		
32	+	+	+	+	+	+	+	+	+	+	+	+		+												+	+	
33	+	+	+	+	+	+	+	+	+	+	+	+								+							+	
34	+	+	+	+	+	+	+	+	+	+	+	+						+									+	
35	+	+	+	+	+	+	+	+	+	+	+	+							+								+	
36	+	+	+	+	+	+	+	+	+	+			+	+														+
37	+	+	+	+	+	+	+	+	+	+			+		+													+
38	+	+	+	+	+	+	+	+	+	+			+						+									+
39	+	+	+	+	+	+	+	+	+	+		+						+										+
40	+	+	+	+	+	+	+	+	+	+		+							+									+
41	+	+	+	+	+	+	+	+	+	+		+			+													+
42	+	+	+	+	+	+	+	+	+	+		+					+											+
43	+	+	+	+	+	+	+	+	+	+		+		+														+
44	+	+	+	+	+	+	+	+	+	+			+				+											+
45	+	+	+	+	+	+	+	+	+	+		+								+								+
46	+	+	+	+	+	+	+	+	+	+			+					+										+
47	+	+	+	+	+	+	+	+	+	+			+							+								+

Table 7.6: Genes in `sak`, `san` and `coh` for which the enzyme 1.5.1.12 was predicted as a suboptimal hit and the results of reannotations in second-line models. The columns from left to right show the gene or strain identifiers, KEGG annotations, PRIAM best hits with the E-values, E-values for 1.5.1.12 and the increase in the number of working genes as a result of a reannotation.

ID, strain	**KEGG**	**best**	**1.5.1.12**	+
sak				
SAK_1211	1.2.1.16	1.2.1.16 : 1e-122	8e-58	4
SAK_0947	1.2.1.9	1.2.1.9 : 0.0	5e-77	4
san				
gbs0841	1.2.1.9	1.2.1.9 : 0.0	3e-76	4
gbs1192	1.2.1.16	1.2.1.16 : 1e-122	8e-58	4
coh				
SAN_0923	NA	1.2.1.9 : 0.0	1e-76	4
SAN_1246	NA	1.2.1.16 : 1e-122	8e-58	4

Similarly to 3.2.1.85, this enzyme was also predicted as a suboptimal hit in the PRIAM annotations of the other strains and a reannotation in each case leads to an improvement of model quality (see Table 7.6).

Each of the remaining predictions in the table is confirmend by at least one of the experimental sources, except phenylalanine, which is essential in all experiments but non-essential in the second and third-line models. However, Figure 7.9 demonstrates that the differences between the modelling predictions and experimental results are not larger than those based on the results from different experiments. Hence, the quality of modelling predictions is at least comparable to that of experimental results.

7.4 Discussion and conclusion

In the current chapter, we introduced the concepts of elementary substrate and product compositions - a structurally invariant property of metabolic networks. We proposed algorithms identifying these compositions, as well as elementary net conversions, in a reversible network. In addition, we introduced the concept of a standard network and proposed a method for the detection of minimal substrate and product sets in such networks. The following remarks describe application fields, possible limitations and theoretical implications of the methods introduced.

The results presented demonstrate that the calculation of elementary compositions under the assumption of reversibility is computationally tractable in

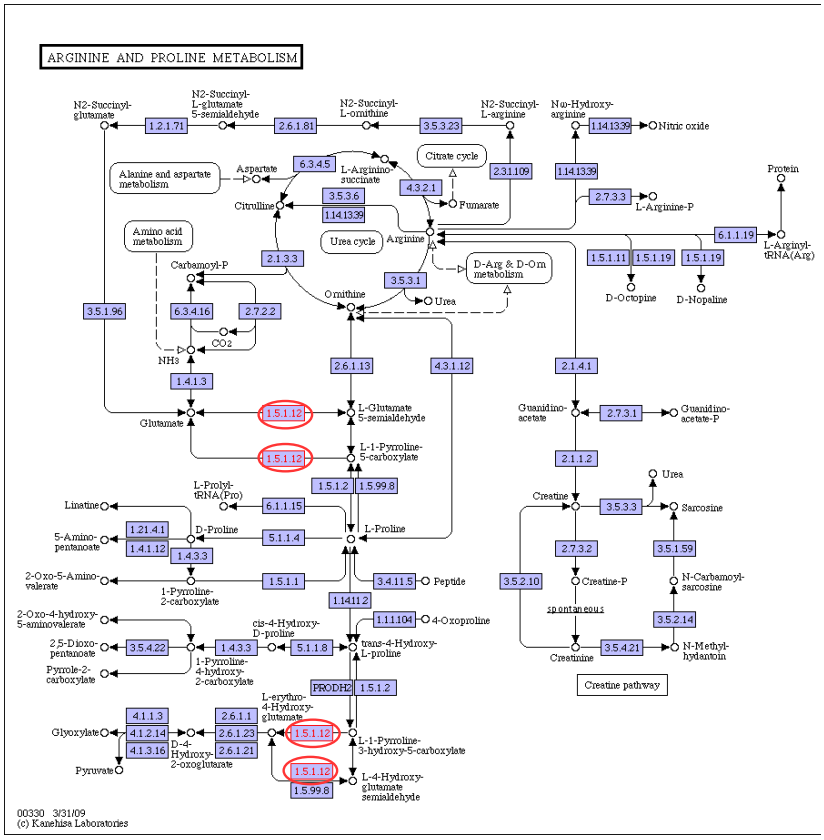


Figure 7.8: KEGG pathway diagram showing the biological functions of the enzyme 1.5.1.12 (circled).

Table 7.7: Essential amino acids identified in the model of *S. agalactiae* (columns 2 - 4) compared to the experimental results reported by [72], [91] and [98]. In the fifth column: cystine was used instead of cysteine, threonine is essential under aerobic conditions only.

source	1-st line	2-nd line	3-rd line	*Milligan*	*Rajagopal*	*Samen*
strain	cross-strain	cross-strain	sag	cross-strain	Ia, A909	Ia, O90R
Valine	+	+	+	+	+	+
Lysine	+	+	+	+		+
Leucine	+	+	+	+	+	+
Isoleucine	+	+	+	+	+	+
Tyrosine	+	+	+	+	+	+
Tryptophan	+	+	+	+	+	+
Cysteine	+	+	+	+		
Methionine	+	+	+	+	+	
Histidine	+	+	+	+	+	+
Arginine	+	+	+	+	+	+
Proline	+	+				
Phenylalanine	+			+	+	+
Threonine	+			+	+	
Glycine	+				+	
Aspartate						
Asparagine						
Glutamate				+		
Glutamine						
Alanine						
Serine					+	

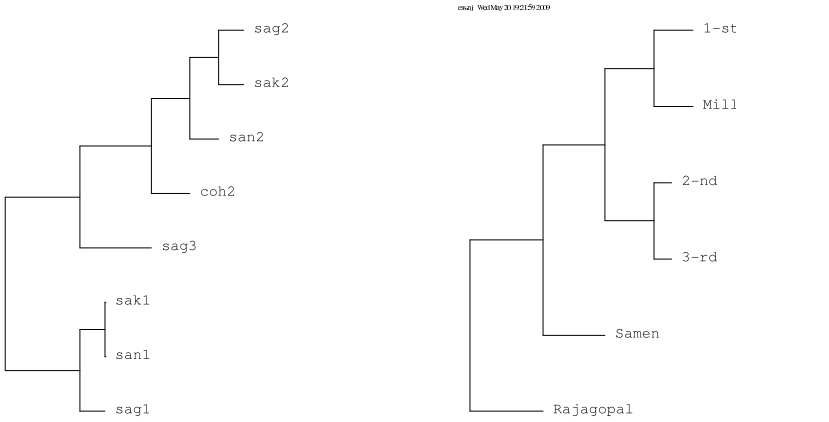


Figure 7.9: Set distance trees: a) based on the feasibility of minimal amino acid compositions (Table 7.5; b) based on the essentiality of amino acids (Table 7.7).

genome-scale models, due to the relatively small size of an external left nullspace matrix in comparison with a stoichiometry matrix. It must be stressed, however, that the computational feasibility of the algorithm strongly depends on the number of external metabolites. Therefore, it is preferable to select the minimal set of external metabolites necessary for the solution of a given problem (i.e. amino acids do not need to be declared external for the prediction of glucose fermentation products).

Unfortunately, the methods described do not enable the calculation of a complete set of elementary compositions with respect to irreversible reactions. The solution of this problem is subject to future developments and would provide more precise characterisation of the networks analysed. However, our results demonstrate that the calculation of elementary compositions in the reversible version of a network provides a range of practically relevant data. In fact, any feasible composition is a convex combination of the elementary compositions calculated in the reversible version, regardless of their feasibility.

As an alternative, minimal substrate and product sets can be calculated using linear programming methods, which take irreversibility of reactions into account. Although the methods described are applicable to standard networks only, this does not diminish their universality, since any non-standard network can be easily converted into a standard one by appending additional transporters. The results can be further analysed using set-theoretical and logical methods (e.g. the combination of sets {A, B} and {A, C} can be represented as a logical expression 'A and (B or C)').

Detection of elementary substrate compositions enables a comprehensive analysis of nutritional requirements of the modelled organisms. It must be taken into consideration, however, that the correctness of the results depends on the exact definition of the target product compositions. In our models, we defined protein simply as a combination of twenty amino acid molecules; more precise results could be obtained if the proportions of the amino acids in the organism's proteome were known. Considering a hypothetical metabolite representing the whole biomass of an organism (e.g. BIOMASS in the present work) as the target product would enable the prediction of elementary substrate compositions representing complete media sufficient for growth. This perspective appears to be widely applicable to various biotechnological problems.

Elementary substrate compositions represent the extreme environments in which an organism is able to survive and to synthesise the desired products. These conditions are not necessarily optimal from an economic and energetic point of view; in fact, the optimal conditions are more likely to correspond to the interior points of the substrate composition polyhedron then to its vertices. For instance, the most favourable environment for any organism would include all amino acids in those proportions in which they are present in the proteome. In this environment, protein biosynthesis would require no amino acid interconversions consuming energy and resulting in loss of carbon and nitrogen atoms.

The number and variability of the elementary substrate compositions for biomass (or geometrically, the volume of $\mathcal{S}_p$) can be considered as a measure of the ecological versatility of an organism; e.g. Table 7.5 makes clear that the

third-line model of `sag` is capable of synthesising protein in a broader range of environments than the other models. A comparison of the sets of elementary or minimal compositions can be used for the classification of organisms into ecotypes (see Figure 7.9a).

Equations 2.13 and 5.4 define a connection between flux modes and conservation relationships and hence between left and right nullspace analyses. Although the left nullspace of a stoichiometry matrix is commonly used in metabolic modelling for the detection of relationships between metabolite concentrations [37, 73], we demonstrated that its analysis provides a wide range of data characterising nutritional requirements and functional capabilities of the modelled organisms. Future developments in this direction may lead to the discovery of other interesting and practically useful properties of metabolic networks.

Chapter 8

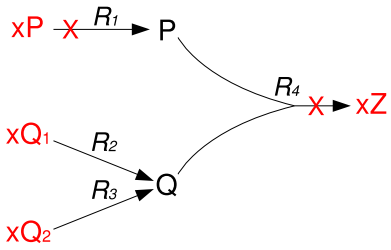
Essentiality Analysis

8.1 Introduction

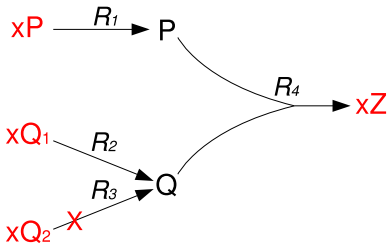
Metabolic enzymes are particularly attractive as drug target candidates [6]. However, the identification of essential enzymes *in vitro* is laborious and expensive and can be considerably facilitated by computational prediction methods. The existing algorithms for the prediction of essential enzymes include the detection of 'chokepoint reactions', which uniquely consume or produce specific metabolites [127], and the calculation of reaction damage, defined as the set of metabolites whose production becomes impossible in a system by deleting a given reaction [61]. These methods are based on graph-theoretical analysis of networks and hence do not take into account the ability of reactions to carry steady-state fluxes. Further, both methods use quantitative criteria of essentiality, namely the uniqueness of the consuming or producing reaction and the size of the damage, regardless of the necessity of these reactions for survival and growth. However, even a non-'chokepoint' reaction with a small damage is essential for an organism if its deletion leads to inability to produce protein, DNA or energy.

In this chapter, we present a target-oriented approach to the identification of essential reactions and enzymes. We define essentiality of reactions as their necessity for vital cellular processes, such as protein biosynthesis or energy production, which are represented by single hypothetical reactions.

We denote by $\mathcal{N}(\mathcal{R})$ a network with a reaction set $\mathcal{R}$. Let us consider reactions $R_q, R_t \in \mathcal{R}$, named the *query* and the *target* reaction, respectively. R_q is *essential* for R_t if R_q is live in $\mathcal{N}(\mathcal{R})$ and dead in $\mathcal{N}(\mathcal{R} \setminus \{\mathcal{R}_t\})$. In other words, the presence of an essential reaction in the network is necessary for the ability of the target reaction to carry steady state flux (Figure 8.1a, b). Essentiality is a non-reflexive, transitive binary relation over the set of reactions and can be represented as a square binary matrix (Figure 8.1c).



a



b

	R_1	R_2	R_3	R_4
R_1		+	+	+
R_2				
R_3				
R_4	+	+	+	

c

$$\begin{array}{c} \\ R_1 \\ R_2 \\ R_3 \\ R_4 \end{array} \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$$

d

R_1: {R_4}, {R_2, R_3}
R_2: {R_4}, {R_1}
R_3: {R_4}, {R_1}
R_4: {R_1}, {R_2, R_3}

e

Figure 8.1: a) Network of four reactions. Considering R_4 as the target reaction, R_1 is essential (a) and R_3 is not essential (b). c) Essentiality matrix: each row represents the reactions for which the given reaction is essential; each column represents the reactions which are essential for the given reaction. d) Right null space matrix: R_1 and R_4 comprise a reaction subset. e) Reactions followed by the lists of their minimal cut sets.

The numbers of reactions essential for vital functions tend to be very limited, since most of the removals can be substituted by redundant pathways. However, an objective function may be blocked by the removal of a group of reactions. A subset $\mathcal{R}' \subset \mathcal{R}$ is called a *cut set* for a reaction R_t if the latter is live in $\mathcal{N}(\mathcal{R})$ and dead in $\mathcal{N}(\mathcal{R} \setminus \mathcal{R}')$. $\mathcal{R}'$ is a *minimal cut set (MCS)* for R_t if none of its proper subsets is a cut set for R_t (see Figure 8.1e). Clearly, any essential reaction comprises a singleton minimal cut set. The application of the concept of MCS to metabolic networks has been suggested by Klamt *et al.* [55], who also proposed an algorithm for their calculation. Unfortunately, this algorithm is computationally inefficient, since it involves the calculation of elementary modes. In this chapter, we propose an alternative algorithm which calculates a complete list of disjoint (i.e. non-overlapping) minimal cut sets.

8.2 Methods

Below we present methods for the detection of essential reactions and minimal cut sets.

8.2.1 Detection of essential reactions

Some information about essential reactions can be derived by means of right nullspace analysis. In particular, the elements of a reaction subset are essential for each other, since they are involved simultaneously in all elementary flux modes (Figure 8.1c). The converse statement is true only in a network of reversible reactions (Poolman, unpublished). In such a network, essentiality is a symmetric relation and mutually essential reactions are obviously involved simultaneously in all elementary modes, thus being elements of the same reaction subsets. In a network containing irreversible reactions, the essentiality relation is asymmetric. So, in Figure 8.1, R_4 is essential for R_2 and R_3, but neither of them is essential for and involved in a common reaction subset with R_4.

The essentiality of a reaction q for a given target t can be tested by complementing the linear program shown in Equation 3.12 (used to test the liveness of t) with two lines stating that both directions of q are blocked:

$$\begin{array}{rl} \text{Minimise} & \\ \text{Subject to} & \ddot{\mathbf{N}}\ddot{\mathbf{v}} = \mathbf{0}, \\ & {}^{-}\mathbf{v}^{irr} = \mathbf{0}, \\ & \ddot{v}_t \geq \epsilon, \\ & \ddot{v}_{opp(t)} = 0, \\ & \ddot{v}_q = 0, \\ & \ddot{v}_{opp(q)} = 0 \\ \text{Where} & \ddot{\mathbf{v}} \geq \mathbf{0} \end{array} \tag{8.1}$$

Similarly to Equation 3.12, the program must be solved twice, with the forward and backward fluxes of the target reaction taken as $\ddot{v}_t$; q is qualified as essential

if the program is not solvable for both directions (e.g. the target reaction cannot operate in either direction).

8.2.2 Detection of minimal cut sets

Algorithm 10 presents a method for the detection of a complete set of disjoint minimal cut sets for a given live target reaction. Similarly to Algorithm 6, the list R of candidate reactions is ordered by descending absolute values of the correlation coefficients with the target reaction, ensuring that the candidates with stronger correlations with the target are tested first (lines 2-14). To accelerate the algorithm, the essential reactions for the target are detected, removed from the candidate list and included into the result as singleton sets.

In each outer while-loop (lines 17-43), a cut set $\mathcal{R}'$ is constructed and then reduced, if possible. In the first inner while-loop (lines 22-27), the candidate reactions are iteratively included into the set $\mathcal{R}'$ and the target reaction is tested for liveness in the network $\mathcal{N}(\mathcal{R} \setminus \mathcal{R}')$. After the target becomes dead, in the second while-loop (lines 30-37) the reactions are iterated in the order of ascending correlation coefficients and in each iteration it is tested whether after removing a given reaction from $\mathcal{R}'$ the latter remains a cut set for t. Thus, a cut set is minimised. Then the minimal cut set is appended to the result and its elements are removed from the candidate list, thus ensuring that all detected sets are disjoint. If after the first inner while-loop, the target is still live (see the IF operator in line 28), the algorithm terminates, since no further cut sets can be detected.

8.3 Applications to genome-scale models

The methods described above were applied to the models constructed as described in Chapter 4, 5 and 6. The minimal cut sets (including the ones consisting of essential reactions) were identified for the reactions producing the biomass components and for the ATPase reaction. The results are summarised in Table 8.1. Figure 8.2 shows the results of the comparison of the sets of essential reactions and MCS detected in the models for all targets except for the peptidoglycan synthesis reaction (since the latter is dead in the first-line models). The differences between these sets are discussed in detail in the following subsections.

8.3.1 Protein biosynthesis

In all models, the only essential reactions for protein biosynthesis are the transporters of essential amino acids (see Table 7.7). The smallest remaining minimal cut sets are shown in Table 8.2.

The table reveals some differences in the sets of MCS between the first-line and other models. In particular, MCS 4 is not a cut set in the first-line models due to the presence of alanine transaminase, which catalyses the conversion of

Algorithm 10 Given a network $\mathcal{N}(\mathcal{R})$ and a live reaction t, detect a set S of disjoint MCS for t.

$S = \{\}$
$\mathbf{K} := orthogonal\_null\_space(\mathbf{N})$
$del\_zero\_rows(\mathbf{K})$
$R := row\_names(\mathbf{K}) - \langle t \rangle$
for all $q \in R$ **do**
  //Equation 8.1
  if $is\_essential(q, t, \mathcal{N}(\mathcal{R}))$ **then**
    $S := S \cup \{q\}$
    $delete(R, q)$
  end if
end for
$corr(i) := abs(cos(\theta^K_{i,t}))$
$cmp(i, j) := cmp(corr(i), corr(j)))$
$sort(R, cmp)$

$l := False$
while $l = False$ *and* $length(R) > 0$ **do**
  $i := 0$
  $l := True$
  //cut set construction
  $\mathcal{R}' := \{\}$
  while $l = True$ *and* $i < length(R)$ **do**
    $q := R[i]$
    $\mathcal{R}' := \mathcal{R}' \cup \{q\}$
    $l := is\_live(\mathcal{N}(\mathcal{R} \setminus \mathcal{R}'), t)$
    $i := i + 1$
  end while
  if $l := False$ **then**
    //cut set reduction
    while $i > 0$ **do**
      $i := i - 1$
      $q := R[i]$
      $R'' := \mathcal{R}' \setminus \{q\}$
      if *not* $is\_live(\mathcal{N}(\mathcal{R}' \setminus \mathcal{R}''), t)$ **then**
        $\mathcal{R}' := \mathcal{R}''$
      end if
    end while
    for all $r \in \mathcal{R}'$ **do**
      $delete(R, r)$
    end for
    $S := S \cup \{\mathcal{R}'\}$
  end if
end while

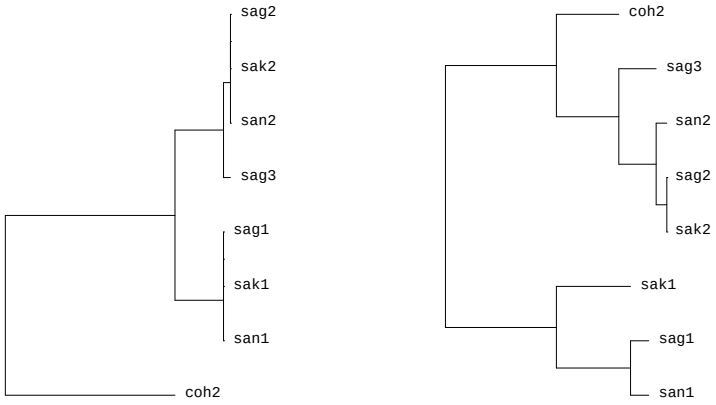


Figure 8.2: Set distance trees based on the sets of essential reactions (a) and MCS (b).

Table 8.1: Numbers of minimal cut sets detected for different target reactions. The columns 3-13 show the numbers of MCS of given lengths.

Target	all	1	2	3	4	5	6	7	8	9	10	>10
syn_PROTEIN	40	12	8	4	6	2	0	0	2	0	0	6
syn_RNA	101	8	14	13	12	13	3	12	5	3	1	17
syn_DNA	120	11	24	18	9	20	9	3	3	5	4	14
syn_MEMBRANE	74	12	6	7	3	12	5	6	1	5	3	14
syn_PG	89	28	9	7	3	9	9	3	3	0	4	14
ATPase	47	0	3	6	3	2	3	9	2	1	3	15

Table 8.2: Minimal cut sets of length 2-3 for protein biosynthesis.

			1-st			2-nd				3-rd
	Minimal cut sets		sag	sak	san	sag	sak	san	coh	sag
1	**C02057_tx:**	X_Phe ↔ Phe	+	+	+	+	+	+	+	+
	R03083:	DAHP ↔ Pi + 3-Dehydroquinate								
2	**C02057_tx:**	X_Phe ↔ Phe	+	+	+	+	+	+	+	+
	R03084:	3-Dehydroquinate ↔ H_2O + 3-Dehydroshikimate								
3	**C00188_tx:**	X_Thr ↔ Thr	+	+	+	+	+	+	+	+
	R01466:	H_2O + O-Phospho-homoserine → Pi + Thr								
4	**C01401_tx:**	X_Ala ↔ Ala				+	+	+	+	+
	R00401:	Ala ↔ D-Ala								
5	**C02057_tx:**	X_Phe ↔ Phe	+	+	+	+	+	+	+	+
	R02413:	$NADP^+$ + Shikimate ↔ NADPH + 3-Dehydroshikimate								
6	**C00302_tx:**	X_Glu ↔ Glu	+	+	+	+	+	+	+	+
	C00303_tx:	X_Gln ↔ Gln								
7	**C02057_tx:**	X_Phe ↔ Phe	+	+	+	+	+	+	+	+
	R01826:	Erythrose 4-P + H_2O + PEP ↔ Pi + DAHP								
8	**C01401_tx:**	X_Ala ↔ Ala	+	+	+					
	R00258:	2-Oxoglutarate + Ala ↔ Glu + Pyr								
9	**C00033_tx:**	X_Acetate ↔ Acetate	+	+	+	+	+	+	+	+
	C00152_tx:	X_Asn ↔ Asn								
	R00658:	2PG ↔ H_2O + PEP								
10	**C00716_tx:**	X_Ser ↔ Ser	+	+						
	R00220:	Ser ↔ NH_3 + Pyr								
	R00943:	THF + Formate + ATP → Pi + 10-FormylTHF + ADP								
11	**C00148_tx:**	X_Pro ↔ Pro	+	+	+	+	+	+	+	+
	R00707:	(S)-1-Pyrroline-5-carboxylate + NAD^+ + 2 H_2O ↔ NADH + Glu								
	R00708:	$NADP^+$ + 2 H_2O + (S)-1-Pyrroline-5-carboxylate ↔ NADPH + Glu								
12	**C00049_tx:**	X_Asp ↔ Asp	+	+	+					
	R00345:	CO_2 + PEP + H_2O → Oxaloacetate + Pi								
	R00485:	Asn + H_2O → Asp + NH_3								

glutamate into alanine:

R00258: 2-Oxoglutarate + Alanine ↔ Glutamate + Pyruvate

However, this enzyme was predicted in all RPS-BLAST annotations as a suboptimal hit. Hence, this difference may be due to an artefact in the second-line models.

MCS 8 is not a cut set in the second and third-line models due to the presence of glycine oxidase, which catalyses the following reaction:

R05861: O_2 + D-Alanine + H_2O ↔ H_2O_2 + NH_3 + Pyruvate

MCS 10 is not a cut set in the second and third-line models due to the presence of aminomethyltransferase, which converts gycine to 5,10-MethyleneTHF, which is further used by glycine hydroxymethyltransferase to synthesise serine:

R01221: THF + Glycine + NAD^+ ↔ NH_3 + CO_2 + 5,10-MethyleneTHF + NADH
R00945: 5,10-MethyleneTHF + H_2O + Glycine ↔ Serine + THF

In the first-line model of `san`, this MCS is not a cut set due to the presence of phosphoserine transaminase (see Figure 8.3).

MCS 12 is not a cut set in the second and third-line models due to the presence of malate dehydrogenase, which produces oxaloacetate. The latter is further used by aspartate transaminase to produce aspartate:

R00342: NAD^+ + Malate ↔ Oxaloacetate + NADH
R00355: Aspartate + 2-Oxoglutarate ↔ Oxaloacetate + Glutamate

8.3.2 Nucleic acid biosynthesis

The smallest minimal cut sets for RNA and DNA biosynthesis are shown in Tables 8.3, 8.4 and 8.5. The essential reactions include the phosphate transporter; apart from the amino acids, phosphate is the only essential substrate for all models.

A number of MCS listed in these tables are only found in the model of `coh`, because of the absence of cytidylate kinase, which catalyses the following reaction:

R00158: UMP + ATP → ADP + UDP

This enzyme was predicted by RPS-BLAST in the `coh` annotation for a number of genes as a suboptimal hit, but with E-values higher than 10^{-8}, so its presence in the strain `coh` is under question.

MCS 16 in Table 8.3 is a cut set in the first-line models of `sag` and `sak` because of the absence of phosphoketolase.

MCS 23 in Table 8.4 and MCS 24 in Table 8.5 both contain the reaction R00945. MCS 23 is found in the second and third-line models because of the absence of methylenetetrahydrofolate dehydrogenase ($NADP^+$), which catalyses

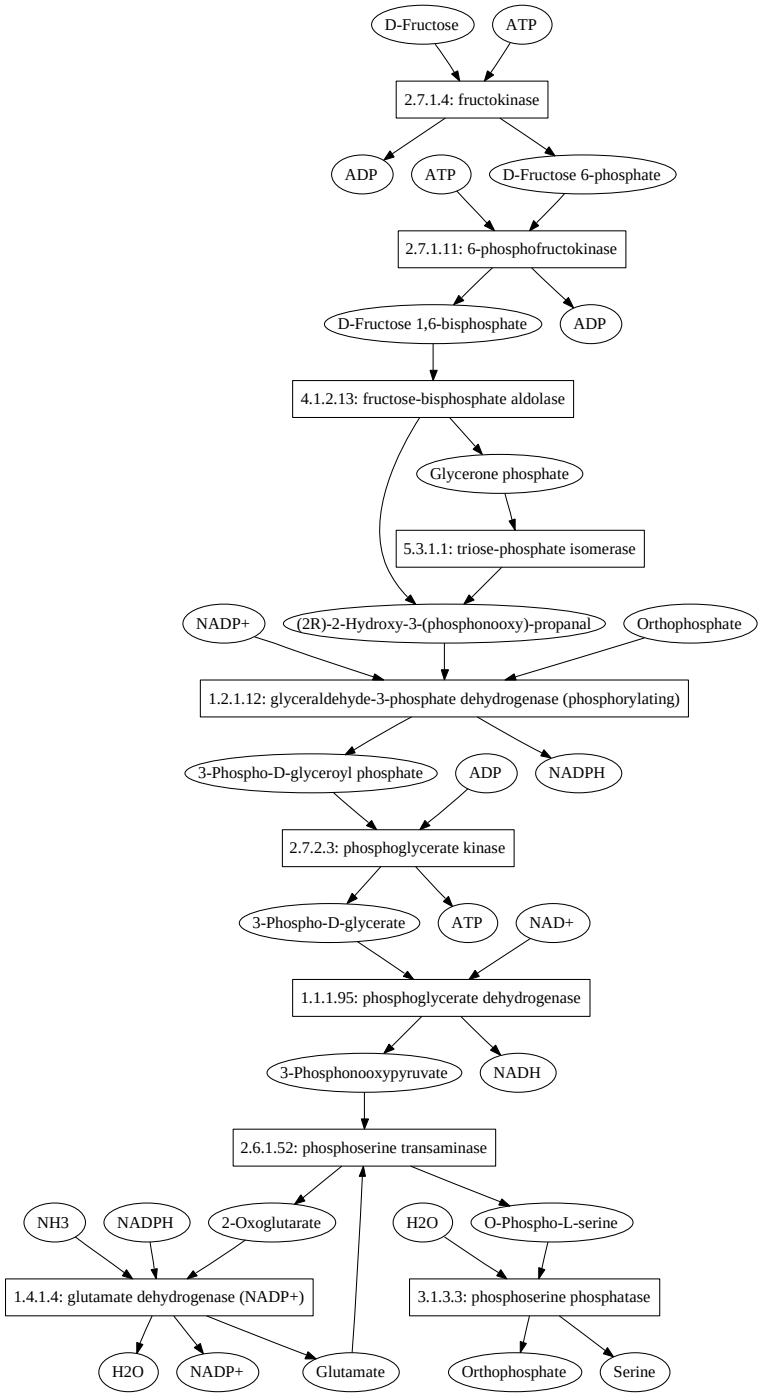


Figure 8.3: Serine biosynthesis using phosphoserine transaminase in the 1-st line model of `san`.

Table 8.3: Minimal cut sets of length 1-2 for RNA biosynthesis.

		1-st			2-nd				3-rd
	Minimal cut sets	sag	sak	san	sag	sak	san	coh	sag
1	**R01066:** 2-Deoxy-ribose 5-P ↔ Acetaldehyde + G3P							+	
2	**R02483:** Deoxyuridine + Pi ↔ Uracil + 2-Deoxy-ribose 1-P							+	
3	**C00009_tx:** X_Pi ↔ Pi	+	+	+	+	+	+	+	+
4	**R00127:** AMP + ATP → 2 ADP	+	+	+	+	+	+	+	+
5	**R02099:** Deoxyuridine + ATP → dUMP + ADP							+	
6	**R00004:** PPi + H_2O → 2 Pi	+	+	+	+	+	+	+	+
7	**R02098:** dUMP + ATP → dUDP + ADP							+	
8	**R02749:** 2-Deoxy-ribose 1-P ↔ 2-Deoxy-ribose 5-P							+	
9	**R00315:** Acetate + ATP ↔ Acetyl phosphate + ADP **R01512:** 3PG + ATP ↔ BPG + ADP	+	+	+	+	+	+	+	+
10	**R00158:** UMP + ATP → ADP + UDP **R02749:** 2-Deoxy-ribose 1-P ↔ 2-Deoxy-ribose 5-P	+	+	+	+	+	+	+	+
11	**R00190:** PPi + AMP ↔ 5-Phospho-α-ribose 1-diphosphate + Adenine **R01135:** GTP + Asp + IMP → GDP + Pi + N6-(1,2-Dicarboxyethyl)-AMP	+	+	+	+	+	+	+	+
12	**C00033_tx:** X_Acetate ↔ Acetate **R00658:** 2PG ↔ H_2O + PEP	+	+	+	+	+	+	+	+
13	**R02018:** dUDP + Oxidized thioredoxin + H_2O ↔ Thioredoxin + UDP **R02023:** Oxidized thioredoxin + dUTP + H_2O ↔ Thioredoxin + UTP							+	
14	**R00332:** ATP + GMP → GDP + ADP **R01066:** 2-Deoxy-ribose 5-P ↔ G3P + Acetaldehyde	+	+	+	+	+	+	+	+
15	**R00332:** ATP + GMP → GDP + ADP **R02749:** 2-Deoxy-ribose 1-P ↔ 2-Deoxy-ribose 5-P	+	+	+	+	+	+	+	+
16	**R00230:** Acetyl-CoA + Pi ↔ CoA + Acetyl phosphate **R00658:** 2PG ↔ H_2O + PEP	+		+					
17	**R00158:** UMP + ATP → ADP + UDP **R01066:** 2-Deoxy-ribose 5-P ↔ Acetaldehyde + G3P	+	+	+	+	+	+	+	+
18	**C00122_tx:** X_Fumarate ↔ Fumarate **C00147_tx:** X_Adenine ↔ Adenine	+	+	+	+	+	+	+	+
19	**R02016:** $NADP^+$ + Thioredoxin ↔ NADPH + Oxidized thioredoxin **R02101:** 5,10-MethyleneTHF + dUMP → dTMP + Dihydrofolate							+	
20	**R00571:** NH_3 + ATP + UTP ↔ Pi + ADP + CTP **R00573:** ATP + H_2O + UTP + Gln ↔ Pi + ADP + CTP + Glu	+	+	+	+	+	+	+	+
21	**C00033_tx:** X_Acetate ↔ Acetate **R01518:** 2PG ↔ 3PG	+	+	+	+	+	+	+	+
22	**C00033_tx:** X_Acetate ↔ Acetate **R01512:** 3PG + ATP ↔ BPG + ADP	+	+	+	+	+	+	+	+

Table 8.4: Minimal cut sets of length 1-2 for DNA biosynthesis.

		1-st			2-nd				3-rd
	Minimal cut sets	sag	sak	san	sag	sak	san	coh	sag
1	**R01066:** 2-Deoxy-ribose 5-P ↔ Acetaldehyde + G3P							+	
2	**R02749:** 2-Deoxy-ribose 1-P ↔ 2-Deoxy-ribose 5-P							+	
3	**R02483:** Deoxyuridine + Pi ↔ Uracil + 2-Deoxy-ribose 1-P							+	
4	**R02101:** 5,10-MethyleneTHF + dUMP → dTMP + Dihydrofolate	+	+	+	+	+	+	+	+
5	**R02099:** Deoxyuridine + ATP → dUMP + ADP							+	
6	**C00009_tx:** X_Pi ↔ Pi	+	+	+	+	+	+	+	+
7	**R00127:** AMP + ATP → 2 ADP	+	+	+	+	+	+	+	+
8	**R00004:** PPi + H_2O → 2 Pi	+	+	+	+	+	+	+	+
9	**R02098:** dUMP + ATP → dUDP + ADP							+	
10	**R02094:** dTMP + ATP → ADP + dTDP	+	+	+	+	+	+	+	+
11	**R02093:** dTDP + ATP ↔ dTTP + ADP	+	+	+	+	+	+	+	+
12	**R00190:** PPi + AMP ↔ 5-Phospho-α-ribose 1-diphosphate + Adenine **R01135:** GTP + Asp + IMP → GDP + Pi + N6-(1,2-Dicarboxyethyl)-AMP	+	+	+	+	+	+	+	+
13	**C00033_tx:** X_Acetate ↔ Acetate **R00658:** 2PG ↔ H_2O + PEP	+	+	+	+	+	+	+	+
14	**R00315:** Acetate + ATP ↔ Acetyl phosphate + ADP **R01518:** 2PG ↔ 3PG	+	+	+	+	+	+	+	+
15	**C00122_tx:** X_Fumarate ↔ Fumarate **R00190:** PPi + AMP ↔ 5-Phospho-α-ribose 1-diphosphate + Adenine	+	+	+	+	+	+	+	+
16	**R02018:** dUDP + Oxidized thioredoxin + H_2O ↔ Thioredoxin + UDP **R02023:** Oxidized thioredoxin + dUTP + H_2O ↔ Thioredoxin + UTP							+	
17	**R02100:** dUTP + H_2O → PPi + dUMP **R02483:** Deoxyuridine + Pi ↔ Uracil + 2-Deoxy-ribose 1-P	+	+	+	+	+	+	+	+
18	**R00158:** UMP + ATP → ADP + UDP **R02099:** Deoxyuridine + ATP → dUMP + ADP	+	+	+	+	+	+	+	+
19	**R00937:** Dihydrofolate + NADH ↔ NAD^+ + THF **R00939:** NADPH + Dihydrofolate ↔ $NADP^+$ + THF	+	+	+	+	+	+	+	+
20	**R02099:** Deoxyuridine + ATP → dUMP + ADP **R02100:** dUTP + H_2O → PPi + dUMP	+	+	+	+	+	+	+	+
21	**R00315:** Acetate + ATP ↔ Acetyl phosphate + ADP **R00658:** 2PG ↔ H_2O + PEP	+	+	+	+	+	+	+	+
22	**R00158:** UMP + ATP → ADP + UDP **R02483:** Deoxyuridine + Pi ↔ Uracil + 2-Deoxy-ribose 1-P	+	+	+	+	+	+	+	+
23	**R00945:** Gly + 5,10-MethyleneTHF + H_2O ↔ Ser + THF **R01221:** Gly + THF + NAD^+ ↔ NH_3 + CO_2 + 5,10-MethyleneTHF + NADH				+	+	+	+	+

the reaction R01220, involved in MCS 24. The latter, in turn, is found in the first-line models because of the absence of aminomethyltransferase, which catalyses R012201, involved in MCS 23. Interestingly enough, these enzymes are predicted for different genes, so the discrepancies between the models are not caused by alternative annotations.

8.3.3 Membrane biosynthesis

The smallest minimal cut sets for membrane biosynthesis are shown in Table 8.6. The only non-hypothetical essential reaction is R00004, catalysed by inorganic diphosphatase.

The reaction R00512 is essential in the second and third-line models because of the absence of 5'-nucleotidase, which catalyses the following reaction:

$$\text{R00511:} \quad \text{CMP} + \text{H}_2\text{O} \rightarrow \text{Cytidine} + \text{Orthophosphate}$$

MCS 16 is found in the first-line models of `sag` and `sak` because of the absence of phosphoketolase and in the model of `coh` because of the absence of cytidylate kinase.

8.3.4 Cell wall biosynthesis

Since the peptidoglycan synthesis reaction is dead in the first-line models, the minimal cut sets were only detected in the second and third-line models. The smallest MCS are shown in Tables 8.7 and 8.8. The only discrepancies are between the model of `coh` and the other models; all of them are caused by the absence of cytidylate kinase in the model of `coh`.

8.3.5 ATP production

The smallest minimal cut sets for ATP production are shown in the Table 8.9. No essential reactions were found and only three MCS consist of pairs of reactions, namely (i) phosphoglycerate kinase and acetate kinase, (ii) acetate transporter and phosphoglycerate kinase and (iii) acetate transporter and phosphopyruvate hydratase. The only discrepancies between the columns are caused by the absence of phosphoketolase in the first-line models of `sag` and `san`.

8.4 Discussion

The framework of essentiality analysis presented in this section is based on the assumption that the survival and growth of an organism depend on its ability to carry out certain metabolic processes, which can be represented in a model by a relatively small group of target reactions. Hence, essentiality analysis is a hypothesis-driven and reductionist method, since the target reactions are selected or defined by the investigator and they by far do not cover all the essential functions of a cell. However, the production of biomass and energy

Table 8.5: Minimal cut sets of length 1-2 for DNA biosynthesis (continued).

		1-st			2-nd				3-rd
	Minimal cut sets	sag	sak	san	sag	sak	san	coh	sag
24	**R00945:** 5,10-MethyleneTHF + H_2O + Gly ↔ Ser + THF **R01220:** $NADP^+$ + 5,10-MethyleneTHF ↔ NADPH + 5,10-MethenylTHF	+	+	+					
25	**R00332:** ATP + GMP → GDP + ADP **R02090:** dGMP + ATP → ADP + dGDP	+	+	+	+	+	+	+	+
26	**R00332:** ATP + GMP → GDP + ADP **R01066:** 2-Deoxy-ribose 5-P ↔ G3P + Acetaldehyde	+	+	+	+	+	+	+	+
27	**C00122_tx:** X_Fumarate ↔ Fumarate **C00147_tx:** X_Adenine ↔ Adenine	+	+	+	+	+	+	+	+
28	**R02014:** H_2O + Oxidized thioredoxin + dATP ↔ Thioredoxin + ATP **R02017:** Oxidized thioredoxin + H_2O + dADP ↔ Thioredoxin + ADP	+	+	+	+	+	+	+	+
29	**R01066:** 2-Deoxy-ribose 5-P ↔ G3P + Acetaldehyde **R02016:** $NADP^+$ + Thioredoxin ↔ NADPH + Oxidized thioredoxin	+	+	+	+	+	+	+	+
30	**R00332:** ATP + GMP → GDP + ADP **R01967:** Deoxyguanosine + ATP → dGMP + ADP	+	+	+	+	+	+	+	+
31	**R00571:** NH_3 + ATP + UTP ↔ Pi + ADP + CTP **R00573:** ATP + H_2O + UTP + Gln ↔ Pi + ADP + CTP + Glu	+	+	+	+	+	+	+	+
32	**R02022:** dCTP + Oxidized thioredoxin + H_2O ↔ CTP + Thioredoxin **R02024:** Oxidized thioredoxin + dCDP + H_2O ↔ Thioredoxin + CDP	+	+	+	+	+	+	+	+
33	**C00033_tx:** X_Acetate ↔ Acetate **R01518:** 2PG ↔ 3PG	+	+	+	+	+	+	+	+
34	**R00332:** ATP + GMP → GDP + ADP **R01969:** Deoxyguanosine + Pi ↔ 2-Deoxy-ribose 1-P + Guanine	+	+	+	+	+	+	+	+
35	**R02016:** $NADP^+$ + Thioredoxin ↔ NADPH + Oxidized thioredoxin **R02749:** 2-Deoxy-ribose 1-P ↔ 2-Deoxy-ribose 5-P	+	+	+	+	+	+	+	+

Table 8.6: Minimal cut sets of length 1-2 for membrane biosynthesis.

		1-st			2-nd				3-rd
	Minimal cut sets	sag	sak	san	sag	sak	san	coh	sag
1	**syn_PHOSPHATIDATE**	+	+	+	+	+	+	+	+
2	**R00004:** PPi + H_2O → 2 Pi	+	+	+	+	+	+	+	+
3	**syn_CDP_DIACYLGLYCEROL**	+	+	+	+	+	+	+	+
4	**R00742:** Acetyl-CoA + HCO_3^- + ATP → Malonyl-CoA + Pi + ADP	+	+	+	+	+	+	+	+
5	**syn_CARDIOLIPIN**	+	+	+	+	+	+	+	+
6	**syn_PHOSPHATIDYL_SERINE**	+	+	+	+	+	+	+	+
7	**C00009_tx:** X_Pi ↔ Pi	+	+	+	+	+	+	+	+
8	**R00512:** CMP + ATP → ADP + CDP				+	+	+	+	+
9	**Carboxyanhydrase:** HCO_3^- ↔ CO_2 + H_2O	+	+	+	+	+	+	+	+
10	**syn_FATTY_ACID**	+	+	+	+	+	+	+	+
11	**syn_PHOSPHATIDYL_GLYCEROL**	+	+	+	+	+	+	+	+
12	**syn_PHOSPHATIDYL_ETHANOLAMINE**	+	+	+	+	+	+	+	+
13	**R00315:** Acetate + ATP ↔ Acetyl phosphate + ADP **R01512:** 3PG + ATP ↔ BPG + ADP	+	+	+	+	+	+	+	+
14	**R00315:** Acetate + ATP ↔ Acetyl phosphate + ADP **R00658:** 2PG ↔ H_2O + PEP	+	+	+	+	+	+	+	+
15	**C00033_tx:** X_Acetate ↔ Acetate **R01518:** 2PG ↔ 3PG	+	+	+	+	+	+	+	+
16	**R00230:** Acetyl-CoA + Pi ↔ CoA + Acetyl phosphate **R00658:** 2PG ↔ H_2O + PEP	+		+				+	
17	**R00512:** CMP + ATP → ADP + CDP **R01878:** Cytidine + H_2O → NH_3 + Uridine	+	+	+	+	+	+	+	+
18	**C00033_tx:** X_Acetate ↔ Acetate **R00658:** 2PG ↔ H_2O + PEP	+	+	+	+	+	+	+	+

Table 8.7: Essential reactions for peptidoglycan biosynthesis.

		2-nd				3-rd
	Minimal cut sets	sag	sak	san	coh	sag
1	**R05030:** NH_3 + C05893 + ATP → Pi + ADP + C05894	+	+	+	+	+
2	**R01066:** 2-Deoxy-ribose 5-P ↔ Acetaldehyde + G3P				+	
3	**R04573:** Alanyl-Ala + C05892 + ATP → Pi + ADP + C04702	+	+	+	+	+
4	**R03193:** UDP-N-acetylmuramate + Ala + ATP → Pi + ADP + UDP-N-acetylmuramoyl-alanine	+	+	+	+	+
5	**R00742:** Acetyl-CoA + ATP + HCO_3^- → Malonyl-CoA + Pi + ADP	+	+	+	+	+
6	**R00660:** UDP-N-acetyl-glucosamine + PEP ↔ C04631 + Pi	+	+	+	+	+
7	**R00768:** F6P + Gln ↔ Glucosamine 6-P + Glu	+	+	+	+	+
8	**R00260:** Glu ↔ Glu	+	+	+	+	+
9	**Carboxyanhydrase:** HCO_3^- ↔ CO_2 + H_2O	+	+	+	+	+
10	**syn_UDC_P**	+	+	+	+	+
11	**syn_FATTY_ACID**	+	+	+	+	+
12	**R02483:** Deoxyuridine + Pi ↔ Uracil + 2-Deoxy-ribose 1-P				+	
13	**C00047_tx:** X_Lys ↔ Lys	+	+	+	+	+
14	**R01150:** 2 Ala + ATP → Pi + Alanyl-Ala + ADP	+	+	+	+	+
15	**R02783:** Glu + UDP-N-acetylmuramoyl-alanine + ATP → C00692 + Pi + ADP	+	+	+	+	+
16	**R02060:** α-Glucosamine 1-P ↔ Glucosamine 6-P	+	+	+	+	+
17	**R02786:** C00692 + Lys + ATP ↔ Pi + ADP + C05892	+	+	+	+	+
18	**C00009_tx:** X_Pi ↔ Pi	+	+	+	+	+
19	**R00966:** PPi + UMP ↔ Uracil + 5-Phospho-α-ribose 1-diphosphate				+	
20	**R05629:** Undecaprenyl phosphate + C04702 ↔ C04851 + UMP	+	+	+	+	+
21	**R00127:** AMP + ATP → 2 ADP				+	
22	**R06173:** UDP-N-acetyl-glucosamine + C04851 ↔ C05893 + UDP	+	+	+	+	+
23	**R05332:** Acetyl-CoA + α-Glucosamine 1-P ↔ CoA + N-Acetyl-glucosamine 1-P	+	+	+	+	+
24	**R00004:** PPi + H_2O → 2 Pi	+	+	+	+	+
25	**R02098:** dUMP + ATP → dUDP + ADP				+	
26	**R00415:** N-Acetyl-glucosamine 1-P + UTP ↔ UDP-N-acetyl-glucosamine + PPi	+	+	+	+	+
27	**R02099:** Deoxyuridine + ATP → dUMP + ADP				+	
28	**R02749:** 2-Deoxy-ribose 1-P ↔ 2-Deoxy-ribose 5-P				+	

Table 8.8: Minimal cut sets of length 2 for peptidoglycan biosynthesis.

		2-nd				3-rd
	Minimal cut sets	sag	sak	san	coh	sag
29	**R00315:** Acetate + ATP ↔ Acetyl phosphate + ADP **R01512:** 3PG + ATP ↔ BPG + ADP	+	+	+	+	+
30	**R00158:** UMP + ATP → ADP + UDP **R02749:** 2-Deoxy-ribose 1-P ↔ 2-Deoxy-ribose 5-P	+	+	+	+	+
31	**R02018:** dUDP + Oxidized thioredoxin + H_2O ↔ Thioredoxin + UDP **R02331:** dUDP + ATP ↔ ADP + dUTP				+	
32	**C01401_tx:** X_Ala ↔ Ala **R05861:** O_2 + Ala + H_2O ↔ H_2O_2 + NH_3 + Pyr	+	+	+	+	+
33	**R00158:** UMP + ATP → ADP + UDP **R01066:** 2-Deoxy-ribose 5-P ↔ Acetaldehyde + G3P	+	+	+	+	+
34	**R03191:** NAD^+ + UDP-N-acetylmuramate ↔ C04631 + NADH **R03192:** $NADP^+$ + UDP-N-acetylmuramate ↔ NADPH + C04631	+	+	+	+	+
35	**R00158:** UMP + ATP → ADP + UDP **R02483:** Deoxyuridine + Pi ↔ Uracil + 2-Deoxy-ribose 1-P	+	+	+	+	+
36	**R02016:** $NADP^+$ + Thioredoxin ↔ NADPH + Oxidized thioredoxin **syn_DNA**				+	
37	**C00302_tx:** X_Glu ↔ Glu **C00303_tx:** X_Gln ↔ Gln	+	+	+	+	+

Table 8.9: Minimal cut sets of length 2-4 for ATP production.

		1-st			2-nd				3-rd
	Minimal cut sets	sag	sak	san	sag	sak	san	coh	sag
1	**R00315:** Acetate + ATP ↔ Acetyl phosphate + ADP **R01512:** 3PG + ATP ↔ BPG + ADP	+	+	+	+	+	+	+	+
2	**C00033_tx:** X_Acetate ↔ Acetate **R00658:** 2PG ↔ H_2O + PEP	+	+	+	+	+	+	+	+
3	**C00033_tx:** X_Acetate ↔ Acetate **R01512:** 3PG + ATP ↔ BPG + ADP	+	+	+	+	+	+	+	+
4	**R00315:** Acetate + ATP ↔ Acetyl phosphate + ADP **R01015:** G3P ↔ Glycerone phosphate **R01529:** Ribulose 5-P ↔ Xylulose 5-P	+	+	+	+	+	+	+	+
5	**R00230:** Acetyl-CoA + Pi ↔ CoA + Acetyl phosphate **R01061:** Pi + NAD^+ + G3P ↔ NADH + BPG **R01063:** $NADP^+$ + G3P + Pi ↔ NADPH + BPG	+		+					
6	**C00033_tx:** X_Acetate ↔ Acetate **R01015:** G3P ↔ Glycerone phosphate **R01529:** Ribulose 5-P ↔ Xylulose 5-P	+	+	+	+	+	+	+	+
7	**C00033_tx:** X_Acetate ↔ Acetate **R01015:** G3P ↔ Glycerone phosphate **R01056:** Ribose 5-P ↔ Ribulose 5-P	+	+	+	+	+	+	+	+
8	**R00315:** Acetate + ATP ↔ Acetyl phosphate + ADP **R01015:** G3P ↔ Glycerone phosphate **R01056:** Ribose 5-P ↔ Ribulose 5-P	+	+	+	+	+	+	+	+
9	**C00033_tx:** X_Acetate ↔ Acetate **R01061:** Pi + NAD^+ + G3P ↔ NADH + BPG **R01063:** $NADP^+$ + G3P + Pi ↔ NADPH + BPG	+	+	+	+	+	+	+	+
10	**C01401_tx:** X_Ala ↔ Ala **R00220:** Ser ↔ NH_3 + Pyr **R00658:** 2PG ↔ H_2O + PEP **R00704:** NAD^+ + Lactate ↔ NADH + Pyr	+		+					
11	**R00230:** Acetyl-CoA + Pi ↔ CoA + Acetyl phosphate **R01058:** $NADP^+$ + G3P + H_2O → NADPH + 3PG **R01061:** Pi + NAD^+ + G3P ↔ NADH + BPG **R01063:** $NADP^+$ + G3P + Pi ↔ NADPH + BPG	+		+	+	+	+	+	+
12	**C00011_tx:** X_CO_2 ↔ CO_2 **C00058_tx:** X_Formate ↔ Formate **R01061:** Pi + NAD^+ + G3P ↔ NADH + BPG **R01063:** $NADP^+$ + G3P + Pi ↔ NADPH + BPG	+		+					

can be safely assumed to be vital in most of the microorganisms. Therefore, the method is reliable and universal enough to be applicable to a wide range of genome-scale metabolic reconstructions.

Algorithm 10 has some important differences from the original algorithm for the detection of minimal cut sets [55]. Firstly, the resulting set of MCS is incomplete, since only disjoint MCS are detected. Further, although the reactions are ordered by correlation coefficients with the target, the result may depend on the initial order of reactions and the low-level implementation of the sorting function used, since some reactions may have equal coefficients. On the other hand, the algorithm is highly computationally efficient, since it does not involve a calculation of elementary modes. Finally, due to the ordering of reactions, the algorithm favours the smaller minimal cut sets.

Small minimal cut sets (involving two or three reactions or corresponding enzymes) have a great practical significance as potential drug targets. Apart from the rarity of single essential reactions, such cut sets are advantageous due to the fact that it is almost impossible for an organism to develop resistance to multiple enzymes simultaneously. Hence, the detection of minimal cut sets promises to be a productive method of rational drug design.

Chapter 9

General Discussion

Although the ultimate objective of the present work was to create and analyse a genome-scale metabolic reconstruction of *Streptococcus agalactiae*, considerable efforts were made to develop methods enabling the achievement of this objective. The original methods introduced in the scope of the work include quantitative evaluation of model quality, optimised genome annotation, detection of stoichiometric inconsistencies, calculation of elementary substrate and product compositions and detection of non-overlapping minimal cut sets. These methods are intended to be widely applicable to computational analysis of metabolic networks in general; their practical and theoretical implications are discussed in the final sections of the corresponding chapters. In this chapter, we summarise and discuss the results of reconstruction and analysis of the modelled organism. Further, we give an outlook of the possible future directions.

9.1 Modelling results

Genome-scale metabolic models of four strains of *Streptococcus agalactiae* were constructed. The primary input data were obtained from two alternative sources: publicly available annotations from the KEGG database and *de novo* annotations generated using the tools RPS-BLAST and PRIAM. These data were used to construct two alternative lines of models for each strain (except for `coh`, for which no KEGG annotation is available).

A range of curation techniques were developed and applied to the input data. In particular, standardised identifiers were used to avoid name conflicts between metabolites defined on different semantic levels, such as glucose and α-D-glucose. Isostoichiometric reactions, those involving polymers or generic metabolites and those violating the balance of atomic species other than hydrogen were excluded from the input. Irreversible reactions were defined manually, based on a literature search. A number of hypothetical reactions were included into the models, such as transporters, generic ATPase, NADH and NADPH oxidases and reactions representing the synthesis of major biomass components, namely protein,

RNA, DNA, membrane and peptidoglycan. Hence, the resulting models represented small molecule metabolism, were atomically balanced (except for hydrogen), purified of redundancies and ambiguities, included some thermodynamic constraints and covered the processes of substrate uptake, growth and energy production.

Indicators of model quality were introduced, based on consistency with fundamental physical and biological constraints, such as mass conservation, network connectedness and ability to achieve a steady state. As a result of applying these indicators, the second-line models were reduced to the threshold E-value of 10^{-13}, which appeared to be an optimal tradeoff between the coverage and the quality of the models (see Table 5.4 and 5.5).

The proton was found to be the only unconserved metabolite in all models; after its removal, the models became stoichiometrically consistent. Although the removal of the proton increases the percentage of unbalanced reactions and may possibly result in violations of redox balance, these did not appear to affect any of the further analysis results. Algorithm 6 was used to block all internal cycles involving ATPase, thus enabling the calculation of energetic efficiency of catabolic pathways. The values of quality indicators depending on the steady-state constraint remained relatively low (see Table 5.6) and required a further improvement.

Several reannotations were made in the second-line models using two alternative approaches: (i) manual inspection of the 'failed' pathways and (ii) search-space reduction followed by stochastic optimisation (namely simulated annealing). The former approach enabled restoring the peptidoglycan biosynthesis pathway by means of two reannotations. The repeated application of the simulated annealing algorithm resulted in the detection of five consensus enzymes, two out of which were thereafter assigned to certain genes and a further two were included as orphan enzymes. Thus an optimised third-line model of `sag` was constructed. The biological significance of two consensus enzymes, namely 6-phospho-beta-galactosidase and 1-pyrroline-5-carboxylate dehydrogenase was revealed by the further analysis.

Functional analysis was used to identify the possible fermentation substrates, and minimal compositions of fermentation products and of amino acids. The ability to ferment lactose was found in some of the models depending on the inclusion of 6-phospho-beta-galactosidase. We are not aware whether the experimentally confirmed ability of the organism to ferment lactose after multiple passages in a lactose-rich substrate [46] is due to a spontaneous mutation or increased expression of an enzyme. A similar process has been observed in *E.coli*, where a beta-galactosidase activity was restored after the deletion of the lacZ gene due to a sequence of mutations in other genes [3]. Hence, the optimised model represents an over-adapted state of the organism, which may be different from its known physiology *in vitro* but predicts the highly probable mutations or over-expressions, which may arise, in particular, during a disease. This agrees with the clinical observations confirming the role of breast milk in the late-onset infection [30, 123, 60]. The fact that in the KEGG annotations, 6-phospho-beta-galactosidase is predicted only for `san` and not for `sag` and `sak`

could be a possible explanation of the role of serotype III (which includes `san`) in the epidemiology of the late-onset disease. However, the difference in the ability of the strains to grow in milk should be confirmed experimentally.

The analysis of minimal compositions of fermentation products revealed some pathways which strongly diverge from the typical glycolytical routes. In particular, an elementary mode converting one mole of glucose into three moles of acetate was found (see Figure 7.5), which involves a number of enzymes typically involved in the pentose-phosphate pathway and photosynthesis. Further, fermentation pathways yielding three moles of ATP per one mole of glucose in anaerobic conditions were detected (see Figure 7.6). The fact that these conversions have (to our knowledge) not been observed experimentally may be possibly explained by the lack of thermodynamic gradient and the high numbers of enzymes involved, which increases the expression costs (so that it cannot be compensated even by the extra molecule of ATP).

Although the oxidative phosphorylation pathway was not explicitly included into the models, the inclusion of oxygen into the NADH and NADPH oxidase reactions (see Equation 7.25) enabled the coverage of the respiration process. The analysis of minimal compositions of fermentation products in aerobic conditions revealed two important differences from anaerobic conditions: (i) decoupling of acetate, ethanol and formate due to the availability of $NADP^+$ and (ii) increase of the maximal yield of ATP per mole of glucose to four moles. However, a maximal yield of five moles in aerobic conditions has been observed experimentally [71].

The predictions of essential amino acids (see Table 7.7) demonstrated a good agreement with experimental results. Only two predictions were not confirmed by any of the available experimental data: Firstly, proline was found to be essential in the first and second-line models, but the prototrophy for proline was restored in the third-line model by including 1-pyrroline-5-carboxylate dehydrogenase. Secondly, phenylalanine was found to be non-essential in the second and third-line models. Interestingly enough, these models could be made auxotrophic for phenylalanine by reducing the threshold E-value to 10^{-15}. However, in terms of essentiality analysis and drug target prediction, it is safer to overestimate rather than to underestimate the metabolic capabilities of an organism. On the other hand, the auxotrophy for phenylalanine *in vivo* could be also caused by thermodynamic or regulatory constraints not included into the models.

Essentiality analysis was performed using biosynthesis reactions and ATPase as targets. The results presented in Chapter 8 suggest a number of essential reactions and small minimal cut sets which can be screened as potential drug targets. In general, DNA and cell wall synthesis appear to be the most vulnerable target functions (see Table 8.1), whereas ATP production is extremely robust (no essential reactions and only three cut sets of length two were detected).

No separate chapter was dedicated to the comparative analysis of the models and strains, but the results described in Chapter 7 and 8 elucidate some important differences, which can be attributed to the variations in the presence of relatively few enzymes, summarised in Table 9.1. Note that this is only a small

Table 9.1: Variations in the presence of enzymes in the models, which affect the results of functional and essentiality analysis.

		1-st			2-nd				3-rd
EC	**name**	sag	sak	san	sag	sak	san	coh	sag
3.1.3.5	5'-nucleotidase	+	+	+					
3.2.1.85	6-phospho-beta-galactosidase			+			+		+
1.1.1.5	acetoin dehydrogenase		+		+	+	+	+	+
2.1.2.10	aminomethyltransferase				+	+	+	+	+
2.7.4.14	cytidylate kinase	+	+	+	+	+	+		+
3.1.3.11	fructose-bisphosphatase	+	+	+					
4.1.2.9	phosphoketolase		+		+	+	+	+	+
1.5.1.12	1-pyrroline-5-carboxylate dehydrogenase								+

part of all the enzymes which are present in some of the models and absent in the others. Hence, the results of functional and essentiality analysis are much more sensitive in terms of detecting the crucial differences between the models than a simple comparison of enzyme sets.

The inspection of Table 9.1 and of the distance trees shown in Table 7.9a and 8.2 leads to one of the principal conclusions of the whole work: the models of different strains constructed using the same methods are more similar than the models of the same strains constructed using different methods. Hence, the analysis results of metabolic reconstructions strongly depend on the sources of input data and the differences in the behaviour of the models are largely attributable to artefacts. On the other hand, Table 7.9b demonstrates that the accuracy of computational predictions is comparable with that of experimental results. This observation confirms the informativeness and reliability of genome-scale metabolic reconstructions.

9.2 Future developments

A better quality of the metabolic reconstructions could be achieved by the inclusion of some additional data, which are not currently available. A precise definition of the biomass composition (including cofactors and ions) would enable the prediction of minimal media using the methods described in Chapter 7. The inclusion of the oxidative phosphorylation pathway would help to elucidate the behaviour of the organism in aerobic conditions, such as neonatal lung. The definition of transporters based on experimental data rather than hypotheses would strongly increase the reliability of all flux-balance analysis results.

The group contribution method [68, 45] calculates the free energy changes in reactions based on the analysis of the molecular structures of the reactants. Unfortunately, the results derived using this method were not included into the models because of the lack of time. In the future, these results can be used to define the feasible reaction directions more precisely. Moreover, the free energy changes in net conversions can be calculated and used as important characteristics of flux modes.

The following developments can possibly improve the results of optimised genome annotation: Firstly, Figure 6.3 indicates the possibility of a further movement of the systems towards the global optimum, which can be achieved by changing the annealing parameters, e.g. by extending the temperature range. Secondly, it may be possible to obtain better results by using other methods of stochastic optimisation, such as genetic algorithms. Further, the adaptation of the organism in extreme conditions can be simulated using alternative objective functions based on biological assumptions, e.g. minimisation of nutritional requirements. Finally, instead of hypothetical assumptions, the objective function can evaluate the consistency of a model with experimental results, e.g. by comparing the correlation of reactions in a model with the coexpression of the genes in a microarray.

Experimental validation would immensely increase the value of the results presented in the thesis. This should include, in particular, the verification of the reannotations and essentiality predictions by means of gene knockout experiments. The minimal substrate and product compositions can be tested by metabolomic methods.

The results of essentiality analysis include hundreds of small minimal cut sets, which should undergo a computational screening before being proposed to cheminformaticians and experimentalists. The enzymes with a high similarity to human proteins should be detected by sequence analysis and excluded from the list of drug target candidates. Similarly, a comparison with the sequences of other bacteria of the normal microflora would help to develop narrow-spectrum antibiotics. The essential enzymes already known as drug targets would be of particular interest, since the corresponding drugs can be possibly used as alternative methods of chemotherapy. The ability of the organism to develop resistance against the deletion of the given cut sets can be tested by means of stochastic optimisation, namely by simulating the mutations restoring the blocked functions. The cut sets with the highest 'resistance cost' (measured as the number of iterations required for developing resistance) could be considered as potential targets for highly effective drugs. The author hopes that the methods and results presented in the thesis will attract the interest of the pharmaceutical community.

Bibliography

[1] B. Apgar, G. Greenberg, and G. Yen. Prevention of group B streptococcal disease in the newborn. *Am Fam Physician.*, 71(5):903–10, 2005.

[2] M. Arita. The metabolic world of Escherichia coli is not small. *Proc Natl Acad Sci U S A.*, 101(6):1543–7, 2004.

[3] J. Arraj and J. Campbell. Isolation and characterization of the newly evolved ebg beta-galactosidase of Escherichia coli K-12. *J Bacteriol.*, 124(2):849–56, 1975.

[4] A. Bairoch. The ENZYME database in 2000. *Nucleic Acids Res.*, 28(1):304–5, 2000.

[5] B. Bakker, H. Westerhoff, F. Opperdoes, and P. Michels. Metabolic control analysis of glycolysis in trypanosomes as an approach to improve selectivity and effectiveness of drugs. *Mol Biochem Parasitol.*, 106(1):1–10, 2000.

[6] D. Becker, M. Selbach, C. Rollenhagen, M. Ballmaier, T. Meyer, M. Mann, and D. Bumann. Robust Salmonella metabolism limits possibilities for new antimicrobials. *Nature.*, 440(7082):303–7, 2006.

[7] S. Becker and B. Palsson. Context-specific metabolic networks are consistent with experiments. *PLoS Comput Biol.*, 4(5), 2008.

[8] R. E. Behrman, R. M. Kliegman, and H. B. Jenson, editors. *Nelson Textbook of Pediatrics*. W. B. Saunders Company, Philadelphia, 16 edition, 2000.

[9] B. Bonde. *Metabolism and Bioinformatics: The Relationship Between Metabolism and Genome Structure*. PhD thesis, Oxford Brookes University, 2006.

[10] D. Budgen. *Software Design*. Addison-Wesley, 1994.

[11] A. P. Burgard, E. V. Nikolaev, C. H. Schilling, and C. D. Maranas. Flux coupling analysis of genome-scale metabolic network reconstructions. *Genome Research*, 14(2):301–312, 2004.

[12] C. Claudel-Renard, C. Chevalet, T. Faraut, and D. Kahn. Enzyme-specific profiles for genome annotation: PRIAM. *Nucleic Acids Research*, 31(22):6633–6639, 2003.

[13] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley & Sons, Inc., New York, 1998.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2 edition, 2001.

[15] T. Dandekar, S. Schuster, B. Snel, M. Huynen, and P. Bork. Pathway alignment: application to the comparative analysis of glycolytic enzymes. *Biochem.J.*, 343:115–124, 1999.

[16] L. L. Dines. On Positive Solutions of a System of Linear Equations. *The Annals of Mathematics*, 28(1):386–392, 1926.

[17] O. Ebenhöh and T. Handorf. Expanding metabolic networks: A novel method for structural analysis. Oral presentation at 11th Workshop of the BioThermoKinetics group's meeting 'Developing Concept for Systems Biology' Oxford, UK., 3 Sept. 2004.

[18] O. Ebenhöh and T. Handorf. Structural analysis of expanding metabolic networks. *Genome Informatics*, 15(1):35–45, 2004.

[19] J. Edwards, R. Ibarra, and B. Palsson. In silico predictions of Escherichia coli metabolic capabilities are consistent with experimental data. *Nat Biotechnol.*, 19(2):125–30, 2001.

[20] J. Edwards and B. Palsson. The E.coli MG1655 in silico metabolic genotype: its definition, characteristics and capabilities. *PNAS*, 97:5528–5533, 2000.

[21] J. S. Edwards, M. Covert, and B. Palsson. Metabolic modelling of microbes: the flux-balance approach. *Environmental Microbiology*, 4(3):133–140, 2002.

[22] A. Eliens. *Object-oriented software development*. Addison-Wesley, 2000.

[23] I. Famili and B. Palsson. The convex basis of the left null space of the stoichiometric matrix leads to the definition of metabolically meaningful pools. *Biophys.J.*, 85(1):16–26, 2003.

[24] D. Fell. *Understanding the Control of Metabolism*. Portland Press, London, 1997.

[25] D. Fell. *Biological Networks*. World Scientific, 2006.

[26] D. B. Fogel. *Evolutionary Computation*. IEEE Press, 2000.

[27] J. Forster, I. Famili, P. Fu, B. Palsson, and J. Nielsen. Genome-Scale Reconstruction of the Saccharomyces cerevisiae Metabolic Network. *Genome Res.*, 13(2):244–253, 2003.

[28] C. Francke, R. J. Siezen, and B. Teusink. Reconstructing the metabolic network of a bacterium from its genome. *Trends in Microbiology*, 13(11):550–8, 2005.

[29] J. Gagneur, D. B. Jackson, and G. Casari. Hierarchical analysis of dependency in metabolic networks. *Bioinformatics*, 19(8):1027–1234, 2003.

[30] V. Gajdos, A. Domelier, C. Castel, M. Guibert, F. Perreaux, A. Mollet, L. Lebrun, R. Quentin, and P. Labrune. Late-onset and recurrent neonatal Streptococcus agalactiae infection with ingestion of infected mother's milk. *Eur J Obstet Gynecol Reprod Biol.*, 136(2):265–7, 2006.

[31] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns. Elements of reusable object-oriented software.* Addison-Wesley, 1997.

[32] D. Garfinkel and B. Hess. Metabolic control mechanisms VII. A detailed computer model of the glycolytic pathway in ascites cells. *J. Biol. Chem*, 239:971–983, 1964.

[33] A. Gevorgyan, M. Poolman, and D. Fell. Detection of stoichiometric inconsistencies in biomolecular models. *Bioinformatics*, 24(19):2245–51, 2008.

[34] P. Glaser, C. Rusniok, C. Buchrieser, F. Chevalier, L. Frangeul, T. Msadek, M. Zouine, E. Couve, L. Lalioui, C. Poyart, P. Trieu-Cuot, and F. Kunst. Genome sequence of Streptococcus agalactiae, a pathogen causing invasive neonatal disease. *Mol Microbiol.*, 45(6):1499–513, 2002.

[35] S. Goto, T. Nishioka, and M. Kanehisa. LIGAND: chemical database of enzyme reactions. *Nucleic Acids Res.*, 28(1):380–2, 2000.

[36] T. Handorf, N. Christian, O. Ebenhöh, and D. Kahn. An environmental perspective on metabolism. *J Theor Biol.*, 252(3):530–7, 2008.

[37] R. Heinrich and S. Schuster. *The Regulation of Cellular Systems.* Chapman & Hall, London, England, 1996.

[38] V. Hemming, K. Nagarajan, L. Hess, G. Fischer, S. Wilson, and L. Thomas. Rapid in vitro replication of group B streptococcus in term human amniotic fluid. *Gynecol Obstet Invest.*, 19(3):124–9, 1985.

[39] M. Herbert, C. Beveridge, D. McCormick, E. Aten, N. Jones, L. Snyder, and N. Saunders. Genetic islands of Streptococcus agalactiae strains NEM316 and 2603VR and their presence in other Group B streptococcal strains. *BMC Microbiol.*, 5(1):31, 2005.

[40] M. Herbert, C. Beveridge, and N. Saunders. Bacterial virulence factors in neonatal sepsis: group B streptococcus. *Curr Opin Infect Dis.*, 17(3):225–9, 2004.

[41] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI–a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, Dec 2006.

[42] M. Hucka and et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical networks. *Bioinformatics*, 19(4):524–531, 2003.

[43] R. Ibarra, J. Edwards, and B. Palsson. Escherichia coli K-12 undergoes adaptive evolution to achieve in silico predicted optimal growth. *Nature.*, 420(6912):186–9, 2002.

[44] J. M. Berg, J. L. Tymoczko, L. Stryer, and N. D. Clarke. *Biochemistry.* W. H. Freeman and Company, 5 edition, 2002.

[45] M. Jankowski, C. Henry, L. Broadbelt, and V. Hatzimanikatis. Group contribution method for thermodynamic analysis of complex metabolic networks. *Biophys J.*, 95(3):1487–99, 2008.

[46] N. Jensen. Epidemiological aspects of human/animal interrelationship in GBS. *Antibiot Chemother.*, 35(40), 1985.

[47] X. Jia. *Object-oriented software development.* Addison-Wesley, 2000.

[48] A. Jones, K. Knoll, and C. Rubens. Identification of Streptococcus agalactiae virulence genes in the neonatal rat sepsis model using signature-tagged mutagenesis. *Mol Microbiol.*, 37(6):1444–55, 2000.

[49] M. Kanehisa, S. Goto, M. Hattori, K. F. Aoki-Kinoshita, M. Itoh, S. Kawashima, T. Katayama, M. Araki, and M. Hirakawa. From genomics to chemical genomics: new developments in KEGG. *Nucleic Acids Research*, 34(Database Issue):D354–D357, 2006.

[50] P. Karp, M. Riley, M. Saier, I. Paulsen, S. Paley, and A. Pellegrini-Toole. The EcoCyc and MetaCyc databases. *Nucleic Acids Res.*, 28(1):56–59, 2000.

[51] P. D. Karp, S. Paley, and P. Romero. The pathway tools software. *Bioinformatics*, 18:S225–32, 2002.

[52] G. Keefe. Streptococcus agalactiae mastitis: a review. *Can Vet J.*, 38(7):429–37, 1977.

[53] S. Kirkpatrick, C. D. Gelatt, Jr., and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[54] H. Kitano. Biological robustness. *Nat Rev Genet.*, 5(11):826–37, 2004.

[55] S. Klamt and E. Gilles. Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20(2):226–234, 2004.

[56] S. Klamt, J. Saez-Rodriguez, and E. Gilles. Structural and functional analysis of cellular networks with CellNetAnalyzer. *BMC Syst Biol*, 1:2, 2007.

[57] S. Klamt and J. Stelling. Two approaches for metabolic pathway analysis? *TIBS*, 21(2):64–69, 2003.

[58] L. Alberghina and H. V. Westerhoff, editors. *Systems Biology*. Springer, Berlin Heidelber, 2005.

[59] A. Laisk and D. A. Walker. A mathematical model of electron transport. Thermodynamic neccessity for PSII regulation: "light stomata". *Proc. R. Soc. Lond. Ser. B*, 237:417–444, 1989.

[60] M. Lanari, L. Serra, F. Cavrini, G. Liguori, and V. Sambri. Late-onset group B streptococcal disease by infected mother's milk detected by polymerase chain reaction. *New Microbiol.*, 30(3):253–4, 2007.

[61] N. Lemke, F. Herédia, C. K. Barcellos, A. N. dos Reis, and J. C. M. Mombach. Essentiality and damage in metabolic networks. *Bioinformatics*, 20(1):115 119, 2004.

[62] A. M. Lesk. *Introduction to Bioinformatics*. Oxford University Press, 2002.

[63] C. Lloyd, J. Lawson, P. Hunter, and P. Nielsen. The CellML Model Repository. *Bioinformatics*, 24(18):2122–3, 2008.

[64] M. Lutz and D. Ascher. *Learning Python*. O'Reilly & Associates, 1999.

[65] M. Sirava, T. Schafer, M. Eiglsperger, M. Kaufmann, O. Kohlbacher, E. Bornberg-Bauer, and H. P. Lenhof. BioMiner – modeling, analyzing and visualizing biochemical pathway and networks. *Bioinformatics*, 1(1):1–12, 2002.

[66] H.-W. Ma and A.-P. Zeng. The connectivity structure, giant strong component and centrality of metabolic networks. *Bioinformatics*, 19(11):1423 1430, 2003.

[67] A. Marchler-Bauer, A. Panchenko, B. Shoemaker, P. Thiessen, L. Geer, and S. Bryant. CDD: a database of conserved domain alignments with links to domain three-dimensional structure. *Nucleic Acids Res.*, 30(1):281–3, 2002.

[68] M. L. Mavrovuniotis. Estimation of standard Gibbs energy changes in biotransformations. *J Biol Chem.*, 266(22):14440–5, 1991.

[69] P. Mendes. Gepasi - a software package for modeling the dynamics, steady-states and control of biochemical and other systems. *Comp. Appl. Biosci.*, 9(5):563–571, 1993.

[70] G. Michal. *Biochemical pathways: an atlas of biochemistry and molecular biology*. Wiley, 1999.

[71] M. Mickelson. Glucose degradation, molar growth yields, and evidence for oxidative phosphorylation in Streptococcus agalactiae. *J Bacteriol.*, 109(1):96–105, 1972.

[72] T. Milligan, T. Doran, D. Straus, and S. Mattingly. Growth and amino acid requirements of various strains of group B streptococci. *J Clin Microbiol.*, 7(1):28–33, 1978.

[73] E. Nikolaev, A. Burgard, and C. Maranas. Elucidation and structural analysis of conserved pools for genome-scale metabolic reconstructions. *Biophys J.*, 88(1):37–49, 2005.

[74] M. Oberhardt, J. Puchalka, K. Fryer, V. Martins dos Santos, and J. Papin. Genome-scale metabolic network analysis of the opportunistic pathogen Pseudomonas aeruginosa PAO1. *J Bacteriol.*, 190(8):2790–803, 2008.

[75] H. Ogata, S. Goto, K. Sato, W. Fujibuchi, H. Bono, and M. Kanehisa. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res.*, 27:29–34, 1999.

[76] B. Olivier, J. Rohwer, and J.-H. Hofmeyr. Modelling cellular systems with PySCeS. *Bioinformatics*, Epub., 2004.

[77] J. Papin, J. Stelling, N. Price, S. Klamt, S. Schuster, and B. Palsson. Comparison of network-based pathway analysis methods. *Trends Biotechnol*, 22(8):400–405, Aug 2004.

[78] A. Papin J, D. Price N, S. Edwards J, and O. Palsson B. The genome-scale metabolic extreme pathway structure in Haemophilus influenzae shows significant network redundancy. *J Theor Biol.*, 215:67–82, 2002.

[79] T. Pfeiffer, I. Sanchez-Valdenebro, J. Nuno, F. Montero, and S. Schuster. Metatool: for studying metabolic networks. *Bioinformatics*, 15(3):251–257, 1999.

[80] C. Phalakornkule, S. Lee, T. Zhu, R. Koepsel, M. Ataai, I. Grossmann, and M. Domach. A MILP-based flux alternative generation and NMR experimental design strategy for metabolic engineering. *Metab Eng.*, 3(2):124–37, 2001.

[81] J. Pinney, M. Shirley, G. McConkey, and D. Westhead. metaSHARK: software for automated metabolic network prediction from DNA sequence and its application to the genomes of Plasmodium falciparum and Eimeria tenella. *Nucleic Acids Res.*, 33(4):1399–409, 2005.

[82] M. G. Poolman. ScrumPy - metabolic modelling with Python. *IEE Proceedings Systems Biology*, 153(5):375–378, 2006.

[83] M. G. Poolman, H. E. Assmus, and D. A. Fell. Applications of metabolic modelling to plant metabolism. *J.Exp.Bot.*, 55(400):1177–1186, 2004.

[84] M. G. Poolman, B. K. Bonde, A. Gevorgyan, H. H. Patel, and D. A. Fell. Challenges to be faced in the reconstruction of metabolic networks from public databases. *IEE Proceedings Systems Biology*, 153(5):379–384, 2006.

[85] M. G. Poolman, H. Ölcer, J. C. Lloyd, C. A. Raines, and D. A. Fell. Computer modelling and experimental evidence for two steady states in the photosynthetic calvin cycle. *Eur. J. Biochem.*, 268:2810–2816, 2001.

[86] M. G. Poolman, C. Sebu, M. K. Pidcock, and D. A. Fell. Modular decomposition of metabolic systems via null-space analysis. *Journal of Theoretical Biology*, 249:691–705, 2007.

[87] L. Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, 2000.

[88] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipies in C*, chapter 9. Cambridge University Press, Cambridge, 1989. Newts method.

[89] N. Price, J. Papin, C. Schilling, and B. Palsson. Genome-scale microbial in silico models: the constraints-based approach. *Trends Biotechnol*, 21(4):162–169, Apr 2003.

[90] R. Urbanczik and C. Wagner. An improved algorithm for stoichiometric network analysis: theory and applications. *Bioinformatics*, 21(7):1203–1210, 2005.

[91] L. Rajagopal, A. Vo, A. Silvestroni, and C. Rubens. Regulation of purine biosynthesis by a eukaryotic-type kinase in Streptococcus agalactiae. *Mol Microbiol.*, 56(5):1329–46, 2005.

[92] E. Ravasz, A. Somera, D. Mongru, Z. Oltvai, and A.-L. Barabási. Hierarchical Organization of Modularity in Metabolic Networks. *Science*, 297:1551–5, 2002.

[93] J. Reed, T. Patel, K. Chen, A. Joyce, M. Applebee, C. Herring, O. Bui, E. Knight, S. Fong, and B. Palsson. Systems approach to refining genome annotation. *Proc Natl Acad Sci U S A*, 103(46):17480–4, 2006.

[94] J. L. Reed and B. O. Palsson. Genome-scale in silico models of E.coli have multiple equivalent phenotypic states: assessment of correlated reaction subsets that comprise network states. *Genome Research*, 14:1797–1805, 2004.

[95] J. L. Reed, T. D. Vo, C. H. Schilling, and B. O. Palsson. An expanded genome-scale model of E.coli K-12 (iJR904 GSM/GPR). *Genome Biology*, 4:R54:1–12, 2003.

[96] O. Richter, A. Betz, and C. Giersch. The response of oscillating glycolysis to perturbations in the NAD/NADH system: a comparison between experiments and a computer model. *Biosystems*, 7:137–146, 1975.

[97] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[98] U. Samen, B. Gottschalk, B. Eikmanns, and D. Reinscheid. Relevance of peptide uptake systems to the physiology and virulence of Streptococcus agalactiae. *J Bacteriol.*, 186(5):1398–408., 2004.

[99] K. V. Satish, M. Dasika, and C. Maranas. Optimization based automated curation of metabolic reconstructions. *BMC Bioinformatics.*, 20(8):212, 2007.

[100] H. Sauro. Jarnac: a system for interactive metabolic analysis. In *Hofmeyr, J.H., Rohwer, J.M., Snoep, J.L. (Eds.). Animating the Cellular Map: Proceedings of the 9th International Meeting on BioThermoKinetics*, pages 221–228. Stellenbosch University Press, 2000.

[101] H. Sauro and D. Fell. SCAMP: a metabolic simulator and control analysis program. *Mathl. Comput. Modelling*, 15:15–28, 1991.

[102] C. Schilling, M. Covert, I. Famili, G. Church, J. Edwards, and B. Palsson. Genome-Scale metabolic model of Helicobacter pylori 26695. *J. Bacteriology*, 184(16):4582–4593, 2002.

[103] C. Schilling, D. Letscher, and B. Palsson. Theory for the systemic definition of metabolic pathways and their use in interpreting metabolic function from a pathway oriented perspective. *J.Theor.Biol.*, 203:229–248, 2000.

[104] C. Schilling and B. Palsson. Assessment of the metabolic capabilities of Haemophilus influenzae Rd through a genome-scale pathway analysis. *J.Theor.Biol.*, 203(3):249–283, 2000.

[105] I. Schomburg, A. Chang, and D. Schomburg. BRENDA, enzyme data and metabolic information. *Nucleic Acids Res.*, 30(1):47–9, 2002.

[106] S. Schrag, R. Gorwitz, K. Fultz-Butts, and A. Schuchat. Prevention of perinatal group B streptococcal disease. Revised guidelines from CDC. *MMWR Recomm Rep.*, 51(RR-11):1–22, 2002.

[107] A. Schuchat. Group B streptococcus. *Lancet.*, 353(9146):51–56, 1999.

[108] S. Schuster, D. Fell, and T. Dandekar. A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks. *Nature Biotech.*, 18:326–332, 2000.

[109] S. Schuster and C. Hilgetag. On Elementary Flux Modes in Biochemical Systems at Steady State. *J.Biol.Syst.*, 2:165–182, 1994.

[110] S. Schuster, C. Hilgetag, J. Woods, and D. Fell. Reaction routes in biochemical reaction systems: algebraic properties, validated calculation procedure and example from nucelotide metabolism. *J.Math.Biol.*, 45:153–181, 2002.

[111] S. Schuster and T. Höfer. Determining all extreme semi-positive conservation relations in chemical reaction systems: a test criterion for conservativity. *J.Chem.Soc.Faraday Trans.*, 87:2561–2566, 1991.

[112] S. Schuster, S. Klamt, W. Weckwerth, M. Moldenhauer, and T. Pfeiffer. Use of network analysis of metabolic systems in bioengineering. *Bioprocess Biosys. Eng.*, 24:363–373, 2002.

[113] S. Schuster and R. Schuster. Detecting strictly detailed balanced subnetworks in open chemical-reaction networks. *J.Math.Chem.*, 6(1):17–40, 1991.

[114] R. Schwarz, P. Musch, A. von Kamp, B. Engels, H. Schirmer, S. Schuster, and T. Dandekar. YANA - a software tool for analyzing flux modes, gene-expression and enzyme activities. *BMC Bioinformatics*, 6:135, 2005.

[115] H. Tettelin, V. Masignani, M. Cieslewicz, C. Donati, D. Medini, N. Ward, S. Angiuoli, J. Crabtree, A. Jones, A. Durkin, R. Deboy, T. Davidsen, M. Mora, M. Scarselli, I. Margarit y Ros, J. Peterson, C. Hauser, J. Sundaram, W. Nelson, R. Madupu, L. Brinkac, R. Dodson, M. Rosovitz, S. Sullivan, S. Daugherty, D. Haft, J. Selengut, M. Gwinn, L. Zhou, N. Zafar, H. Khouri, D. Radune, G. Dimitrov, K. Watkins, K. O'Connor, S. Smith, T. Utterback, O. Whitem, C. Rubens, G. Grandi, L. Madoff, D. Kasper, J. Telford, M. Wessels, R. Rappuoli, and C. Fraser. Genome analysis of multiple pathogenic isolates of Streptococcus agalactiae: implications for the microbial "pan-genome". *Proc Natl Acad Sci U S A.*, 102(39):13950–5, 2005.

[116] A. B. Tucker and R. E. Noonan. *Programming Languages. Principles and Paradigms.* McGraw-Hill, 2007.

[117] R. Urbanczik. SNA–a toolbox for the stoichiometric analysis of metabolic networks. *BMC Bioinformatics.*, 7(129), 2006.

[118] R. Urbanczik and C. Wagner. Functional stoichiometric analysis of metabolic networks. *Bioinformatics.*, 21(22):4176–80, Nov. 2005.

[119] L. Vaserstein. *Introduction to Linear Programming.* Pearson Education, Inc., New Jersey, US, 2003.

[120] W. Vollmer, D. Blanot, and M. de Pedro. Peptidoglycan structure and architecture. *FEMS Microbiol Rev.*, 32(2):149–67., 2008.

[121] K. Voss, M. Heiner, and I. Koch. Steady state analysis of metabolic pathways using Petri nets. *In Silico Biol.*, 3(3):367–87, 2003.

[122] A. Wagner and D. Fell. The small world inside large metabolic networks. *Proc. Roy. Soc. London B*, 268:1803–1810, 2001.

[123] L. Wang, C. Chen, W. Liu, and Y. Wang. Recurrent neonatal group B streptococcal disease associated with infected breast milk. *Clin Pediatr (Phila).*, 46(6):547–9, 2007.

[124] D. Whitley. A genetic algorithm tutorial. Technical report, Colorado State University, 1993.

[125] N. Willett and G. Morse. Long-chain fatty acid inhibition of growth of Streptococcus agalactiae in a chemically defined medium. *J Bacteriol.*, 91(2245-50):6, 1966.

[126] Y. Yamamoto, C. Poyart, P. Trieu-Cuot, G. Lamberet, A. Gruss, and P. Gaudu. Respiration metabolism of Group B Streptococcus is activated by environmental haem and quinone and contributes to virulence. *Mol Microbiol.*, 56(2):525–34, 2005.

[127] I. Yeh, T. Hanekamp, S. Tsoka, P. Karp, and R. Altman. Computational analysis of Plasmodium falciparum metabolism: organizing genomic information to facilitate drug discovery. *Genome Res.*, 14(5):917–24, 2004.

[128] I. Zevedei-Oancea and S. Schuster. Topological analysis of metabolic networks based on Petri net theory. *In Silico Biol.*, 3(3):323–45, 2003.

Appendix A

Published material