Doctoral Thesis

Large scale methods to enumerate extreme rays and elementary modes

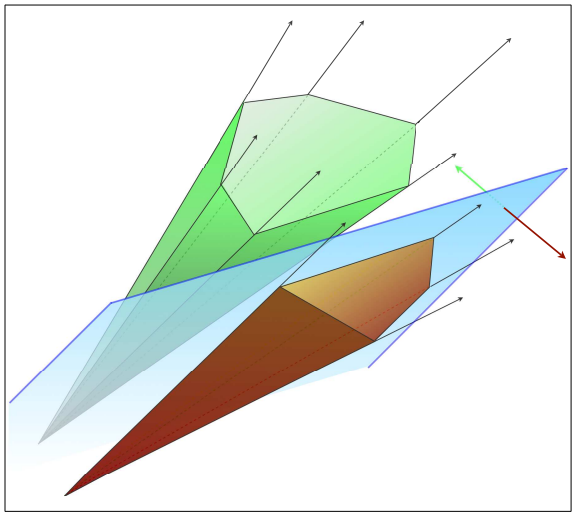
Author(s):
Terzer, Marco

Publication Date:
2009

Permanent Link:
https://doi.org/10.3929/ethz-a-005945733 →

Diss. ETH No. 18538 2009

Large Scale Methods to Enumerate Extreme Rays and Elementary Modes



Marco Terzer

Diss. ETH No. 18538

Large Scale Methods to Enumerate Extreme Rays and Elementary Modes

A dissertation submitted to the
Swiss Federal Institute of Technology, Zurich

for the degree of
Doctor of Sciences

presented by
Marco Terzer
Dipl. Informatik-Ing. ETH

born February 2, 1973
citizen of Ebnat-Kappel (SG), Switzerland

accepted on the recommendation of
Prof. Dr. Jörg Stelling, examiner
Prof. Dr. Komei Fukuda, co-examiner
Prof. Dr. Gaston H. Gonnet, co-examiner
Prof. Dr. Leen Stougie, co-examiner

2009

Abstract

Approaching biological systems quantitatively using mathematical modeling techniques has gained increasing popularity in recent years. In systems biology, the biological complex is considered *as a whole*, as opposed to studying individual components and interactions. However, this can be very challenging even for simple bacteria, and various modeling techniques have been proposed approaching the enormous complexity at different levels. Simple methods at a high level of abstraction many times lack predictability and significance; detailed models capturing thermodynamic particulars are often limited due to gappy mechanistic knowledge and unknown kinetic parameters. Structural analysis as an intermediate attempt is based on usually wellknown network stoichiometries. Constraint-based methods like linear or convex optimization are able to predict reaction fluxes, growth rates or viability of knockout mutants with high confidence. However, their significance depends heavily on the underlying objectives, and alternative solutions in the flux space are mostly ignored.

Comprehensive approaches exist, aiming at analyzing the flux space *as a whole*. The flux space is a high dimensional solution space for reaction fluxes, bounded by linear constraints on the flux values. Pathway analysis methods define minimal functional modes in the network that are able to comply with the constraints. All possible operation modes are superpositions of such *elementary modes*. Methods performing the analysis are transforming the descriptive constraints into generative basis vectors. Mathematically, the flux cone shapes a *polyhedral cone*, and algorithms arise from computational geometry, performing a representation conversion for the cone. Extreme ray enumeration, facet enumeration, vertex enumeration and convex hull are representatives of this family, and they are all related, if not equivalent. Unfortunately, the computation struggles with combinatorial explosion and is computationally intensive. At the beginning of this work, no implementation was applicable to genome scale metabolic networks. Improving the algorithms towards genome scale application is the declared goal of this thesis.

The double description method is an algorithm to enumerate extreme rays of a polyhedral cone—or of elementary modes in biological terminology. It has proven efficient especially for degenerate problems, where points and constraints are not in general (e.g. random) position. Most biochemical networks lead to degenerate problems, hence the double description method is usually chosen for pathway computations. Our own implementation is also based on this method, and the present work describes the most important aspects to attain an efficient implementation.

Various parts of the algorithm are performance critical and must be considered: it starts with input data, and we review and propose methods to sort and compress the data structures. We also show how to deal with different number types, since we need exact arithmetic for certain ill conditioned problem cases. A central part of the algorithm deals with *elementarity* of the

computed generators. Most of the computation time is spent on elementarity testing, and we propose new methods speeding up this part considerably. They are based on multidimensional search trees, and we execute superset lookup queries for combinatorial elementarity tests. A new two-tree traversal method further improves elementarity testing, and we show how to perform rank update strategies as an alternative method for testing.

As a next step, we aim at scaling up our methods to larger cases: memory becomes an issue, and we introduce out-of-core strategies storing intermediary results in files—out of the core memory. We also introduce two approaches to parallelize the algorithm, utilizing modern multi-core architectures. Finally, we apply our algorithm to real-world biological networks and demonstrate the capability of our methods. In the largest application, we compute over 390 million generators for a genome scale network of *Staphylococcus aureus*. Even if we cannot yet analyze all available genome scale networks, this work is a huge step in this direction. Since our implementation is not specific to biological applications, it is also a remarkable advancement for the representation conversion of highly degenerate polyhedral cones and polyhedra.

Zusammenfassung

Biologische Systeme werden seit neuestem gerne mit quantitativen Methoden angegangen. In der Systembiologie werden biologische Objekte ausserdem als ganzes betrachtet, ganz im Gegensatz zu früheren Ansätzen, wo oft nur einzelne Teile oder individuelle Interaktionen studiert wurden. Allerdings können gerade quantitative Modelle selbst für einfache Bakterien schnell sehr komplex werden. Deshalb wurden zahlreiche Techniken vorgeschlagen, um solche Systeme zu modellieren. Einfache Methoden operieren auf stark abstrahierten Modellen, sind aber auch oft nicht im Stande, verlässliche und biologisch relevante Vorhersagen zu liefern. Auf der anderen Seite der Skala haben sehr detaillierte Modelle das Problem, dass man die modellierten Vorgänge oft nicht genau genug versteht, oder aber die wesentlichen kinetischen Parameter schlicht nicht bekannt sind. Als Kompromiss werden deshalb of strukturelle Analysemethoden verwendet, die im Wesentlichen auf meist bekannten stöchiometrischen Netzwerken beruhen. Randbedingungen auf den Reaktionswerten definieren den Lösungsraum für mögliche Reaktionsflüsse, und Optimierungsmethoden liefern im allgemeinen recht gute und zuverlässige Vorhersagen für Flussverteilungen, Wachstumsraten oder die Lebensfähigkeit von Knockout–Mutanten. Allerdings basieren solche Methoden immer auf einer Optimierungsfunktion, und alternative Lösungen werden meist ausgeblendet.

Es gibt auch Ansätze, die sich einer umfassenderen Betrachtung des Lösungsraumes widmen. Der hochdimensionale sogenannte *Flussraum* ist typischerweise linear be-schränkt, was durch Grenzwerte auf den Reaktionsflüssen und auf Kombinationen von Flüssen begründet ist. *Pathway analysis* Methoden wandeln diese den Lösungsraum beschreibenden Beschränkungen in Basisvektoren um, die den Raum aufspannen. Dabei werden minimal funktionale *Modi* oder *Pathways* gefunden, die allen Randbedingungen entsprechen. Aus solchen *Elementary Modes* können dann alle möglichen gültigen Lösungen durch Superposition ermittelt werden. Mathematisch betrachtet stellt der Lösungsraum einen *Polyhedral Cone* dar, und Methoden, die eine solche Transformation durchführen, sind in der computergestützten Geometrie zu finden. Typische Vertreter sind in etwa *Extreme Ray Enumeration*, *Facet Enumeration*, *Vertex Enumeration* oder das Berechnen der konvexen Hülle. Die Methoden sind alle verwandt oder sogar äquivalent. Allerdings kann die Zahl der Basisvektoren – auch *generators* genannt – durch kombinatorische Explosion exponenziell sein gemessen an der Grösse der Eingabe. Zu Beginn dieser Arbeit war kein verfügbarer Algorithmus in der Lage, auf Netzwerke angewandt zu werden, die das ganze Genom umfassen (sogenannte *genome scale models*). Das erklärte Ziel dieser Dissertation war daher, die Methoden soweit voranzubringen, dass sie auf *genome scale networks* angewandt werden können.

Ein Vertreter der erwähnten Algorithmus–Familie ist die *Double Description Method*, welche die *Extreme Rays* eines *Polyhedral Cones* berechnet – oder die Elementarmoden, wie sie im biologischen Kontext auch genannt werden. Der Algorithmus ist speziell geeignet für degene-

rierte Probleme, bei denen Randbedingungen und Vektoren nicht in allgemeiner Position liegen (zufällige Punkte liegen zum Beispiel immer in allgemeiner Position). Degenerierte Probleme sind speziell schwierig, und Aufgaben aus der Biologie sind leider meistens von dieser Art. Deshalb wird in der Regel die *Double Description* Methode angewendet, und auch unsere Implementierung ist eine Variante davon. Die vorliegende Arbeit beschreibt die wichtigsten Aspekte, die für eine effiziente Implementierung unabdingbar sind.

Die Effizienz des Algorithmus hängt von sehr verschiedenen Teilbereichen ab, die alle sorgfältig angegangen werden müssen. Das fängt schon bei den Eingabedaten an: Zum Beispiel spielt die Sortierung der Randbedingungen eine wichtige Rolle, und wir beschreiben, wie die Daten sortiert und komprimiert werden können. Wir zeigen auch, wie mit unterschiedlichen numerischen Datentypen umgegangen wird, was für einige Probleme essenziell ist, da sie nicht mittels Fliesskomma–Arithmetik gelöst werden können. Ein zentraler Aspekt betrifft die *Elementarheit* der berechneten *generators*. Am meisten Zeit wird für *Elementarity Testing* verwendet, und wir stellen neue Techniken vor, die diesen Prozess erheblich beschleunigen. Wir verwenden dazu multi–dimensionale Suchbäume, auf denen wir den sogenannten *kombinatorischen* Test mittels Superset–Suchabfragen durchführen. Eine Strategie zur gleichzeitigen Traversierung zweier Suchbäume bringt weitere Verbesserungen, und wir zeigen auch, wie dieses Traversieren auch für den *algebraischen Elementarity Test* verwendet werden kann. Anstelle von Rang–Berechnungen auf ganzen Matrizen optimieren wir den Prozess mittels einer *rank updating* Strategie.

Als nächstes widmen wir uns den grossen Skalen: Als erstes führen wir eine Methode ein, die Zwischenresultate nicht mehr im Hauptspeicher ablegt, sondern diese in Dateien – *out-of-core memory* – auslagert. Ausserdem zeigen wir, wie der Algorithmus parallelisiert werden kann, z.B. um heutzutage handelsübliche Mehrkern–Prozessorarchitekturen sinnvoll auszunutzen. Schliesslich wenden wir unseren Algorithmus auf reelle biologische Netzwerke an und zeigen, dass sich unsere Methoden auch in der Praxis bewähren. Unsere grösste Berechnung liefert mehr als 390 Millionen *generators* für ein *genome scale* metabolisches Netzwerk von *Staphylococcus aureus*. Auch wenn wir noch nicht im Stande sind, alle verfügbaren *genome scale* Netzwerke zu analysieren, so glauben wir dennoch, einen Meilenstein auf diesem Weg erreicht zu haben. Da unsere Implementierung ausserdem nicht spezifisch für biologische Anwendungen konzipiert wurde, sind unsere Neuerungen ein grosser Vorwärtsschritt ganz allgemein für die *Representation Conversion* von degenerierten *Polyhedral Cones* und Polyhedra.

Acknowledgements

First and foremost, I would like to thank my supervisor Jörg Stelling for giving me the opportunity to work on this fascinating topic, for his constant support and interest in the project, and for many interesting and fruitful discussions—both scientific and personal. He gave me a lot of freedom to follow my own research ideas, which contributes in large parts to my enjoyment of the last four years.

I am also grateful to Gaston Gonnet, who agreed to co-supervise my thesis and who was always a source of new ideas and an authority concerning mathematical or scientific questions.

I would also like to thank Prof. Komei Fukuda, whose paper was my first contact with the topic of this thesis—a throw-in at the deep end after five years in industry. I greatly appreciate his commitment as a co-examiner.

I thank Prof. Peter Widmayer for chairing my examination. He pointed me to Jörg Stelling when I applied for a PhD position. Without him, this thesis would not have happend.

A very special thank-you goes to Leen Stougie, an expert in the field of elementary modes & pathways. He was the scientific host for my HPC Europa project at Centrum Wiskunde & Informatica (CWI) in Amsterdam, together with Frank Bruggeman. Some of the parallelization ideas presented in this thesis were born during this stay. I am particularly happy to have him on board as a co-examiner.

Several people were directly or indirectly involved in my projects. Many thanks go to Dirk Müller, my first collegue in our group, always my primary choice for questions relating to systems biology or engineering. We had many and excellent discussions, from science to politics, and from family to the meaning of life. I would also like to thank Markus Uhr for his mathematical support, Mathias Ganter and Chris Mayer for the nice collaboration with the FluxViz tool, and all the others in the group, especially those in my office, for ignoring my occasional verbal reactions to a dissent with my PC.

Thanks to Steven, a good friend and our administrator in the group, for always keeping the systems running, and for his support and effort also for special requests.

I would also like to thank Christophe Dessimoz, my buddy to do sports, but also a good friend and discussion partner at countless coffee breaks.

I also wish to convey thanks to my collaborators, Robert Schütz and Stefan Jol from the Institute of Molecular Systems Biology at ETH Zürich. Thanks go to my students, Samuel Pasquier, David Lamparter and Thomas Liphardt, for their contribution in form of semester projects.

My biggest thank goes to my friends, with Daniel Furer leading the way. He was always there for me, for a glass of wine and a good talk, or for two beers and lots of silly nonsense. I'd like to thank to my musician friends, Dirk, Nik and Pesche, for having a good time almost every Wednesday evening. Thanks to all other friends and co-workers—past and present—

which are too numerous to list. But not forgotten are the WG times, the wine or beer evenings and barbeque parties.

Thanks to my parents, who made all this possible, and who are still there if I really need them.

Above all, I want to thank Karin, my beloved companion, for her continuous support, for her patience and her love. She encouraged me to do this thesis, and she always stood by me. To her, this work is dedicated.

Zürich in June 2009
Marco

Contents

List of Abbreviations

β	Bit set representation of a zero set ζ
$\mathcal{C}$	Convex set or convex cone
$\mathcal{F}$	Face
$\mathcal{H}$	Hyperplane or halfspace
$\mathcal{L}$	Lineality space
$\mathcal{N}$	Nullspace
$\mathcal{O}$	Big-O symbol of Bachmann–Landau notation
$\mathcal{P}$	Polyhedral cone or polyhedron
$\mathbb{R}$	Real numbers
$\mathbb{R}^d$	Euclidean space in d dimensions
$\mathcal{R}$	Range
$\mathcal{S}$	Subspace or affine set
$\mathcal{T}$	Tree, usually bit pattern tree; also subtree or tree node
ζ	Zero set
COV	Coefficient of variation
EFM	Elementary flux mode
EM	Elementary mode
EP	Extreme pathway
FBA	Flux balance analysis
GCD	Greatest common divisor
I/O	Input/Output, data exchange with peripheral devices, such as disks
iFBA	Integrated flux balance analysis
IP	Integer programming
LCM	Least common multiple
LP	Linear programming
MCS	Minimal cut set

MoMA	Minimization of metabolic adjustment
MPI	Message passing interface
NET	Network-embedded thermodynamic analysis
ODE	Ordinary differential equation
QP	Quadratic programming
rFBA	Regulatory flux balance analysis
ROOM	Regulatory on/off minimization
SBML	Systems Biology Markup Language
STD	Standard deviation
SVD	Singular value decomposition
VAR	Variance

Please refer to the Index in the Appendix on page C-1 for the list of occurrence.

1 Introduction

In recent years, systems biology enjoyed gaining popularity not least because of the failure to explain and comprehend complex mechanisms simply by characterizing single parts or subcomponents of a biological system. Instead, the system is considered *as a whole*, requiring the integration of diverse components and interactions at different levels and scales. This can become very complex even for simple bacteria: at genome, transcriptome, proteome and metabolism level, there are roughly 4,000 genes, ≈5,000 mRNA molecules with >5,000 interactions, 6–10,000 proteins with over 100,000 protein interactions, and 1,000–2,000 metabolites with more than 25,000 reactions and around 1,000 catalyzing enzymes, respectively.

Developing mathematical models for such large-scale systems is thus a major challenge. Many modeling approaches for studying biological networks at different levels have been proposed—and are still a current field of research in systems biology. The integration of different approaches and modeling levels is probably even more ambitious. There arises the question of the best level for modeling and of the most appropriate method. To answer this question, it is important to keep the desired application in mind, and to know about the questions to be approached by the model. Consequently it is also intuitive to ask for significance and behavioral predictability to assess the (many times subjective) quality of a method.

In many cases, this is a major weakness of simple approaches, such as graph analytical methods. On the contrary, detailed deterministic or stochastic dynamic approaches are often limited by insufficient knowledge on mechanisms and parameters. At an intermediary level, reaction stoichiometries are usually well known even for genome scale metabolic networks. Methods like constraint based approaches are popular, for instance because they are able to predict reaction fluxes, growth rates or viability of knockout mutants. However, most of these methods apply certain objective criteria to the constrained flux space and search for an optimal solution. More comprehensive approaches—so-called *pathway analysis* methods—explore and characterize the flux space without the need of a (sometimes artificial) objective function.

The flux space—a high dimensional convex space with usually linear boundaries—is typically approached with algorithms from computational geometry. Optimization based methods include *linear programming* (LP), *quadratic programming* (QP) and related methods, and they are usually quite efficient and can easily be applied to genome scale networks. Pathway analysis methods are closely related to *extreme ray enumeration* of a *polyhedral cone*, or to the non-homogeneous equivalent, *vertex enumeration* of a *polyhedron*. Unfortunately, they scale poorly and as of today, no efficient (time-polynomial) algorithm is known.

It is the main topic of this thesis to analyze, implement and improve pathway analysis methods and apply them to realistic biological networks. Pathway methods aim at finding the minimal set of operation modes for a metabolic network, such that all feasible modes can be derived from the minimal set. The application is limited to networks of moderate size, and it is the goal of this work to evolve the methods towards genome scale application. Not so much the theoretical structure of the problems, nor analysis of algorithm complexity are emphasized. Instead, we focus more on practical and implementational aspects, with the designated aim that the improved algorithms can be used for whole-cell metabolic networks. No such implementation was available at the beginning of my thesis, with the consequence that our improved algorithm—to be seen as integral part of this work—generally enhances the scope in different application areas.

Previous Work

The most important preliminaries for our work is the double description method invented by Motzkin et al. (1953). However, we have based a lot of the mathematical background on Rockafellar (1970), the revisitation paper of Fukuda and Prodon (1995) and on the thesis of Malkin (2007). The application of extreme ray enumeration to biochemical reaction networks goes back to Schuster and Hilgetag (1994) and is popular at least since Stelling et al. (2002), where it was applied in a larger scale for the first time. The method was able to predict mutant lethality with high confidence, and key aspects of functionality and regulation could be addressed.

The most important algorithmic improvements have been introduced by Wagner (2004) by his *nullspace approach*, and by Gagneur and Klamt (2004) with the *binary approach*. Both techniques are also used in our implementation. The compression techniques presented in Gagneur and Klamt (2004) were also the basis for our compression methods described in Section 3.1. *Bit pattern trees*, a central element of our own achievements, are variants and advancements of k-d-trees invented by Bentley (1975).

Of course, there are many other important aspects and achievements related to polyhedral computation, for instance the *upper bound theorem* by McMullen (1970), which we apply to *extreme pathways* and *elementary modes* in Section 2.4.4. A recent result points towards the hardness of vertex enumeration and related methods (Khachiyan et al., 2006). However, the main goal of this work is a practical implementation for the enumeration of extreme rays for highly degenerate problems, hence theoretical aspects are not the primary focus. We give an overview of complexity, alternative algorithms and related problems in Section 2.3.3.

Main Results

In our view, the main result of this work is clearly the improved algorithm *per se*. We think that it is one of the fastest double description implementations available today. This thesis can be seen as a report of the most important steps needed to get to an efficient implementation of the double description method. Various aspects of the algorithm are critical for its performance, starting with the input structures and with row ordering strategies. With an inappropriate ordering of the input, the best implementation will fail badly. For metabolic networks, it is also important to remove redundant elements and compress the network before the computation. We address this topic in Section 3.1.

From an algorithmic point of view, the most significant improvement was probably the *candidate narrowing* approach with *bit pattern trees* treated in Section 4.1.4. The performance improvement was tremendous, but using binary trees is also nice since memory demand is bounded linearly in the number of intermediary rays. Some previous implementations needed quadratic space to eliminate duplicate candidates. We also use bit pattern trees for the adjacency tests, in its combinatorial form with subset searching as discussed in Section 4.1.2. The described procedure is readily applicable to any other problem where subset or superset searching is an issue. For rank based adjacency tests, we introduce a rank updating procedure (Section 4.1.5), and we perform the computation with integers and residue arithmetic (Section 4.2). Using residue arithmetic for rank computations is of course not completely new, but we haven't seen a concrete report on implementation details either. Since it is efficient and circumvents numerical issues, it might as well be a candidate for other problems utilizing rank computations.

In Chapter 5, we deal with large-scale computations. We haven't seen any other double description implementation that uses multi-threading techniques, an astonishing fact, since multi core CPUs are nowadays present in every desktop machine. We introduce a very simple multi-core approach in Section 5.2. A more complex parallelization technique is presented in Section 5.2.2, which we think is in particular interesting because it breaks the inherent sequentiality of the double description algorithm at least partially.

On the application side, we think that the 2% suboptimality threshold discussed in the context of an optimal metabolic operation point (Section 6.1) is our most interesting discovery. It might be useful to predict reaction fluxes more accurately, or to better understand the "design goals" that evolutionary pressure imposed to bacterial cells and other organisms. Our proof that *feasibility classifiers* can be applied to elementary modes in Section 6.2 might be a useful tool in the future, even if we couldn't derive new biological insights yet. Finally, we are back at efficiency of the implementation, when we apply our algorithm to genome scale networks for the first time (Section 6.3). The largest generating set contains over 390 million vectors—definitely a proof of concept for the collection of algorithmic improvements in its entirety.

Structure of the Thesis

In the next chapter, we first review and summarize methods used to construct and formalize metabolic networks. Common tools are presented, for instance suitable to check the consis-

tency of a network model, or to identify particular solutions in the flux space. We also give an overview of pathway methods and show how dynamics and regulation can be integrated into such static modeling approaches. The section concludes with an overview about current challenges in the field, like automated network construction and large-scale approaches to integrate different modeling techniques into a single model.

In the second part of the review section, we introduce mathematical tools and definitions used throughout the remainder of the thesis. The standard *Double Description Method* is described, and we show how to transform between different formats for polyhedral cone definitions. We review common definitions for cones in the flux space related to metabolic networks, and we derive upper limits for different generator types using the upper bound theorem of McMullen (1970). We conclude the chapter with notes about complexity and alternative algorithms.

In the next chapter, we review and introduce data structures needed for the algorithm, as well as pre– and postprocessing steps such as compression strategies for the input data. We also show how to implement standard linear algebra algorithms in a generic way for different number types (e.g. addressing floating point and exact arithmetic).

The next chapter deals with elementarity, one of the core parts of the algorithm. Elementarity testing is essentially checking whether a ray is an extreme ray, and we introduce bit pattern trees, a new concept to improve the elementarity testing process. We show how matrix ranks can be computed with residue arithmetic, in many cases a suitable and efficient alternative to standard floating point methods. The presented rank algorithm is virtually a byproduct of our optimization efforts to improve elementarity testing.

The *scale up* chapter approaches the vast quantity of extreme rays possible for large problems—we computed over 390 million rays for large metabolic networks. Matrices of that size can no longer be kept in memory, even if the machine is equiped with over 100G RAM. Hence, out-of-core strategies are discussed. Furthermore, modern processors usually have a multi core architecture, allowing the simultaneous execution of several tasks. We show how this can be exploited for our problem and introduce two multi-core parallelization approaches. In the last section of the chapter, we finally discuss strategies to distribute the computation to multiple host machines, for instance to run the program on a computer cluster.

In the last chapter, we close the circle and come back to biology. We show how to employ the enhanced computational potential of our implementation in concrete applications. In a first application, we hypothesize that a cell probably operates on a suboptimal operation mode to keep a certain variability and flexibility, and we strengthen our hypothesis with predicted flux values derived from our assumption. In another application, we compute elementary modes for a yeast network and show the effects on the feasibility of the modes if additional constraints are added. The third application focuses on large networks. We first show that approaches to simplify the computation usually lead to incomplete and misleading results. Then, we apply our algorithm to three genome scale networks resulting in over 390 million generators for the largest case.

2 Background

2.1 Metabolic Networks

The following sections are joint work with N. Maynard, M. Covert and J. Stelling. It was published in Terzer et al. (2009).

2.1.1 Model Development

2.1.1.1 The Stoichiometric Matrix

Although there are a variety of metabolic modeling approaches, all of them share a fundamental requirement: a stoichiometric matrix based on a reconstructed metabolic network. Each column of the stoichiometric matrix corresponds to a chemical or transport reaction, with non-zero values that identify the metabolites which participate in the reaction, as well as the stoichiometric coefficients that correspond to each metabolite (Fig. 2.1). The matrix also contains directionality: substrate and product metabolites in the matrix have negative and positive coefficients, respectively. By considering the matrix rows instead of the columns, the stoichiometric matrix can also be thought of as the list of reactions in which a given metabolite participates. This interpretation is useful when defining mass balances for each metabolite in the network.

These mass balances are expressed by a system of differential equations written for all the metabolite concentrations $\boldsymbol{c}$ as follows:

$$\frac{d\boldsymbol{c}}{dt} = \boldsymbol{S} \cdot \boldsymbol{v}(t) \tag{2.1}$$

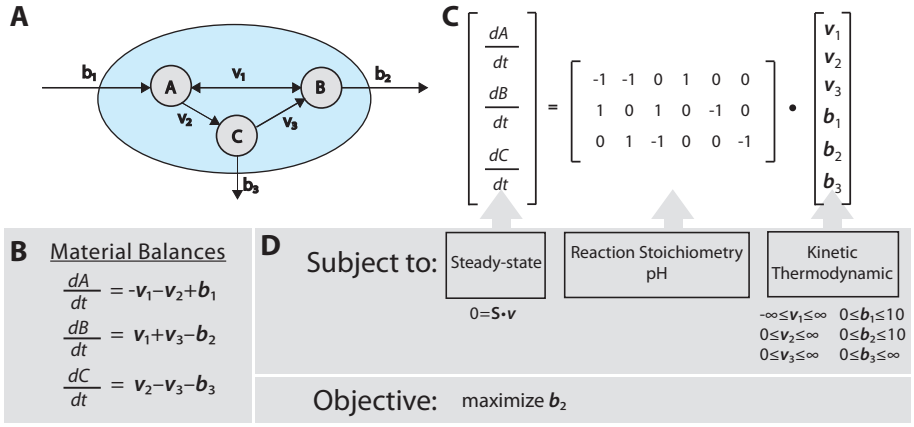


Figure 2.1: **A.** A small reaction network consisting of three metabolites (A,B,C), three transport reactions, and three enzymatic reactions is constructed. v_i indicates the flux through reaction i, and b_j represents the flux through transport protein j. **B.** Material balance equations are shown for each metabolite. **C.** A stoichiometric matrix is populated according to eq. (2.1). **D.** Assumptions, constraints, and an objective are listed for the system.

where $\boldsymbol{S}$ is the stoichiometric matrix and $\boldsymbol{v}(t)$ is the vector of reaction rates. Note that metabolism operates on a much faster time scale than regulatory or cell division events. It is thus often reasonable to assume that metabolic dynamics have reached a *quasi* or *pseudo steady state*, where metabolite concentrations do not change. This leads to the *metabolite balancing equation*

$$\boldsymbol{S} \cdot \boldsymbol{v}(t) = \boldsymbol{0} \tag{2.2}$$

Equation (2.2) is a homogeneous system of linear equations. It requires that each metabolite is consumed in the same quantity as it is produced, and is the basis for further analysis of metabolic fluxes based on the stoichiometric matrix.

The process of building stoichiometric matrices has been amply described and reviewed elsewhere (Covert et al., 2001a). Briefly, this process involves gathering a variety of genomic, biochemical and physiological data from the primary literature as well as databases, such as UniProt (Apweiler et al., 2004), Brenda (Schomburg et al., 2002), BioCyc (Karp et al., 2005), Kegg (Ogata et al., 1999), and the Enzyme Commission database (Bairoch, 2000). This information is used to synthesize a list of chemical and transport reactions together with their metabolite participants for a given cell. Additionally, the chemical formula and charge of each metabolite should be inspected to verify that the chemical reaction is balanced. A reconstructed network model is only as good as its corresponding stoichiometric matrix, and the amount and quality of experimental evidence supporting inclusion of a reaction in the

matrix can vary significantly. Therefore, careful curation and continual updates to the matrix are critical.

2.1.1.2 Constraints on Reaction Rates

The stoichiometric matrix can be annotated by including further important information about either the reactions or the metabolites. The most common matrix annotations include the reversibility of each reaction, and the cellular compartment in which each reaction occurs. More generally, reaction rates are bounded as a consequence of kinetic constants, measured or estimated concentration ratios or to reflect the experimental setup. Upper and lower limits can apply to fluxes of individual reactions ($v_{\min} \leq v \leq v_{\max}$) and reaction directions can be defined by simply setting $v_{\min} = 0$ or $v_{\max} = 0$ for forward or backward irreversible reactions. Additional matrix annotation might include more detailed information about reaction kinetics. For example, it is possible to calculate metabolite uptake or secretion rates without difficulty in several cases; these can be used as part of the metabolic model. Additionally, ^{13}C chase experiments measure concentrations of internal metabolites, from which enzymatic fluxes can often be inferred (Wiechert, 2001; Wittmann, 2002). At the present such "fluxomic" approaches can only be applied to highly reduced metabolic networks, although substantial progress has been made in recent years.

2.1.1.3 Links to Genomic Information

The stoichiometric matrix can also be linked explicitly to the genome and gene expression data for use in certain applications. First, one can connect each chemical or transport reaction to the proteins and genes which enable it to occur. These relationships are often complicated, because proteins are frequently made up of multiple subunits, multiple enzymes sometimes catalyze the same chemical reaction, and certain enzymes catalyze more than one reaction. Delineating these gene-protein-reaction relationships enables the comparison of computational network analyses to experimentally determined gene knockout phenotypes. Gene expression data and transcription factor-gene relationships can also be critical to understanding more complex metabolic behaviors, as described below.

2.1.1.4 Existing Genome-scale Models

To date, dozens of large-scale metabolic reconstructions have been published (Reed et al., 2006a). Most reconstructions are of human pathogens, model organisms, or organisms used in the biotechnology industry to produce valuable chemical products. For the purposes of this review we will briefly highlight three of the most extensively-tested reconstructions: *Escherichia coli* (Feist et al., 2007), *Geobacter sulferreducens* (Mahadevan et al., 2006) and *Saccharomyces cerevisiae* (Herrgård et al., 2008).

In terms of gene coverage, the most complete of these reconstructions is that of *E.coli* (Feist et al., 2007), which has been the product of over a decade of sustained effort (Reed and Palsson, 2003). The most recent reconstruction includes 1260 of *E. coli's* 4405 genes, together with regulatory events, compartmentalization, P/O ratio, reaction thermodynamics, energetic requirements for maintenance, and known kinetic effects. The model has been used for a variety of applications, including the prediction of growth rates, substrate uptake and

by-product secretion for thousands of combinations of knockout strains and culture conditions (Feist and Palsson, 2008).

Geobacter species network reconstructions have also been developed due to their remarkable bioremediative potential (Lovley, 2003). These organisms have somewhat unique metabolic properties enabling them to oxidize organic compounds using a variety of toxic or radioactive metals as electron acceptors. The *Geobacter sulferreducens* metabolic model of metabolism (Mahadevan et al., 2006) has been used to engineer a potentially useful strain with high respiration capacity and low growth rate (Izallalen et al., 2008).

The *Saccharomyces cerevisiae* model is notable in particular for the recent efforts by the yeast community to develop a consensus network. Several reconstructions had previously been made, and varied significantly in their content (Herrgård et al., 2006; Kuepfer et al., 2005; Caspi et al., 2006). As a result, leading researchers in the yeast field along with metabolic modeling experts were brought together for a weekend "jamboree" to agree on the specifics of yeast metabolism. The resulting consensus metabolic network reconstruction was represented in a standard format, Systems Biology Markup Language (SBML: http://www.sbml.org/) (Hucka et al., 2003) and made publicly available. This consensus network is distinct from an actual *in silico* model, because it must be associated with analytical approaches to produce metabolic simulations. Many of the significant modeling approaches are detailed below.

2.1.2 Stoichiometric Network Analysis

The analysis of genome-scale metabolic models relies on three basic approaches: (1) characterizing the general network structure, (2) identifying particular flux distributions, or (3) analyzing all possible flux distributions in a network. The following sections are organized according to the corresponding analysis methods.

2.1.2.1 Characterizing the Nullspace

Nullspace analysis is the first simple tool to perform consistency validations with a metabolic network. Two or more metabolites are called *conserved moieties* if the overall concentration of all remains constant. In such cases, consumption of either metabolite involves production of the other. Some examples of conserved moieties include NAD^+ and NADH, or ATP, ADP and AMP. Detecting conserved moieties requires only the stoichiometric matrix $\boldsymbol{S}$. Several metabolites are part of a *conserved group* if the corresponding rows in the stoichiometric matrix are linearly dependent. Different methods exist to evaluate dependent rows, for instance *Gaussian elimination* or *Singular Value Decomposition* (SVD) (Sauro and Ingalls, 2004). All of these methods compute a basis for the *left nullspace* of $\boldsymbol{S}$, or equivalently, a basis for the nullspace of the transpose of $\boldsymbol{S}$. The set of basis vectors calculated by these methods—and the conserved moieties—are unfortunately non-unique. However, a *convex basis* can be constructed to derive a unique minimal definition for conserved groups (for an application example, see Famili and Palsson (2003)).

Analysis of balanced metabolites also uses nullspace basis vectors. The metabolite balancing equation (2.2) defines a subspace of $\mathbb{R}^n$, where n denotes the number of reactions, and is also the dimensionality of the space. The kernel matrix $\boldsymbol{K}$ is a basis for the nullspace, but it is not unique. However, any valid flux vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)^T$ that defines a flux value v_i for

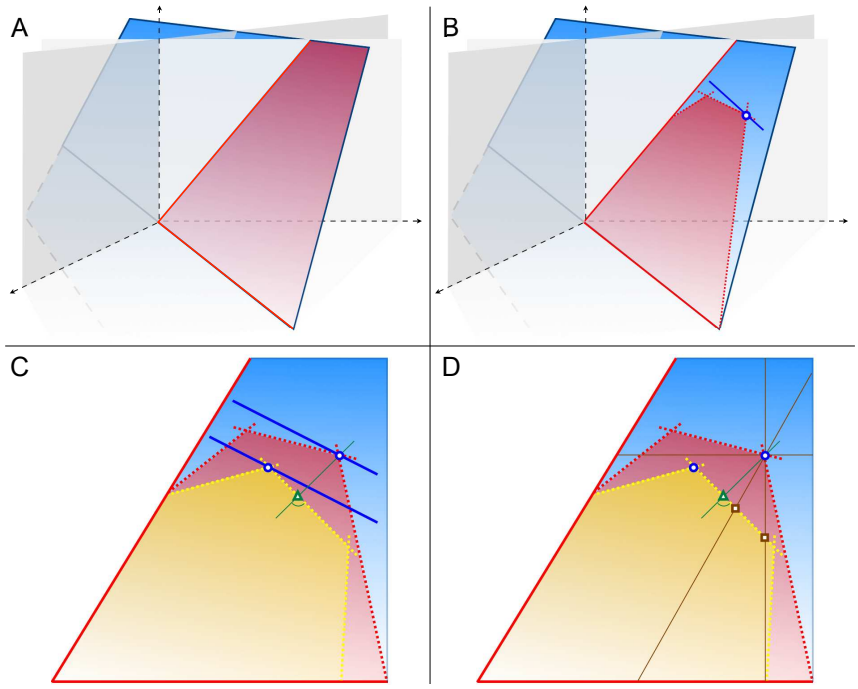


Figure 2.2: **A.** Nullspace (blue hyperplane) and the two-dimensional cone as intersection of the nullspace with the positive orthant. **B.** Additional boundary constraints (dotted lines) shape a bounded convex region. The FBA objective function (blue solid line) touches the region in the optimal point (blue circle). **C.** The same cone, now in a 2D view, with feasible regions for wild type (red area) and mutant (yellow). The FBA objective function touches the regions at the optimal points (blue circles). If MoMA is used instead, the distance to the best wild type value is minimized, resulting in a different optimal value for the mutant (green triangle). **D.** As a third alternative, ROOM minimizes the number of necessary changes. The brown lines indicate that one variable is kept constant, implying a minimal number of changes for this example. Here, two alternative optimal values are possible (brown squares).

each reaction i is a linear combination of column vectors of $\boldsymbol{K}$, that is,

$$\boldsymbol{v} = a_1 \cdot \boldsymbol{K}_{.1} + a_2 \cdot \boldsymbol{K}_{.2} + \ldots + a_k \cdot \boldsymbol{K}_{.k} = \boldsymbol{K} \cdot \boldsymbol{a} \tag{2.3}$$

Note that k is the dimension of the nullspace, the nullity of $\boldsymbol{S}$, namely $k = n - \mathbf{rank}\ \boldsymbol{S}$ and hence usually $k < n$. Equation (2.3) enables the identification of key reaction properties, such as *zero flux reactions* that cannot operate under steady state conditions, or *coupled reactions* whose fluxes have a direct linear dependency. We are using properties of the kernel matrix in Section 3.1, where we discuss *model consistency* and *model compression* in a broader context.

2.1.2.2 Identifying Particular Solutions: Flux Balance Analysis (FBA)

If reaction reversibilities and maximum throughput rates are incorporated as constraints on the steady state model, the solution space reduces from the nullspace to a *convex polyhedral cone* called the *flux cone*. Because these constraints can be represented as linear equations, *linear programming (LP)* methods can be used to identify points with optimal values of a given objective function (for an introduction to LP, see for instance (Cormen et al., 2001)). In the simplest case, a single reaction flux is optimized. Many different objective functions have been tested for their utility in predicting phenotypic behavior, relying on smaller-scale network models (Schuetz et al., 2007; Savinell and Palsson, 1992). One common practice is to define an artificial *growth reaction* that takes the chemical dry-weight components of the cell, in their proper ratios, and "produces" cellular biomass . This reaction can be used as the objective function under certain conditions, depending on the cell type and environmental conditions (Fischer and Sauer, 2005). The term *flux balance analysis* stands for the application of linear programming (LP) methods to analyze fluxes under balanced metabolite conditions (Fig. 2.2B).

As FBA was introduced, the objective function was probably the most discomforting component of the method. Objections to this approach ranged from the biological to the mathematical (Schuetz et al., 2007; Pramanik and Keasling, 1998). One major biological concern was whether the objective function made biological sense—do cells actually have objectives? Use of an objective function was also unsatisfying to those who doubted that FBA could predict cellular phenotypes in the absence of kinetic parameters. The FBA approach seemed to undermine traditional metabolic models which were composed of equations describing the properties of, and interactions between, small molecules and proteins.

To address these concerns, some of the earliest work in experimentally testing FBA focused on whether or not optimization of growth was a reasonable objective that could be observed in culture. Initial results were promising: model predictions of *E.coli* optimal growth and by-product secretion rates matched experimental measurements (Edwards et al., 2001). However, this did not hold true for certain substrates, like glycerol, where *E.coli* growth was suboptimal. This discrepancy between model and experiment was eventually reconciled when *E.coli* was grown on glycerol for several hundred generations and it was shown to evolve towards the computationally predicted optimal growth rate (Ibarra et al., 2002). Interestingly, the phenotypic evolutionary endpoint was shown to be reproducible between cultures, while the underlying gene expression states varied (Fong et al., 2005). The link between optimal growth and evolution was further strengthened when FBA was able to predict *a priori* the endpoint growth rates for several *E.coli* deletion strains (Fong and Palsson, 2004).

Note that often the optimal flux distributions are not unique. Rather, in general a set of *alternate optima* satisfy the linear programming problem, and the available solvers may therefore return different results. Sometimes, it is desirable to enumerate all alternate optima for a given objective, but this is computationally challenging (Lee et al., 2000; Phalakornkule et al., 2001). Another way to deal with this ambiguity is *flux variability analysis*, which examines how individual fluxes can be changed without affecting optimality (Mahadevan and Schilling, 2003; Bilu et al., 2006). Some linear programming packages directly report variability by *sensitivity ranges*. The sensitivity range can also be computed by determining the optimal value for a given objective, fixing the optimal objective value (or a desired optimality range),

and minimizing and maximizing fluxes for reactions of interest.

Finally, multiple iterations of FBA can be used to generate dynamic simulations. An initial optimization can be run for any given starting conditions, and using the resulting flux distribution, external and internal initial concentrations can be updated over a given time step. Critically, this time step must be large enough that the quasi-steady state assumption in Equation (2.1) still holds. However, in practice a time step of 1 second should be large enough. The new conditions define the environment for the next iteration step, leading to a time-course for the environmental conditions as well as for optimal flux patterns. With this type of modeling, glucose uptake was predicted on minimal media under aerobic and anaerobic conditions (Varma and Palsson, 1994). Some groups have also integrated kinetic information with flux-balance models (Covert et al., 2008; Mahadevan et al., 2002; Smallbone et al., 2007; Yugi et al., 2005).

Deletion Strain Phenotypes

A common use of FBA is to compute the essentiality of all the genes in the network, by constraining the corresponding reactions fluxes to zero, and comparing to observed deletion strain phenotypes (Edwards and Palsson, 2000). However, some alternate methods have since been proposed. Using a reference flux vector of the wild type (determined experimentally or estimated by an FBA simulation of the wild-type strain), these optimization strategies minimize the adjustment compared to wild type fluxes. *Minimization of metabolic adjustment (MoMA)* (Segré et al., 2002) uses *quadratic programming (QP)* to minimize the sum-of-squares difference between mutant and wild-type reference flux distribution. Note that the MoMA solution is not optimal in terms of the wild type objective. For instance, if the wild type maximizes for biomass production, the mutant type might not exploit its full growth potential (see Fig. 2.2C). Furthermore, because it depends on an initial non-unique FBA solution, the MoMA solution is also non-unique. *Regulatory on/off minimization (ROOM)* (Shlomi et al., 2005) minimizes the number of (significant) flux changes associated with regulation effort. ROOM defines a set of Boolean *on/off* variables for all reactions. An *on* state means that the corresponding reaction is up- or down-regulated. For *off* variables, an additional constraint ensures that the mutant flux lies within a predefined interval around the wild type flux; deviations inside the interval are regarded *insignificant*. Due to the binary variables, *mixed integer linear programming (MILP)* is used to minimize the number of significant regulation changes (see Fig. 2.2D). Overall, the main difference between MoMA and ROOM is in the motivation behind the approaches. MoMA has a mathematical origin in the formulation of a minimal response to perturbations. ROOM uses a more qualitative, biological approach to control of gene expression, assuming that a cell, in the long run, tries to minimize the number of significant flux changes.

2.1.2.3 Comprehensive Approaches

It is also possible to analyze flux cones without the need to define an objective function. Next, we introduce two approaches that treat the flux cone as a whole through *minimal sets of elements*: *pathway analysis* and the determination of *minimal cut sets*.

Pathway Analysis

As described above, all steady-state flux distributions can be constructed from nullspace basis vectors. *Elementary (flux) modes* have similar properties, but also some important differences: Elementary modes are feasible vectors, whereas nullspace basis vectors might violate reaction reversibility constraints. In addition, all feasible flux vectors—and only feasible ones—can be constructed from *nonnegative* combinations of elementary modes. From nullspace vectors, feasible and infeasible vectors can be constructed, using any *linear* combination.

To calculate a unique, smallest possible set of elementary modes, an additional constraint is introduced: *minimality*. A feasible flux vector is minimal (or *elementary*) if no other flux vector has the same reactions with zero flux plus additional ones. As a consequence, elementary modes (EMs) describe basic non-decomposable operation modes of the network (Gagneur and Klamt, 2004; Terzer and Stelling, 2008). Extreme pathways (EPs) constitute a very similar concept, associated with the "shortest" possible paths in the network (note that "shortest" is misleading). Elementary modes are actually a superset of extreme pathways, arising from a slightly different treatment of reversible transport reactions (see (Klamt and Stelling, 2003; Wagner and Urbanczik, 2005) and Section 2.4 for exact definitions). Moreover, in a strong mathematical sense, neither EMs nor EPs are minimal, and hence also not a minimal set of generators. EMs and EPs operate in a reconfigured network with enhanced reaction dimensionality, and they are only minimal in the augmented flux space. We discuss the differences in Section 2.4 and focus here on elementary modes. The concepts also apply to extreme pathways unless stated otherwise.

Unfortunately, these *pathway sets* grow exponentially with increasing network size. It is not yet possible to compute elementary modes for genome-scale networks under general conditions (Terzer and Stelling, 2008; Klamt and Stelling, 2002). Nevertheless, elementary modes have interesting properties because they are minimal, and hence contain information about the smallest functional units of complex networks. Pathway analysis uncovers *all* alternative pathways in contrast to FBA, where only a single (optimal) pathway is found. All modes of the network are superpositions of EMs, and an *alpha-spectrum*—the contribution of individual pathways to an observed (*in vivo*) flux pattern—has been analyzed for a simplified core metabolic network and for human red blood cell metabolism (Wiback et al., 2003). Reaction participation in EMs also indicates a reaction's importance for different substrate/product conversions.

Furthermore, the number of active reactions in a single EM is related to the necessary enzymes, and a cost function can be derived. In combination with benefit functions known from FBA, we get a *multiple objective*, which might indeed be better suited to represent the complex optimization strategy of the cell (Terzer and Stelling, 2008; Stelling et al., 2002). To analyze *flux variability*, only EMs above a certain optimality criterion can be considered, and the *coefficient of variation*—the relative standard deviation of reaction fluxes—indicates variability or flexibility of the fluxes (Terzer and Stelling, 2008) (see Fig. 2.3 for an example). In addition, EMs can be used to reliably predict viability of gene deletion mutants. There, only EMs are kept where the reaction of interest does not participate. An empty set for the perturbed network indicates that the network is structurally unable to operate under steady state conditions (Stelling et al., 2002).

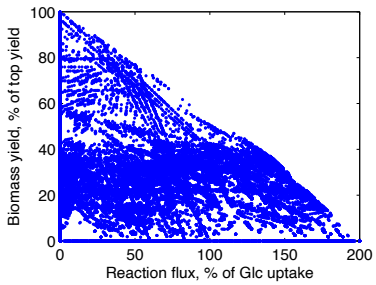


Figure 2.3: Ethanol excretion rate related to biomass yield for 178,575 elementary modes of an E. coli central metabolism network with 97 metabolites and 120 reactions; only growth on glucose (Glc) was considered. Ethanol production is possible only for suboptimal growth.

Minimal Cut Sets

Minimal cut sets give an opposite view of a metabolic network, as compared to *elementary modes*. EMs describe minimal requirements to operate the network; minimal cut sets define smallest possible reaction sets that cause network failure with respect to a specific function such as biomass production (Klamt and Gilles, 2004). In fact, both concepts completely describe the metabolic network (of course without involved regulation); formally, they are dual (Klamt, 2006). The simplest minimal cut sets are essential reactions, since they have minimal size, and their removal disables any target reaction. *Synthetic lethals* are minimal cut sets of size two: lethality is caused by the simultaneous knockout of two non-essential reactions. We can proceed with this concept for larger sets, requiring minimality or *irreducibility*, similar to the non-decomposability of elementary modes. Not surprisingly, it is computationally expensive to enumerate all minimal cut sets for large networks.

Network dysfunction is associated with minimal cut sets. Such dysfunction may be caused internally through spontaneous mutations, or have external reasons and be intentional in the case of gene deletions or RNA interference. Using minimal cut sets, internal structural fragility and robustness can be analyzed. In metabolic engineering, minimal cut sets identify potential drug targets, driving new hypotheses or narrowing down test candidates for expensive experiments. Finally, minimal cut sets have practical applications in the design and optimization of biotechnological processes. Methods such as OptKnock (Burgard et al., 2003) can be used to study minimal intervention strategies for overproduction of target biochemicals in microbial strains.

2.1.3 Incorporating Regulation

Constraint-based analysis has for the most part only focused on metabolism, with a notable exception in signal transduction (Papin and Palsson, 2004). For instance, it is unclear whether the assumptions required for FBA can be justifiably applied to other biological processes. Instead, some studies have integrated FBA-based metabolic models with possibly more appropriate models for transcriptional regulatory and signaling networks. Also, these approaches start to address the dynamic behavior of metabolic networks.

However, gene expression has a dramatic additional effect on metabolic networks. For example, only about 50% of the *E.coli* genome is expressed under typical culture conditions, and therefore half of the stoichiometric matrix might be significantly constrained at any given time. This has major ramifications for convex-based analyses that ignore gene expression. The calculated pathways will include several false positives which never occur in the living cell. Furthermore, the optimal solutions determined by FBA will most likely be incorrect in all but relatively simple growth conditions.

To explicitly account for the effects of gene expression on metabolic behavior, an integrated procedure was developed whereby transcriptional regulatory events were described using Boolean logic and used to constrain the solution space further (Covert et al., 2001b). The status of regulated transcription is found by evaluating intra- or extra-cellular conditions. Transcription may be switched on or off by presence or absence of particular metabolites, proteins or signaling molecules. Sometimes, concentrations above a certain threshold are required to trigger regulatory events. If an activating or inhibiting expression changes—caused by updated concentration values—the rate constraints for the regulated reaction change, resulting in up/down-regulations. In the simplest case, inhibition sets the constraints to zero, and activation causes a reset back to the original boundary values. This method was found to significantly reduce the number of extreme pathways calculated in a simple system (Covert and Palsson, 2003).

Combination of the method with FBA in an approach called *regulatory flux balance analysis*, or rFBA (Fig. 2.4) produced dramatically more accurate model predictions in organisms such as *E.coli* (Covert et al., 2004) and *yeast* (Herrgård et al., 2006). More specifically, Covert and Palsson (2002) have used rFBA to derive time courses for *E.coli* with a genome-scale model and to correctly predict viability for 106 of 116 mutant strain/growth medium conditions. In (Covert et al., 2004), an extended model with 1010 genes was used in an iterative process to generate and test hypotheses, and missing components and interactions in regulatory or metabolic networks could be identified.

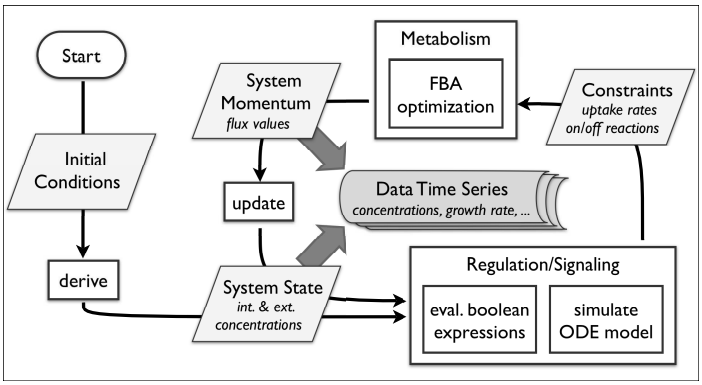


Figure 2.4: Schematic representation of regulated Flux Balance Analysis (rFBA) using Boolean expressions to simulate regulatory elements. The concept is generalized to integrated FBA (iFBA), also incorporating ordinary differential equations (ODEs) to simulate regulation. The algorithm is an iterative procedure, generating time series output at each iteration.

The initial work focused on reconstructing transcriptional regulatory networks based directly on findings from the primary literature. Boolean logic best represented the typically qualitative conclusions of experimental studies (e.g., "transcription factor X was observed to repress expression of target gene Y"). However, subsequent studies have used microarray data to constrain FBA (Akesson et al., 2004), as well as machine learning techniques to generate more sophisticated regulatory network models (Bonneau et al., 2007). rFBA models stand to gain significantly from these developments.

2.1.4 Current Challenges

2.1.4.1 Automated Network Reconstruction

The earliest metabolic network reconstructions were manually generated. Since that time, several tools that support network reconstruction have been developed, such as Pathway Tools (Karp et al., 2002), Kegg Pathways (Kanehisa et al., 2002), Puma2 (Maltsev et al., 2006), and SimPheny (Price et al., 2003). However, unknown reactions and the necessary validation of database entries still result in time-intensive manual network curation (Ott and Vriend, 2006). Therefore, although significant effort is dedicated to developing computational approaches for fully automatic network reconstruction (Nikoloski et al., 2008) and reconciliation (Gevorgyan et al., 2008), curation-based efforts such as the jamboree currently produce the "gold standard" reconstructions.

One automated reconstruction method aims to identify necessary reactions from an organismwide database such that these reactions could allow growth of mutants that are experimentally viable, but predicted to be inviable by an existing stoichiometric model (Reed et al., 2006b). This approach considers only one experimental condition at a time, it yields (potentially large) sets of candidate reactions, and it is computationally expensive because for each

condition and candidate reaction FBA analysis has to be performed. Other methods rely on information fusion from pathway databases to reconstruct models *de novo* but so far they have not yielded functional models (DeJongh et al., 2007), or only prototypes that lack validation with experimental data (Arakawa et al., 2006). In addition, powerful methods exist to identify metabolic genes for a given enzymatic function (Kharchenko et al., 2006, 2004)—but this function has to be already contained in the model. Optimization-based methods help identifying gaps in metabolic network reconstructions, and they consolidate the models by introducing new reactions, or by modifying existing reactions. Existing algorithms, however, do not consider global changes in network structures, or potential effects on the quality of model predictions (Kumar et al., 2007). Available methods, thus, have limitations in automatically generating predictive network models and novel concepts such as (Clauset et al., 2008) are needed.

2.1.4.2 Cellular Optimality and Design

The choice of a biologically meaningful objective function is critical for FBA. Identifying the objective function—or cellular design principles—can be regarded as the *inverse problem* of FBA. Where FBA finds an optimal flux vector given some objective function, a more challenging problem is to infer the objective for an experimentally determined reference flux vector. Early work on this problem used a bi-level optimization approach to test existing hypothetical objectives (Burgard and Maranas, 2003). The *OptFind* method solves two optimizations in one step, using the duality theorem of linear programming to flatten the two optimality layers. In an application to *E.coli* under aerobic and anaerobic conditions, a high coefficient of importance was found for the biomass function (Burgard and Maranas, 2003). Method refinements allowed to derive objectives de novo (Gianchandani et al., 2008). Similar techniques—in combination with binary variables, requiring the use of *mixed integer linear programming (MILP)* —can find optimal knockout strategies leading to the overproduction of a desired product (Burgard et al., 2003). By automatically querying reaction databases, an optimal strain is composed and the method yields optimal substrates for different microbes (Pharkya et al., 2004). Similar approaches have been established using stochastic optimization (Patil et al., 2005) and sensitivity analysis of metabolic flux distributions (Park et al., 2007).

Another way to analyze biological networks begins with the hypothesis that cells have optimized their operation over evolutionary time-scales, as assumed in FBA. However, FBA does not provide insight into specific control mechanisms. Further elaborations of principles of optimal control theory (already present in MoMA) could elucidate large-scale metabolic control circuits. For instance, the cybernetic approach (Kompala et al., 1984) treats metabolic control as a set of dynamic, optimal resource allocation problems that are solved in parallel with the mass balances. Predictions on gene expression and enzyme activity result from choices between competing alternatives, each with a relative cost and benefit for the organism. In addition, postulates for specific pathway architectures have resulted from this approach (Young et al., 2008).

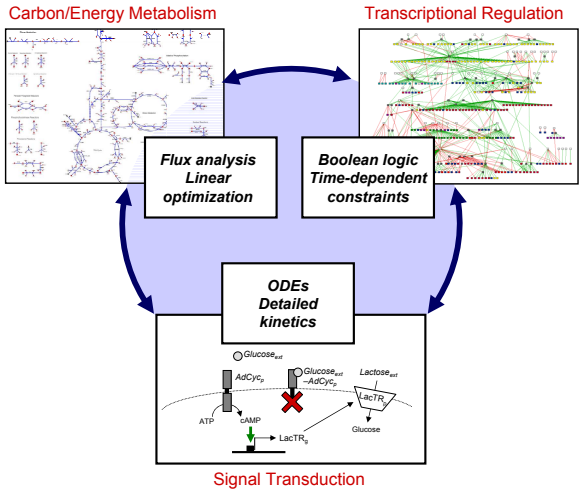


Figure 2.5: Models have been constructed which build on the constraint-based framework to integrate metabolic, transcriptional regulatory and signal transduction networks (each represented using different mathematics) into a single model.

2.1.4.3 Towards Large-scale Network Integration and Dynamics

To build on these successes toward creation of a whole-cell model, approaches to model integration must be developed, in particular with large-scale kinetic models (Jamshidi and Palsson, 2008). The integration of metabolism with transcriptional regulation was already described above. More recently, these integration methods were expanded to include ordinary differential equations (Covert et al., 2008). The method (called "integrated FBA", or iFBA; see Fig. 2.5) was used to build an FBA model of *E.coli* central metabolism, together with a Boolean logic-based regulatory network and a set of ODEs that described catabolite repression (Kremling et al., 2007). The resulting model had significant advantages over either the rFBA or ODE-based models alone, particularly in predicting the consequences of gene perturbation. Another approach has been developed which incorporates coarse-grained timescale information about signaling dynamics with metabolic models (Lee et al., 2008). Finally, recent advances towards large-scale dynamic models include those that rely on simplified reaction kinetics (Liebermeister and Klipp, 2006a,b) or on ensembles of models (Tran et al., 2008).

In addition to explicitly modeling cellular regulation, one can exploit that stoichiometric constraints restrict the systems dynamics, for instance, by conserved moieties (Famili and Palsson, 2003). Early work addressed this topic for chemical reaction networks. For instance, in the 1970's Feinberg, Horn and Jackson started deriving theorems to determine the pos-

sible dynamic regimes, such as multistability and oscillations, based on network structure alone. The specific challenges posed by biological systems lead to application studies as well as further theory development (Craciun et al., 2006). For instance, the theory can be used in stability analysis and for model discrimination by safely rejecting hypotheses on reaction mechanisms; this analysis relies on a modular approach where subnetworks that correspond to EMs are investigated individually (Conradi et al., 2007). Other algorithms for the identification of dependent species in large biochemical systems—to be employed, for instance, in model reduction-have recently become available (Vallabhajosyula et al., 2006).

2.1.5 Conclusion

In sum, large-scale metabolic network modeling has matured as a field, with a library of computational techniques, published networks for a large and increasing number of organisms, and an extensive body of supporting experimental evidence. It is clear that such modeling can be extremely useful, notwithstanding its limitations. These studies have also established the essentiality of other biological models—metabolism alone cannot explain most observed phenotypic behaviors. It seems unrealistic to expect that data which would allow detailed genome-scale modeling of other biological networks will arrive in the next few years. However, we re-emphasize that scientists felt the same way about metabolism just over a decade ago.

2.2 Polyhedral Sets & Cones

We first review the relation between subspaces, affine sets, convex cones and convex sets. Then, some fundamental theorems about polyhedral cones are given, leading to the statement that a finitely constrained polyhedral cone is finitely generated by its extreme rays. This statement is central for this thesis as we dedicate our full attention to the problem of converting the constraint repesentation of a cone to the generating representation. We conclude with a section about polyhedra, the non-homogeneous counterpart to polyhedral cones.

2.2.1 Affine & Convex Sets

Subspaces and affine sets are closed under linear and affine combinations, respectively. If we restrict the scalars in the combinations to nonnegative values, we get more general sets with operations that are more strictly constrained. In this sense, convex cones and convex sets are a natural generalization of subspaces and affine sets.

Subspaces

Definition 2.1. *Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r$ be r points in $\mathbb{R}^d$, and $\lambda_1, \lambda_2, \ldots, \lambda_r \in \mathbb{R}$ be r scalars. Then, any point*

$$\boldsymbol{y} = \lambda_1 \boldsymbol{x}_1 + \lambda_2 \boldsymbol{x}_2 + \ldots + \lambda_r \boldsymbol{x}_r = \sum_{i=1}^{r} \lambda_i \boldsymbol{x}_i \tag{2.4}$$

is called linear combination *of the points $\boldsymbol{x}_i$.*

Definition 2.2. *A subset $\mathcal{S} \subseteq \mathbb{R}^d$ is called* subspace *iff it is closed under* linear combinations*, that is,*

$$\boldsymbol{x}_i \in \mathcal{S}, \quad i \in \{1, \ldots, r\} \quad \Longrightarrow \quad \sum_{i=1}^{r} \lambda_i \boldsymbol{x}_i \in \mathcal{S} \quad \forall \lambda_i \in \mathbb{R} \tag{2.5}$$

Definition 2.3. *The set of all linear combinations of points $\boldsymbol{x}_i \in \mathcal{S} \subseteq \mathbb{R}^d$ is called* span *or* linear hull *of $\mathcal{S}$:*

$$\textbf{span}\ \mathcal{S} = \left\{ \textstyle\sum \lambda_i \boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{S}, \lambda_i \in R \right\} \tag{2.6}$$

We may also use $\textbf{span}\ \boldsymbol{M}$ for a matrix $\boldsymbol{M}$, in which case we mean the linear combinations of the column vectors of $\boldsymbol{M}$. Usually, however, we use $\textbf{im}\ \boldsymbol{M}$ in this case, denoting the *image* of the matrix $\boldsymbol{M}$ (see also Appendix A1.1).

Definition 2.4. *A subset $\mathcal{S} \subseteq \mathbb{R}^d$ is called* linearly independent *iff no point is a linear combination of the other points, and* linearly dependent *otherwise. Equivalently, $\mathcal{S}$ is* linearly independent *iff there exists no $\boldsymbol{x} \in \mathcal{S}$ such that $\boldsymbol{x} \in \textbf{span}\,(\mathcal{S} \setminus \boldsymbol{x})$.*

Remarks

i) Any subspace $\mathcal{S}$ is closed under addition and scalar multiplication:

$$\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{S} \Longrightarrow \boldsymbol{x}_1 + \boldsymbol{x}_2 \in \mathcal{S} \tag{2.7}$$
$$\boldsymbol{x} \in \mathcal{S}, \lambda \in \mathbb{R} \Longrightarrow \lambda \boldsymbol{x} \in \mathcal{S} \tag{2.8}$$

ii) Any nonempty subspace contains the origin.

iii) The intersection of subspaces is again a subspace.

iv) **span** $\mathcal{S}$ is the smallest subspace containing $\mathcal{S}$, or equivalently, the intersection of all subspaces containing $\mathcal{S}$.

v) A linearly independent set $\mathcal{S} \subseteq \mathbb{R}^d$ contains at most d points.

vi) *Row space*, *range*, and *nullspace* of a matrix are subspaces (see Appendix A1.1).

Affine Sets

Definition 2.5. *Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r$ be r points in $\mathbb{R}^d$, and $\lambda_1, \lambda_2, \ldots, \lambda_r \in \mathbb{R}$ be r scalars satisfying $\sum \lambda_i = 1$. Then, any point*

$$\boldsymbol{y} = \lambda_1 \boldsymbol{x}_1 + \lambda_2 \boldsymbol{x}_2 + \ldots + \lambda_r \boldsymbol{x}_r = \sum_{i=1}^{r} \lambda_i \boldsymbol{x}_i \tag{2.9}$$

is called affine combination *of the points $\boldsymbol{x}_i$.*

Definition 2.6. *A subset $\mathcal{S} \subseteq \mathbb{R}^d$ is called* affine set[1] *iff it is closed under* affine combinations*, that is,*

$$\boldsymbol{x}_i \in \mathcal{S}, \quad i \in \{1, \ldots, r\} \quad \Longrightarrow \quad \sum_{i=1}^{r} \lambda_i \boldsymbol{x}_i \in \mathcal{S} \quad \forall \lambda_i \in \mathbb{R} \quad \text{with} \quad \sum \lambda_i = 1 \tag{2.10}$$

Definition 2.7. *The set of all affine combinations of points $\boldsymbol{x}_i \in \mathcal{S} \subseteq \mathbb{R}^d$ is called* affine hull *of $\mathcal{S}$:*

$$\mathbf{aff}\, \mathcal{S} = \left\{ \sum \lambda_i \boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{S}, \lambda_i \in R \quad \text{with} \quad \sum \lambda_i = 1 \right\} \tag{2.11}$$

We may also use $\mathbf{aff}\, \boldsymbol{M}$ for a matrix $\boldsymbol{M}$, in which case we mean the affine combinations of the column vectors of $\boldsymbol{M}$.

Definition 2.8. *A subset $\mathcal{S} \subseteq \mathbb{R}^d$ is called* affinely independent *iff no point is an affine combination of the other points, and* affinely dependent *otherwise. Equivalently, $\mathcal{S}$ is* affinely independent *iff there exists no $\boldsymbol{x} \in \mathcal{S}$ such that $\boldsymbol{x} \in \mathbf{aff}(\mathcal{S} \setminus \boldsymbol{x})$.*

[1] An affine set is sometimes also called *affine (sub)space*

Remarks

i) Any subspace is also an affine set.

ii) The intersection of affine sets is again an affine set.

iii) **aff** $\mathcal{S}$ is the smallest affine set containing $\mathcal{S}$, or equivalently, the intersection of all affine sets containing $\mathcal{S}$.

iv) An affine set can be seen as a translated subspace.

v) An affine set in $\mathbb{R}^d$ can be seen as the intersection of a subspace and a hyperplane in $\mathbb{R}^{d+1}$.

vi) Linear independence implies affine independence.

vii) An affinely independent set $\mathcal{S} \subseteq \mathbb{R}^d$ contains at most $d+1$ points.

Convex Cones

Definition 2.9. *Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r$ be r points in $\mathbb{R}^d$, and $\lambda_1, \lambda_2, \ldots, \lambda_r \in \mathbb{R}_{\geq 0}$ be r nonnegative scalars. Then, any point*

$$\boldsymbol{y} = \lambda_1 \boldsymbol{x}_1 + \lambda_2 \boldsymbol{x}_2 + \ldots + \lambda_r \boldsymbol{x}_r = \sum_{i=1}^{r} \lambda_i \boldsymbol{x}_i \tag{2.12}$$

is called conic combination[1] *of the points $\boldsymbol{x}_i$.*

Definition 2.10. *A subset $\mathcal{C} \subseteq \mathbb{R}^d$ is called* convex cone *iff it is closed under* conic combinations*, that is,*

$$\boldsymbol{x}_i \in \mathcal{C}, \quad i \in \{1, \ldots, r\} \quad \Longrightarrow \quad \sum_{i=1}^{r} \lambda_i \boldsymbol{x}_i \in \mathcal{C} \quad \forall \lambda_i \in \mathbb{R}_{\geq 0} \tag{2.13}$$

Definition 2.11. *The set of all conic combinations of points $\boldsymbol{x}_i \in \mathcal{S} \subseteq \mathbb{R}^d$ is called* conic hull *of $\mathcal{S}$:*

$$\textbf{cone}\ \mathcal{S} = \left\{ \textstyle\sum \lambda_i \boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{S}, \lambda_i \in \mathbb{R}_{\geq 0} \right\} \tag{2.14}$$

We may also use **cone** $\boldsymbol{M}$ for a matrix $\boldsymbol{M}$, in which case we mean the conic combinations of the column vectors of $\boldsymbol{M}$.

Definition 2.12. *A subset $\mathcal{C} \subseteq \mathbb{R}^d$ is called* conically independent *iff no point is a conic combination of the other points, and* conically dependent *otherwise. Equivalently, $\mathcal{C}$ is* conically independent *iff there exists no $\boldsymbol{x} \in \mathcal{C}$ such that $\boldsymbol{x} \in$ **cone** $(\mathcal{C} \setminus \boldsymbol{x})$.*

[1] Other authors use the term *nonnegative (linear) combination* instead of conic combination

Remarks

i) Any convex cone $\mathcal{C}$ is closed under addition and nonnegative scalar multiplication:

$$\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{C} \Longrightarrow \boldsymbol{x}_1 + \boldsymbol{x}_2 \in \mathcal{C} \tag{2.15}$$

$$\boldsymbol{x} \in \mathcal{C}, \lambda \in \mathbb{R}_{\geq 0} \Longrightarrow \lambda \boldsymbol{x} \in \mathcal{C} \tag{2.16}$$

ii) Any nonempty convex cone contains the origin.

iii) Any subspace is also a convex cone.

iv) The intersection of convex cones is again a convex cone.

v) **cone** $\mathcal{S}$ is the smallest convex cone containing $\mathcal{S}$, or equivalently, the intersection of all convex cones containing $\mathcal{S}$.

vi) Linear independence implies conic independence.

Convex Sets

Definition 2.13. *Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r$ be r points in $\mathbb{R}^d$, and $\lambda_1, \lambda_2, \ldots, \lambda_r \in \mathbb{R}_{\geq 0}$ be r nonnegative scalars satisfying $\sum \lambda_i = 1$. Then, any point*

$$\boldsymbol{y} = \lambda_1 \boldsymbol{x}_1 + \lambda_2 \boldsymbol{x}_2 + \ldots + \lambda_r \boldsymbol{x}_r = \sum_{i=1}^{r} \lambda_i \boldsymbol{x}_i \tag{2.17}$$

is called convex combination *of the points $\boldsymbol{x}_i$.*

Definition 2.14. *A subset $\mathcal{C} \subseteq \mathbb{R}^d$ is called* convex set *iff it is closed under* convex combinations*, that is,*

$$\boldsymbol{x}_i \in \mathcal{C}, \quad i \in \{1, \ldots, r\} \quad \Longrightarrow \quad \sum_{i=1}^{r} \lambda_i \boldsymbol{x}_i \in \mathcal{S} \quad \forall \lambda_i \in \mathbb{R}_{\geq 0} \quad \text{with} \quad \sum \lambda_i = 1 \tag{2.18}$$

Definition 2.15. *The set of all convex combinations of points $\boldsymbol{x}_i \in \mathcal{S} \subseteq \mathbb{R}^d$ is called* convex hull *of $\mathcal{S}$:*

$$\textbf{conv}\ \mathcal{S} = \left\{ \sum \lambda_i \boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{S}, \lambda_i \in \mathbb{R}_{\geq 0} \quad \text{with} \quad \sum \lambda_i = 1 \right\} \tag{2.19}$$

We may also use **conv** $\boldsymbol{M}$ for a matrix $\boldsymbol{M}$, in which case we mean the convex combinations of the column vectors of $\boldsymbol{M}$.

Definition 2.16. *A subset $\mathcal{C} \subseteq \mathbb{R}^d$ is called* convexly independent *iff no point is a convex combination of the other points, and* convexly dependent *otherwise. Equivalently, $\mathcal{C}$ is* convexly independent *iff there exists no $\boldsymbol{x} \in \mathcal{C}$ such that $\boldsymbol{x} \in$ **conv** $(\mathcal{C} \setminus \boldsymbol{x})$.*

Remarks

i) Any subspace is also a convex set.

ii) Any affine set is also a convex set.

iii) Any convex cone is also a convex set.

iv) The intersection of convex sets is again a convex set.

v) **conv** $\mathcal{S}$ is the smallest convex set containing $\mathcal{S}$, or equivalently, the intersection of all convex sets containing $\mathcal{S}$.

vi) A convex set in $\mathbb{R}^d$ can be seen as the intersection of a convex cone and a hyperplane in $\mathbb{R}^{d+1}$.

vii) Linear, affine or conic independence implies convex independence.

Relations Between the Sets

Set properties and relations between the four sets are summarized in Fig. 2.6.

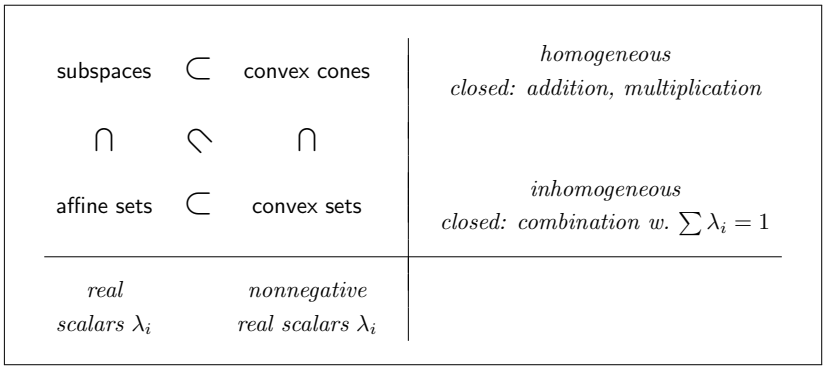


Figure 2.6: Subsets of $\mathbb{R}^d$: properties and relationships. The subset relation $\subset$ relates *sets of sets*, for instance: the set of all subspaces is a subset of the set of convex sets. It can also be read as an *is-a* relation for set instances: every subspace *is a* convex set.

Hyperplanes & Halfspaces

Definition 2.17. *A* hyperplane $\mathcal{H} \subseteq \mathbb{R}^d$ *is the solution set of one nontrivial linear equation:*

$$\mathcal{H} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}^T \boldsymbol{x} = b \quad \text{with} \quad \boldsymbol{a} \in \mathbb{R}^d, \boldsymbol{a} \neq \boldsymbol{0}, b \in \mathbb{R} \right\} \tag{2.20}$$

Remarks

i) Any vector in $\mathcal{H}$ is normal to $\boldsymbol{a}$.
ii) The constant b determines the offset of $\mathcal{H}$ from the origin.
iii) Any hyperplane is an affine set.
iv) Any homogeneous hyperplane ($b = 0$) contains the origin and is thus a subspace.

Definition 2.18. *A* (closed) halfspace *is the solution set of one nontrivial linear inequality:*

$$\mathcal{H}_{\geq} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}^T \boldsymbol{x} \geq b \quad \text{with} \quad \boldsymbol{a} \in \mathbb{R}^d, \boldsymbol{a} \neq \boldsymbol{0}, b \in \mathbb{R} \right\} \tag{2.21}$$

$$\mathcal{H}_{\leq} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}^T \boldsymbol{x} \leq b \quad \text{with} \quad \boldsymbol{a} \in \mathbb{R}^d, \boldsymbol{a} \neq \boldsymbol{0}, b \in \mathbb{R} \right\} \tag{2.22}$$

Definition 2.19. *An* open halfspace *is the solution set of one nontrivial linear strict inequality:*

$$\mathcal{H}_{>} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}^T \boldsymbol{x} > b \quad \text{with} \quad \boldsymbol{a} \in \mathbb{R}^d, \boldsymbol{a} \neq \boldsymbol{0}, b \in \mathbb{R} \right\} \tag{2.23}$$

$$\mathcal{H}_{<} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}^T \boldsymbol{x} < b \quad \text{with} \quad \boldsymbol{a} \in \mathbb{R}^d, \boldsymbol{a} \neq \boldsymbol{0}, b \in \mathbb{R} \right\} \tag{2.24}$$

Remarks

i) A hyperplane divides the space into two halfspaces.
ii) A halfspace is *closed* if it contains its *boundary* (the *separating hyperplane*), and *open* otherwise.
iii) An open halfspace is the *interior* of a corresponding closed halfspace.
iv) Any halfspace is a convex set.
v) Any convex set can be seen as the intersection of halfspaces.
vi) Any closed homogeneous halfspace ($b = 0$) contains the origin and is thus a convex cone.
vii) Any convex cone can be seen as the intersection of closed homogeneous halfspaces.

2.2.2 Polyhedral Cones

Here, we introduce definitions and some important theorems about polyhedral cones. A very fundamental achievement is the theorem of Minkowski and Weyl, stating that there is a one-to-one correspondence between finitely constrained and finitely generated cones. In the sequel, we define extreme rays and show that they are the minimal generators of a pointed polyhedral cone.

Polyhedral Cones

Definition 2.20. *A* finitely constrained cone *is a convex cone defined as the solution set of finitely many homogeneous linear equalities and inequalities:*

$$\mathcal{P} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0} \ , \ \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \right\} \tag{2.25}$$

Definition 2.21. *A* finitely generated cone *is a convex cone defined as the conic hull of finitely many generating points (columns in the generating matrix $\boldsymbol{R}$):*

$$\begin{aligned}\mathcal{P} &= \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x} = \boldsymbol{R}\,\boldsymbol{c} \quad \textit{with} \quad \boldsymbol{c} \geq \boldsymbol{0} \,\} \\ &= \mathbf{cone}\ \boldsymbol{R}\end{aligned} \tag{2.26}$$

Theorem 2.1 (Minkowski-Weyl's Theorem for Polyhedral Cones)**.** *A convex cone is finitely generated iff it is finitely constrained.*

Due to this fundamental theorem, we subsume finite cones (Def. 2.20 and 2.21) as follows:

Definition 2.22. *A finitely constrained/generated convex cone is called* polyhedral cone.

In the remainder of this thesis, we might use the simpler term *cone* meaning *polyhedral cone* unless otherwise indicated.

Lineality & Pointedness

We have already seen that a cone is not necessarily pointed, for instance recalling that subspaces are also cones. For many applications, however, it is advantageous to deal with pointed cones, e.g. because pointed cones have a unique generating representation (up to positive scaling). The following statements establish that we can indeed restrict our consideration to pointed cones.

Definition 2.23. *The largest subspace contained within a cone $\mathcal{P}$ is called* lineality space *of $\mathcal{P}$, written as* $\mathbf{lin}\ \mathcal{P}$.

The lineality space is the intersection of the cone with its negation, that is,

$$\mathbf{lin}\ \mathcal{P} = \mathcal{P} \cap -\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x} \in \mathcal{P}, -\boldsymbol{x} \in \mathcal{P} \,\} \tag{2.27}$$

For a polyhedral cone $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \,\}$, we have

$$\mathbf{lin}\ \mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}\,\boldsymbol{x} = \boldsymbol{0} \,\} \tag{2.28}$$

Definition 2.24. *A cone $\mathcal{P}$ is called* pointed *iff it has a trivial lineality space* $\mathbf{lin}\ \mathcal{P} = \{\boldsymbol{0}\}$.

Lemma 2.1. *A polyhedral cone $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \,\}$ is pointed if and only if* $\mathbf{rank} \left[\begin{smallmatrix} \boldsymbol{A} \\ \boldsymbol{B} \end{smallmatrix}\right] = d$.

Proof. Follows immediately from eq. (2.28). □

Lemma 2.2. *Any cone $\mathcal{P}$ can be seen as the direct sum of its lineality space $\mathcal{L} = \mathbf{lin}\ \mathcal{P}$ and the intersection of $\mathcal{P}$ with the orthogonal complement*[1] *$\mathcal{L}^\perp$ of $\mathcal{L}$, where $\mathcal{P} \cap \mathcal{L}^\perp$ is a pointed cone:*

$$\begin{aligned}\mathcal{P} &= \mathcal{L} \oplus (\mathcal{P} \cap \mathcal{L}^\perp) \\ &= \{\, \boldsymbol{x}_1 + \boldsymbol{x}_2 \mid \boldsymbol{x}_1 \in \mathcal{L},\, \boldsymbol{x}_2 \in (\mathcal{P} \cap \mathcal{L}^\perp) \,\}\end{aligned} \tag{2.29}$$

Note that if $\mathcal{L} = \mathcal{R}(\boldsymbol{L}) = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid x = \boldsymbol{L}\,\boldsymbol{\lambda} \,\}$, then $\mathcal{L}^\perp = \mathcal{N}(\boldsymbol{L}^T) = \{\, \boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{0} \,\}$, that is, we know a constraint representation of the pointed cone $\mathcal{P} \cap \mathcal{L}^\perp$.

[1] see Definition A.1 in Appendix A1.1

Faces & Extreme Rays

Faces are the "boundary" of polyhedral cones, and rays are "half lines" entirely contained within the cone. Below, we define *extreme rays* – rays that represent a one-dimensional face – and we will see that they are minimal generators of the cone. To work towards this result, we need the following formal definitions.

Definition 2.25. *A subset $\mathcal{F} \subseteq \mathcal{P}$ of a cone $\mathcal{P}$ is called* face *of $\mathcal{P}$ if $\mathcal{F} = \mathcal{P}$ or if $\mathcal{F}$ is the intersection of $\mathcal{P}$ with a hyperplane $\mathcal{H}$, such that $\mathcal{P}$ is entirely contained in one of the halfspaces separated by $\mathcal{H}$. More formally, $\mathcal{F}$ is a face of $\mathcal{P}$ iff*

$$\mathcal{F} = \{\, \boldsymbol{x} \in \mathcal{P} \mid \boldsymbol{a}^T \boldsymbol{x} = 0 \,\} \quad \text{for some } \boldsymbol{a} \text{ with} \quad \boldsymbol{a}^T \boldsymbol{x} \leq 0 \quad \forall \boldsymbol{x} \in \mathcal{P} \tag{2.30}$$

Theorem 2.2. *Let $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0} \;,\; \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \,\}$ be a polyhedral cone and $\mathcal{F} \subseteq \mathcal{P}$. Then, $\mathcal{F}$ is a face of $\mathcal{P}$ iff*

$$\mathcal{F} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0} \;,\; \boldsymbol{B}_{\zeta\star}\boldsymbol{x} = \boldsymbol{0} \;,\; \boldsymbol{B}_{\bar{\zeta}\star}\boldsymbol{x} \geq \boldsymbol{0} \,\} \tag{2.31}$$

where $\boldsymbol{B}_{\zeta\star}$ denotes a row submatrix of $\boldsymbol{B}$, meaning that some inequalities ζ of $\mathcal{P}$ are restrained to equalities.

Remarks

i) Every polyhedral cone has a finite number of faces.
ii) Faces of polyhedral cones are again cones.
iii) A face $\mathcal{F}$ of a cone $\mathcal{P}$ is called *proper face* if $\mathcal{F} \neq \mathcal{P}$ and $\mathcal{F} \neq \mathbf{lin}\ \mathcal{P}$.
iv) A zero-dimensional face is called *vertex* (the origin of a pointed cone).
v) A face of dimension $(n-1)$ of a cone $\mathcal{P}$ is called *facet*, where n denotes the dimension of $\mathcal{P}$, defined as the number of linearly independent points in $\mathcal{P}$, or equivalently, as the dimension of the smallest subspace containing $\mathcal{P}$.

Definition 2.26. *A vector $\boldsymbol{r}$ is said to be a* ray *of a cone $\mathcal{P}$ iff*

$$\boldsymbol{r} \neq \boldsymbol{0} \quad \textit{and} \quad \lambda\,\boldsymbol{r} \in \mathcal{P} \quad \textit{for every} \quad \lambda > 0\,.$$

Two rays $\boldsymbol{r}$ and $\boldsymbol{r}'$ are equivalent*, that is, $\boldsymbol{r} \simeq \boldsymbol{r}'$, if $\boldsymbol{r} = \lambda\,\boldsymbol{r}'$ for some positive λ.*

Definition 2.27. *A ray $\boldsymbol{r}$ of a cone $\mathcal{P}$ is called* extreme ray *iff $\mathcal{F} = \{\, \boldsymbol{x} = \lambda \boldsymbol{r} \mid \lambda \in \mathbb{R}_{>0} \,\}$ is a one-dimensional face of $\mathcal{P}$.*

Definition 2.28. *Let $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0} \;,\; \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \,\}$ be a polyhedral cone, and $\boldsymbol{r}$ be a ray of $\mathcal{P}$. Then, the set $\zeta(\boldsymbol{r})$ with $\boldsymbol{B}_{\zeta\star}\boldsymbol{r} = \boldsymbol{0}$ is called* zero set *of $\boldsymbol{r}$, where $\boldsymbol{B}_{\zeta\star}$ denotes the* tight *inequalities of $\mathcal{P}$, that is, the inequalities that are fulfilled with equality by $\boldsymbol{r}$. If b stands for the number of rows in $\boldsymbol{B}$, we have*

$$\zeta(\boldsymbol{r}) = \left\{\, i \mid \boldsymbol{B}_{i\star}\boldsymbol{r} = 0 \;,\; i \in \{1, \ldots, b\} \,\right\} \tag{2.32}$$

Lemma 2.3 (Algebraic Test for Extreme Rays)**.** *Let $\boldsymbol{r}$ be a ray of a pointed cone $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \,\}$, and ζ the zero set associated with $\boldsymbol{r}$. Then, $\boldsymbol{r}$ is an* extreme ray *of $\mathcal{P}$ iff*

$$\mathbf{rank} \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{B}_{\zeta\star} \end{bmatrix} = d - 1 \tag{2.33}$$

Proof. Let $\mathcal{F} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}_{\zeta\star}\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}_{\bar{\zeta}\star}\boldsymbol{x} \geq \boldsymbol{0} \,\}$ be a face of $\mathcal{P}$ containing $\boldsymbol{r}$, and $\mathcal{S} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}_{\zeta\star}\boldsymbol{x} = \boldsymbol{0} \,\}$ be the subspace containing $\mathcal{F}$. $\mathcal{S}$ is the smallest subspace that contains $\mathcal{F}$, and hence $\mathbf{dim}\ \mathcal{F} = \mathbf{dim}\ \mathcal{S}$. Note that $\mathcal{S}$ is the nullspace of the matrix $\boldsymbol{Z} = \left[\begin{smallmatrix} \boldsymbol{A} \\ \boldsymbol{B}_{\zeta\star} \end{smallmatrix}\right]$, that is, $\mathbf{dim}\ \mathcal{S} = d - \mathbf{rank}\ \boldsymbol{Z}$.

If eq. (2.33) holds, $\mathbf{rank}\ \boldsymbol{Z} = d - 1$ and hence $\mathbf{dim}\ \mathcal{S} = \mathbf{dim}\ \mathcal{F} = 1$, and $\boldsymbol{r}$ is an extreme ray. On the other hand, if $\boldsymbol{r}$ is an extreme ray, and since $\mathcal{F}$ is the minimal face containing $\boldsymbol{r}$, $\mathbf{dim}\ \mathcal{F} = \mathbf{dim}\ \mathcal{S} = 1$, and consequently, $\mathbf{rank}\ \boldsymbol{Z} = d - 1$. □

Corollary 2.1. *Let $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \,\}$ be a pointed cone and $\boldsymbol{r}$ an extreme ray of $\mathcal{P}$. Then, the zero set $\zeta(\boldsymbol{r})$ contains at least $d - 1 - \mathbf{rank}\ \boldsymbol{A}$ indices.*

Proof. Follows immediately from eq. (2.33). □

Lemma 2.4 (Combinatorial Test for Extreme Rays)**.** *Let $\boldsymbol{r}$ and $\boldsymbol{r}'$ be rays of a pointed cone $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \,\}$. Then, $\boldsymbol{r}$ is an* extreme ray *of $\mathcal{P}$ iff*

$$\zeta(\boldsymbol{r}') \supseteq \zeta(\boldsymbol{r}) \quad \Longrightarrow \quad \boldsymbol{r}' \simeq \boldsymbol{r} \tag{2.34}$$

Proof. Let $\zeta = \zeta(\boldsymbol{r})$, and $\mathcal{F} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}_{\zeta\star}\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}_{\bar{\zeta}\star}\boldsymbol{x} \geq \boldsymbol{0} \,\}$ be the minimal face of $\mathcal{P}$ containing $\boldsymbol{r}$.

If eq. (2.34) holds, the zero set ζ is maximal, and no ray $\boldsymbol{r}'$ other than $\boldsymbol{r}' \simeq \boldsymbol{r}$ is contained in $\mathcal{F}$. Consequently, $\mathcal{F} = \{\, \boldsymbol{r}' = \lambda\boldsymbol{r} \mid \lambda \in \mathbb{R}_{>0} \,\}$, and by Definition 2.27, $\boldsymbol{r}$ is an extreme ray. This proves ($\Leftarrow$).

On the contrary, if $\boldsymbol{r}$ is an extreme ray, and $\zeta(\boldsymbol{r}') \supseteq \zeta$, the minimal face $\mathcal{F}$ of $\mathcal{P}$ containing $\boldsymbol{r}$ also contains $\boldsymbol{r}'$. By Def. 2.27, we have also $\mathcal{F} = \{\, \boldsymbol{x} = \lambda\boldsymbol{r} \mid \lambda \in \mathbb{R}_{>0} \,\}$, that is, $\boldsymbol{r}' = \lambda\boldsymbol{r}$, and thus $\boldsymbol{r}' \simeq \boldsymbol{r}$. □

The next lemma is central for the computation of generators. It states that extreme rays are generating a pointed cone. Due to Lemma 2.4, we know that extreme rays are essential, that is, they cannot be combined from other rays. It is not obvious, however, that they are indeed sufficient to generate the cone (Malkin, 2007).

Lemma 2.5. *A pointed polyhedral cone $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \,\}$ is generated by its extreme rays, that is,*

$$\mathcal{P} = \left\{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x} = \boldsymbol{R}\,\boldsymbol{c} \quad \text{with} \quad \boldsymbol{c} \geq \boldsymbol{0} \,\right\} \tag{2.35}$$

where the columns of $\boldsymbol{R}$ are the extreme rays of $\mathcal{P}$.

Proof. By theorem 2.1, we know that for some matrix $\boldsymbol{G}$, $\mathcal{P} = \{\, \boldsymbol{x} = \boldsymbol{G}\,\boldsymbol{c} \;,\; \boldsymbol{c} \geq \boldsymbol{0} \,\}$. Let $\mathcal{P}' = \{\, \boldsymbol{x} = \boldsymbol{R}\,\boldsymbol{c} \;,\; \boldsymbol{c} \geq \boldsymbol{0} \,\}$, and obviously $\mathcal{P}' \subseteq \mathcal{P}$. If we assume that $\mathcal{P}' \neq \mathcal{P}$, we can choose a ray $\boldsymbol{r} \in \mathcal{P} \setminus \mathcal{P}'$ with maximal $\zeta(\boldsymbol{r})$. It is not an extreme ray since $\boldsymbol{r} \notin \mathcal{P}'$, and by Lemma 2.4, there must exist another extreme ray $\boldsymbol{r}'$ with $\zeta(\boldsymbol{r}') \supseteq \zeta(\boldsymbol{r})$. Choose a row i of $\boldsymbol{B}$ with $\beta' = \boldsymbol{B}_{i\star}\boldsymbol{r}' > 0$ (and hence also $\beta = \boldsymbol{B}_{i\star}\boldsymbol{r} > 0$), such that $\frac{\beta}{\beta'}$ is minimal. Consider the ray $\boldsymbol{r}'' = \boldsymbol{r} - \frac{\beta}{\beta'}\boldsymbol{r}'$, a ray of $\mathcal{P}$ since all inequalities are still valid due to the minimal choice of $\frac{\beta}{\beta'}$. Furthermore, $\boldsymbol{r}'' \notin \mathcal{P}'$ since $\boldsymbol{r}' \in \mathcal{P}'$, otherwise $\boldsymbol{r} = \boldsymbol{r}'' + \frac{\beta}{\beta'}\boldsymbol{r}' \in \mathcal{P}'$. By construction of $\boldsymbol{r}''$, we have $\zeta(\boldsymbol{r}'') \supsetneq \zeta(\boldsymbol{r})$, conflicting with the assumption that $\zeta(\boldsymbol{r})$ is maximal in $\mathcal{P} \setminus \mathcal{P}'$. □

Constraint Transformations

We have introduced a very general definition for a finitely constrained cone (eq. 2.25). In this section, we give alternative forms of this definition and show how to transform between them. Depending on the application, the alternative forms lead to simpler expressions and data structures. We highlight the simplifications that are most important for this work. The transformations are adapted from Malkin (2007).

Proposition 2.1. *Every cone $\mathcal{P}$ given as in eq. (2.25) can be transformed into a cone solely described by inequalities:*

$$\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x} = \boldsymbol{T}'\,\boldsymbol{x}' \;,\; \boldsymbol{B}'\,\boldsymbol{x}' \geq \boldsymbol{0}\} \tag{2.36}$$

Proof. A simple way to achieve this is to set $\boldsymbol{B}' = [\boldsymbol{A}^T \;,\; -\boldsymbol{A}^T \;,\; \boldsymbol{B}^T]^T$, but we would like to remove all equalities from the new matrix $\boldsymbol{B}'$. We use a transformation adapted from Malkin (2007): we can assume that $\boldsymbol{A}$ in (2.25) has full row rank, since we can remove linearly dependent rows without changing the cone. Let $a = \operatorname{rank}(\boldsymbol{A})$ be the number of rows of $\boldsymbol{A}$, and choose a independent columns $\tau \subseteq \{1, \ldots, d\}$ from $\boldsymbol{A}$. Let $\boldsymbol{A}_{\star\tau} \in \mathbb{R}^{a\times a}$ be the columns of $\boldsymbol{A}$ indexed by τ, and $\boldsymbol{T} \in \mathbb{R}^{a\times a}$ be the inverse matrix of $\boldsymbol{A}_{\star\tau}$ (that is, $\boldsymbol{T}\boldsymbol{A}_{\star\tau} = \boldsymbol{I}$). We get

$$\begin{aligned}
\mathcal{P} &= \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0} \;,\; \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0}\} \\
&= \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{T}\,\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0} \;,\; \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0}\} \\
&= \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{T}\,\boldsymbol{A}_{\star\tau}\,\boldsymbol{x}_\tau + \boldsymbol{T}\,\boldsymbol{A}_{\star\overline{\tau}}\,\boldsymbol{x}_{\overline{\tau}} = \boldsymbol{0} \;,\; \boldsymbol{B}_{\star\tau}\,\boldsymbol{x}_\tau + \boldsymbol{B}_{\star\overline{\tau}}\,\boldsymbol{x}_{\overline{\tau}} \geq \boldsymbol{0}\} \\
&= \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x}_\tau = -\boldsymbol{T}\,\boldsymbol{A}_{\star\overline{\tau}}\,\boldsymbol{x}_{\overline{\tau}} \;,\; (-\boldsymbol{B}_{\star\tau}\,\boldsymbol{T}\,\boldsymbol{A}_{\star\overline{\tau}} + \boldsymbol{B}_{\star\overline{\tau}})\,\boldsymbol{x}_{\overline{\tau}} \geq \boldsymbol{0}\}
\end{aligned}$$

□

The cone $\mathcal{P}' = \{\, \boldsymbol{x}' \in \mathbb{R}^{d-a} \mid \boldsymbol{B}'\boldsymbol{x}' \geq \boldsymbol{0} \,\}$ with $\boldsymbol{B}' = (-\boldsymbol{B}_{\star\tau}\,\boldsymbol{T}\,\boldsymbol{A}_{\star\overline{\tau}} + \boldsymbol{B}_{\star\overline{\tau}})$ is a projection of the original cone $\mathcal{P}$ to the variables $\boldsymbol{x}_{\overline{\tau}}$. For us, in particular the influence on zero sets is important since we are interested in the extreme rays of the cones. Note that $\mathcal{P}'$ has essentially the same inequalities as $\mathcal{P}$, and hence also $\zeta'(\boldsymbol{x}') = \zeta(\boldsymbol{x})$. The combinatorial extreme ray test (eq. 2.34) has not changed, but since we have eliminated equality matrix $\boldsymbol{A}$ with $a = \operatorname{rank}(\boldsymbol{A})$, the algebraic test (eq. 2.33) for an extreme ray $\boldsymbol{r}'$ of $\mathcal{P}'$ simplifies to

$$\textbf{rank}\; \boldsymbol{B}'_{\zeta\star} = d - a - 1 \tag{2.37}$$

where $\zeta = \zeta'(\boldsymbol{r}')$ denotes the zero set of $\boldsymbol{r}'$.

Proposition 2.2. *Every cone $\mathcal{P}$ given as in eq. (2.25) or eq. (2.36) can be transformed into a cone solely described by equalities and non-negative variables:*

$$\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x} = \boldsymbol{T}''\,\boldsymbol{x}''\ ,\ \boldsymbol{A}''\,\boldsymbol{x}'' = \boldsymbol{0}\ ,\ \boldsymbol{x}'' \geq \boldsymbol{0}\} \tag{2.38}$$

Proof. We assume that the cone is defined as $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0}\}$, using the transformation of Prop. 2.1 if necesssary. For simplicity, we furthermore assume that the cone is pointed, that is, the origin $\boldsymbol{0}$ is an extreme point of $\mathcal{P}$, or equivalently, $\operatorname{rank}(\boldsymbol{B}) = d$. Note that according to Lemma 2.2, any non-pointed cone can be reduced to the orthogonal complement of its lineality space $\{\boldsymbol{x} \mid \boldsymbol{B}\,\boldsymbol{x} = \boldsymbol{0}\}$.

The transformation (adapted from Malkin (2007)) introduces slack variables $\boldsymbol{y}$ to change inequality to equality constraints, and eliminates the original variables $\boldsymbol{x}$. Let b be the number of rows in $\boldsymbol{B}$, and note that $\operatorname{rank}(\boldsymbol{B}) = d$ since $\mathcal{P}$ is pointed. Then, there exists a matrix $\boldsymbol{T} \in \mathbb{R}^{b\times b}$ such that $\boldsymbol{T}\,\boldsymbol{B} = [\boldsymbol{I}, \boldsymbol{0}]^T$, and hence

$$\begin{aligned}
\mathcal{P} &= \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0}\} \\
&= \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{B}\,\boldsymbol{x} - \boldsymbol{I}\,\boldsymbol{y} = \boldsymbol{0}\ ,\ \boldsymbol{y} \geq \boldsymbol{0}\} \\
&= \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{T}\,\boldsymbol{B}\,\boldsymbol{x} - \boldsymbol{T}\,\boldsymbol{y} = \boldsymbol{0}\ ,\ \boldsymbol{y} \geq \boldsymbol{0}\} \\
&= \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x} = \boldsymbol{T}_{\rho\star}\,\boldsymbol{y}\ ,\ \boldsymbol{T}_{\overline{\rho}\star}\,\boldsymbol{y} = \boldsymbol{0}\ ,\ \boldsymbol{y} \geq \boldsymbol{0}\}
\end{aligned}$$

where $\rho = \{1, \ldots, d\}$, that is, $\boldsymbol{T}_{\rho\star}$ are the first d, and $\boldsymbol{T}_{\overline{\rho}\star}$ are the last $b - d$, rows of $\boldsymbol{T}$, respectively. □

The cone $\mathcal{P}'' = \{\,\boldsymbol{y} \in \mathbb{R}^b \mid \boldsymbol{A}''\boldsymbol{y} = \boldsymbol{0}\ ,\ \boldsymbol{y} \geq \boldsymbol{0}\,\}$ with $\boldsymbol{A}'' = \boldsymbol{T}_{\overline{\rho}\star}$ does not change the zero sets. For a ray $\boldsymbol{r}$ of the original cone $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0}\}$ and a ray $\boldsymbol{r}'' \in \mathcal{P}''$, we have $\zeta''(\boldsymbol{r}'') = \zeta(\boldsymbol{r})$. However, the zero set Definition 2.28 simplifies to the indices corresponding to zero entries in $\boldsymbol{r}''$:

$$\zeta''(\boldsymbol{r}'') = \{\, i \mid r''_i = 0\ ,\ i \in \{1, \ldots, b\}\,\} \tag{2.39}$$

The algebraic extreme ray test (eq. 2.33) can be simplified as follows:

Proposition 2.3. *A ray $\boldsymbol{r}''$ of a polyhedral cone $\mathcal{P}'' = \{\,\boldsymbol{y} \in \mathbb{R}^d \mid \boldsymbol{A}''\boldsymbol{y} = \boldsymbol{0}\ ,\ \boldsymbol{y} \geq \boldsymbol{0}\,\}$ is an* extreme ray *of $\mathcal{P}''$ if and only if*

$$\operatorname{\mathbf{rank}} \boldsymbol{A}''_{\star\overline{\zeta}} = \left|\,\overline{\zeta}\,\right| - 1 \tag{2.40}$$

where $\overline{\zeta}$ are the indices corresponding to non-zero entries in $\boldsymbol{r}''$, that is, $\overline{\zeta} = \{\, i \mid r''_i \neq 0\,\}$.

Proof. According to the algebraic test Lemma 2.3, $\boldsymbol{r}''$ is an extreme ray if and only if $\operatorname{\mathbf{rank}} \left[\begin{smallmatrix} \boldsymbol{A}'' \\ \boldsymbol{I}_{\zeta\star} \end{smallmatrix}\right] = d - 1$. We derive the rank by transforming the matrix $\boldsymbol{C} = \left[\begin{smallmatrix} \boldsymbol{I}_{\zeta\star} \\ \boldsymbol{A}'' \end{smallmatrix}\right]$ into upper triangular form using Gaussian elimination. Without loss of generality, we can assume that $\zeta = \{1, \ldots, z\}$, i.e. $\boldsymbol{I}_{\zeta\star} = [\,\boldsymbol{I}\,,\,\boldsymbol{0}\,]$. To triangulize the first z rows and columns of $\boldsymbol{C}$, we eliminate all values for rows $i > z$ and columns j with $1 \leq j \leq z$ by substracting an appropriate multiple of a row of $\boldsymbol{I}_{\zeta\star}$, resulting in a matrix $\boldsymbol{C}_z = \left[\begin{smallmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}''' \end{smallmatrix}\right]$. We have $\operatorname{\mathbf{rank}} \boldsymbol{C} = \operatorname{\mathbf{rank}} \boldsymbol{C}_z = \left|\,\zeta\,\right| + \operatorname{\mathbf{rank}} \boldsymbol{A}''' \overset{!}{=} d - 1$. For general ζ, we still have $\boldsymbol{A}''' = \boldsymbol{A}''_{\star\overline{\zeta}}$, and (2.40) follows since $\left|\,\overline{\zeta}\,\right| = d - \left|\,\zeta\,\right|$. □

2.2.3 Polyhedra

Definition 2.29. *A* finitely constrained convex set *is a convex set defined as the solution set of finitely many linear equalities and inequalities:*

$$\mathcal{P} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{a}\ ,\ \boldsymbol{B}\,\boldsymbol{x} \leq \boldsymbol{b} \right\} \tag{2.41}$$

Definition 2.30. *A* finitely generated convex set *is a convex set defined as* Minkowski sum *of the conic hull of finitely many generating directions, plus the convex hull of finitely many vertices (columns in the extreme ray matrix $\boldsymbol{R}$ and in the vertex matrix $\boldsymbol{V}$):*

$$\begin{aligned} \mathcal{P} &= \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x} = \boldsymbol{R}\,\boldsymbol{c} + \boldsymbol{V}\boldsymbol{\lambda} \quad \textit{with} \quad \boldsymbol{c} \geq \boldsymbol{0}\ ,\ \boldsymbol{\lambda} \geq 0\ ,\ \textstyle\sum \lambda_i = 1 \right\} \\ &= \left\{ \boldsymbol{r} + \boldsymbol{v} \mid \boldsymbol{r} \in \textbf{cone}\ \boldsymbol{R}\ ,\ \boldsymbol{v} \in \textbf{conv}\ \boldsymbol{V} \right\} \\ &= \textbf{cone}\ \boldsymbol{R} + \textbf{conv}\ \boldsymbol{V} \end{aligned} \tag{2.42}$$

Theorem 2.3 (Minkowski-Weyl's Theorem)**.** *A convex set is finitely generated iff it is finitely constrained.*

We summarize Def. 2.29 and Def. 2.30 as follows:

Definition 2.31. *A finitely constrained/generated convex set is called* (convex) polyhedron.

Definition 2.32. *A polyhedron $\mathcal{P} \subseteq \mathbb{R}^d$ is* bounded *if every coordinate is bounded, i.e. if there exists some finite $b \in \mathbb{R}_{\geq 0}$ such that $-b \leq x_i \leq b, i \in \{1, \ldots, d\}$ for every $\boldsymbol{x} \in \mathcal{P}$.*

Definition 2.33. *A bounded polyhedron is called* polytope.

Remarks

i) Any polyhedral cone is also a polyhedron.

ii) A polyhedron $\mathcal{P}$ is bounded iff it contains no directions, that is, if the *recession cone* $\mathcal{C} = \{\boldsymbol{r} \mid \boldsymbol{v} + \lambda\,\boldsymbol{r} \in \mathcal{P}$ for every $\boldsymbol{v} \in \mathcal{P}, \lambda \geq 0\}$ is the trivial cone $\{\boldsymbol{0}\}$.

iii) A polyhedron in $\mathbb{R}^d$ can be seen as the intersection of a polyhedral cone and a hyperplane in $\mathbb{R}^{d+1}$.

2.3 Polyhedral Computation

We first describe related enumeration problems that can be solved with algorithms to compute extreme rays. Then, we discuss the double description method in detail, an enumeration procedure that proved efficient especially for degenerate problems. We conclude with notes and comments on complexity, alternative algorithms and related problems.

2.3.1 Enumeration Problems

In this section, we shortly describe enumeration problems that are important for this thesis. All problems are equivalent, and the same algorithms are usually applied.

Extreme Ray Enumeration

This is our principal enumeration problem. *Extreme ray enumeration* describes an algorithm for finding the extreme rays of a polyhedral cone given its constraint representation. Referring to our definitions, it converts a cone specified with constraints as in Def. 2.20 into the generating form used in Def. 2.21.

We will describe an algorithm to solve extreme ray enumeration in Section 2.3.2. It is particularly well suited for highly degenerate problems, such as those arising in systems biology. For certain cases, however, more efficient (i.e. polynomial) methods are known. We summarize alternative algorithms for extreme ray enumeration in Section 2.3.3.

Facet Enumeration (aka Convex Hull)

Facet enumeration is the reverse process of extreme ray enumeration, that is, a finitely generated cone is converted into a finitely constrained cone. It turns out that conversion algorithms are exactly the same as those for extreme ray enumeration. Using Farkas' lemma, one can show that we can apply extreme ray enumeration to the transpose of the extreme ray matrix. More formally, given the extreme ray matrix $\boldsymbol{R}$ of a polyhedral cone as in eq. (2.26), we first compute the linealities $\boldsymbol{A}$ in eq. (2.25) by finding a basis $\boldsymbol{A}^T$ for the nullspace of $\boldsymbol{R}^T$:

$$\mathcal{N}(\boldsymbol{R}^T) = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{R}^T \boldsymbol{x} = \boldsymbol{0} \,\} \tag{2.43}$$

$$= \mathcal{R}(\boldsymbol{A}^T) = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x} = \boldsymbol{A}^T \boldsymbol{\lambda} \,\} \tag{2.44}$$

The inequality matrix $\boldsymbol{B}$ (eq. 2.25) can be derived by computing the extreme rays $\boldsymbol{B}^T$ of the following pointed polyhedral cone:

$$\mathcal{P} = \left\{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0} \;,\; \boldsymbol{R}^T \boldsymbol{x} \geq \boldsymbol{0} \,\right\} \tag{2.45}$$

$$= \mathbf{cone}\ \boldsymbol{B}^T \tag{2.46}$$

The term facet enumeration is also used for the enumeration of the facets of a polyhedron. As we will see in the next section, we usually embed a d dimensional polyhedron into a polyhedral cone in $\mathbb{R}^{d+1}$ to perform vertex or facet enumeration.

Methods performing facet enumeration are sometimes also called *convex hull* algorithms, since they compute the inequalities defining the convex hull from extreme points and directions. We avoid this term since it may be confused with another (much simpler) problem of identifying non-redundant vertices generating the convex hull of a point set. This problem can be solved with linear programming (LP), and we call it *redundancy removal* problem, which also includes the dual problem of finding the non-redundant constraints of a polyhedron stated in constraint form.

Vertex Enumeration

Vertex enumeration usually means an algorithm for enumerating the vertices of a polytope. Note that for a general unbounded polyhedron, we also need the extreme rays to generate it. Therefore, we usually compute vertices together with extreme rays, and in fact, we can use a standard extreme ray enumeration procedure for polyhedral cones for this purpose. To do so, we embed a d dimensional polyhedron into $\mathbb{R}^{d+1}$ as follows:

$$\mathcal{P}^{(d)} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{a}\ ,\ \boldsymbol{B}\,\boldsymbol{x} \leq \boldsymbol{b} \right\} \tag{2.47}$$

$$\mathcal{P}^{(d+1)} = \left\{ \boldsymbol{x} \in \mathbb{R}^{d+1} \mid -\boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{a}\,x_{d+1} = \boldsymbol{0}\ ,\ -\boldsymbol{B}\,\boldsymbol{x} + \boldsymbol{b}\,x_{d+1} \geq \boldsymbol{0}\ ,\ x_{d+1} \geq 0 \right\} \tag{2.48}$$

Note that eq. (2.48) is a finitely constrained cone (see eq. (2.25)), and we have

$$\mathcal{P}^{(d)} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid (\boldsymbol{x}, 1) \in \mathcal{P}^{(d+1)} \right\} \tag{2.49}$$

To compute the extreme rays and vertices of a polyhedron, we transform eq. (2.47) into eq. (2.48). After computing the extreme rays $\boldsymbol{Q}$ for $\mathcal{P}^{(d+1)}$, we get the extreme rays $\boldsymbol{R}$ and the vertices $\boldsymbol{V}$ of $\mathcal{P}^{(d)}$ after reordering and scaling the columns in $\boldsymbol{Q}$ into the following form:

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{V} \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{2.50}$$

2.3.2 Double Description Method

The *double description method* was initially invented by Motzkin et al. (1953) and uses the notion of a *double description pair* or *DD pair* for short. A DD pair denotes a pair $(\boldsymbol{B}, \boldsymbol{R})$ of real matrices such that

$$\boldsymbol{B}\,\boldsymbol{x} \geq \boldsymbol{0} \quad \text{if and only if} \quad \boldsymbol{x} = \boldsymbol{R}\,\boldsymbol{c} \ \text{ for some } \ \boldsymbol{c} \geq \boldsymbol{0} \tag{2.51}$$

The term "double description" is natural in the sense that both matrices describe the same object. The second description in eq. (2.51) using $\boldsymbol{R}$ corresponds to the generating definition of a polyhedral cone (eq. 2.26); the inequality description reflected by matrix $\boldsymbol{B}$ is a special form of eq. (2.25) for a constrained cone (e.g. attained by using the transformation given in Prop. 2.1).

The double description method takes $\boldsymbol{B}$ as input and returns $\boldsymbol{R}$. It starts with an initial DD pair $(\boldsymbol{B}^0, \boldsymbol{R}^0)$ and iteratively constructs a pair $(\boldsymbol{B}^j, \boldsymbol{R}^j)$ from the previous pair

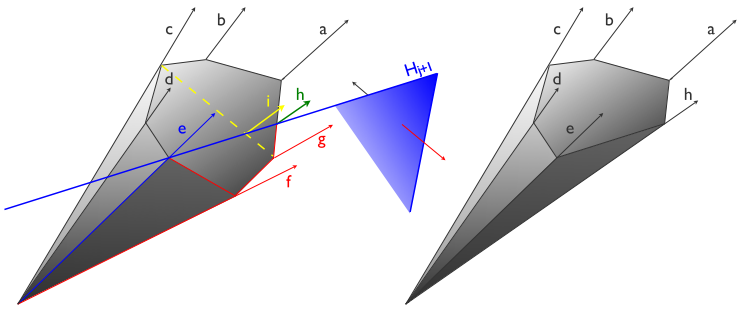


Figure 2.7: Iteration step of the double description method: the preliminary polyhedral cone is intersected with the halfspace representing the added constraint. The hyperplane H_{j+1} separating valid from invalid rays is shown in blue. New extreme rays generated at this step (here: h) are lying in the hyperplane and are generated from adjacent ancestor rays (a and g). Descendants of non-adjacent rays, such as i from c and g, are not extreme rays.

$(\boldsymbol{B}^{j-1}, \boldsymbol{R}^{j-1})$ ending up in the final pair $(\boldsymbol{B}, \boldsymbol{R})$. The DD method was rediscovered many times under different names, such as Chernikova's algorithm (Chernikova, 1965) or the dual method for computing $\boldsymbol{R}$ from $\boldsymbol{B}$ known as *beneath-and-beyond method* (Edelsbrunner, 1987; Mulmuley, 1993).

We will describe the DD method for a cone defined by equalities and non-negativity of the variables. The algorithm implemented as part of this thesis uses this form, first because it simplifies algorithm and data structures, and second because our main application, the computation of elementary modes, is usually stated in this form. Note that any polyhedral cone can be transformed into this form (see Prop. 2.1 and (2.2)). The algorithm for the direct conversion from $\boldsymbol{B}$ to $\boldsymbol{R}$ as in eq. (2.51) is given in Fukuda and Prodon (1995).

It turns out that it is easy to find initial DD pairs $(\boldsymbol{B}^0, \boldsymbol{R}^0)$ such that certain constraints of $\boldsymbol{B}$ are already considered. All remaining constraints are added iteratively, yielding a new cone at each iteration step. Subsequent cones are contained in preliminary ones and are obtained by intersecting the predecessor cone with the halfspace associated with the added constraint. An intersection of this kind is illustrated in Fig. 2.7.

Here, we start with a cone defined as

$$\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{x} \geq \boldsymbol{0} \,\} \tag{2.52}$$

As proposed by Wagner (2004), we use a row-echelon kernel matrix for the nullspace $\mathcal{N}(\boldsymbol{A}) = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\}$ to derive an initial DD pair. Setting $\boldsymbol{B} = \left[\begin{smallmatrix} \boldsymbol{A} \\ -\boldsymbol{A} \\ \boldsymbol{I} \end{smallmatrix}\right]$, our initial pair $(\boldsymbol{B}^0, \boldsymbol{R}^0)$ is defined as

$$\boldsymbol{B}^0 = \left[\begin{smallmatrix} \boldsymbol{A} \\ -\boldsymbol{A} \\ \boldsymbol{I}_{\rho\star} \end{smallmatrix}\right] \qquad \boldsymbol{R}^0 = \boldsymbol{K} = \left[\begin{smallmatrix} \boldsymbol{I} \\ \boldsymbol{K}^* \end{smallmatrix}\right] \quad \text{with} \quad \boldsymbol{A}\boldsymbol{K} = \boldsymbol{0} \tag{2.53}$$

where $\boldsymbol{I}_{\rho\star}$ denotes the rows ρ of the identity matrix corresponding to already considered non-negativity constraints. Since we use nullspace basis vectors, the initial DD pair already

considers all equality constraints. By choosing the row echelon form, $r = |\rho| = d - \mathbf{rank}\ \boldsymbol{A}$ non-negativity constraints are also considered, and $d - r = \mathbf{rank}\ \boldsymbol{A}$ constraints are left for the iteration phase. The columns in $\boldsymbol{K}$ span the nullspace, and every vector $\boldsymbol{v} \in \{\, \boldsymbol{x} \mid \boldsymbol{B}^0 \boldsymbol{x} \geq \boldsymbol{0} \,\}$ can be constructed as a non-negative (conic) combination of columns in $\boldsymbol{K}$, hence $(\boldsymbol{B}^0, \boldsymbol{R}^0)$ is a DD pair. Since $\boldsymbol{K}$ are basis vectors, the columns are independent and $(\boldsymbol{B}^0, \boldsymbol{R}^0)$ is a *minimal* DD pair. The following statement shows that they are indeed extreme rays of the initial cone.

Proposition 2.4. *Let $(\boldsymbol{B}^0, \boldsymbol{R}^0)$ be an initial DD pair as defined in eq. (2.53). Then, the columns in $\boldsymbol{K}$ are extreme rays of the polyhedral cone $\mathcal{P}^0 = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{B}^0 \boldsymbol{x} \geq \boldsymbol{0} \,\}$.*

Proof. Consider the matrix $\boldsymbol{C} = \left[\begin{smallmatrix} \boldsymbol{A} \\ \boldsymbol{I} \end{smallmatrix}\right]$, and note that $\boldsymbol{C}\boldsymbol{K} = \left[\begin{smallmatrix} \boldsymbol{0} \\ \boldsymbol{I} \\ \boldsymbol{K}^* \end{smallmatrix}\right]$. If we consider only rows ρ of the identity matrix in $\boldsymbol{C}$ corresponding to already considered non-negativity constraints, we have $\boldsymbol{C}_\rho := \left[\begin{smallmatrix} \boldsymbol{A} \\ \boldsymbol{I}_{\rho\,\star} \end{smallmatrix}\right]$ and $\boldsymbol{C}_\rho \boldsymbol{K} = \left[\begin{smallmatrix} \boldsymbol{0} \\ \boldsymbol{I} \end{smallmatrix}\right]$. Note that the rows $\boldsymbol{I}_{\rho\,\star}$ are independent in $\boldsymbol{C}_\rho$, otherwise, they would be mapped into zero by $\boldsymbol{K}$. Hence, we have $\mathbf{rank}\ \boldsymbol{C}_\rho = \mathbf{rank}\ \boldsymbol{A} + |\rho| = d$. Take any column $\boldsymbol{r}$ of $\boldsymbol{K}$ and consider only rows of $\boldsymbol{I}_{\rho\,\star}$ that correspond to the zero set $\zeta = \zeta(\boldsymbol{r})$, yielding $\boldsymbol{C}_\zeta := \left[\begin{smallmatrix} \boldsymbol{A} \\ \boldsymbol{I}_{\zeta\,\star} \end{smallmatrix}\right]$. Since $\boldsymbol{r}$ has only one non-zero entry in ρ, the matrix $\boldsymbol{C}_\zeta$ has lost one independent row of $\boldsymbol{C}_\rho$. We have $\mathbf{rank}\ \boldsymbol{C}_\zeta = d - 1$, and by eq. (2.33), $\boldsymbol{r}$ is an extreme ray of $\mathcal{P}^0$. □

At iteration step j, we construct a DD pair $(\boldsymbol{B}^j, \boldsymbol{R}^j)$ from the previous pair $(\boldsymbol{B}^{j-1}, \boldsymbol{R}^{j-1})$ by adding an additional non-negativity constraint. The rays of the cone $\mathcal{P}^{j-1}$ are partitioned into three parts:

$$\begin{aligned} \mathcal{P}_> &= \{\, \boldsymbol{x} \in \mathcal{P}^{j-1} \mid x_j > 0 \,\} \\ \mathcal{P}_0 &= \{\, \boldsymbol{x} \in \mathcal{P}^{j-1} \mid x_j = 0 \,\} \\ \mathcal{P}_< &= \{\, \boldsymbol{x} \in \mathcal{P}^{j-1} \mid x_j < 0 \,\} \end{aligned} \tag{2.54}$$

The rays of the cone $\mathcal{P}^j$ are those in $\mathcal{P}_> \cup \mathcal{P}_0$. Let τ be the column indices of $\boldsymbol{R}^{j-1}$ and partition extreme rays $\boldsymbol{r}_i$ with $(i \in \tau)$ accordingly:

$$\begin{aligned} \tau_> &= \{\, i \in \tau \mid \boldsymbol{r}_i \in \mathcal{P}_> \,\} \\ \tau_0 &= \{\, i \in \tau \mid \boldsymbol{r}_i \in \mathcal{P}_0 \,\} \\ \tau_< &= \{\, i \in \tau \mid \boldsymbol{r}_i \in \mathcal{P}_< \,\} \end{aligned} \tag{2.55}$$

The next DD pair $(\boldsymbol{B}^j, \boldsymbol{R}^j)$ can be obtained from $(\boldsymbol{B}^{j-1}, \boldsymbol{R}^{j-1})$ as follows:

Lemma 2.6 (Main Lemma for Double Description Method)**.** *Let $(\boldsymbol{B}^{j-1}, \boldsymbol{R}^{j-1})$ be a DD pair and j be the index of the non-negativity constraint added at iteration j. Let $\boldsymbol{r}_i$ be column vectors of $\boldsymbol{R}^{j-1}$ with $i \in \tau' \subseteq \tau$, and let us denote by $[\boldsymbol{r}_i]$ the matrix received by concatenating all column vectors $\boldsymbol{r}_i$. Then, $(\boldsymbol{B}^j, \boldsymbol{R}^j)$ is a DD pair defined by*

$$\begin{aligned} & \boldsymbol{B}^j && = && \left[\begin{smallmatrix} \boldsymbol{B}^{j-1} \\ \boldsymbol{I}_{j\,\star} \end{smallmatrix}\right] \ , \ \boldsymbol{R}^j = \left[\, [\boldsymbol{r}_i] \ , \ [\boldsymbol{r}_h] \ , \ [\boldsymbol{r}_{(ik)}] \,\right] \\ \text{for all} \quad & i && \in && \tau_> \ , \ h \in \tau_0 \ , \ k \in \tau_< \\ \text{with} \quad & \boldsymbol{r}_{(ik)} && = && p_j\, \boldsymbol{q} - q_j\, \boldsymbol{p} \quad \text{and} \quad \boldsymbol{p} = \boldsymbol{r}_i \ , \ \boldsymbol{q} = \boldsymbol{r}_k \end{aligned} \tag{2.56}$$

Proof. See also Fukuda and Prodon (1995). $\boldsymbol{B}^j$ is clear, since we only add non-negativity constraint j to the previous matrix $\boldsymbol{B}^{j-1}$. Let $\mathcal{P} = \{\, \boldsymbol{x} \mid \boldsymbol{B}^j \boldsymbol{x} \geq \boldsymbol{0} \,\}$ and $\mathcal{P}'$ be the cone generated by $\boldsymbol{R}^j$. We must show that $\mathcal{P} = \mathcal{P}'$. Clearly, all rays $[\boldsymbol{r}_i]$ and $[\boldsymbol{r}_h]$ are in $\mathcal{P}$. Note that $[\boldsymbol{r}_{(ik)}]$ are non-negative combinations of vectors in $\mathcal{P}^{j-1}$, hence all $\boldsymbol{r}_{(ik)} \in \mathcal{P}^{j-1}$. By construction, $r_{(ik)_j} = 0$, such that also $\boldsymbol{r}_{(ik)} \in \mathcal{P}$, and we have $\mathcal{P}' \subseteq \mathcal{P}$.

Now, we show that $\boldsymbol{x} \in \mathcal{P}$ implies $\boldsymbol{x} \in \mathcal{P}'$, and hence $\mathcal{P} \subseteq \mathcal{P}'$. Since $\boldsymbol{x} \in \mathcal{P}$, also $\boldsymbol{x} \in \mathcal{P}^{j-1}$, and there exist some $\lambda \geq \boldsymbol{0}$ such that

$$\boldsymbol{x} = \sum_{k \in \tau} \lambda_k \boldsymbol{r}_k \tag{2.57}$$

If $\lambda_k = 0$ for all $k \in \tau_<$ in eq. (2.57), $\boldsymbol{x} \in \mathcal{P}'$ and we are done. Suppose that $\lambda_k > 0$ for some $i \in \tau_<$. Then, there is also some $\lambda_i > 0$ with $i \in \tau_>$, otherwise $x_j < 0$ and $\boldsymbol{x} \notin \mathcal{P}$. By eq. (2.56) and for the same i and k, there must also exist some $\boldsymbol{r}_{(ik)} = p_j\, \boldsymbol{q} - q_j\, \boldsymbol{p}$ with $\boldsymbol{p} = \boldsymbol{r}_i$ and $\boldsymbol{q} = \boldsymbol{r}_k$. We isolate $\boldsymbol{p}$ and $\boldsymbol{q}$ in eq. (2.57) and get

$$\begin{aligned} \boldsymbol{x} &= \underbrace{\lambda_i \boldsymbol{r}_i}_{\lambda_i \boldsymbol{p}} + \underbrace{\lambda_k \boldsymbol{r}_k}_{\lambda_k \boldsymbol{q}} + \underbrace{\sum_{i' \in \tau \setminus \{i,k\}} \lambda_{i'} \boldsymbol{r}_{i'}}_{\boldsymbol{x}'} \\ &= \boldsymbol{x}' + \lambda_i \boldsymbol{p} + \lambda_k \boldsymbol{q} - \lambda \boldsymbol{r}_{(ik)} + \lambda \boldsymbol{r}_{(ik)} \\ &= \boldsymbol{x}' + \underbrace{(\lambda_i + \lambda\, q_j)}_{*^1} \boldsymbol{p} + \underbrace{(\lambda_k - \lambda\, p_j)}_{*^2} \boldsymbol{q} + \lambda \boldsymbol{r}_{(ik)} \end{aligned} \tag{2.58}$$

If we choose $\lambda = \min(-\frac{\lambda_i}{q_j}, \frac{\lambda_k}{p_j}) \geq 0$, at least one of $*^1$ and $*^2$ in eq. (2.58) vanishes, and the other stays non-negative. Compared to eq. (2.57), we use strictly fewer $\lambda_k > 0$ for $k \in \tau_> \cup \tau_<$ to compose $\boldsymbol{x}$ in eq. (2.58), and we can repeat this procedure until all $k \in \tau_> \cup \tau_0$. Consequently, $\boldsymbol{x} \in \mathcal{P}'$ and hence $\mathcal{P} \subseteq \mathcal{P}'$. □

The DD pair $(\boldsymbol{B}^j, \boldsymbol{R}^j)$ obtained by the instructions defined in Lemma 2.6 is not minimal. Many if not most of the new rays $\boldsymbol{r}_{(ik)}$ in eq. (2.56) are redundant, that is, they are not extremal. To derive a minimal DD pair, we can remove all $\boldsymbol{r} \in [\boldsymbol{r}_{(ik)}]$ which are not extreme rays, e.g. using Lemma 2.3 or 2.4. First, we show that $\boldsymbol{r}_i$ $(i \in \tau_> \cup \tau_0)$ is an extreme ray of $\mathcal{P}^j$ if it was an extreme ray of $\mathcal{P}^{j-1}$:

Proposition 2.5. *Let $(\boldsymbol{B}^{j-1}, \boldsymbol{R}^{j-1})$ be a minimal DD pair with associated cone $\mathcal{P}^{j-1}$, that is, $\boldsymbol{R}^{j-1}$ are the extreme rays of $\mathcal{P}^{j-1}$. Then, following the notation in eq. (2.56), every $\boldsymbol{r}_i$ $(i \in \tau_>)$ and $\boldsymbol{r}_h$ $(h \in \tau_0)$ is an extreme ray of the cone $\mathcal{P}^j$ described by the DD pair $(\boldsymbol{B}^j, \boldsymbol{R}^j)$.*

Proof. Let $\boldsymbol{r}$ be one ray of $\boldsymbol{r}_i$ $(i \in \tau_>)$ or $\boldsymbol{r}_h$ $(h \in \tau_0)$, and $\zeta(\boldsymbol{r})$ be the zero set of $\boldsymbol{r}$ associated with $\mathcal{P}^{j-1}$. From the combinatorial extreme ray test (Lemma 2.4) we know that there is no $\boldsymbol{r}' \in \mathcal{P}^{j-1}$ with $\zeta(\boldsymbol{r}') \supseteq \zeta(\boldsymbol{r})$ (other than $\boldsymbol{r}' \simeq \boldsymbol{r}$). Hence, there is no such $\boldsymbol{r}' = \boldsymbol{r}_{i'}$ $(i' \in \tau_> \cup \tau_0)$ with $\zeta'(\boldsymbol{r}') \supseteq \zeta'(\boldsymbol{r})$, where $\zeta'(\boldsymbol{x})$ denotes the zero set of $\boldsymbol{x}$ associated with $\mathcal{P}^j$. This becomes evident since we add at most index j to ζ', that is $\zeta'(\boldsymbol{x}) \supseteq \zeta(\boldsymbol{x})$ and

$$\zeta(\boldsymbol{r}') \not\supseteq \zeta(\boldsymbol{r}) \quad \Longrightarrow \quad \{\, \{j\} \cup \zeta(\boldsymbol{r}') \,\} \not\supseteq \zeta(\boldsymbol{r}) \quad \forall j \notin \zeta(\boldsymbol{r}) \tag{2.59}$$

It remains to show that there is also no $\boldsymbol{r}_{(ik)}$ in $\boldsymbol{R}^j$ with $\zeta'(\boldsymbol{r}_{(ik)}) \supseteq \zeta'(\boldsymbol{r})$. To see this, note that all rays $\boldsymbol{r}_{(ik)}$ are non-negative combinations of rays $\boldsymbol{p}$ and $\boldsymbol{q}$ in $\mathcal{P}^{j-1}$. Observe also that

$$\zeta'(\boldsymbol{r}_{(ik)}) = \big\{\ \{j\} \cup \{\, \zeta(\boldsymbol{p}) \cap \zeta(\boldsymbol{q}) \,\}\ \big\}. \tag{2.60}$$

If $\boldsymbol{r} \not\simeq \boldsymbol{p}$, we have $\zeta(\boldsymbol{r}) \not\subseteq \zeta(\boldsymbol{p}) \cap \zeta(\boldsymbol{q})$, otherwise also $\zeta(\boldsymbol{r}) \subseteq \zeta(\boldsymbol{p})$ and $\boldsymbol{r}$ would not be an extreme ray in $\mathcal{P}^{j-1}$. On the other hand, if $\boldsymbol{r} \simeq \boldsymbol{p}$, and since $\zeta(\boldsymbol{r}) = \zeta(\boldsymbol{p}) \not\subseteq \zeta(\boldsymbol{q})$, also $\zeta(\boldsymbol{r}) \not\subseteq \zeta(\boldsymbol{p}) \cap \zeta(\boldsymbol{q})$. Using the same argument as in eq. (2.59), we get $\zeta'(\boldsymbol{r}) \not\subseteq \zeta'(\boldsymbol{r}_{(ik)})$. □

Next, we restrict our choice of extreme ray pairs $(\boldsymbol{p}, \boldsymbol{q})$ in eq. (2.56) to *adjacent extreme rays*. As we will see, this has the consequence that only extreme rays $\boldsymbol{r}_{(ik)}$ are added to $\boldsymbol{R}^j$, and hence $(\boldsymbol{B}^j, \boldsymbol{R}^j)$ is a *minimal* DD pair.

Definition 2.34. *Let $\boldsymbol{r}$ and $\boldsymbol{r}'$ be two distinct extreme rays of a polyhedral cone $\mathcal{P}$, and let $\mathcal{F}$ be the minimal face of $\mathcal{P}$ containing them. We call $\boldsymbol{r}$ and $\boldsymbol{r}'$* adjacent *if no other extreme ray $\boldsymbol{r}''$ is contained in $\mathcal{F}$, that is*

$$\boldsymbol{r}'' \in \mathcal{F} \quad \Longrightarrow \quad \boldsymbol{r}'' \simeq \boldsymbol{r} \ \text{ or } \ \boldsymbol{r}'' \simeq \boldsymbol{r}' \tag{2.61}$$

Proposition 2.6 (Adjacency Test)**.** *Let $\boldsymbol{r}'$ and $\boldsymbol{r}''$ be two distinct extreme rays of a polyhedral cone $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{x} \geq \boldsymbol{0} \,\}$, and let $\zeta = \zeta(\boldsymbol{r}') \cap \zeta(\boldsymbol{r}'')$ be the intersection of the zero sets associated with $\boldsymbol{r}'$ and $\boldsymbol{r}''$. Then, the following statements are equivalent:*

(a) *$\boldsymbol{r}'$ and $\boldsymbol{r}''$ are adjacent*

(b) $\mathbf{rank} \left[\begin{smallmatrix} \boldsymbol{A} \\ \boldsymbol{I}_{\zeta\star} \end{smallmatrix}\right] = d - 2$

(c) $\mathbf{rank}\ \boldsymbol{A}_{\star\overline{\zeta}} = \big|\,\overline{\zeta}\,\big| - 2$

(d) $\zeta(\boldsymbol{r}''') \supset \zeta \quad \Longrightarrow \quad \boldsymbol{r}''' \simeq \boldsymbol{r}' \ \text{ or } \ \boldsymbol{r}''' \simeq \boldsymbol{r}''$ *for any ray $\boldsymbol{r}''' \in \mathcal{P}$.*

Proof. The equivalence of (b) and (c) follows with the same argumentation as in the proof of Prop. 2.3.
If (a) holds, we have a face $\mathcal{F}$ containing $\boldsymbol{r}'$ and $\boldsymbol{r}''$ but no other extreme ray. Hence, every ray in $\mathcal{F}$ is a nonnegative combination of $\boldsymbol{r}'$ and $\boldsymbol{r}''$, and since they are distinct and linearly independent, $\dim \mathcal{F} = 2$. Furthermore, for every ray $\boldsymbol{r} \in \mathcal{F}$, we have $i \in \zeta \Rightarrow r_i = 0$, and $\mathcal{S} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{I}_{\zeta\star}\,\boldsymbol{x} = \boldsymbol{0} \,\}$ is the minimal subspace containing $\mathcal{F}$. Note that $\mathcal{S}$ is the nullspace of the matrix $\boldsymbol{Z} = \left[\begin{smallmatrix} \boldsymbol{A} \\ \boldsymbol{I}_{\zeta\star} \end{smallmatrix}\right]$, that is, $\mathbf{dim}\ \mathcal{S} = d - \mathbf{rank}\ \boldsymbol{Z}$, and hence $\mathbf{rank}\ \boldsymbol{Z} = d - 2$. This proves (a) ⇒ (b). On the other hand, if (b) holds, $\dim \mathcal{S} = 2$, and since both extreme rays $\boldsymbol{r}'$ and $\boldsymbol{r}''$ are contained in $\mathcal{S}$, also $\mathcal{F} = \mathcal{S} \cap \mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{I}_{\zeta\star}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{I}_{\overline{\zeta}\star}\,\boldsymbol{x} \geq \boldsymbol{0} \,\}$ has dimension 2. Consequently, no other extreme ray can be contained in $\mathcal{F}$, and hence $\boldsymbol{r}'$ is adjacent to $\boldsymbol{r}''$.
To prove (a)⇔(d), assume that (a) holds. If there exists an extreme ray $\boldsymbol{r}'''$ with $\zeta(\boldsymbol{r}''') \supset \zeta$, then it also lies in the face containing $\boldsymbol{r}'$ and $\boldsymbol{r}''$, since $\mathcal{F} = \mathcal{S} \cap \mathcal{P}$, and clearly $\boldsymbol{r}''' \in \mathcal{P}$ and $\boldsymbol{r}''' \in \mathcal{S} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{I}_{\zeta\star}\,\boldsymbol{x} = \boldsymbol{0} \,\}$. If the extreme ray $\boldsymbol{r}''' \not\simeq \boldsymbol{r}'$ and $\boldsymbol{r}''' \not\simeq \boldsymbol{r}''$, $\boldsymbol{r}'$ and $\boldsymbol{r}''$ are not adjacent, which contradicts our assumption (a). On the other hand, if (d) holds, no extreme ray $\boldsymbol{r}'''$ exists with $\zeta(\boldsymbol{r}''') \supset \zeta$ distinct from $\boldsymbol{r}'$ and $\boldsymbol{r}''$. Since all rays lying in $\mathcal{F}$ also lie in $\mathcal{S} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{I}_{\zeta\star}\,\boldsymbol{x} = \boldsymbol{0} \,\}$, we have no third extreme ray in $\mathcal{F}$, yielding that $\boldsymbol{r}'$ and $\boldsymbol{r}''$ are adjacent. □

Corollary 2.2. *Let $\boldsymbol{r}'$ and $\boldsymbol{r}''$ be two adjacent extreme rays of a polyhedral cone $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{x} \geq \boldsymbol{0} \,\}$. Then, the set $\zeta = \zeta(\boldsymbol{r}') \cap \zeta(\boldsymbol{r}'')$ contains at least $d - 2 - \mathbf{rank}\ \boldsymbol{A}$ indices.*

Proof. Follows immediately from Prop. 2.6b. □

Finally, we prove that rays $\boldsymbol{r}_{(ik)}$ as defined in eq. (2.56) are extreme rays if and only if the ancestor rays $\boldsymbol{p}$ and $\boldsymbol{q}$ are adjacent.

Proposition 2.7. *Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be two extreme rays of the polyhedral cone $\mathcal{P}^{j-1} = (\boldsymbol{B}^{j-1}, \boldsymbol{R}^{j-1})$ used to generate a new ray $\boldsymbol{r}_{(ik)}$ as defined in eq. (2.56). Then, the new ray $\boldsymbol{r}_{(ik)}$ is an extreme ray of the cone $\mathcal{P}^{j} = (\boldsymbol{B}^{j}, \boldsymbol{R}^{j})$ iff $\boldsymbol{p}$ is adjacent to $\boldsymbol{q}$ in $\mathcal{P}^{j-1}$.*

Proof. Consider the matrix $\boldsymbol{C} = \left[\begin{smallmatrix} \boldsymbol{A} \\ \boldsymbol{I} \end{smallmatrix}\right]$ and let us denote by $\boldsymbol{C}_\sigma$ the matrix $\left[\begin{smallmatrix} \boldsymbol{A} \\ \boldsymbol{I}_{\sigma\star} \end{smallmatrix}\right]$ containing only the rows $\sigma \subseteq \{1, \ldots, d\}$ of the identity matrix in $\boldsymbol{C}$. Note that for any ray $\boldsymbol{r}'$ of the cone, $\boldsymbol{C}\,\boldsymbol{r}' = \left[\begin{smallmatrix} \boldsymbol{0} \\ \boldsymbol{r}' \end{smallmatrix}\right]$ since $\boldsymbol{r}'$ lies in the nullspace of $\boldsymbol{A}$. For the zero set $\zeta' = \zeta(\boldsymbol{r}')$, we have $\boldsymbol{C}_{\zeta'}\,\boldsymbol{r}' = \boldsymbol{0}$. Note also that for any row $k \in \{1, \ldots, d\} \setminus \zeta'$, we get $\boldsymbol{C}_{\zeta' \cup \{k\}}\,\boldsymbol{r}' = \left[\begin{smallmatrix} \boldsymbol{0} \\ r'_k \end{smallmatrix}\right]$, that is, the row $\boldsymbol{I}_{k\star}$ is independent in $\boldsymbol{C}_{\zeta' \cup \{k\}}$.

Let now $\boldsymbol{r}' = \boldsymbol{r}_{(ik)}$ be a ray of the cone $\mathcal{P}^j$ constructed from two adjacent rays $\boldsymbol{p}$ and $\boldsymbol{q}$ from $\mathcal{P}^{j-1}$ as stated in eq. (2.56). Since $\boldsymbol{p}$ and $\boldsymbol{q}$ are adjacent, $\mathbf{rank}\ \boldsymbol{C}_\zeta = d-2$ for $\zeta = \zeta(\boldsymbol{p}) \cap \zeta(\boldsymbol{q})$. We also know that $r'_j = 0$ and $j \notin \zeta$ since $p_j > 0$ and $q_j < 0$. Hence, row $\boldsymbol{I}_{j\star}$ is independent in $\boldsymbol{C}_{\zeta \cup \{j\}}$, and consequently $\mathbf{rank}\ \boldsymbol{C}_{\zeta'} > d - 1$ for $\zeta' = \zeta(\boldsymbol{r}')$. Since j is the only element added to ζ' compared to ζ, we have $\mathbf{rank}\ \boldsymbol{C}_{\zeta'} = d - 1$, and $\boldsymbol{r}'$ is an extreme ray of $\mathcal{P}^j$.

On the other hand, assume that $\boldsymbol{r}' = \boldsymbol{r}_{(ik)}$ is an extreme ray of $\mathcal{P}^j$, hence $\mathbf{rank}\ \boldsymbol{C}_{\zeta'} = d-1$ for $\zeta' = \zeta(\boldsymbol{r}')$. Since $\boldsymbol{p}$ and $\boldsymbol{q}$, from which $\boldsymbol{r}'$ has been constructed, are extreme rays of $\mathcal{P}^{j-1}$, we have $\mathbf{rank}\ \boldsymbol{C}_{\zeta_p} = \mathbf{rank}\ \boldsymbol{C}_{\zeta_q} = d - 1$ for $\zeta_p = \zeta(\boldsymbol{p})$ and $\zeta_q = \zeta(\boldsymbol{q})$. Furthermore, we know that $\zeta' = \zeta \cup \{j\}$ with $\zeta = \zeta_p \cap \zeta_q$, and since $j \notin \zeta_p$ and $j \notin \zeta_q$, also $j \notin \zeta$, and the row $\boldsymbol{I}_{j\star}$ is independent in $\boldsymbol{C}_{\zeta'}$. Removing it reduces the rank of the matrix by one, and we have $\mathbf{rank}\ \boldsymbol{C}_\zeta = d - 2$, hence $\boldsymbol{p}$ and $\boldsymbol{q}$ are adjacent in $\mathcal{P}^{j-1}$. □

The following procedure (Alg. 1: `DDStandard`) in pseudo code outlines a standard implementation of the double description algorithm, and it summarizes the key properties of the method that were derived above.

2.3.3 On Complexity & Alternatives

There are two approaches to express the complexity of *extreme ray*, *vertex* or *facet enumeration algorithms*. Here, we concentrate on *extreme ray enumeration*, measuring the computational effort for solutions of a problem in d dimensions given by m inequalities yielding n extreme rays. One popular measure expresses the worst case computation time for fixed d in terms of the problem size, i.e. the size of the input data. For this *worst-case input measure*, an optimal algorithm is known, running in time $\mathcal{O}(m^{\lfloor \frac{d}{2} \rfloor})$ for any fixed $d \geq 4$ (Chazelle, 1993). It cannot be better since the largest output is of the same order by the upper bound theorem (McMullen (1970), see also Section 2.4.4). Even if this result is a remarkable theoretical achievement, it is not very useful in practice. The output size n varies largely depending on the concrete problem instance, and the upper bound almost never occurs in problems of biological/practical interest. Hence, we prefer an *output sensitive* measure that also includes n, or more generally, the size

Algorithm 1: DDStandard($\mathbf{A}$)

desc : Standard implementation of double description method
input : Matrix $\boldsymbol{A}$, defining a polyhedral cone $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{x} \geq \boldsymbol{0} \,\}$
output: Matrix $\boldsymbol{R}$, extreme rays of $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x} = \boldsymbol{R}\boldsymbol{c}\ ,\ \boldsymbol{c} \geq \boldsymbol{0} \,\}$
begin
1 $\quad \boldsymbol{R} \leftarrow \boldsymbol{K}$ `/* row-echelon kernel matrix, basis for` $\mathcal{N}(\boldsymbol{A}) = \{\, \boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0} \,\}$ `*/`
2 $\quad \rho \leftarrow \ldots$ `/* indices of already considered non-negativity constraints */`
3 $\quad$ **while** $(\rho \neq \{1, 2, \cdots, d\})$ **do**
4 $\quad\quad j \leftarrow$ choose from $\{1, 2, \cdots, d\} \setminus \rho$
5 $\quad\quad \tau_> \leftarrow \{i \mid R_{j,i} > 0\}$
6 $\quad\quad \tau_0 \leftarrow \{h \mid R_{j,h} = 0\}$
7 $\quad\quad \tau_< \leftarrow \{k \mid R_{j,k} < 0\}$
8 $\quad\quad \tau_{adj} \leftarrow \{(i,k) \mid (i,k) \in (\tau_> \times \tau_<) : \boldsymbol{R}_{\star i}$ is adjacent to $\boldsymbol{R}_{\star k}\}$
9 $\quad\quad \boldsymbol{R}_{new} \leftarrow [\,]$ `/* empty matrix to store new rays */`
10 $\quad\quad$ **foreach** $(i,k) \in \tau_{adj}$ **do**
11 $\quad\quad\quad \boldsymbol{p} \leftarrow \boldsymbol{R}_{\star i}$
12 $\quad\quad\quad \boldsymbol{q} \leftarrow \boldsymbol{R}_{\star k}$
13 $\quad\quad\quad \boldsymbol{r}_{(ik)} \leftarrow p_j\boldsymbol{q} - q_j\boldsymbol{p}$
14 $\quad\quad\quad$ append column $\boldsymbol{r}_{(ik)}$ to $\boldsymbol{R}_{new}$
15 $\quad\quad$ **end**
16 $\quad\quad \boldsymbol{R} \leftarrow \big[\, [\boldsymbol{R}_{\star i}]\ ,\ [\boldsymbol{R}_{\star h}]\ ,\ \boldsymbol{R}_{new}\, \big]$ $\quad$ with $i \in \tau_>$ and $h \in \tau_0$
17 $\quad\quad \rho \leftarrow \rho \cup \{j\}$
18 $\quad$ **end**
end

of the input *and* output data. In particular, if an algorithm runs in *polynomial total time*, it has a running time that is polynomial in the input and output size (here in m, d and n).

The total time complexity of extreme ray enumeration, or generally of *representation conversion* of polyhedral cones, is an open question in computational geometry. Efficient algorithms are known for special cases: for non-degenerate problems, an algorithm with total time complexity $\mathcal{O}(mdn)$ has been introduced by Avis and Fukuda (1991). The algorithm has *space complexity* $\mathcal{O}(md)$ not depending on the output size—a property called *compact*. Non-degeneracy means for vertex enumeration that no point lies in more than d facets, and for facet enumeration, no $d + 1$ points lie in a common facet. Unfortunately, many problems arising in practical applications are degenerate, in particular problems that occur in systems biology. Another special case involves polyhedra associated with network LP's (Provan, 1994). Network LP's are special linear programming instances associated with a loopless graph, vertices are associated with supplies/demands, and edges with costs and lower/upper bounds.

Unfortunately, no polynomial algorithm is known for the general case. In Avis and Bremner (1995), existing algorithms are classified into three groups: algorithms based on pivoting, on triangulation and insertion algorithms. For instance, the *reverse search algorithm* (Avis and Fukuda, 1991) for non-degenerate cases is a pivoting algorithm, whereas the *double description method* belongs to the class of insertion algorithms. Avis and Bremner (1995) construct nasty polytopes for each category and show that all algorithms of these classes have exponential running time in the worst case. Consequently, an efficient algorithm must be based on a completely new idea, or it combines two or more existing techniques and adapts to the favorable variant depending on the problem instance.

Another indicator for the hardness of extreme ray enumeration has been presented recently in Khachiyan et al. (2006). With a sophisticated construction, they prove NP-completeness for the problem of deciding whether a still unknown negative cycle exists in a weighted graph, given some cycles that are already known. From this decision problem, they conclude that negative cycle enumeration is NP-hard and show that enumerating negative cycles is equivalent to enumeration of the vertices of a polyhedron. What matters is that they enumerate vertices of an unbounded polyhedron, ommitting the extreme rays. Hence, the complexity for enumerating vertices of a (bounded) polytope, or equivalently, extreme rays of a polyhedral cone, remains unknown.

The complexity of the double description method is poorly understood. The difficulty in the analysis has two main sources: first, the number of intermediary rays during the computation can grow exponentially compared to the final number of extreme rays. Second, the algorithm is extremely sensitive to the insertion order of the halfspaces. Avis and Bremner (1995) study insertion orderings performing well for certain polytopes, but failing badly for others. The average-case complexity of the double description method is studied thoroughly in Borgwardt (2007) for non-degenerate cases using various random distributions. They conclude that under their stochastic model and average-case analysis, it cannot be as efficient as a pivoting algorithm studied elsewhere. However, for our purposes, random cases are not of great importance, and furthermore, the double description method has proven efficient *in particular* for degenerate problems.

In Acuña et al. (2009), some interesting problems have been addressed that are related to extreme ray enumeration, and even more closely to *elementary modes* (EMs) and *minimal cut sets* (MCSs). Different problems are investigated, starting with finding *one* EM or MCS, and they show that this can easily be solved with linear programming (LP). Then, they focus on specific EMs and MCSs, for instance proving that it is NP-complete to decide whether an EM with a certain support exists. The proof is a nice reduction from Directed Hamiltonian Circuit. They also show that finding an EM with k non-zero entries is already NP-complete, using a reduction from 3-Dimensional Matching (for $k \geq 2$, and k smaller than the upper bound resulting from the rank condition, see Corollary 2.1). Furthermore, they address the problem of *counting elementary modes*, proving that this is ♯P-complete, with a reduction from Counting Perfect Matchings in a Bipartite Graph. Still, the enumeration problem remains open, with the exception that enumerating EMs in a network is polynomial in the output size if all reactions are reversible (Acuña et al., 2009).

2.4 The Flux Cone

Different names have been used for similar polyhedral cones arising from constraints applied to biological reaction networks. For most problems—and for all problems discussed here—quasi steady state is assumed (see eq. (2.1)), leading to m equality constraints for a stoichiometric matrix $\boldsymbol{S} \in \mathbb{R}^{m \times r}$ with m metabolites and r reactions:

$$\boldsymbol{S}\,\boldsymbol{v} = \boldsymbol{0} \tag{2.62}$$

Some authors distinguish between internal (balanced) metabolites and external (unbalanced) metabolites, but we circumvent this problem by adding an excretion (or uptake) reaction to external metabolites.

The second constraint type is based on thermodynamic restrictions for the reaction rates $\boldsymbol{v}$, and in the most general case, we simply assume that all reactions are irreversible. Note that any reversible reaction can be split into two irreversible reactions, which is exactly performed if *elementary modes* are computed (see Section 2.4.3):

$$\boldsymbol{v} \geq \boldsymbol{0} \tag{2.63}$$

In a more general setting, reaction upper and lower bounds can be set for the reactions, due to physical limits or deduced from measurements:

$$\boldsymbol{v}_{\min} \leq \boldsymbol{v} \leq \boldsymbol{v}_{\max} \tag{2.64}$$

If boundary constraints (other than zero) are added, the solution space is a polyhedron rather than a polyhedral cone. To compute the generators of this space, we can use the transformation described in Section 2.3.1.

In the following sections, we introduce flux cone versions that are most commonly used in the literature. Wagner and Urbanczik (2005) investigate the nature of the flux cone and summarize the terminology also used here. The smallest generator set are the minimal generators described in Section 2.4.1, where reversible reactions are not constrained and hence corresponding flux values can be positive or negative, i.e. those reactions are not split into a forward and backward part. All other approaches split certain or all reversible reactions into a forward and backward irreversible reaction, and the generators are computed in this *augmented flux space* of higher dimension. After computation, they are projected back into the original reaction space, where the generators are not minimal. Therefore, these generator sets are always a superset of the minimal generators, and tests to distinguish them from minimal generators can be developed, usually based on elementarity tests in the original and augmented reaction dimensionality space. Such a test is also proposed by Jevremovic et al. (2008), where extreme pathways are distinguished from elementary modes.

2.4.1 Minimal Generators

As mentioned in the previous section, *minimal generators* are the smallest possible set of generators for the flux cone. For all irreversible reactions, we add a non-negativity (or non-positivity) constraint; reversible reactions are left unconstrained.

Definition 2.35. *Let $\boldsymbol{S} \in \mathbb{R}^{m\times r}$ be the stoichiometric matrix for a network with m metabolites and r reactions, and let us denote by I the set of reaction indices corresponding to irreversible reactions, i.e. we have $I \subseteq \{1,\dots,r\}$. Then, the* flux cone $\mathcal{P}^{(\mathrm{flux})}$ *is defined by the following constraints:*

$$\begin{aligned} \boldsymbol{S}\,\boldsymbol{v} &= \boldsymbol{0} \\ v_i &\geq 0 \quad \forall\, i \in I \end{aligned} \tag{2.65}$$

Backward irreversible reactions have been reverted for simplicity. Note that the flux cone is not necessarily pointed, that is, it might also contain lines (see Def. 2.24 and Lemma 2.2). To generate an unpointed cone, we also need the linealities (see Def. 2.23).

Definition 2.36. *The extreme rays and the linealities of the flux cone are called* minimal generators *of $\mathcal{P}^{(\mathrm{flux})}$. Minimal generators of $\mathcal{P}^{(\mathrm{flux})}$ are sometimes also called* convex basis *of $\mathcal{P}^{(\mathrm{flux})}$.*

2.4.2 Extreme Pathways

To compute *extreme pathways*, all *internal reversible reactions* are split into a forward and backward part. The dimension of the *augmented flux space* is incremented by the number of internal reversible reactions. The original flux cone is projected to the augmented space, and we define the cone $\mathcal{P}^{(\mathrm{EP})}$ in this space as follows:

Definition 2.37. *Let us consider a metabolic network with stoichiometric matrix $\boldsymbol{S} \in \mathbb{R}^{m\times r}$ and index set I for irreversible reactions as specified for the flux cone (Def. 2.35), and let E be the indices associated with exchange reactions, i.e. columns in $\boldsymbol{S}$ with only non-negative or only non-positive entries. Let furthermore $R = \{i \mid i \notin (I \cup E)\}$ be the indices of internal reversible reactions. Then, the* EP cone $\mathcal{P}^{(\mathrm{EP})}$ *is defined by the following constraints:*

$$\begin{aligned} [\,\boldsymbol{S}\ ,\ -\boldsymbol{S}_{\star R}\,]\,\boldsymbol{v}' &= \boldsymbol{0} \\ v'_i &\geq 0 \quad \forall\, i \in I \\ v'_j &\geq 0 \quad \forall\, j \in R \\ v'_{r+j} &\geq 0 \quad \forall\, j \in R \end{aligned} \tag{2.66}$$

Also the EP cone is not necessarily pointed, and linealities might be needed to generate the cone. However, if reversible exchange reactions are linearly independent—which is for instance the case if each exchanges a different metabolite—the EP cone is pointed.

To get extreme pathways, linealities and extreme rays of the EP cone are projected back to the flux cone. Split reactions are recombined, negating the flux value if it corresponds to the backward direction of the reaction:

$$\begin{aligned} v_i &= v'_i && \forall\, i \notin R \\ v_j &= v'_j - v'_{r+j} && \forall\, j \in R \end{aligned} \tag{2.67}$$

We get a *futile cycle* for every split reaction, containing nonzero values only for the forward and backward part. Futile cycles have no biological meaning, and the projection back according to eq. (2.67) leads to a zero vector. Futile cycles are therefore eliminated during the back projection.

Definition 2.38. *The extreme rays and linealities of the EP cone $\mathcal{P}^{(\mathrm{EP})}$, projected back to the flux cone $\mathcal{P}^{(\mathrm{flux})}$ and eliminating the futile cycles described above, are called* extreme pathways (EPs) *of the flux cone $\mathcal{P}^{(\mathrm{flux})}$. EPs associated with linealities of the EP cone are sometimes called* reversible extreme pathways.

2.4.3 Elementary Modes

Definition 2.39. *Let us consider a metabolic network with stoichiometric matrix $\boldsymbol{S} \in \mathbb{R}^{m\times r}$ and index set I for irreversible reactions as specified for the flux cone (Def. 2.35). Then, the* EM cone $\mathcal{P}^{(\mathrm{EM})}$ *is defined by the following constraints:*

$$\begin{array}{rcl} [\ \boldsymbol{S}\ ,\ -\boldsymbol{S}_{\star\bar{I}}\ ]\,\boldsymbol{v}'' & = & \boldsymbol{0} \\ \boldsymbol{v}'' & \geq & \boldsymbol{0} \end{array} \tag{2.68}$$

Note that the EM cone is always pointed since $\boldsymbol{B}$ in Lemma 2.1 is the identity matrix resulting from the nonnegativity constraints in eq. (2.68). To get elementary modes, rays of the EM cone are projected back to the flux cone, again remerging the split reactions. Futile cycles containing nonzero values only for the two parts of split reactions are eliminated.

$$\begin{array}{rcll} v_i & = & v_i'' & \forall\, i \in I \\ v_j & = & v_j'' - v_{r+j}'' & \forall\, j \notin I \end{array} \tag{2.69}$$

Definition 2.40. *The extreme rays of the EM cone $\mathcal{P}^{(\mathrm{EM})}$, projected back to the flux cone $\mathcal{P}^{(\mathrm{flux})}$ according to eq. (2.69) eliminating the futile cycles described above, are called* elementary modes (EMs) *of the flux cone $\mathcal{P}^{(\mathrm{flux})}$.*

2.4.4 Upper Bound Theorem

The *upper bound theorem* of McMullen (1970) is a main achievement in the theory of convex polytopes. It defines upper bounds for the number of facets in a d-dimensional polytope, achieved for instance by the *cyclic polytope*, which was also used for the proof. We omit details of the proof and are also not explaining the cyclic polytope here, but simply state the theorem in its dual form for an upper bound of extreme rays:

Theorem 2.4 (Upper Bound Theorem for Extreme Rays)**.** *Let $f_0(\mathcal{P})$ be the number of extreme rays of a polyhedron $\mathcal{P} \subseteq \mathbb{R}^d$ with n facets. Then,*

$$f_0(\mathcal{P}) \leq \binom{n - \lfloor \frac{d+1}{2} \rfloor}{n-d} + \binom{n - \lfloor \frac{d+2}{2} \rfloor}{n-d} \tag{2.70}$$

We are interested in the application of the theorem to the flux cone to estimate the maximum number of *minimal generators*, *extreme pathways* (EPs) and *elementary modes* (EMs). Before we apply Theorem 2.4, we remove the equalities from the flux cone definition using the transformation described in Prop. 2.1.

Let us assume that the flux cone is specified by the stoichiometric matrix $\boldsymbol{S} \in \mathbb{R}^{m\times r}$, i.e. the network has m metabolites and r reactions. Furthermore, let $I \subseteq \{1,\ldots,r\}$ be the index set

associated with irreversible reactions, and $D \subseteq \overline{I}$ duplicated irreversible reactions, i.e. those reactions that are split when the flux cone is projected to a higher dimensional space. After projection, we have the following constraints:

$$\begin{aligned} [\, \boldsymbol{S} \;,\; -\boldsymbol{S}_{\star D} \,]\, \boldsymbol{v}' &= \mathbf{0} \\ v'_i &\geq 0 \qquad \forall\, i \in I \\ v'_j &\geq 0 \qquad \forall\, j \in D \\ v'_{r+j} &\geq 0 \qquad \forall\, j \in D \end{aligned} \tag{2.71}$$

For simplicity, we assume that the cone defined by eq. (2.71) is pointed. For an unpointed cone, we compute the linealities and add them as additional equality constraints. Furthermore, we assume that $\boldsymbol{S}' := [\boldsymbol{S}\;,\; -\boldsymbol{S}_{\star D}]$ has full row rank m, since we could simply remove dependent rows without changing the cone. Now, we apply the transformation described in Prop. 2.1, yielding a transformed cone solely defined by inequalities. Note the following terminology analogies from Prop. 2.1 to the current notation:

$$\begin{aligned} \boldsymbol{A} \in \mathbb{R}^{a\times d} &= \boldsymbol{S}' \in \mathbb{R}^{m\times(r+|D|)} \\ \boldsymbol{B} \in \mathbb{R}^{b\times d} &= \boldsymbol{I}_{\rho\star} \in \mathbb{R}^{(|I|+2\,|D|)\times(r+|D|)} \\ \rho &= I \cup D \cup \{r+j \mid j \in D\} \\ d &= r + |D| \\ |\tau| &= m \\ |\overline{\tau}| &= r + |D| - m \end{aligned}$$

The transformed cone $\mathcal{P}' \subseteq \mathbb{R}^{d'}$ with $d' := |\overline{\tau}| = r+|D|-m$ is defined by $n' := b = |I|+2\,|D|$ inequality constraints. Note that the true dimension of $\mathcal{P}'$ might be smaller than d'. In this case, we find at least one hyperplane containing $\mathcal{P}'$ using linear programming.[1] Hence we can append the additional hyperplane equations to the definition of the cone and reapply the above transformation. In the following, we assume that $\mathcal{P}'$ is a d'-polytope. Inserting into eq. (2.70), we get

$$f_0(\mathcal{P}) \leq \binom{|I|+2\,|D| - \lfloor\frac{r+|D|-m+1}{2}\rfloor}{|I|+|D|-r+m} + \binom{|I|+2\,|D| - \lfloor\frac{r+|D|-m+2}{2}\rfloor}{|I|+|D|-r+m} \tag{2.72}$$

$$\leq 2\binom{|I|+2\,|D| - \lfloor\frac{r+|D|-m}{2}\rfloor}{|I|+|D|-r+m}. \tag{2.73}$$

In particular, for *minimal generators*, no reactions are split, i.e. we have $|D| = 0$ and hence

$$f_0(\mathcal{P}) \leq 2\binom{|I| - \lfloor\frac{r-m}{2}\rfloor}{|I|-(r-m)}. \tag{2.74}$$

[1] We bound the cone, e.g. by constraining the variables to $[-1, 1]$. Find a first feasible point by maximizing and minimizing along any objective direction. Then, optimize for a direction orthogonal to the ray defined by the first point, again minimizing and maximizing in the bounded cone. Each new ray further restricts the search directions to the space orthogonal to the found rays. Hence, we find $i \leq d'$ linearly independent points by solving at most $2d'$ LPs. If $i < d'$, we get $d' - i$ hyperplane equations if we compute a basis for the left nullspace of the matrix consisting of the found points.

For elementary modes, all reversible reactions are split. We have $|D| = r - |I|$, yielding the following upper bound:

$$f_0(\mathcal{P}) \leq 2 \binom{r - \lfloor \frac{|I|-m}{2} \rfloor}{m}. \tag{2.75}$$

We conclude with the last two special cases: the worst case, yielding the largest generating set, occurs for elementary modes if all reactions are reversible:

$$f_0(\mathcal{P}) \leq 2 \binom{r + \lceil \frac{m}{2} \rceil}{m}. \tag{2.76}$$

For this case, Klamt and Stelling (2002) provide the upper bound $2 \cdot \binom{r}{m+1}$. Their bound is tighter than our bound (2.76) in almost all cases.

If all reactions are irreversible, minimal generators, EPs and EMs coincide, and we get

$$f_0(\mathcal{P}) \leq 2 \binom{\lceil \frac{r+m}{2} \rceil}{m}. \tag{2.77}$$

3

Pre–/Postprocessing & Data Structures

In this chapter, we review and discuss issues and processing steps that are not part of the main algorithm. Still, they are crucial for the computation, for instance because also the fastest implementation of the double description method is almost worthless if the constraints are sorted unfavorably. Furthermore, in particular for biological applications, the input structures contain a lot of redundant or even inconsistent information. Preprocessing steps are necessary to compress the networks and remove unneeded elements. The first section addresses this issue, and we review and summarize constraint sorting strategies in the second section. We also review the binary approach by Gagneur and Klamt (2004) reducing memory demands significantly. Parts of the intermediary rays can be stored in binary form, hence this choice has a direct influence on the internal data structures. Finally, we discuss different number data types, because double precision arithmetic is not sufficient for certain difficult problem cases. The impact of the decision to support different number types is huge, since it affects most data structures throughout the whole computation procedure. We illustrate our approach to deal with multiple data types in the last section, where we present our generic implementation of simple linear algebra functions.

3.1 Model Consistency & Compression

Parts of this section have been published in Terzer et al. (2009).

An important step in model building is determining the consistency of the network, meaning technical consistency without considering experimental data or biological interpretation. For elementary mode computation, it is also crucial to remove model redundancies before the main loop. This usually reduces the size of data structures and demanded memory, but also affects performance since operations on smaller structures are faster, and compacted stoichiometric matrices typically lead to fewer iteration steps.

The simplest consistency and compression method is mainly based on the kernel matrix (Section 3.1.1), meaning that resulting restrictions are a consequence of the quasi steady state assumption (eq. 2.2). In Section 3.1.2, linear programming is used to check the feasibility of reactions using additional constraints like reaction irreversibilities or lower and upper bounds for flux values. A third approach operates on the reaction graph and eliminates or combines graph elements if possible (Section 3.1.3). Our current implementation iteratively applies the approaches based on nullspace analysis together with graph consistency methods. Computing a nullspace basis matrix is somewhat time-consuming especially if exact arithmetic is used, but we can detect coupled reactions that are not directly connected in the hypergraph. Graph based methods on the other hand are fast and very efficient for connected elements in the graph, but they miss correlations of elements that are far apart. We stop the procedure if both methods fail to find further compressible elements. We are not using LP based consistency methods since they are relatively expensive, and we could not observe significantly improved compression.

Related techniques have been proposed by other authors, for instance based on the analysis of variability and coupling of metabolic fluxes (Burgard et al., 2004; Mahadevan and Schilling, 2003). Another method focuses on stoichiometric inconsistencies and was recently described (Gevorgyan et al., 2008). Good overviews of compression techniques are also given in Gagneur and Klamt (2004) and in the Appendix B of Urbanczik and Wagner (2005).

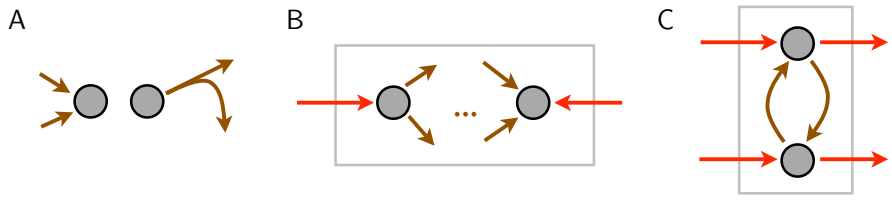


Figure 3.1: Network inconsistencies and compression. **A:** Inconsistency due to dead-end metabolites and reactions. **B:** Inconsistent reaction coupling with coupling factor -1. At least one of the fluxes through the red exchange reactions should be negative due to the coupling factor, but both reactions are irreversible. **C:** A subnetwork with $r = 6$ reactions and $m = 2$ metabolites. The internal cycle and each input/output combination is an elementary mode (EM), hence the subnetwork contains 5 EMs. Replacing the subnetwork with the EMs reduces r by one and m by two.

3.1.1 Nullspace Analysis

Analyzing the kernel matrix $\boldsymbol{K}$ is based on the metabolite balancing equation (2.2) to reduce the flux space to the nullspace of the stoichiometric matrix $\boldsymbol{S}$. Any valid flux vector is a linear combination of column vectors of $\boldsymbol{K}$ (eq. 2.3), enabling the identification of key reaction properties as those shown in the box below.

Results of Nullspace Analysis

1. If the kernel matrix contains a zero-row, the corresponding reaction cannot carry a (non-zero) flux. We can remove this reaction for all analysis employing the steady state assumption.
2. If two matrix rows differ only by a constant factor, the two reactions are coupled, that is, the flux through one reaction is always a multiple of the flux through the other reaction; consequently either both reactions are active or both are passive. Such reactions are presumably co-regulated (Notebaart et al., 2008).
3. Also using information about reaction reversibilities, we can detect *inconsistent reaction coupling*. For example, two coupled forward-only reactions with a negative coupling factor cannot carry a non-zero flux without violating an irreversibility constraint, since one reaction would have to operate in backward mode.

For elementary mode computation, all inconsistent model parts (items 1 and 3 in the box) are removed. Coupled reactions (item 2) can be combined into a single reaction, using the constant *coupling factor* to weight the reactions in the composite reaction. Examples for *zero flux* reactions and *inconsistent reaction coupling* are shown in Fig. 3.1.

3.1.2 Feasibility Analysis using LP

Linear programming (LP) is a more powerful tool to analyze infeasibility of reactions, since inequality constraints are also considered. Reaction irreversibilities ($v \geq 0$ or $v \leq 0$) and flux bounds ($v_{\min} \leq v \leq v_{\max}$) are added before determining the minimal and maximal flux value for each reaction:

Results of Feasibility Analysis using LP

1. Minimize and maximize the flux value for each reaction.
   a) If min *and* max value are zero, the reaction is a *zero flux reaction*, that is, it cannot have a flux value other than zero. It can be removed if no model corrections are made, without affecting the outcome of subsequent simulations.
   b) If min *or* max value is zero and the reaction is reversible, we have an *unsatisfied reversibility*. Either, the reversibility constraint is too lax, or another component is missing, disabling the operation in one direction. Tightening this constraint might lead to better simulation performance.
   c) If the minimal and maximal values are non-zero and have equal sign, the reaction is essential. Deletion of the reaction, for example, by gene knockout, is predicted to be lethal.
2. For reactions not of type (1a) or (1c), set the bounds to zero. If biomass cannot be produced, the reaction is essential. Again, reaction removal is associated with lethality.

Note that *zero flux reactions* (item 1a in the box) detected by linear programming are a superset of those detected by nullspace analysis (Section 3.1.1, box item 1). Again, we can remove zero flux reactions safely before computing elementary modes. *Unsatisfied reversibilities* (item 1b) can lead to tighter constraints, and possibly to faster computations. For elementary modes, all reversible reactions are split into a forward and backward part (see Section 2.4.3), and fewer reversible reactions lead to a lower dimensionality of the enhanced flux space – and hence almost surely to faster computations. However, if minimal generators are computed, where nonnegativity constraints are only added for irreversible reactions, tightening reversibilities can also have a negative influence on performance.

3.1.3 Graph Consistency

Instead of analyzing the kernel matrix or running (linear) optimizations to find inconsistencies and couplings, we can also examine the structure of the directed hypergraph associated with the reaction network. Here, we are only interested in the connectedness of the components, stoichiometric coefficients are only important for reaction merging operations. Consequently, we define the hypergraph G as follows:

Definition 3.1. *Let $\boldsymbol{S} \in \mathbb{R}^{m \times r}$ be a stoichiometric matrix associated with m metabolites and r reactions, all reactions assumed irreversible. Then, the $G = (V, H)$ defines the hypergraph*

associated with $\boldsymbol{S}$*, where* V *denotes the set of vertices and* H *the set of directed hyperedges. Each vertex* $v \in V$ *is associated with a metabolite, e.g.* $V = \{v_1, \ldots, v_m\}$*, and a hyperedge* $h \in H$ *points from educt to product metabolites, that is,* $h = (V_e, V_p)$, $V_e \subseteq V$, $V_p \subseteq V$, $V_e \cap V_p = \emptyset$*.*

Note that educt and product sets of a hyperedge can be empty; in particular, an uptake reaction has an empty educt set ($V_e = \emptyset$), and an excretion reaction an empty product set ($V_p = \emptyset$). We also allow edges with emtpy educt and product sets, since they might occur as a consequence of compression operations.

Let us now illustrate the concepts based on two simple compression examples: if a network node has only incoming (outgoing) edges, it can obviously not be consumed (produced) and all edges and the node can be removed. As a second example, consider a node that has only one incoming and one outgoing edge, like in a linear pathway. We can then remove the node and merge the incoming and outgoing edge, taking account of the stoichiometry of the node metabolite in the two reactions. Gagneur and Klamt (2004) generalized this concept and mentioned that any *uniquely procuded (consumed)* metabolite can be removed. The single production (consumption) reaction is also removed and incorporated into every reaction consuming (producing) the removed metabolite—again considering the stoichiometry when merging. We call this type of compression *unique flow compression*. In the Appendix B of Urbanczik and Wagner (2005), an even more general approach is proposed: any subgraph can be replaced by its elementary modes (EMs). Note that this technique also includes unique flow compression. In most cases, however, the number of EMs is larger than the number of elements in the replaced subgraph.

Before we can identify desirable subgraph replacements, we have to define an efficiency measure for compression. We assume that $\boldsymbol{S}$ has full row rank, since we can remove dependent rows without changing the cone. Furthermore, we assume that all reactions are irreversible, yielding a pointed cone (note that we can convert any network into this form using the transformations described in Prop. 2.1 and Prop. 2.2). Using the nullspace approach (see eq. (2.53)), the double description method runs through $\mathbf{rank}\ \boldsymbol{S} = m$ iterations. Hence, we will take m as the primary efficiency measure for our compression method, and we prefer lower dimensionality r if m is equal for two problem cases.

Let us first discuss the mentioned examples:

Dead-end metabolite: metabolite and producing or consuming reactions are removed. Compression decrements m by one and r at least by one. The compressed network is therefore preferable to the uncompressed case.

Linear pathway: we remove an internal metabolite and merge the reactions connected to it, diminishing m and r by one. If we recursively apply this procedure, we end up with an empty 0×1 (sub)matrix, representing a pointed cone consisting of a single ray—the flux through the linear pathway.

Unique flow compression: If we combine the unique producing (consuming) reaction with the consuming (producing) counterparts, we remove one metabolite and one reaction, i.e. m and r are decremented by one.

As we have seen, all mentioned compression procedures are desirable according to our efficiency measure. The question is whether we can find alternative cases with favorable compression. Let us therefore define a subnetwork as a subset of the metabolites, together with all reactions connected to those metabolites. Note that reactions of the subnetwork can still involve metabolites that are not part of the subnetwork. More formally, a subnetwork G_S of $G = (V, H)$, denoted by $G_S \subseteq G$, is defined as follows:

$$G_S := (V_S, H_S) \ , \ V_S \subseteq V \ , \ H_S = \{\, h = (V_e, V_p) \in H \mid (V_e \cup V_p) \cap V_S \neq \emptyset \,\} \tag{3.1}$$

We call a hyperedge $h \in H_S$ *input* of the subnetwork G_S if at least one metabolite in G_S is produced by the reaction represented by h, and *output* if h consumes at least one metabolite from G_S. Note that h can be both input and output of G_S. If the considered subnetwork has at most one input (or at most one output), we can apply one of the special compression techniques discussed above. Hence we focus on networks with two or more inputs and outputs. We can always eliminate all metabolites in G_S by computing the elementary modes of G_S and replacing G_S by the EMs. Note that this compression technique implies essentially a procedure to compute elementary modes: assume we have a method that computes EMs for a network with one metabolite. Iterating over all metabolites, we can now replace single metabolites with the EMs of the subnetwork containing it. Note, however, that this procedure can generate redundant elements, and we must perform an elementarity test for all generated EMs.

Summing up our observations for compression procedures, any replacement that reduces m and r seems favorable for the performance of EM computation. In our implementation, however, we have only considered the special cases discussed above. Example (C) in Fig. 3.1 illustrates compression that reduces m and r, even if the metabolites in the subnetwork are not uniquely produced or consumed.

3.2 Row Ordering

It is known that the double description method is extremely sensitive to the ordering of the constraints (Fukuda and Prodon, 1995). Without ordering strategies, also efficient implementations can fail badly, even for relatively small examples, since the number of intermediary extreme rays can grow exponentially in terms of the output size. Sorting strategies are distinguished by the way they are applied to the problem: *static ordering* strategies are applied in advance and kept fixed throughout the computation. *Dynamic ordering* procedures decide at each iteration which constraint to add next. Fukuda and Prodon (1995) experimentally assess different ordering strategies for constraints, and a strategy called *lex-min*—lexicographically sorting the vectors—almost always performs best. Note that random order is usually very unfavorable. Dynamic ordering examples include *max-cut-off*—a strategy trying to cut off as many extreme rays as possible—and *min-cut-off*, the opposite approach. Fukuda and Prodon (1995) could not verify superiority of dynamic methods, on the contrary, for certain examples, all dynamic strategies performed much worse than *lex-min*. Our own tests confirm this empirical result. Therefore, we only support static ordering strategies in our implementation.

Note that compared to the standard implementation that operates on general inequality constraints, like that in (Fukuda and Prodon, 1995), we do not sort the constraints directly. Since we are using the *nullspace approach* (see Section 2.3.2), our iterations are associated with

single variables, and the method operates on basis vectors from the kernel matrix. Note also that the kernel matrix has row-echelon form, and we should not resort the identity part of the matrix. However, all other rows of the kernel matrix *have* to be sorted for good performance, and similar strategies have proven efficient as if constraints are sorted directly. We tested several sorting methods, some of the best strategies are the following:

Most-zeros: Kernel rows with more zero entries are chosen first.

Lex-min: Lexicographical ordering of the kernel rows from left to right column.

Abs-lex-min: Lexicographical ordering only considering absolute values.

Fewest-neg-pos: Row entries with negative and positive values are counted. The product of positive and negative count is used as criterion, smaller products are perferred. This strategy is related to finding adjacent extreme ray pairs, since rays with positive and negative entries are candidates.

We have also implemented combinations of the above strategies, using one strategy as primary criterion, and a second method to decide if two rows are considered identical according to the primary measure. Our default sorting strategy, *Most-zeros-or-abs-lex-min*, is an example of such a combination.

3.3 Data Structures

3.3.1 Number Types

The choice of an appropriate number type was an important issue during the implementation of the double description method. Nasty examples are known for which the computation fails with double precision arithmetic. Stoichiometric matrices of metabolic networks are sometimes ill conditioned, which might also lead to computation problems or errors. The compression techniques described in Section 3.1 exacerbate the problem further, hence we decided to support exact arithmetic for all phases of the computation.

Currently, three different number types are supported: double precision arithmetic, and exact arithmetic either based on integers of arbitrary length or on rationals, implemented as fraction numbers composed of such integers. It turned out that the core computation is not harmed significantly by exact arithmetic, since the computationally intensive parts are usually based on binary structures (the zero sets, see also Section 3.3.2). However, arbitrary precision numbers need much more memory, since the numbers can grow during the computation. Furthermore, pre– and post–processing is more strongly affected by exact arithmetic, in particular the back-transformation from binary to numeric extreme rays (see Section 3.3.2).

One challenge when supporting multiple number types are the immense dependencies: the data type is central to all data structures, and they are used by almost every algorithm part. Hence, the whole program has to deal with generic structures to support multi-type arithmetic operations. For instance, also adjacency test options must support different data types, and sophisticated design priciples are necessary to avoid "copy-paste" code. A taste of the generic implementation is given in Section 3.4, where we describe how basic linear algebra functionality was implemented in our algorithm.

3.3.2 Binary Approach

Bit Sets

Adjacency tests (Prop. 2.6) are the most expensive part of the double description algorithm. However, as we only need the *zero set* $\zeta(\boldsymbol{r})$ (Def. 2.28) of a ray $\boldsymbol{r}$ for the tests, we can use binary k-tuples called *bit sets* to store this information. Here, we define a *bit set zero set* according to eq. (2.39) as follows:

Definition 3.2. *Let* $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{x} \geq \boldsymbol{0} \,\}$ *be a polyhedral cone, and* $\mathcal{P}^j = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = 0\ ,\ x_i \geq 0\ ,\ 1 \leq i \leq j \,\}$ *be the intermediary cone at iteration* j*. For any ray* $\boldsymbol{r} \in \mathcal{P}^j$*, the set*

$$\beta^j(\boldsymbol{r}) = \{\, b_1 b_2 \cdots b_i \cdots b_j \mid 1 \leq i \leq j \,\} \quad \textit{with} \quad b_i = \begin{cases} 1 & \textit{if } r_i = 0 \\ 0 & \textit{otherwise} \end{cases}$$

is called the bit set representation *of the zero set* $\zeta^j(\boldsymbol{r})$*.*

The *bitwise and* operation ($\wedge$) for bit sets corresponds to the intersection of the represented sets, because for every bit position in the bit set, the position in the resulting set is 1 iff the position was 1 in both source sets. Accordingly, the subset (or superset) operation can be computed as follows:

$$\zeta(\boldsymbol{x}) \subseteq \zeta(\boldsymbol{y}) \iff \beta(\boldsymbol{x}) \wedge \beta(\boldsymbol{y}) \equiv \beta(\boldsymbol{x}) \tag{3.2}$$

Proposition 3.1. *Let* $\mathcal{P}^j = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = 0\ ,\ x_i \geq 0\ ,\ 1 \leq i \leq j \,\}$ *be an intermediary polyhedral cone at iteration* j*, and* $\mathcal{P}^{j+1} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\boldsymbol{x} = 0\ ,\ x_i \geq 0\ ,\ 1 \leq i \leq j+1 \,\}$ *the cone at iteration* $j+1$*. The bit set zero set* β^{j+1} *can then be derived from* β^j *by*

$$\beta^{j+1}(\boldsymbol{r}) = \begin{cases} \beta^j(\boldsymbol{r}) + 1 & \textit{if } r_{j+1} = 0 \\ \beta^j(\boldsymbol{r}) + 0 & \textit{if } r_{j+1} > 0 \end{cases} \tag{3.3}$$

for retained extreme rays $\boldsymbol{r}$ *fulfilling the new non-negativity constraint* $j+1$*, and*

$$\beta^{j+1}(\boldsymbol{p}, \boldsymbol{q}) = \{\beta^j(\boldsymbol{p}) \wedge \beta^j(\boldsymbol{q}) + 1 \mid p_{j+1} > 0, q_{j+1} < 0\ ,\ \boldsymbol{p} \textit{ adjacent to } \boldsymbol{q}\} \tag{3.4}$$

for new rays combined from adjacent rays $\boldsymbol{p}$ *and* $\boldsymbol{q}$*, where* $+$ *stands for* concatenation*,* $\wedge$ *for the* bitwise and *operation.*

Proof. Follows immediately from Def. 3.2 and eq. (2.60). □

The bit set representation of zero sets has two main advantages: It requires little space in memory, and set operations (*bitwise and*, subset tests) for adjacency can be performed efficiently (a single CPU operation for set sizes of 32/64 elements on a 32/64 bit architecture).

Binary Approach

If a non-negativity constraint has been enforced, it is sufficient to store only binary information for the corresponding values (Gagneur and Klamt, 2004). Those values are either zero or

positive, and if new rays are generated, it is always a non-negative combination of two existing rays (eq. 2.56). Hence we can always update the bit representation according to eq. (3.3) and eq. (3.4) without loss of information. For values associated with a non-negativity constraint j that has not yet been enforced, we need the numeric values to derive the factors p_j and q_j in eq. (2.56). After processing the constraint j, the values r_j are changed from numeric to binary, and after the last iteration step, only binary extreme rays are left. In (Gagneur and Klamt, 2004), the binary extreme rays contain ones for positive values and zeros for zero values. Here, we use the binary complement and store bit set zero sets for the binary part of the extreme rays. From the final (bit set) zero sets, the real extreme rays can be reconstructed as follows:

Proposition 3.2. *Let β be the bit set representation of a zero set $\zeta = \zeta(\boldsymbol{r})$ for an extreme ray $\boldsymbol{r}$ of a polyhedral cone $\mathcal{P} = \{\, \boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{x} \geq \boldsymbol{0} \,\}$, and let $\boldsymbol{C}_\zeta = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{I}_{\zeta\star} \end{bmatrix}$. Then, the nullspace $\mathcal{N}(\boldsymbol{C}_\zeta)$ is 1-dimensional, and either $\boldsymbol{r} \simeq \boldsymbol{v}$ or $\boldsymbol{r} \simeq -\boldsymbol{v}$ for any basis $\boldsymbol{v}$ of $\mathcal{N}(\boldsymbol{C}_\zeta)$.*

Proof. Note that $\boldsymbol{C}_\zeta\,\boldsymbol{r} = \boldsymbol{0}$, that is, $\boldsymbol{r}$ lies in the nullspace of $\boldsymbol{C}_\zeta$. Since $\boldsymbol{r}$ is an extreme ray, $\mathbf{rank}\ \boldsymbol{C}_\zeta = d - 1$ by eq. (2.33), and the nullspace $\mathcal{N}(\boldsymbol{C}_\zeta)$ is 1-dimensional. Hence, $\mathcal{N}(\boldsymbol{C}_\zeta) = \{\, \boldsymbol{x} \mid \boldsymbol{x} = \lambda\boldsymbol{v} \,\}$ for some basis vector $\boldsymbol{v}$ and $\lambda \in \mathbb{R}$, and there exists some positive λ such that either $\boldsymbol{r} = \lambda\boldsymbol{v}$ or $\boldsymbol{r} = -\lambda\boldsymbol{v}$. □

3.4 Generic Linear Algebra Algorithms

A major disadvantage of using Java as a developing language for the *polyhedral computation tool* implemented as part of this thesis is the lack of efficient and generic utility functions for mathematical operations. For C or C++, libraries such as BLAST or LAPACK are available for scientific computations, and optimized and parallelized versions exist for many platforms. On the other hand, such libraries are usually specialized for floating point arithmetic, and supporting multiple data types is not straightforward in most cases. For polyhedral computations, we need a collection of linear algebra functions, preferably supporting different data types, including exact arithmetic with arbitrary precision. Here, we present our approach for a generic implementation of some basic linear algebra functions—generic meaning that almost all functions, independent of the data type—utilize a few fundamental routines. This is possible with type specific callback structures for basic type specific operations. We emphasize that the resulting code is not optimal in terms of efficiency, since highly specialized implementations always allow for better performance. But performance is not critical for those functions in our case, and the benefit of code reusability and maintainability is worth presenting some of these concepts, at least according to our opinion.

3.4.1 Generic Gaussian Elimination

Gaussian elimination is a simple procudure that can be applied to several related linear algebra problems, like matrix inversion, rank computation, matrix triangularization and many more. It has also severe disadvantages, especially if it is applied to floating point numbers due to numeric instability. However, its performance is quite competitive with alternative implementations, and with good pivoting strategies, the most serious problems can be circumvented. Gaussian elimination is the algorithm of our choice mainly for its simplicity and generality, making it a good candidate also for alternative number types.

Our central method transforms a matrix $\boldsymbol{A}$ into row-echelon form:

$$\boldsymbol{A}^{m\times n} \xrightarrow{\text{Gauss. elim.}} \begin{cases} \begin{pmatrix} \boldsymbol{U} & \boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} & \text{if } m \leq n \\ \\ \begin{pmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{R} & \boldsymbol{0} \end{pmatrix} & \text{if } m > n \end{cases} \tag{3.5}$$

Here, $\boldsymbol{U}$ and $\boldsymbol{L}$ denote *upper* and *lower triangular* square matrices, and $\boldsymbol{R}$ is a rectangular full matrix. Since $\boldsymbol{U}$ and $\boldsymbol{L}$ have non-zero diagonal elements, this structure immediately discloses the rank of the matrix $\boldsymbol{A}$ (the size of $\boldsymbol{U}$ or $\boldsymbol{L}$). For a full-rank square matrix $\boldsymbol{A}$, the elimination process leads to an empty matrix $\boldsymbol{R}$, and there are no zero rows and columns. In the following, we will concentrate on the first form of the above transformation for the case $m \leq n$. Note that we can restrict our implementation to this form, since we can simply apply it to the transposed matrix $\boldsymbol{A}^T$, and transpose the resulting matrix again to get the transformation for $m > n$.

Type Specific Operators

To support multiple data types, basic mathematical operations like addition, multiplication etc. are encapsulated in a type specific `Operators` object. Since we use mathematical operations throughout the whole polyhedral computation algorithm, we want to provide an interface that is as flexible as possible. Hence, the `Operators` interface must support different standard operators, classified as follows:

Nullary Operator An operator that delivers a result without any input operand. Typical instances are constant values. Another candidate is the random function.

Unary Operator Operates on one operand. Typical representatives are absolute value or negation. Trigonometric functions are also candidates, but they have not been implemented here.

Binary Operator An operation on two operands, like summation, division, subtraction and multiplication.

Note that all of the above operators return a value of the same type as the operands (if any), i.e. of the type for which the operators are defined. However, some important operators return special values, for instance *comparison operators* comparing two value instances, returning a boolean result if the first operand is larger/equal/smaller than the second. We use the following special operator classes:

Boolean Unary Operator An operator that returns a boolean value, taking a single numeric value as operand. Typical instances compare a value with zero, with one, or tell you whether a given value is positive or negative.

Boolean Binary Operator Returns a boolean value based on two numeric operands. Typical operators return the result of comparing the first with the second operand (e.g. "is the first value larger than the second?")

Integer Unary Operator Operates on one numeric operand and returns an integer value. A typical instance is the signum function.

Integer Binary Operator An operation on two numeric operands, returning an integer value. The comparison operator is a typical representative, returning -1, 0 or 1 if the first value is smaller, equal or larger than the second value, respectively.

The next operator class is not mandatory, but practical and efficient especially for linear algebra functions. These operators perform an operation on vectors, but return a single value. Hence, we call them *aggregating operators*:

Aggregating Unary Operator Performs an operation on a vector of numeric values, and returns a numeric value. A typical instance returns the minimum, maximum, sum, absolute sum or sum of squares. Another important operation of this kind deals with *vector normalization*: depending on the number type, different normalization techniques might be appropriate. For instance, normalizing a floating point vector might trim the vector to length one. For an integer vector, however, we cannot apply this operation. Instead, it might be appropriate to divide all vector elements by the greatest common divisor (GCD) of all elements. We have defined and implemented two such operators to provide transparent support for vector normalization throughout our programs. To normalize a vector, all elements are divided by the value returned by the operator.

Aggregating Binary Operator Performs an operation on two numeric vectors and returns a numeric value. A typical implementation is the inner product of the two vectors, which is also used to implement matrix multiplication.

The `Operators` interface provides also *Converter Operators* that are used to convert between different number types. Furthermore, an *Expression Composer* facilitates the composition of expressions, using a series of the standard operators.

Pivoting Strategies

Gaussian elimination or the related LU factorization is known to be sensitive to roundoff errors. To avoid numerical instability, one usually permutes matrix $\boldsymbol{A}$ to find favorable pivot elements during the triangularization. Typical pivoting strategies for floating point matrices prefer numbers with larger absolute values over others, to avoid large results after division by small pivot elements. In (Golub and Van Loan, 1996), partial and complete pivoting is discussed, meaning strategies that permute only rows or both rows and columns, respectively. Row and column swap operations are expressed by permutation matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$ as follows:

$$\boldsymbol{P}\boldsymbol{A}\boldsymbol{Q} = \boldsymbol{L}\boldsymbol{U} \tag{3.6}$$

Note that matrix $\boldsymbol{U}$ in eq. (3.5) corresponds to the $\boldsymbol{U}$ matrix in a generalized LU factorization for rectangular matrices. Matrix $\boldsymbol{L}$ contains the multiplication terms used for the elimination. It is for instance useful to solve a system of linear equations. For our purposes, $\boldsymbol{L}$ is not important and we ommit it here.

With partial pivoting, the matrix $\boldsymbol{Q}$ in eq. (3.6) is the identity. In most cases, Gaussian elimination with partial pivoting is considered sufficiently stable, and with complete pivoting, it is stable (Golub and Van Loan, 1996). Partial pivoting is thus usually chosen over complete pivoting, but since rank determination is important in our case, and since performance of this procedure is not critical, we use complete pivoting for all Gaussian algorithms.

For fraction numbers or integers, choosing large pivots is an extremely bad idea: since we divide by the pivot elements, the numbers are growing rapidly. Favorable pivot elements are thus small numbers. We implemented a strategy that prefers numbers with a small bit length considering the absolute value of non-zero elements. If a one value is found, we abort the pivoting search. For fraction numbers, we compute the product of numerator and denominator bit lengths.

Exploiting Sparseness

Many constraint matrices for polyhedral cones occuring in practice are sparse, and they contain mostly small integers. This is true in particular for stoichiometric matrices, hence an implementation that adapts to sparseness is highly desirable. Note that we are not using sparse data structures to store the matrices, even if this should be considered in a revised implementation. Our algorithm detects sparseness within the dense matrix structure using two very simple ideas:

1. Rows below the pivot element with a zero entry in the pivoted column are skipped
2. The pivot row is pre-scanned for non-zero entries; zero entries can thus be skipped when subtracting the pivot row from rows below it.

Even if the strategies are trivial, the effect is significant, as shown in Fig. 3.2. For random double matrices, sparseness leads to considerably faster computation times, especially up to around 1% density. For higher densities above 10%, the speedup is not significant. Note, however, that stoichiometric matrices have typically around $5n$ non-zero entries, leading to a density of $5/n$, if n denotes the size of a square matrix. Hence, the speedup is relevant for computations with stoichiometric matrices. In Fig. 3.2B, tests with 100×100 integer matrices are shown, using exact arithmetic for the computations. Running times react very sensitive to the density of the matrix and the size of the integers. Stoichiometric matrices almost exclusively contain small integers, which also has a beneficial effect on the performance of the algorithms.

3.4.2 Functions Derived from Gaussian Elimination

As already mentioned, the rank of the matrix can immediately be derived from the transformed matrices, namely the size of square matrices $\boldsymbol{U}$ and $\boldsymbol{L}$ in eq. (3.5). Two other functions that

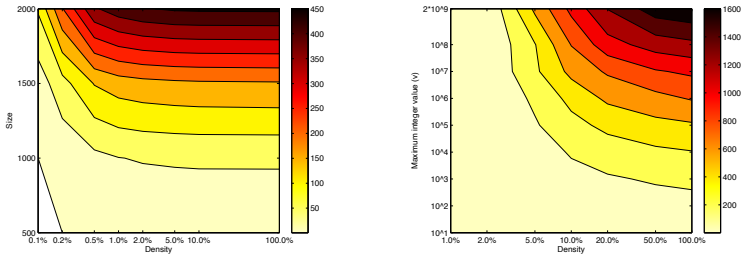


Figure 3.2: Computation times in seconds for matrix inversion, shown as colored areas in the plots. To avoid singular matrices, diagonal elements were always non-zero, in addition to values set randomly according to the density. **A:** Inversion of random matrices using double precision floating point arithmetic for different matrix sizes and densities. **B:** Inversion of 100×100 integer matrices, yielding rational result matrices with fraction numbers of arbitrary precision, again using different matrix densities. The value v on the y-axis indicates that the starting matrix contains random integers constrained to the interval $\{-v, \ldots, v\}$.

are important for our algorithm are implemented on the basis of this Gaussian elimination process.

Inverse of Maximal Submatrix

To perform matrix inversion functions, the Gaussian elimination procedure is slightly extended: instead of producing upper (or lower) triangular matrices, we also eliminate the elements above the diagonal. The matrices $\boldsymbol{U}$ and $\boldsymbol{L}$ in eq. (3.5) are now diagonal, and by dividing each row of the complete matrix by the diagonal element, we get identity matrices for $\boldsymbol{L}$ and $\boldsymbol{U}$. We focus on the case $m \leq n$, and start the elimination process with $\left(\begin{array}{cc} \boldsymbol{A} & \boldsymbol{I} \end{array}\right)$. If we allow only columns from matrix $\boldsymbol{A}$ in the pivoting procedure, we get

$$\left(\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{I} \end{array}\right) \xrightarrow{\text{Gauss. elim.}} \left(\begin{array}{cc|cc} \boldsymbol{I} & \boldsymbol{R} & \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{array}\right) \tag{3.7}$$

If $\boldsymbol{A}$ is square and regular, the matrices $\boldsymbol{R}$ and $\boldsymbol{C}$ are empty and we have no zero rows. Furthermore, we have $\boldsymbol{B} = \boldsymbol{A}^{-1}$, neglecting the permutation matrices for simplicity. The method implements the standard "schoolbook procedure" for matrix inversion. If $\boldsymbol{A}$ is singular or rectangular, $\boldsymbol{B}$ contains the inversion of a maximal submatrix of $\boldsymbol{A}$, the submatrix itself is reflected by the permutation matrices. Submatrix inversion has for instance been used to implement the constraint transformations for polyhedral cones described in Prop. 2.1 and 2.2.

Kernel Matrix

Another important function is the computation of the *kernel matrix*, a basis for the nullspace of $\boldsymbol{A}$. It is for instance used for the consistency and compression methods described in Section 3.1.1, but also to compute the initial generating matrix for the double description method using the nullspace approach (see Section 2.3.2). As for matrix inversion, we use the reduced form of eq. (3.5), i.e. the matrix $\boldsymbol{U}$ is an identity matrix. We get

$$\boldsymbol{A} \xrightarrow{\text{Gauss. elim.}} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \quad \Longrightarrow \quad \boldsymbol{K} = \ker \boldsymbol{A} = \begin{pmatrix} -\boldsymbol{R} \\ \boldsymbol{I} \end{pmatrix} \tag{3.8}$$

4
Elementarity & Adjacency

Rays that are generated during iterations of the double description method are only retained if they are *elementary*, meaning that they must be extreme rays of the cone. Generating rays and testing elementarity absorbs most of the computation time of the algorithm, and it is therefore inevitable to provide efficient strategies for any suitable implementation. Different approaches are possible to improve the process: first, we have two ways to test elementarity or adjacency of two rays. The combinatorial test is based on a superset search using the zero sets of the involved rays. We discuss general strategies allowing for indexed subset and superset search queries. The indexing is based on *bit pattern trees*, a specialization and extension of multi-dimensional search trees (k-d-trees) invented by Bentley (1975). The second attempt to speedup the enumeration process of adjacent extreme rays approaches the all-against-all loop directly. Instead of looping through all pair candidates, we propose a recursive enumeration procedure again based on bit pattern trees. We show that we can avoid individual tests in many cases, especially if ray pairs match badly in terms of adjacency. Since we eliminate many adjacent ray pair candidates before we perform the actual adjacency test, we call this procedure *candidate narrowing*. Our last improvement addresses the algebraic adjacency test method. We show that we can use bit pattern trees to perform rank computations with an updating strategy. Similar tests benefit from each other since commonalities in the rank computations are exploited. Finally, we show that residue arithmetic has many advantages for rank computations compared to floating point arithmetic. We conclude this chapter with computational benchmarks, demonstrating the efficiency of the introduced methods.

4.1 Binary Trees for Bit Sets

Applying binary trees to bit sets has been introduced in Terzer and Stelling (2006) and refined in Terzer and Stelling (2008). The idea would probably not have been born without the input of Prof. Gonnet, who pointed me to the direction of k-d-trees.

4.1.1 Bit Set Trees

The zero sets (Defs. 2.28 and 3.2) can be seen as k-tuples of boolean values, and search operations on a set of bit sets coincide with queries on a collection of k-dimensional records. An efficient structure for storage and retrieval of multidimensional (k-dimensional) data has been invented in (Bentley, 1975) in the form of k-d-trees.

To perform adjacency tests for two rays, we can try to find a superset for a given bit set, as imposed by the combinatorial test (Prop. 2.6d). Two rays $\boldsymbol{r}$ and $\boldsymbol{r}'$ with corresponding zero sets $\zeta(\boldsymbol{r})$ and $\zeta(\boldsymbol{r}')$ are adjacent iff we find no superset of $\zeta(\boldsymbol{r}) \cap \zeta(\boldsymbol{r}')$ other than $\zeta(\boldsymbol{r})$ and $\zeta(\boldsymbol{r}')$. We can execute this type of query on a binary k-d-tree, similar to the *partial match queries* given in (Bentley, 1975).

Tree Construction

Given a set of bit sets (our zero sets), the algorithm returns a binary k-d-tree or *bit set tree*. The input of the algorithm is a *set* of bit sets, that is, a collection without duplicates, conforming with the actual problem. This simplifies the partition of bit sets (Alg. 2: `CreateBitSetTree`, lines 4–6), since we can be sure that they can always be separated.

4.1.2 Indexed Subset and Superset Searching

Superset-Existence

The method `hasSuperSet` is defined for every node of the bit set tree, as indicated through the definition in the abstract class `Node`. The method is individually implemented by the subclasses `InterNode` and `LeafNode` as shown below. To test whether two extreme rays $\boldsymbol{p}$ and $\boldsymbol{q}$ are adjacent, we use their bit set zero sets β_p and β_q and invoke the test method on the root node returned by the tree creation algorithm:

```
root = CreateBitSetTree(sets)
adjacent = not root.hasSuperSet(β_p, β_q)
```

For the general problem of finding a superset of a given test set β in a (duplicate free) set of sets, we directly pass the test set β to `hasSuperSet` (instead of passing β_p and β_q and computing β at line 1 of the functions below). Note that bit set trees can also be used to find a subset instead of a superset. Since $\zeta' \supseteq \zeta \Leftrightarrow \overline{\zeta}' \subseteq \overline{\zeta}$, we can simply create the tree with the complement sets and invoke the test method with $\overline{\beta}$.

Algorithm 2: CreateBitSetTree(*BitSet*[ ] *sets*)

```
   desc   : Creates a bit set tree and returns the root node of the tree
   input  : Set of bit set zero sets (duplicate free)
   output: Root node of the tree
   const  : threshold ← 4 (limit for number of bit sets per leaf node)
   define
       class Node
           /* adjacency test method, defined by Node, but implemented in subclasses InterNode
              and LeafNode                                                                  */
           abstract boolean hasSuperSet(β_p, β_q)

       class InterNode extends Node
           int bit            /* index of bit used to separate sets in left and right subtree */
           Node left                          /* left subtree with sets not containing bit  */
           Node right                         /* left subtree with sets not containing bit  */

       class LeafNode extends Node
           BitSet[] sets

   end
   begin
1      if (length(sets) < threshold) then        /* create leaf node containing the bit sets */
2          node ← new LeafNode(sets)
3      else                                        /* create intermediary node with subtrees */
4          bit ← some index not yet used to separate bit sets
5          ls ← {β ∈ sets | β(bit) = 0}
6          rs ← {β ∈ sets | β(bit) = 1}
7          left ← CreateBitSetTree(ls)              /* recursive invocation for left subtree */
8          right ← CreateBitSetTree(rs)            /* recursive invocation for right subtree */
9          node ← new InterNode(bit, left, right)
10     end
11     return node
   end
```

Correctness and Complexity

By the way of constructing the tree, the `left` subtree of an intermediary node with selective bit j contains those bit sets that have $\beta(j) = 0$ (see Alg. 2: `CreateBitSetTree`, line 4 ff.). Thus, if the set β' to be tested contains j, that is, $\beta'(j) = 1$, the bit sets in `left` cannot be supersets of β' and only the bit sets in `right` are superset candidates, conforming with the recursion condition (line 2, 3 and 7 in Fct. `InterNode.hasSuperSet()`, p. 62).

It is very difficult to estimate the number of intermediary rays, and hence also the total time complexity of the double description algorithm. However, for each step, we know from Corollary 2.1 that each zero set must contain at least $d-\mathbf{rank}\ \boldsymbol{A}-1$ elements. Let $a = \mathbf{rank}\ \boldsymbol{A}$, then, each bit set in the tree has at least $d-a-1$ one-bits. For the test set, we know by Corollary 2.2 that every set that passes the test has at least $d-a-2$ one-bits, and we do not test sets with fewer one-bits. Hence, we approximate the chance for a one-bit in a tree set of length l by $\frac{d-a-1}{l}$, and for a one bit in the test set by $\frac{d-a-2}{l}$, where $l \in \{d-a, \ldots, d-1\}$ for the a iteration steps. If n is the number of bit sets in the tree, we can now estimate the

Function `InterNode.hasSuperSet(`*BitSet* β_p, *BitSet* β_q`)`

```
   desc   : Test method of an intermediary node of a bit set tree, checking adjacency of two extreme rays
            p and q using the combinatorial test
   input  : The bit set zero set for each extreme ray
   output: true if a superset is found (meaning that p and q are not adjacent), false otherwise
   begin
1  |  β = β_p ∧ β_q                                          /* intersection of zero sets */
2  |  if (β(bit) = 0) then                                  /* superset candidates if bit not in β */
3  |  |  if (left.hasSuperSet(β_p, β_q)) then
4  |  |  |  return true
5  |  |  end
6  |  end
7  |  if (right.hasSuperSet(β_p, β_q)) then                 /* always superset candidates */
8  |  |  return true
9  |  end
10 |  return false
   end
```

Function `LeafNode.hasSuperSet(`*BitSet* β_p, *BitSet* β_q`)`

```
   desc   : Test method of a leaf node of a bit set tree, checking adjacency of two extreme rays p and q
            using the combinatorial test
   input  : The bit set zero set for each extreme ray
   output: true if a superset is found (meaning that p and q are not adjacent), false otherwise
   begin
1  |  β = β_p ∧ β_q                                          /* intersection of zero sets */
2  |  foreach set ∈ sets do
3  |  |  if (set ≠ β_p) and (set ≠ β_q) then
4  |  |  |  if (β ∧ set = β) then                           /* subset test */
5  |  |  |  |  /* β is a subset of set                                */
6  |  |  |  |  return true
7  |  |  |  end
8  |  |  end
9  |  end
10 |  return false
   end
```

remaining sets to test if we recurse one or both subtrees:

$$\begin{cases} n \cdot \frac{d-a-1}{l} & \text{remaining sets to test with probability} \quad \frac{d-a-2}{l} \\ n & \text{remaining sets to test with probability} \quad 1 - \frac{d-a-2}{l} \end{cases} \tag{4.1}$$

We set $\omega_1 = \frac{d-a-1}{l}$ and $\omega_2 = \frac{q-m-2}{l}$ and estimate the number of bit sets in the tree to consider per adjacency test. If the tree is well-balanced, it has height $h = \log_2(n)$. If we take into account that the tree is asymmetrical since $\omega_1 \neq 1/2$, we can estimate its height by

$$h = \log_k(n) \quad \text{with} \quad k = \min(\frac{1}{\omega_1}, \frac{1}{1-\omega_1}) \tag{4.2}$$

We can now approximate the time complexity by

$$n \cdot (1 - \omega_2 + \omega_1\omega_2)^h = n^{1+\log_k(1-\omega_2+\omega_1\omega_2)} = \mathcal{O}\big(n^\sigma\big) \tag{4.3}$$

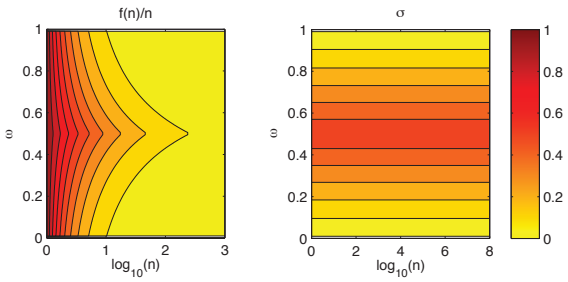


Figure 4.1: Estimated time complexity $f(n) = \mathcal{O}(n^{\sigma})$ for superset search with bit set trees. The one-bit probability ω is assumed equal for the bit sets in the tree and for the test set.

Note that eq. (4.3) is sublinear since the logarithm in the exponent is always negative, yielding an estimated time complexity $f(n) = \mathcal{O}(n^{\sigma})$ with $\sigma = \frac{\log(f(n))}{\log(n)} = 1 + \log_k(1 - \omega_2 + \omega_1\omega_2)$ and $k = \min(\frac{1}{\omega_1}, \frac{1}{1-\omega_1})$. If we assume that $\omega \approx \omega_1 \approx \omega_2$, we get the poorest complexity for $\omega = 1/2$ with $\sigma = \log_2 3 - 1 \approx 0.5850$ (see Fig. 4.1).

We have no influence on the sets to test, meaning that we can hardly manipulate ω_2. However, since $n \ll 2^l$, some bits are never used for the separation of subtrees, and we have a certain degree of freedom to choose the most appropriate bits and bit ordering. For instance, we could try to select those bits that separate the subtrees more evenly, that is, we try to influence ω_1 in eq. (4.3). In Fig. 4.2, different choices for ω_1 are shown, together with all possible values for ω_2. It turns out that it is indeed advantageous to construct the tree in such a way that we have more bit sets in the left subtree (not containing the separating bit). Test sets with many one-bits are favorable, which is quite obvious since more frequently, only one subtree is traversed. Note that eq. (4.3) is actually a conservative approximation since we stop the traversal process immediately if a superset is found. In practice, we had good experience with $\omega_1 \approx 1/2$, probably because *candidate narrowing* (see Section 4.1.4) performs best with balanced trees.

If we assume that every intermediary node in the tree has two subtrees, a total of $n - 1$ intermediary nodes are needed to hold n leaves. Note that we simply remove intermediary nodes with one subtree and connect the child tree to the parent node. Hence, $c \cdot 2n$ memory is required to store n bit sets in the tree, where c is a small constant. Optimizations could be applied, but these memory demands are far from being critical for our purposes. However, we can reduce the memory needed for the tree by storing multiple bit sets per leaf (see `threshold` constant in Alg. 2: `CreateBitSetTree`). In the current implementation, each leaf node contains up to four bit sets.

4.1.3 Bit Pattern Trees

The general idea of *pattern trees* ties up to the *bit set trees*, where bit sets are separated in every intermediary node using some designated *selective bit* as criterion for partitioning. In pattern trees, all tree nodes additionally store a *union pattern* reflecting the commonalities

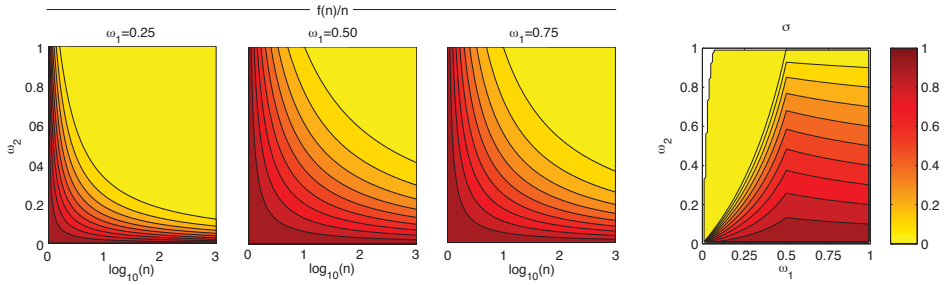


Figure 4.2: Estimated time complexity $f(n) = \mathcal{O}(n^{\sigma})$ for superset search with balanced bit set trees. The choice of bits used to separate left and right subtrees influences the ω_1 parameter, the probability of a one at the separating bit. The one-bit probability in the test set is reflected by ω_2.

of all bit sets contained in the subtree or leaf node. Note that at least the selective bit used in the parent node is common to all bit sets stored in the subtree. However, since the actual bit sets constitute only a small fraction of all possible bit combinations, it is likely that other common 0-bits occur in the pattern. This allows for a more restrictive pre-rejection of test sets as follows:

Proposition 4.1. *Let ζ' be a test set, $\mathcal{T}$ a collection of zero sets (like a subtree) and $\zeta^{U} = \{\bigcup \zeta \mid \zeta \in \mathcal{T}\}$ the union of all sets in $\mathcal{T}$. Then $\zeta' \subseteq \zeta^{U}$ is a necessary condition for $\{\zeta \mid \zeta' \subseteq \zeta\ ,\ \zeta \in \mathcal{T}\} \neq \emptyset$, i.e. that a superset ζ of ζ' exists in $\mathcal{T}$.*

Proof. If $\zeta' \not\subseteq \zeta^{U}$, ζ' contains at least some element $e \notin \zeta^{U}$. Thus, for all sets $\zeta \in \mathcal{T}$, also $e \notin \zeta$ and consequently $\zeta' \not\subseteq \zeta$ holds for all $\zeta \in \mathcal{T}$. □

Tree Construction

We use the same algorithm as for bit set trees, but additionally, we calculate the *union bit pattern* β^{U} when a new node is created (Alg. 2: `CreateBitSetTree`, line 2 for leaf nodes, and lines 7–8 for intermediary nodes). The union bit pattern is calculated as $\beta^{U} = \bigvee \beta$ for all bit set zero sets $\beta \in \mathcal{T}$, where $\vee$ stands for the *bitwise or* operation, and $\mathcal{T}$ corresponds to the variables `sets` for leaves and to `ls` and `rs` for intermediary nodes in Alg. 2: `CreateBitSetTree`.

Superset-Existence

The procedure is essentially the same as for bit set trees, but instead of bit testing (line 3 in Fct. `InterNode.hasSuperSet()`, p. 62 above), we check the union pattern condition (line 2 below). Note that we always recurse both subtrees in the intermediary node, since the union

Function `InterNode.hasSuperSet(`*BitSet* β_p, *BitSet* β_q`)`

```
   desc   : Test method of an intermediary node of a bit pattern tree, checking adjacency of two extreme
            rays p and q using the combinatorial test
   input  : The bit set zero set for each extreme ray
   output: true if a superset is found (meaning that p and q are not adjacent), false otherwise
   begin
1      β = β_p ∧ β_q                                              /* intersection of zero sets */
2      if (β ∧ β^U = β) then                  /* test whether β is a subset of union pattern β^U */
3          if (left.hasSuperSet(β_p, β_q)) then
4              return true
5          end
6          if (right.hasSuperSet(β_p, β_q)) then
7              return true
8          end
9      end
10     return false
   end
```

pattern condition aborts the traversing process anyway at the next recursion level if necessary. An exemplarily pattern tree with search path for a test set is shown in Fig. 4.3.

For leaf nodes, we also perform the union pattern test (line 2) and discontinue if the test fails (added after line 1 in Fct. `LeafNode.hasSuperSet()`, p. 62).

Correctness and Complexity

We only abort the test process if the union pattern test fails. Then, we know by Prop. 4.1 that no bit set in the subtree can be a superset candidate.

Time and memory complexity for bit pattern trees are very similar to those of bit set trees. To derive more precise estimates for time complexity seems quite difficult, but is not of great help for the overall running time of the algorithm. Hence, we perform no detailed analysis for pattern trees, even if this might be interesting for theoretical purposes.

4.1.4 Candidate Narrowing with Tree All-against-all

A first non-recursive approach for *candidate narrowing* was already presented in (Terzer and Stelling, 2006). The current version described here was published in (Terzer and Stelling, 2008).

At iteration j, the non-negativity constraint j splits all intermediary rays $\boldsymbol{r}$ into three parts, according to the sign of r_j. If τ is the set of all ray indices (column indices in $\boldsymbol{R}^j$), the sets $\tau_>$, $\tau_<$ and τ_0 correspond to indices of rays with positive, negative and zero j-value (see eq. (2.55)). The simplest approach to use bit pattern trees enumerates all ray pair indices $(i,k) \in (\tau_> \times \tau_<)$ and tests adjacency as described in the previous section. However, enumerating all index pairs (i,k) is still an elaborate task in $\mathcal{O}(n^2)$, where n is the number of intermediary rays.

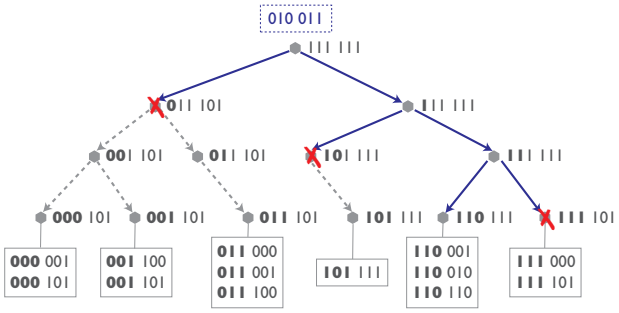


Figure 4.3: Bit pattern tree with union patterns on the right of every tree node and leaf nodes at the bottom of the tree with bit set zero sets in the boxes. Bold values are common for the whole subtree since they were used to separate the sets in the left and right subtrees. The bit set in the dotted box at the top of the tree represents an exemplary test set β. Searching a superset of β, the tree is traversed along the blue arrows. Dotted arrows are not traversed and truncation of searching is indicated by red crosses next to the union pattern causing abortion.

We improve the process by excluding certain pairs of the cartesian product before actually enumerating them. The idea is to perform the all-against-all enumeration process (all elements in $\tau_>$ against all in $\tau_<$) recursively on trees. Analogous to the $\tau_\star$ index sets, we create three bit pattern trees $\mathcal{T}_>$, $\mathcal{T}_<$ and $\mathcal{T}_0$ containing bit set zero sets corresponding to ray indices in $\tau_>$, $\tau_<$ and τ_0, respectively. Enumerating all (i, k) pairs and testing adjacency of the sets β_i and β_k is equivalent to pairing sets $\beta_i \in \mathcal{T}_>$ with $\beta_k \in \mathcal{T}_<$ and testing the pairs (β_i, β_k). Instead of directly building all pairs using the whole trees, we pair the sets *recursively* using the four subtree combinations:

- $\beta_i \in \mathcal{T}_{>,L}$ with $\beta_k \in \mathcal{T}_{<,L}$
- $\beta_i \in \mathcal{T}_{>,L}$ with $\beta_k \in \mathcal{T}_{<,R}$
- $\beta_i \in \mathcal{T}_{>,R}$ with $\beta_k \in \mathcal{T}_{<,L}$
- $\beta_i \in \mathcal{T}_{>,R}$ with $\beta_k \in \mathcal{T}_{<,R}$

Here, $\mathcal{T}_{\star,L}$ and $\mathcal{T}_{\star,R}$ denote the left and right subtree of $\mathcal{T}_\star$, respectively. A recursion occurs if at least one tree node is an intermediary node. If both nodes are leaves, we perform the standard all-against-all loop with the bit sets of the leaf nodes. Recursing this way takes about $4n^2$ steps instead of n^2, since every tree contains approximately $2n$ nodes if we assume unary leaves holding only one bit set.

So far, we have only worsened the enumeration procedure. By introducing a *cut pattern* based on the union patterns of two tree nodes, we can narrow down the adjacency candidates, as imposed by the following proposition:

Proposition 4.2. *Let $\mathcal{T}_a$ and $\mathcal{T}_b$ be collections of zero sets (like subtrees) with corresponding union patterns ζ_a^U and ζ_b^U, according to Prop. 4.1. We define the* cut pattern *ζ^C as*

intersection of the two union patterns, that is, $\zeta^C = \zeta_a^U \cap \zeta_b^U$. *Then, for any set* ζ', *the implication*

$$\zeta^C \subseteq \zeta' \quad \Longrightarrow \quad \zeta \subseteq \zeta' \tag{4.4}$$

holds for every test set $\zeta = \zeta_a \cap \zeta_b$ *with* $\zeta_a \in \mathcal{T}_a$ *and* $\zeta_b \in \mathcal{T}_b$.

Proof. By definition, $\zeta^C = \big((\zeta_{a,1} \cup \cdots \cup \zeta_{a,n_a}) \cap (\zeta_{b,1} \cup \cdots \cup \zeta_{b,n_b})\big)$. Applying the distributive law, we get $\big((\zeta_{a,1} \cap \zeta_{b,1}) \cup (\zeta_{a,1} \cap \zeta_{b,2}) \cup \cdots \cup (\zeta_{a,1} \cap \zeta_{b,n_b}) \cup \cdots \cup (\zeta_{a,n_a} \cap \zeta_{b,n_b})\big)$, that is, the union of all possible intersections ζ. Thus, every ζ is a subset of ζ^C, and if $\zeta^C \subseteq \zeta'$, consequently also $\zeta \subseteq \zeta'$. □

We can use eq. (4.4) as a necessary precondition before recursing two subtrees $\mathcal{T}_a$ and $\mathcal{T}_b$ in the all-against-all loop. If we find any zero set ζ' such that the cut pattern ζ^C of two tree nodes is a subset of ζ', we know that this recursion is not necessary, at least if $\zeta' \notin \mathcal{T}_a$ and $\zeta' \notin \mathcal{T}_b$. If ζ' is a set of one of the subtrees, we could still have adjacent pairs involving ζ'. In that case, we need at least two distinct sets $\zeta' \supseteq \zeta^C$ and $\zeta'' \supseteq \zeta^C$ of the same subtree to refrain from recursing. Pseudo code for the recursive enumeration procedure is shown in Alg. 6: `AddAdjacent`, a simple case study is given in Appendix A4.1.

Instead of the combinatorial adjacency test (Prop. 2.6d), we can also use Prop. 4.2 with alternative preconditions for adjacency to implement `isPreconditionMet` (line 2 in Alg. 6: `AddAdjacent`). For instance, if the rank $d-2$ in Prop. 2.6b cannot be achieved with ζ^C, also no $\zeta \subseteq \zeta^C$ can achieve it. Since the precondition is verified many times in the tree, fast tests are preferable. In the current implementation, we check the minimum cardinality condition $|\zeta^C| \geq d - 2 - \mathbf{rank}\ \boldsymbol{A}$ (see Corollary 2.2).

The final adjacency test for two bit sets (method `areAdjacent` at line 13) is either a rank computation according to Prop. 2.7c, or we perform the combinatorial test (Prop. 2.7d) using the pattern trees for optimized superset searching. If the combinatorial test is used, we search for supersets in all three trees $\mathcal{T}_>$, $\mathcal{T}_0$ and $\mathcal{T}_<$.

Note that the power of this strategy depends mainly on two aspects: primarily on the cut pattern, and hence also the union patterns of the two subtrees, and secondarily on the sizes of the subtrees. Sparse or dissimilar union patterns ζ_a^U and ζ_b^U yield sparse cut patterns ζ^C, which usually enhances the probability that an adjacency precondition fails for ζ^C, and therefore prevents the algorithm from recursing. On the other hand, we would like to eliminate recursions for large subtrees, but larger trees tend to denser union patterns. *Bit pattern trees* comply well with this trade-off, since union patterns for nodes in the upper part of the tree are dense, representing large subtrees. By descending the trees when recursing, the union patterns become sparser, but the subtrees also contain fewer elements. For subtree pairs with widely different union patterns, the recursion might stop early affecting large subtrees; for similar patterns, we have to recurse close or up to the leaf nodes.

4.1.5 Lazy Rank Updating

Lazy rank updating has been published in (Terzer and Stelling, 2008).

Using the algebraic adjacency test Prop. 2.7b has the advantage that it does not depend on the number of intermediary rays. We compute the rank of a submatrix of the equality matrix $\boldsymbol{A}$,

Algorithm 6: AddAdjacent(*Node* $\mathcal{T}_a$, *Node* $\mathcal{T}_b$, *Set adj*)

```
   desc   : Recursively traverses two bit pattern trees T_a and T_b and adds adjacent bit set pairs (β_a, β_b)
            to adj
   input  : Tree nodes T_a and T_b
   output : Adjacent bit set pairs in adj
   begin
1      β^C ← β_a^U ∧ β_b^U                                          /* compute cut pattern */
2      if (isPreconditionMet(β^C)) then
3          if (T_a is InterNode) and (T_b is InterNode) then
4              /* one intermediary nodes:  recurse subtrees                              */
5              AddAdjacent(T_{a,L}, T_{b,L}, adj)                    /* left , left */
6              AddAdjacent(T_{a,L}, T_{b,R}, adj)                   /* left , right */
7              AddAdjacent(T_{a,R}, T_{b,L}, adj)                   /* right , left */
8              AddAdjacent(T_{a,R}, T_{b,R}, adj)                  /* right , right */
9          else if (T_a is LeafNode) and (T_b is LeafNode) then
10             /* both leaf nodes:  enumerate & test                                     */
11             foreach β_a ∈ T_a do
12                 foreach β_b ∈ T_b do
13                     if (areAdjacent(β_a, β_b)) then
14                         adj ← adj ∪ {(β_a, β_b)}
15                     end
16                 end
17             end
18         else
19             /* one leaf, one intermediary node:  recurse subtrees                     */
20             if (T_a is InterNode) then
21                 AddAdjacent(T_{a,L}, T_b, adj)                    /* left , leaf */
22                 AddAdjacent(T_{a,R}, T_b, adj)                   /* right , leaf */
23             else
24                 AddAdjacent(T_a, T_{b,L}, adj)                    /* leaf , left */
25                 AddAdjacent(T_a, T_{b,R}, adj)                   /* leaf , right */
26             end
27         end
28     end
   end
```

and the submatrix is defined by the rays to test. However, rank computation is an expensive procedure for large matrices, which can be optimized especially if we compute many similar submatrices.

Let us recall that we are recursively traversing two trees with little change between two recursion steps. Hence, we can think of an update strategy for the examined matrix. Using Gaussian elimination to compute an upper triangular matrix to derive the rank, we extend the triangular part with every recursion step. Lower recursion levels can then benefit from the precomputed part of the matrix and only need to perform triangularization of the remaining part.

Let us consider two consecutive recursions of Alg. 6: `AddAdjacent` with parent nodes $(\mathcal{T}_{P_a}, \mathcal{T}_{P_b})$ and child nodes $(\mathcal{T}_{C_a}, \mathcal{T}_{C_b})$, e.g. assuming the first recursion at line 5. Descending the trees, union patterns can only have fewer or equally many elements. Thus, the parent cut pattern is a superset of the child cut pattern, i.e. $\zeta^C_{(P)} \supseteq \zeta^C_{(C)}$. The algebraic test (Prop. 2.6c)

$$\left[\begin{array}{c|ccc} \times & \otimes & \otimes & \otimes & \dots \\ \hline 0 & \otimes & \otimes & \otimes & \dots \\ 0 & \otimes & \otimes & \otimes & \dots \\ 0 & \otimes & \otimes & \otimes & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}\right] \quad \rightsquigarrow \quad \left[\begin{array}{ccc|c} \times & \otimes & \otimes & \otimes & \dots \\ 0 & \times & \otimes & \otimes & \dots \\ 0 & 0 & \times & \otimes & \dots \\ \hline 0 & 0 & 0 & \otimes & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}\right]$$

$$\zeta_{(P)} = [0111\, R_b\,] = \{2, 3, 4, R\} \qquad \zeta_{(C)} = [0010\, R_b\,] = \{3, R\}$$

$$\overline{\zeta}_{(P)} = [1000\, \overline{R}_b\,] = \{1, \overline{R}\} \qquad \overline{\zeta}_{(C)} = [1101\, \overline{R}_b\,] = \{1, 2, 4, \overline{R}\}$$

$$\overline{\zeta}_{(C)} \backslash \overline{\zeta}_{(P)} = \zeta_{(P)} \backslash \zeta_{(C)} = [0101\, 0 \cdots 0\,] = \{2, 4\}$$

Figure 4.4: One step of the rank update method: partially triangularized matrix before the current step (left) and continuation after triangularization of columns two and four (right). $\times$ stands for non-zero pivot elements, $\otimes$ for any value. Triangularized columns are reflected by elements in the complementary cut patterns $\overline{\zeta}_{(P)}$ and $\overline{\zeta}_{(C)}$, respectively. Exemplary patterns are given in binary (square brackets) and standard set notation (curly brackets), R stands for any remainder of the sets, R_b for its binary representation. Note that column three and four are swapped during the update step to put the pivots in place, and rows might have been swapped to find non-zero pivot elements. If all remaining elements of an added column are zero, it is put to the end of the matrix and ignored at subsequent steps.

consists of two parts: computing the rank of a column submatrix of $\boldsymbol{A}$, and evaluating the size of the test set $\zeta = \zeta(\boldsymbol{r}\,') \cap \zeta(\boldsymbol{r}\,'')$. During the traversal process, the cut pattern is our test set ζ (we use $\zeta_{(P)} = \zeta^C_{(P)}$ and $\zeta_{(C)} = \zeta^C_{(C)}$ to simplify the notation). Since the child cut pattern $\zeta_{(C)}$ contains at most all elements of $\zeta_{(P)}$, the submatrix $\boldsymbol{A}_{\star\,\overline{\zeta}_{(C)}}$ contains at least all columns of $\boldsymbol{A}_{\star\,\overline{\zeta}_{(P)}}$. The elements $\overline{\zeta}_{(C)} \backslash \overline{\zeta}_{(P)} = \zeta_{(P)} \backslash \zeta_{(C)}$ are thus exactly those columns which are added to the triangularized part of the matrix at this recursion step. A single step of the rank update method is illustrated in Fig. 4.4 (for an example, see Appendix A4.1, Fig. A.17).

It is important to recall that recursive enumeration of adjacent rays aborts early for many node pairs due to the precondition test (invocation of `isPreconditionMet` at line line 2 in Alg. 6: `AddAdjacent`). Therefore, we execute triangularization of the matrix lazily, that is, not before a real rank computation is requested by `areAdjacent` (line 13).

4.1.6 Number Types

Gaussian triangularization with floating point numbers typically uses full pivoting to minimize numeric instability effects (see also Section 3.4). Rank updating uses the same matrix for various rank computations, and instability becomes a serious issue. To circumvent this problem, we stored a copy of the triangularized part before adding new columns and rows. The stored part is recovered before we process another subtree. Neglecting that this procedure uses some more memory, it has still two main disadvantages: (i) restoring significantly affects

performance, and (ii) instability might still be an issue for large or ill conditioned matrices.

Rank updating was therefore implemented with exact arithmetic, using rational numbers with large integer numerators and denominators (see Section 3.3.1). The problem with exact arithmetic is twofold: arithmetic operations such as addition and multiplication are much more expensive, and the integer numbers (numerators and denominators) are possibly growing, even if we choose small integers as pivot elements and continually reduce the fractions. The store/restore approach described above for floating point numbers was also applied to limit the growth, which indeed improved the overall performance. An even better idea is to use residue arithmetic for the rank computations, as explained in the next section.

4.2 Matrix Rank with Residue Arithmetic

> Prof. Gonnet gave me the idea to use residue arithmetic for matrix rank computations. We have presented it in (Terzer and Stelling, 2008).

A powerful method to implement rank computation is to work with integer residues u modulo a prime p. The residues u are constrained to the interval $0 \leq u < p$ using unsigned, and to $-p < u < p$ with signed arithmetic (see for instance Knuth (1997), section 4.3.2). Assuming that the equality matrix $\boldsymbol{A}$ is rational, we compute the residue matrix $\boldsymbol{A'} = [a'_{ij}]$ from $\boldsymbol{A} = [\frac{n_{ij}}{d_{ij}}]$ by multiplying each numerator n_{ij} with the multiplicative inverse of the denominator d_{ij} (modulo p):

$$a'_{ij} = (n_{ij} \bmod p)(d_{ij}^{-1} \bmod p) \bmod p \tag{4.5}$$

Note that multiplicative inverses are only defined for numbers being coprime with p. They are computed using the extended Euclidean algorithm. If any of the denominators is not coprime with p, that is, it is a multiple of the prime—which is very unlikely for large primes—we multiply the whole row of the matrix with p (or a power of it if necessary) and reduce the fractions. We can also avoid multiplicative inverses completely by finding the least common multiple (LCM) of a matrix row or column. Multiplying the row (or column) with its LCM eliminates the denominators without changing the rank of the matrix.

To illustrate the triangularization process, suppose we have chosen pivot element a'_{rc}. The new values a''_{ij} of subsequent rows are then calculated as

$$a''_{ij} = (a'_{ij}a'_{rc} - a'_{rj}a'_{ic}) \bmod p \quad \text{for all} \quad i > r, j \geq c \tag{4.6}$$

resulting in zeros for values a''_{ic} below the pivot element. If we want to use CPU arithmetic operations, the difference of two products in eq. (4.6) has to fit into a register, that is, it has to be below 2^e with $e = 32$ or 64. We can ensure this by choosing our prime $p < \sqrt{2^{e-1}}$.

Suppose that we have reached the point where no non-zero values are left when scanning for the next pivot row/column. This is where rank computation normally stops, but in the residue case, one of the zero elements could theoretically be a multiple of our prime, unequal to zero in the non-residue world. That is, the rank computed by residue arithmetic is at most equal to the real rank. The probability of any non-zero value being zero ($\bmod p$) is p^{-1}, which could cause a problem since we execute many different rank computations. The probability

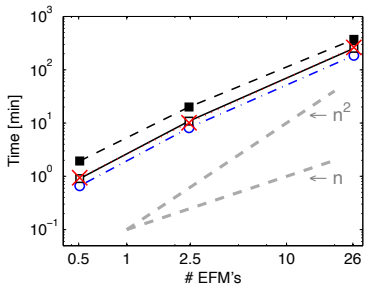


Figure 4.5: Computation times with 4 threads on a system with 2 dual-core AMD Opteron processors for different configurations for an *E. coli* central metabolism network. Adjacency was tested with the combinatorial test (–·○·–), standard rank test (··×··), rank updating with exact arithmetic (–■–) and with residue arithmetic (–□–). Dashed gray lines indicate linear and quadratic computation time. Network configurations, machine specifications and all computation times are given in Appendix A2.1.

can be improved to $\prod p_i^{-1}$ by simultaneously computing the rank modulo different primes p_i, which can indeed be performed in parallel in modern processors by using SIMD instructions (single instruction multiple data). However, in practice, even a single small prime around one hundred never led to a collision.

4.3 Benchmarks

Fig. 4.5 shows a comparison of adjacency test methods, including rank updating with different number types. The tests compute elementary modes for different configurations of an *E. coli* central carbon metabolism network with 106 reactions and 89 metabolites that was also used in (Klamt et al., 2005). Combining recursive enumeration of adjacent rays on bit pattern trees and a combinatorial adjacency test shows best performance for the selected examples. Standard rank computation combined with recursive enumeration of adjacent rays and rank updating with residue arithmetic have similar performance for the shown examples. The superiority of the rank test in Klamt et al. (2005) may be caused by their unoptimized linear search method for the combinatorial test. However, we have also observed cases where rank tests outperformed combinatorial testing, especially for larger networks (data not shown). The performance of the combinatorial test depends on the number of intermediary rays, whereas rank tests only depend on the size of the input matrix. Rank updating is definitely a good choice especially if parallel computation is considered (see Section 5.2). For rank updating, *residue arithmetic* is recommended, actually a method that can be applied to any rank computation algorithm if the matrices are rational (see Section 4.2). However, rank updating performs remarkably well even with exact arithmetic. For our examples, all methods—even exact arithmetic for the large problems—are faster than CellNetAnalyzer / Metatool (Klamt et al., 2005), another common tool for EM computation. For 507,632 / 2,450,787 EMs, we timed 2min 57s / 92min 09s for Metatool (5.0.4) and 1min 30s / 19min 39s for the combinatorial test with one and

40s / 8min 08s with four threads, respectively (see Appendix A2.1 for network configuration details, computation times and machine specifications).

5
Scale Up

As the name suggests, we are now interested in methods expedient for large scale applications. It is a major drawback of the double description method that it is not *compact*, that is, all preliminary results must be kept in memory. Especially for large problems, this can harm the computation even more severely than performance. Hence, we start this chapter with an out-of-core strategy storing the intermediary rays *out-of* the *core memory*. The presented framework is general in a sense that any other application needing simple and fast I/O operations could also utilize it. Compared to conventional database systems, we do not support transactions and sophisticated (e.g. indexed) access strategies. The second section is dedicated to multi-core CPUs, allowing the simultaneous execution of several threads or processes. Most modern CPUs have multiple cores, even simple desktop machines or laptops. We make use of this extra computation power using a shared memory approach. Our first parallelization strategy is a concurrent version of the *candidate narrowing* algorithm based on bit pattern trees that we introduced in the previous chapter. It is a straightforward procedure parallelizing the recursive tree traversion algorithm. Our second variant—the *born/die* approach—tries to break the inherent sequentiality of the double description method. We store the intermediary rays in a born/die matrix, deriving conditions for *pairing jobs* performing the actual generation of new extreme rays based on rays stored in two matrix cells. The two approaches are compared experimentally on a machine with 4 quad-core CPUs. In the last section of this chapter, we briefly discuss possible strategies for distributed computation on multiple host machines. We describe a very simple strategy again using our out-of core framework, but also confess that more work is needed for a useful distributed implementation. Message passing (MPI) should be considered, and we plan to implement a distributed version of the mentioned born/die algorithm. The born/die algorithm might be suitable for distributed computation since it already provides a way to partition the storage of extreme rays. However, it is also important to mention that the double description method—through its data intensity—might simply *not* be the appropriate method for distributed computation.

5.1 Out-of-core Computation

To store intermediary results out of the core memory, a database could be used, but since I/O efficency is critical for the overal performance of the algorithm, we preferred a custom, file-based implementation.

The library is generic, meaning that it can be used by different programs for any kind of data. We organize the storage in a list– (or table–) like structure, as indicated by the following interface:

```
interface Table<Entry> {
    int size()
    Entry get(int index)

    int add(Entry entity)

    void set(int index, Entry entity)
    void swap(int indexA, int indexB)

    void remove(int index)
    void removeAll()
}
```

An `Entry` can be seen as a row in a table representing some data unit of variable or fixed size. Entries are generic types, meaning that they are defined by the user, who has to implement the `Marshaller` interface individually for each entry type:

```
interface Marshaller<Entry> {
    void writeTo(Entry entity, DataOutput out)
    Entry readFrom(DataInput in)
}
```

Note that `DataInput` and `DataOutput` are Java interfaces for I/O of any of the basic data types, offering convenient utility functions to the implementor to store the custom entry data.

Storing fix size entries is straightforward: since the entry size is known, we can compute the offset for an entry—given its index—to access the corresponding bytes in a block data structure. A large byte array or a *random access file* are sample structures suitable for this kind of access. Typical exponents for fixed size entries are integer numbers (of fixed bit length), single or double floating point numbers or strings with a fixed maximum length.

For entries of variable length, such as strings of unlimited length, integers with arbitrary bit length or rationals composed of such integers, a more sophisticated storage approach is needed. We achieve this by using multiple files to store table data: one primary file with a predefined data width, and several secondary files with different power–of–two data widths. To illustrate this concept, consider a table with strings of arbitrary length. The width of the primary file is chosen such that for the majority of the entries, all data can be stored in the primary file. For instance, if names should be stored in our sample table, a width of 10 might be suitable to store most names. To store larger entries, the primary table contains extra space for two indices, one poining to the secondary table, the other to the entry in that table. The

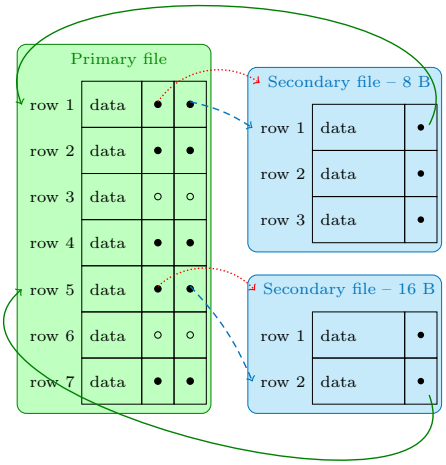


Figure 5.1: File layout used by the *jbase* library. The primary table contains part of the entry data, and two pointers to the secondary table for row data not fitting into the primary table. For small row entries, no record in the secondary table is needed (indicated by empty circles, row 3 and 6 in the example). For larger entries, the smallest possible secondary table that can hold the additional data is used. The primary table contains a pointer to the secondary table (red dotted arrow) and to the corresponding row (blue dashed arrow). The secondary tables contain data not fitting into the primary table, where the data size per row is always a power of 2 (here, tables for 8 and 16 bytes per row are shown). The secondary table contains a pointer back to the primary table row (gray solid arrow).

choice of exponential widths for the secondary tables ensures that (i) only a limited number of secondary files is needed, and (ii) at most half of the storage in the secondary file is unused. The file layout is illustrated in Fig. 5.1; operations and file access characteristics of the library are summarized in Table 5.1.

The library supports concurrency mechanisms, which is important since the current elementary mode implementation uses multiple threads for fine-grained parallelization (see Terzer and Stelling (2008) and Section 5.2), corresponding to a shared-memory approach. Various performance optimizations have been implemented, and the pure Java library is available as *jbase* open source project at `http://www.javasoft.ch`.

Operation	*Files accessed*	*Access operations*
Get Returns an entry identified by index, reading from primary and and secondary file	$1-2$	1 per file
Add Adds a new entry at the end of the table, appending to primary and secondary file	$1-2$	1 per file
Delete Removes an entry by moving the last table entry to the delete position. Implemented as follows: 1. Delete entry in the primary file by moving the last file entry to the position of the deleted entry 2. If the moved entry has data in a secondary file, the back–pointer (to the position in the primary file) is updated 3. If the deleted entry had secondary data, also delete it, again by moving the last entry in the secondary file to the delete position 4. The forward pointer of the entry just moved in (3) is updated in the primary file	$1-3$	$1-4$
Set Replaces an entry. Implemented as insertion with subsequent removal operation: 1. The inserted entry is appended as last entry. 2. Deleting the entry at the target position causes a move of the last entry to the target position.	$1-4$	$2-6$
Swap Two table entries are swapped (entries given by index). 1. Entries in primary files are swapped. 2. If any of the swapped entries has data in a secondary file the back-pointers (pointing to position in primary table) are updated.	$1-3$	$2-4$

Table 5.1: Operations and file access characteristics for variable width tables of the *jbase* library.

5.2 Exploiting Multi-core CPUs

5.2.1 Concurrent Tree All-against-all

The concurrent multi-core version of candidate narrowing has been published in (Terzer and Stelling, 2008). We compare it with an alternative parallelization approach in (Terzer and Stelling, 2009) (accepted).

Most current processors have multi-core architectures, allowing multiple threads or processes to run concurrently. To use this extra computing power, we can use multiple threads and parallelize individual iteration steps. Using the candidate narrowing approach with tree all-against-all as described in Section 4.1.4, we can distribute the recursion calls to multiple threads. However, some of the tree recursions might end much earlier than others, and we would like to keep on using all CPU cores for the whole recursion process. We are therefore maintaining a set of permits to start a new thread: if a permit is available, the recursion is split into two or more parts, one part is executed by the current, the other part(s) by a new—or several new—threads, depending on the number of permits that could be acquired. If no permit is available, all recursions are executed within the current thread, which is actually the normal case. If a thread completes since the leaves of the trees are reached, the permit associated with this thread is released. This triggers the creation of a new thread soon after, since one of the remaining threads will acquire the released permit and parallelize one of its recursions.

Note that since we test availability of permits at every level of the tree recursion, testing and acquisition of permits must be efficient (and of course thread-safe). We implemented permission handling with a semaphore, a concurrency construct usually available for most programming languages and operating systems. In the Java standard library, a thread-safe lock-free semaphore implementation is available.

Another critical point is thread-safety of all data structures that are accessed by multiple threads. For structures that are only read by the threads—such as the trees and the zero sets they contain—concurrency is no big issue. Modified structures need more care, and lock-free implementations are preferable since they are usually faster. For instance, the set `adj` in (Alg. 6: `AddAdjacent`) must allow concurrent append operations, since adjacent pairs may be found by the threads simultaneously. If we use *out-of-core computation* as described in Section 5.1, we must use a concurrent implementation of the `Table` interface.

To further enhance the efficiency of the concurrent tree traversal strategy described above, we create a job pool whenever a new thread is started due to newly acquired permits. The job pool contains executable units that are performing the tree recursion. Instead of partioning the recursion into two or four parts according to the subtree calls (see for instance line 5ff. in algorithm Alg. 6: `AddAdjacent`), we compute smaller recursion units. This ensures that threads that execute jobs from the same pool terminate almost simultaneously, and hence fewer new thread creations are necessary without lowering the rate of parallelization. To provide smaller recursion units, we simply pre-execute the recursions up to a certain depth. In the current implementation, 6 levels are pre-recursed, yielding up to $4^6 = 4096$ units.

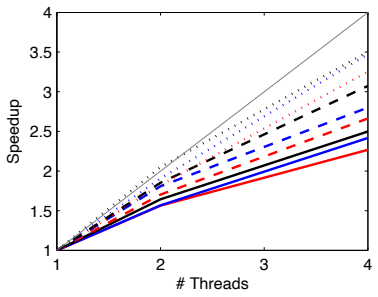


Figure 5.2: Speedup with 2 and 4 threads compared to single thread computation on a system with 2 dual-core AMD Opteron processors. Different configurations for an *E. coli* central metabolism network have been tested, line colors indicate the problem size: red, blue and black stand for 0.5, 2.45 and 26 mio. EFMs, respectively. Line styles correspond to different computation methods: combinatorial test (——), rank updating with exact arithmetic (· · ·) and with residue arithmetic (– –). The thin gray diagonal represents ideal speedup. Details of the network configurations and computation times with machine specifications are given in Appendix A2.1.

A last minor optimization can be achieved if the threads do not release their permit immediately if the job pool contains no further recursion units. Instead, they delay the release operation for a short while, possibly enabling release of several permits simultaneously if another thread has also terminated meanwhile. This has the following advantage: the now available permits trigger a new parallelization operation, that is, the creation of a new job pool with recursion units. If multiple permits are available, several threads can be created at the same time, working off the jobs of one pool. If the same permits are released separately—one shortly after the other—two or more job pools are created, each processed only by two threads.

In Fig. 5.2, we test the efficiency of this parallel approach on a system with two dual core CPUs for different problem sizes. Exploiting multi core CPUs is particularly effective for large problems, where the speedup factors approach the optimum.

Note that this fine-grained parallelization technique is applicable to both adjacency test variants, and it can be applied simultaneously with other parallelization approaches. For instance, we could combine our technique with more coarse-grained methods such as that given in Klamt et al. (2005), where the overall computational task is divided into disjunct subtasks that can be run in parallel. It is also possible to use this approach together with the *born/die* algorithm introduced in the next section.

5.2.2 Born/Die Algorithm

The born/die algorithm has been accepted for publication at PPAM 2009 (Terzer and Stelling, 2009).

Born/Die Matrix and Basic Idea

At iteration t, intermediary extreme rays $\boldsymbol{r}$ with negative value $r_t < 0$ are removed, that is, they *die* at step t. New extreme rays $\boldsymbol{q}$ are created from adjacent ray pairs $(\boldsymbol{r}, \boldsymbol{s})$ with $r_t < 0$ and $s_t > 0$, that is, they are *born* at iteration t. We can thus assign a *born/die* index pair (i, j) to each extreme ray, and for iterations $1, \ldots, n$, we get:

- $i = 0$: initial ray, a column in the initial kernel matrix $\boldsymbol{K}$
- $i \in [1, \ldots, n]$: ray born during iteration $t = i$
- $j \in [0, \ldots, (n-1)]$: ray dies during iteration $t = j + 1$
- $j = n$: ray never dies, final extreme ray, part of algorithm output

Note that $j \geq i$, since a ray can only die *after* being born. Each ray can be associated with a cell in a lower triangular *born/die matrix* (Fig. 5.3B). The $(n+1)$ columns and rows account for born and die indices, respectively. Column 0 contains initial, row n final extreme rays. The die index j of a ray $\boldsymbol{r}$ is determined as follows:

$$j = \begin{cases} n & \text{if } r_k \geq 0 \quad \forall k \in [1,\, n] \\ \min\{k : r_k < 0,\ k \in [1,\, n]\} - 1 & \text{otherwise} \end{cases} \tag{5.1}$$

The general idea of the algorithm exploits that every cell in the born/die matrix contains rays dying at a certain iteration step (except for cells in the last row). The matrix is filled up by pairing the dying rays with rays from other cells. The pairing jobs can be run concurrently, but we have to be careful: newly generated rays are again stored in the born/die matrix, and a pairing job should not be started before the involved cells have gathered all rays.

Definition 5.1. *The generation process of new rays $\boldsymbol{q}$ from rays $\boldsymbol{r}$ in cell (i, j), dying at iteration $j+1$, and rays $\boldsymbol{s}$ adjacent to $\boldsymbol{r}$ from a cell (k, l) with $s_{j+1} > 0$, is called* pairing job *or* pairing task $\theta_{(i,\,j)}\Big((k,\, l)\Big)$. *We call the cell (i, j)* pairing cell, *and (k, l)* partner cell *of θ, and we say that pairing cell (i, j)* initiates *pairing jobs $\theta_{(i,\,j)}(\star)$.*

Cell Stages

Before we can use the rays of a cell, we must ensure that the cell has already collected all rays to be stored in it ((A)ccumulating stage). When the cell has collected all rays, it enters the (B)earing stage. After actively triggering pairing jobs, it is still involved in pairing jobs

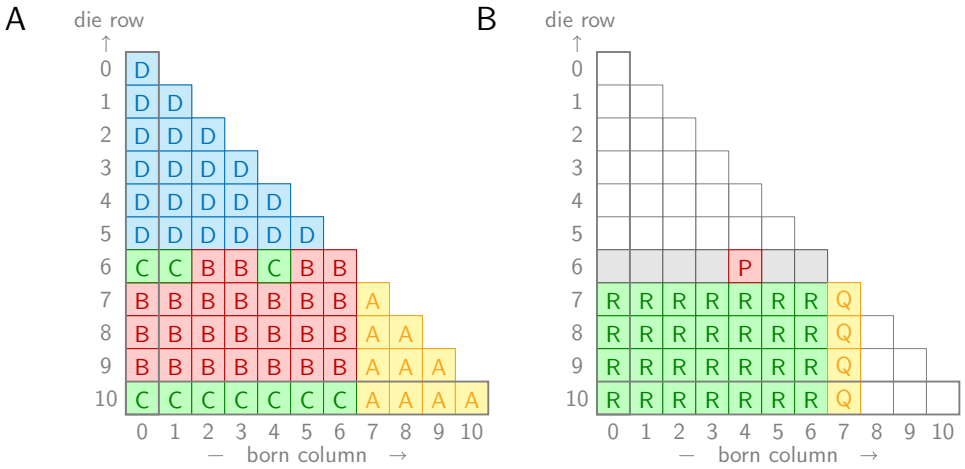


Figure 5.3: **A:** Hypothetical intermediary cell stages (A)ccumulating, (B)earing, (C)ollaborating and (D)one. Collaborating cells can only occur in the final row n, and in the first bearing row ρ (here: $\rho = 6$). Rays in cells of the final row never die, and are thus never bearing; cells in rows between ρ and n have pending partner cells in column $\rho + 1$, which are still accumulating. Cells above row ρ are always in the Done stage, cells with column index $> \rho$ are always accumulating. **B:** Born/die matrix with an exemplarily red pairing cell (P) and yellow destination cells (Q) for pairing jobs involving P. All green cells (R) are partner cells for P. Note that any other pairing cell in row 6 has the same destination and partner cells as P.

triggered by other bearing cells ((C)ollaborating stage). Finally, if the rays in the cell are not used anymore, they are dropped and the cell is in the (D)one stage. The cell stages (A)-(D) are summarized in the table below, and an exemplarily born/die matrix with cells at different stages is shown in Fig. 5.3A.

(A)ccumulating: The cell is collecting rays generated by pairing jobs, but is not involved in pairing jobs in a constitutive way.

(B)earing: The cell is initiating pairing jobs as pairing cell. It might also be involved in other pairing jobs as partner cell.

(C)ollaborating: The cell is involved in pairing jobs as partner cell, but has completed the initiation of pairing jobs.

(D)one: The cell is not involved in pairing jobs anymore. If it is not a final row cell, the rays previously stored in the cell have been dropped.

Conditions for Cell Stage Transitions

Lemma 5.1 (Pairing Lemma)**.** *Let $t = j+1$, and $\boldsymbol{r}$ be a ray in cell $(i,\, j)$ with $i \in [0,\, n-1]$, $j \in [0,\, n-1]$. Then, $\boldsymbol{r}$ is paired with adjacent rays $\boldsymbol{s}$ with $s_t > 0$ found in cells*

$$\Pi_{(i,\,j)} = \{(k,\, l) : k \in [0,\, j]\ ,\ l \in [j+1,\, n]\}$$

Proof. From the main lemma of the double description method (see Fukuda and Prodon (1995)), we know that at iteration t, rays $\boldsymbol{r}$ with $r_t < 0$ are paired with adjacent rays $\boldsymbol{s}$ with $s_t > 0$. Here, we also have $r_t < 0$ since $\boldsymbol{r}$ is stored in row j, that is, it dies at iteration $t = j+1$. It remains to prove that partner rays $\boldsymbol{s}$ can only be found in the cells $\Pi_{(i,\,j)} = \{(k,\, l)\}$. Clearly, $k \leq j$, otherwise, the partner ray would not yet be born at iteration t. Furthermore, $\boldsymbol{s}$ must die *after* step t, that is, $l \geq j+1$. □

Lemma 5.2 (Dependency Lemma)**.** *Every extreme ray $\boldsymbol{q}$ stored in cell $(i,\, j)$ with $i \in [1,\, n]$, $j \in [1,\, n]$ has been generated from an ancestor ray in*

$$\Psi_{(i,\,j)} = \{(k,\, l) : k \in [0,\, i-1],\ l = i-1\}$$

Proof. Since $i \geq 1$, a ray $\boldsymbol{q}$ stored in cell $(i,\, j)$ must have been generated from a dying ray $\boldsymbol{r}$ with $r_t < 0$ and a surviving ray $\boldsymbol{s}$ with $s_t > 0$. The dying ancestor must be dying during the same iteration step as ray $\boldsymbol{q}$ is born, that is, at iteration $t = i$, and we have $l = i-1$. It is not relevant when ray $\boldsymbol{s}$ was born, and hence $k \in [0,\, i-1]$. □

The Dependency Lemma 5.2 not only tells us when to start pairing jobs, but also where the new rays created from those jobs will be stored. The rays stored in cell $(i,\, j)$ are connected to ancestor cells only through column index i, and all ancestor cells with dying rays are contained in row $i-1$ (Fig. 5.3B). This leads to the following corollary:

Corollary 5.1 (Destination Cells)**.** *Rays $\boldsymbol{q}$ generated from a pairing job $\theta_{(\star,\,i)}(\star)$ initiated by a pairing cell from row i are placed in column $i+1$.*

Stages (B) and (C) are *active* since cells are involved in pairing jobs during those stages. The following proposition yields the condition for the activation of a cell.

Proposition 5.1 (Activation Condition: transition A–B)**.** *Cells in column i with $i \in [1,\, n]$ have collected all rays if all pairing jobs $\theta_{(\star,\,k)}(\star)$ involving a pairing cell from row $k = i-1$ have completed.*

Proof. Follows immediately from Dependency Lemma 5.2. □

The sequential algorithm keeps rays which are feasible at least until the next iteration. Infeasible rays are dropped after the generation of new rays with adjacent partner rays. To the same effect, we derive a deallocation condition for rays stored in the born/die matrix:

Proposition 5.2 (Deallocation Condition: transition C–D)**.** *Rays stored in cells of row j with $j \in [0,\, n-1]$ can be dropped if all pairing jobs $\theta_{(\star,\,k)}(\star)$ initiated by a pairing cell from row $k \leq j$ have completed.*

Proof. As a precondition, the cell in row j is not used as pairing cell, and thus also not as partner cell as implied by Pairing Lemma 5.1. Consequently, the rays in the cell are not used in any pairing job, and because $j < n$, they are also not final rays, that is, they can be dropped. □

To account for the last transition from (B) to (C), we count the completed pairing jobs initiated by a cell, and introduce a *remaining job counter* Θ as follows:

$$\begin{aligned} \Theta\Big((i,\,j)\Big) &= \text{remaining jobs, initiated by } (i,\,j) \\ &= |\theta_{(i,\,j)}(\star)|_{\text{total}} \;-\; |\theta_{(i,\,j)}(\star)|_{\text{completed}} \\ &= (j+1)(n-j) \;-\; |\theta_{(i,\,j)}(\star)|_{\text{completed}} \end{aligned} \tag{5.2}$$

The counter $\Theta\Big((i,\,j)\Big)$ is initialized with $(j+1)(n-j)$ according to Pairing Lemma 5.1, and decremented whenever a pairing job $\theta_{(i,\,j)}(\star)$ terminates. We can also derive the activation and deallocation conditions from Θ by computing the minimum index ρ of a row with at least one cell with remaining jobs:

$$\rho \;=\; \begin{cases} n & \text{if} \quad \Theta\Big((i,\,j)\Big) = 0 \quad \forall\, i,j \\ \min\{j : \exists i : \Theta\Big((i,\,j)\Big) > 0\} & \text{otherwise} \end{cases} \tag{5.3}$$

Our observations are summarized in Table 5.2.

Object	*Stage*	*Condition*
$(i,\,j)$	(A)ccumulating	$i > \rho$
$(i,\,j)$	active, (B) or (C)	$i \le \rho \le j$
$(i,\,j)$	(B)earing	$(i,\,j)$ is active and $\Theta\Big((i,\,j)\Big) > 0$
$(i,\,j)$	(C)ollaborating	$(i,\,j)$ is active and $\Theta\Big((i,\,j)\Big) = 0$
$(i,\,j)$	(D)one	$j < \rho$
algorithm	terminates	$\rho = n$

Table 5.2: Conditions for cell stages and algorithm termination

Implementation Aspects

For the concurrent execution of pairing jobs, operations modifying the cell counter Θ must be thread-safe. In our Java implementation, we use variables from the atomic package of the standard library. The classes `AtomicInteger` and `AtomicIntegerArray` offer atomic,

thread-safe and lock-free `decrementAndGet` and `compareAndSet` operations. An additional helper variable $\Omega(j)$ for $j \in [0, n-1]$ is initialized with $j+1$, accounting for the number of cells in row j which are still (or not yet) bearing. The implementation of the conditions in Table 5.2 is illustrated in pseudo-code procedure `PairingJobTermination`. In addition to the state variables $\Theta\Big((i,\, j)\Big), \Omega(j)$ and ρ used to control cell stage conditions and algorithm termination, also adding rays to an accumulating matrix cell and job queue operations must be thread-safe.

Procedure `PairingJobTermination`

```
desc: Called before terminating a concurrently executed pairing job θ_(i,j)(⋆)
begin
    jobsLeft ← decrementAndGet Θ((i, j))
    if (jobsLeft = 0) then
        cellsLeft ← decrementAndGet Ω(j)
        while (cellsLeft = 0) do                          /* no bearing cells left in row j  */
            if (ρ ← compareAndSet ρ = j, j + 1) then                          /* j was ρ  */
                if (ρ = n) then
                    terminate;
                else
                    queue new pairing jobs involving a cell of the activated column ρ
                    j ← j + 1
                    cellsLeft ← Ω(j)
                end
            else
                cellsLeft ← −1                                  /* j was not ρ ⇒ break loop */
            end
        end
    end
end
```

We have now all elements for the born/die algorithm:

1. Initialize: $\Theta\Big((i,\, j)\Big) = (j+1)(n-j)$ and $\Omega(j) = j+1$
2. Add rays from the initial kernel matrix $\boldsymbol{K}$ to column 0 in the born/die matrix.
3. Setup an empty job queue.
4. Start concurrent "worker" threads, processing queued jobs if available.
5. Activate column 0 by setting $\rho = 0$ and queue new pairing jobs involving only cells from column 0. All further pairing jobs will be added to the queue by terminating jobs as illustrated in (Alg. 7: `PairingJobTermination`)
6. Await termination condition $\rho = n$.
7. Terminate worker threads and return resulting extreme rays stored in matrix row n.

5.2.3 Experimental Results

We have tested the two approaches on five large problems from combinatorics and systems biology. Two examples are part of the `cddlib` sample files (Fukuda, 1996): `ccp7` (116,764 facets of a 0/1 polytope in 21 dimensions, constructed as convex hull of edge incidence vectors of cuts in the complete graph over 7 vertices) and `mit71-61` (3,149,579 vertices of a polytope in 60 dimensions with 71 hyperplanes). Three examples concern elementary modes (EMs) of central metabolism of *E.coli*, namely `coli-S2` (507,632 EMs corresponding to extreme rays of a polyhedral cone in 64 dimensions, configuration S2 in Terzer and Stelling (2008)), `coli-S3` (2,450,787 EMs, 68 dimensions) and `coli-L1` (26,381,168 EMs, 76 dimensions).

Both approaches performed similarly well, even if the born/die approach seems to have a larger overhead especially with few threads (Table 5.3). This is probably caused by the numerous fine-grained pairing jobs, but its scale-up behavior is at least as good. Both variants also compete well with the sequential implementations `cddlib` (Fukuda, 1996) and `4ti2` (4ti2 team, 2006), even if a comparison is difficult. The systems biology examples did not terminate with `cddlib` and `4ti2`, possibly due to missing pre-processing e.g. to compress input matrices (Gagneur and Klamt, 2004).

Problem	*Per-step*					*Born/die*					*cddlib*	*4ti2*
Threads	*1*	*2*	*4*	*8*	*16*	*1*	*2*	*4*	*8*	*16*	*(1)*	*(1)*
ccp7	1,117	615	348	164	137	2,088	1,100	610	268	195	22,652	525
mit71-61	1,084	493	369	260	226	1,286	637	385	249	195	>4d	13,253
coli-S2	50	31	21	15	17	78	39	22	16	18	–	–
coli-S3	501	327	238	157	186	814	432	290	151	163	–	–
coli-L1	7,072	4,470	2,659	2,121	2,614	11,296	5,950	4,325	3,395	2,556	–	–

Table 5.3: Computation times for *per-step* and *born/die* approach with 1,2,4,8 and 16 threads (double precision arithmetic, adjacency test with rank updating and modulo arithmetic), and sequential single-threaded times for *cddlib* (V0.94f with double arithmetic (Fukuda, 1996)) and *4ti2* (V1.3.2 with 64 bit integers (4ti2 team, 2006)). The tests are run on a 64bit linux 2.6.18 machine with 4 Intel Xeon X7350 Quad Core CPUs with 2.93GHz. We used a Sun Java 64-Bit Server VM (1.6.0_11) with at most 8GB memory, except for coli-L1 (32GB). All times are in seconds unless otherwise indicated.

5.3 Distributed Computation

During an iteration step of the double description method, new elementary modes (EMs) are born from adjacent modes of the previous step. To find the adjacent EM pairs, the *recursive enumeration* approch is used (see Section 4.1.4, or Appendix A4.1 for an illustrative example). Two bit pattern trees are created, one for EMs with positive and one for those with negative flux value for the reaction that is currently processed. When we descend one step of both binary trees, we end up with at most four recursive calls if both trees have two subtrees (see line 5ff. in Alg. 6: `AddAdjacent`).

A very simple parallelization approach executes the (up to) four recursions in separate threads or processes. Such a strategy has been described for threads in a multi-core environment in Section 5.2. Within the same virtual machine, it is easy to check whether the

current process should be forked and execute the subtask in separate threads. However, in a distributed environment, the processes are executed on different machines, and communication and synchronization costs are much higher. Still, we can use a similar strategy to parallelize the program, but in a more static sense. Insteady of dynamically checking whether we should fork, we statically divide the overall taks into subtasks. We could for instance only parallelize the first recursion, yielding four independent subtasks. Using two recursion levels leads to at most 16 subtasks, and so on. We have implemented such a strategy and tested it on computer clusters.

Note that only the phase of adjacency testing is parallelized, meaning that we re-sychronize all processes between the iteration steps. One machine takes the role of the master, all other machines are clients. To exchange information between client and master processes, all host machines have access to a common network device, where we store the files containing intermediary results needed for the computation. We use the out-of-core framework described in Section 5.1 for that purpose. Note that only the master process needs write access to files containing intermediary results. The adjacent modes found by each client are sent back to the master process.

We summarize the distributed computation procedure for iteration step i, corresponding to reaction R_i, as follows:

1. The computation uses out-of-core memory computation as described in Section 5.1, that is, all intermediary modes are already stored in files.

2. The master process performs the following operations to initialize the current iteration step:
   a) The intermediary modes are partitioned into three files, containing modes with zero, positive and negative flux value r_i for reaction R_i
   b) Two bit pattern trees are constructed, one for the modes with positive flux values, and one for those with negative flux values at reaction R_i. The two trees are stored in files, also using the out-of-core framework described in Section 5.1.
   c) Information corresponding to the current iteration step is stored in a step configuration file.
   d) The master process opens a TCP socket on a free port and waits for incoming requests from clients.

3. Now, the child processes are started on all machines that have been configured for distributed computation. The file names for intermediary modes, bit pattern trees and step configuration are passed as command line arguments, as well as host name and port to connect to the master process.

4. Each child process reads the step configuration file and opens intermediary mode and pattern tree files. Then, a connection is opened to the master process, and an integer is returned by the master process identifying the first job for the child. For instance, if only one recursion level is parallelized, the job ID's $0 \ldots 3$ would identify the subtree recursions $(L, L), (L, R), (R, L)$ and (R, R), where L and R stands for left and right subtree, respectively.

5. If the client process finds two adjacent modes, the new-born mode is constructed and sent to the master process, who writes it to file.

6. After completing a job, the client process requests the next job ID from the master, until no more jobs are available. In the latter case, the client process terminates.

7. The master process uses an active node counter, initialized with the number of client processes. The counter is decremented if a client requests a new job and receives the "end-of-jobs" answer. If the counter reaches zero, all clients have terminated, and the master process closes the socket and continues with the next iteration step.

This simple parallelization approach has several drawbacks: first, distributing the computation to host machines only pays back for ellaborative iteration steps. Second, it can only be used in combination with out-of-core computation, which is of course much slower than in-memory calculations. Third, the communication and synchronization overhead is immense, and the method does not scale well, for instance because we depend on the capacity of the master process who collects all adjacencies computed by the clients. We could reduce communication overhead by starting client processes at the beginning of the computation, instead of restarting them for each iteration. But we have still synchronization points between the iterations, and since the computation is extremely data intensive, distributing and re-collecting data will always be a problem. With this primitive implementation, we could almost never outperform our multi-core implementation. Hence, the method needs definitely more refinement and a more efficient implementation. Use of a message passing interface (MPI) should be discussed, and we would definitely like to implement a distributed version of the *born/die* algorithm invented in Section 5.2.2. It remains an open questions whether the double description method can be implemented and run efficiently in a distributed environment.

6 Application

6.1 Defining an Optimal Metabolic Operation Point

Flux variability analysis for suboptimal flux vectors has been presented in Terzer and Stelling (2008). We have an ongoing collaboration with R. Schütz from the Institute of Molecular Systems Biology at ETH Zurich, who found similar results with another approach, also confirmed experimentally by flux measurements.

6.1.1 Introduction

A major disadvantage of single-objective optimization techniques such as flux balance analysis (FBA) is the restriction to *one* optimal flux vector. Robustness, for instance reflected by a certain degree of flexibility, is disregarded. To account for this, we computed elementary modes (EMs) for different configurations of the central metabolism network of *E.coli* (Stelling et al., 2002) with growth on glucose. We normalize the modes and compute the efficiency for every EM, reflected by *biomass* production in our application. Sorting the EMs according to their efficiency level, we analyze the variability of individual reactions, if only EMs above a certain optimality threshold are considered. The key idea of this approach is that a cell has to find an optimal balance between competing objectives. For instance, fast growth is desirable for the cell as long as there are sufficient nutrients. However, robustness is also essential to react rapidly to environmental changes, or to cope well with variations. We approach robustness by analyzing flux variability of the individual reactions—first for optimal biomass production—then with increasing suboptimality. Interestingly, variability increases rapidly if we relax the optimality condition, but saturates close to optimal yield. We hypothesize that this saturation

point might be an *optimal metabolic operation point* for the cell—and strengthen our findings first by computing flux predictions using our operation point, and second by comparing our approach with work of Schuetz et al. (2009). They use multiple objectives and compare experimentally measured flux values with *pareto optimality*, finding that all measured values are close to the optimality surface in the three dimensions of their objectives.

6.1.2 Methods

We used different configurations of the central metabolism network of *E.coli* (Stelling et al., 2002) with growth on glucose. Besides uptake of glucose, six central amino acids from different biosynthesis families are available, only one at a time in a first experiment. This results in six sets of elementary modes for the amino acids *alanine*, *aspartate*, *glutamate*, *histidine*, *phenylalanine* and *serine*. The sets contain between 321,431 (alanine) and 858,648 (glutamate) EMs. In a second evaluation, we computed the set of EMs for combined uptake that consists of 26,381,168 modes (Terzer and Stelling, 2008).

To measure efficiency for growth on different media, we have to define a normalization method that allows for a fair comparison. A common approach is to normalize flux vectors according to the number of uptaken C atoms. Here, we use a different method, measuring the reduction level of the substrates (Dauner et al., 2002):

$$Y_{b/e} = \frac{v_b}{\sum_{s \in S} v_s \cdot Y_{e/s}} \tag{6.1}$$

$$Y_{e/s} = 4c_s + h_s - 2o_s - \frac{n_s}{n_n}(4c_n + h_n - 2o_n) \tag{6.2}$$

$Y_{b/e}$ denotes the biomass yield (dry cell mass produced per electron equivalents on substrates $s \in S$), v_b (v_s) indicate flux rate of biomass production (substrate uptake) and $Y_{e/s}$ is the number of electrons available from complete combustion of the substrate to CO_2. The terms c_i, h_i, o_i and n_i stand for carbon, hydrogen, oxygen and nitrogen atoms of substrate ($i = s$) and nitrogen source ($i = n$).

After assigning a yield value to every elementary mode of a set, we sort the EMs according to efficiency. We define the suboptimality threshold as the maximally allowed decline of the yield from optimal yield. Using different suboptimality thresholds, we only account for EMs within the allowed suboptimality range. As a measure for robustness, we compute the flux variability for each reaction, simply using the coefficient of variation (CV), the standard deviation relative to the mean value. All modes are weighted equally, independent of yield optimality. To define a global measure for flux variability, we count reactions above a certain CV threshold. For instance, we count how many reactions have a relative standard deviation of more than 50%. The CV threshold is of course somewhat arbitrary, but different choices between 10% and 100% did not have a large effect on the outcome; in particular, the location of the saturation point did not change significantly (see Appendix A3.1).

To provide evidence that the cell might indeed be operating around the saturation point, we fix a suboptimality threshold close to saturation (here, 2% suboptimality was chosen, see Fig. 6.1). We compute flux prediction vectors by computing the mean value and standard deviation for all flux values taking into account only EMs above the threshold. We use a part

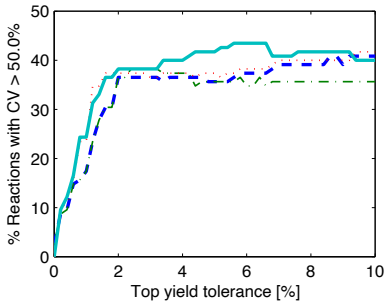


Figure 6.1: Coefficient of variation (CV) for reaction fluxes of different uptake configurations, considering only top yield EMs. The configurations are: glucose without amino acid uptake (dashed line), with phenylalanine (dotted), with glutamate (dash-dotted) and with all selected amino acids (solid line).

of the metabolism for which it is particularly difficult to predict flux values, namely the TCA cycle, and compare the predicted flux ratios with experimental data.

6.1.3 Results

As shown in Fig. 6.1, we observed no flux variation for zero deviance from the top biomass yield since single EMs reach optimality. With increasing suboptimality, the number of reactions with coefficient of variation (CV) over 50% grows rapidly. Around 2% below top yield, reaction variation reaches a saturation point. Interestingly, we find a similar saturation point for different CV threshold values (see Appendix A3.1 for other thresholds). The cell can thus achieve high flexibility (robustness) already for a small decrease in metabolic efficiency. We hypothesize that the *optimal metabolic operation point* for the cell must be around this saturation point.

To gain some evidence for our hypothesis, we compute flux values based on a fixed suboptimality threshold: accounting for EMs with biomass yield at most 2% below top yield, we compute mean and standard deviation for each flux value, each EM weighted equally. Here, we focus on the tricarboxylic acid cycle pathway under glucose scarcity, on the one hand because experimental data is available for these conditions. On the other hand, correct flux predictions are difficult especially for the TCA cycle, and standard optimization approaches like flux balance analysis (FBA) usually produce inaccurate estimates. According to a comprehensive study by Schuetz et al. (2007), biomass yield is an appropriate objective function in particular for growth under nutrient scarcity. The predicted (relative) fluxes agree well with experimental data under conditions of glucose hunger (Fischer and Sauer, 2003). For instance, flux values into the glyoxylate shunt closely match the measured values (Fig. 6.2). Here, LP based approaches cannot produce similar predictions, since a single EM shows highest biomass yield and thus represents the optimal value for FBA implicitly. Specifically, the optimal elementary mode does not explain the flux into the glyoxylate shunt. Hence—at least under conditions of

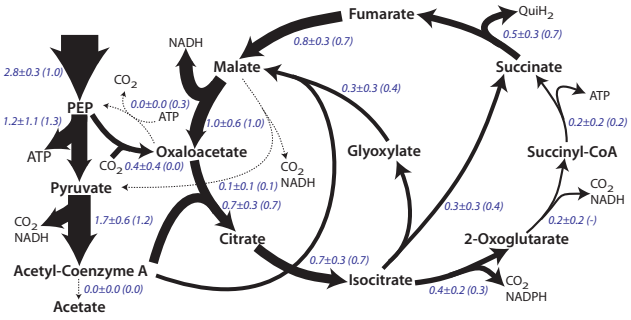


Figure 6.2: Flux predictions for the tricarboxylic acid cycle based on elementary modes (EMs) with high biomass yield. Arrows are drawn proportional to the predicted fluxes, normalized with the largest value in the cycle (malate dehydrogenase flux). Mean value and standard deviation of top 2% yield EMs (267 of 178,575 EMs) are indicated. Values in parentheses denote experimentally determined flux values from (Fischer and Sauer, 2003), again normalized to the flux through malate dehydrogenase. Most measured flux values comply well with the predictions: only two values lie outside one standard deviation compared to predicted values.

glucose hunger—not only maximization of biomass is important, but also a certain degree of robustness—here represented by different EMs leading to flux variability. These findings are not possible with single objective optimization. Furthermore, optimization techniques only provide the optimal value, and the resulting flux vector is not unique if alternate optima exist. Elementary mode analysis is more expensive to apply, but also yields alternate optima and sub-optimal modes and does not depend on sometimes arbitrary objective functions.

6.1.4 Discussion

We have analyzed variability of individual reaction fluxes considering all elementary modes above a certain yield threshold. For all investigated datasets, we observed a steep rise in variability if optimality was relaxed, considering also sub-optimal EMs. However, already at 2% below optimal yield, the increase in variability reaches a saturation point, and only marginal or no additional variability can be achieved if also additional EMs with lower yield values are allowed (see Fig. 6.1). We concluded that it might be a good strategy to choose the metabolic operation point close to this saturation value. Using the 2% suboptimality threshold as a working hypothesis, we derived flux predictions using a simple averaging strategy for EMs complying with the yield constraint. Our flux predictions fitted well with exterimental data (Fig. 6.2).

In collaboration with Robert Schütz from the Molecular Systems Biology Institute at ETH Zurich, we realized that we found similar results independently and with different methods. Schuetz et al. (2009) use three different objective functions to perform flux balance analysis. Using a single objective does not always produce accurate predictions, as was already shown in (Schuetz et al., 2007). However, using a trade-off between maximization of biomass, maxi-

mizal energy (ATP) production and minimal sum of absolute flux values (reflecting efficiency through a minimum of allocated resources to produce the enzymes), they could show that all experimentally determined flux values are close to the so-called *pareto surface* of optimality. Pareto optimality stands for the concept of contradicting objective functions: if a point lies in the pareto surface, an objective value can only be improved (if at all) on the cost of other objectives. In their study, they computed the distance from the pareto surface for 45 measured flux distributions from *E.coli* grown aerobically, anaerobically and with nitrate under nutrient-excess batch conditions and 13 glucose- and nitrogen-limiting chemostat conditions using ^{13}C-labeling experiments. The distances from the pareto surface are significantly closer than random, and they define a metabolic optimality space of near-optimal and optimal combinations of biomass and ATP yield and sum of absolute fluxes with a maximal distance of 0.025 units. A direct comparison with our 2% value is of course difficult, since we are only considering one objective. Furthermore, Schuetz et al. (2009) shrink the flux space by considering only biomass producing fluxes without futile cycles. Still, the magnitue of suboptimality is similar to our observation. Hence, it will be very interesting to expand our tests and also measure yield by means of ATP production and sum of fluxes. Notably, both objectives can be transformed into linear constraints or objectives—a pre-requisite to compute elementary modes.

6.2 Feasibility of Elementary Modes

The following section is joint work with S. J. Jol, A. Kümmel and M. Heinemann of the Institute of Molecular Systems Biology at ETH Zurich.

6.2.1 Introduction

Characterizing the flux space with *elementary modes (EMs)* is based on two assumptions to constrain the flux values: quasi steady state (eq. 2.62) and irreversibility of certain reactions (eq. (2.63) or (2.64)). However, thermodynamic restrictions on flow directions can involve more than one reaction. By combining *elementary modes (EMs)* with metabolome data, the feasibility space for possible flux values can be further constrained. We address this question by computing large EM sets for a compartmentalized metabolic network of *Saccharomyces cerevisiae*. All EMs are tested for feasibility based on the second law of thermodynamics and on measured metabolite concentrations. Similar approaches have been proposed to constrain *flux balance analysis* (FBA, see Section 2.1.2.2) with additional constraints to ensure that only thermodynamically feasible solutions are found (Kümmel et al., 2006; Hoppe et al., 2007; Henry et al., 2007). Unfortunately, these additional constraints are non-linear, and the resulting optimization problem usually involves integer and/or boolean variables, yielding non-convex optimization problems. Besides the complexity to solve such problems, they can also only provide one optimal flux vector, which strongly depends on the assumed objective (such as maximal biomass yield). Hence, we first compute the elementary modes (EMs) to derive an initial feasibility space and subsequently apply the additional constraints to the computed EM set. We show that we can exclude EMs proven infeasible without loosing any (possibly

composite) valid flux vector; in other words, we can shrink the solution space found by EM computation by applying additional thermodynamic constraints.

6.2.2 Methods

The flux direction for a chemical reaction is related to Gibbs free energy as follows:

$$\begin{aligned} \Delta_r G_j < 0 \quad &\Longrightarrow \quad v_j > 0 \\ \Delta_r G_j > 0 \quad &\Longrightarrow \quad v_j < 0 \end{aligned} \tag{6.3}$$

Here, $\Delta_r G_j$ denotes Gibbs free energy for a reaction j, and v_j is the flux through this reaction. The free energy of a reaction can be calculated from Gibbs energies of formation of the participating reactants ($\Delta_f G$):

$$\boldsymbol{\Delta_r G} = \boldsymbol{S}\,\boldsymbol{\Delta_f G} \tag{6.4}$$

Note that $\boldsymbol{\Delta_r G}$ and $\boldsymbol{\Delta_f G}$ are vectors of length r and m for a metabolic network with stoichiometric matrix $\boldsymbol{S} \in \mathbb{R}^{m \times r}$ containing m metabolites and r reactions. Gibbs energy of formation for metabolites can be derived from its standard Gibbs energy and the thermodynamic activity. The *network-embedded thermodynamic (NET) analysis* applied here uses measured metabolite concentrations to derive energy of formation for each metabolite (Kümmel et al., 2006). Allowing for a certain tolerance due to measurement errors, Gibbs formation energies are constrained to an interval:

$$\boldsymbol{\Delta_f G}_{min} \leq \boldsymbol{\Delta_f G} \leq \boldsymbol{\Delta_f G}_{max} \tag{6.5}$$

Given a flux vector $\boldsymbol{v}$, the NET tool performs a feasibility analysis, adding additional flux dependent constraints for every non-zero flux value v_j, restricting Gibbs free energy for reaction j according to eq. (6.3). This yields patterns of reaction directionalities that are not feasible, even if all individual reaction directions are very well possible.

Let us assume for the moment that we have a tool that decides whether a certain flux vector is feasible or not, based on the direction pattern of the flux vector. If we apply the tool to special vectors like elementary modes (EMs), it would be desirable that we could then draw feasibility conclusions also for other flux vectors, for instance for those derived from elementary modes. Ideally, we would like to remove infeasible EMs and shrink the flux cone accordingly. But is this really what we want, i.e. can we still generate all feasible flux vectors from the remaining feasible EMs? The answer is yes, under certain conditions, and to work towards a proof, we have to concretize some definitions.

Definition 6.1. *The* flux pattern $\phi(\boldsymbol{v})$ *of a flux vector* $\boldsymbol{v}^{1\times r}$ *are the indices* $j \in \{1, \ldots, r\}$ *associated with nonzero fluxes, multiplied with the signum of the flux value. More formally,*

$$\phi(\boldsymbol{v}) = \{\, j \cdot \operatorname{sgn}(v_j) \mid v_j \neq 0 \,\} \quad \text{where } \operatorname{sgn}(v_j) = \begin{cases} -1 & \text{if } v_j < 0 \\ 0 & \text{if } v_j = 0 \\ +1 & \text{if } v_j > 0 \end{cases} \tag{6.6}$$

Note that irreversible reactions can be seen as special case for feasibility conditions, preventing a flux direction of a single reaction. Hence, irreversibility constraints classify flux patterns of size one as infeasible. We can apply conditions for such simple infeasibilities *a priori* to

the computation, usually stated as non-negativity constraints for the corresponding variables. More complex conditions are applied *after* the computation of the elementary modes. For those conditions, we need a formal definition of *feasibility classification*:

Definition 6.2. *A function* $f(\boldsymbol{v}) : \mathbb{R}^d \to [0,1]$ *is called* feasibility classifier *for flux vectors* $\boldsymbol{v}^d$ *if for any flux vector* $\boldsymbol{v}'$ *with* $\phi(\boldsymbol{v}') \supseteq \phi(\boldsymbol{v})$*, infeasibility* ($f = 0$) *of* $\boldsymbol{v}$ *implies infeasibility of* $\boldsymbol{v}'$*:*

$$\phi(\boldsymbol{v}') \supseteq \phi(\boldsymbol{v}) \ , \ f(\boldsymbol{v}) = 0 \quad \Longrightarrow \quad f(\boldsymbol{v}') = 0 \tag{6.7}$$

The NET tool meantioned above is an example for a feasibility classifier, but the following is true for *any* classifier conforming with Def. 6.2: infeasible elementary modes can be removed from the generating matrix without losing feasible flux vectors. Equivalently, we can compose any feasible flux vector from feasible elementary modes, as stated formally by the following theorem:

Theorem 6.1. *Let* f *be an feasibility classifier according to Def. 6.2, and let* $\mathcal{P}$ *be a flux cone generated by* n *elementary modes* $\boldsymbol{e}_j$ *(see Def. 2.39 2.40). Let us furthermore denote by* F *be the set of indices associated with feasible EMs, i.e.* $F = \{\, j \mid j \in \{1,\ldots,n\} \ , \ f(\boldsymbol{e}_j) = 1\}$*. Then, any feasible flux vector* $\boldsymbol{v} \in \mathcal{P}$ *can be composed solely from feasible elementary modes* $\boldsymbol{e}_k$ *with* $k \in F$*. More formally, we have:*

$$\boldsymbol{v} \in \mathcal{P}, f(\boldsymbol{v}) = 1 \quad \Longrightarrow \quad \forall k \in F, \exists\, \lambda_k \geq 0 \quad \textit{such that} \quad \boldsymbol{v} = \sum_{k \in F} \lambda_k \boldsymbol{e}_k \tag{6.8}$$

6.2.3 Proof

Let us first establish an important relationship between flux patterns (Def. 6.1) and zero sets (Def. 2.28, eq. (2.39)) associated with flux vectors:

Proposition 6.1. *Let* $\boldsymbol{v}_1$ *and* $\boldsymbol{v}_2$ *be flux vectors in* $\mathbb{R}^r$ *with associated flux patterns* $\phi_1 := \phi(\boldsymbol{v}_1)$ *and* $\phi_2 := \phi(\boldsymbol{v}_2)$*. Let* $\boldsymbol{v}'_1$ *and* $\boldsymbol{v}'_2$ *be the same vectors projected to the augmented dimensionality space of the EM cone as implied by Def. 2.39, and* $\zeta_1 := \zeta(\boldsymbol{v}'_1)$ *and* $\zeta_2 := \zeta(\boldsymbol{v}'_2)$ *be the corresponding zero sets given by eq. (2.39). Then, the following holds:*

$$\phi_1 \supseteq \phi_2 \Longleftrightarrow \overline{\zeta_1} \supseteq \overline{\zeta_2} \tag{6.9}$$

$$\Longleftrightarrow \zeta_2 \supseteq \zeta_1 \tag{6.10}$$

$$\phi_1 \not\supseteq \phi_2 \Longleftrightarrow \overline{\zeta_1} \not\supseteq \overline{\zeta_2} \tag{6.11}$$

$$\Longleftrightarrow \zeta_2 \not\supseteq \zeta_1 \tag{6.12}$$

Proof. Similar to the flux pattern (ϕ), the complement of the zero set ($\overline{\zeta}$) contains indices associated with nonzero flux values. Let us consider any reaction indexed by i. If i is irreversible, we have a one-to-one mapping from ϕ to $\overline{\zeta}$, i.e. both sets contain or do not contain i. If i is reversible and the associated flux vector carries a positive flux value at i, both sets contain i. For a negative flux value, ϕ contains $-i$ and $\overline{\zeta}$ contains $i + r$. For zero flux, neither of the sets contains any of the elements and we have a one-to-one correspondence between the elements in ϕ and $\overline{\zeta}$. From this, eq. (6.9) and (6.11) follow immediately. The remaining equivalences are simple set contrapositions. □

Proposition 6.2. *Let $\boldsymbol{v} = \boldsymbol{q} + \lambda_i \boldsymbol{e}_i$ be a feasible ray composed of a (not necessarily feasible) ray $\boldsymbol{q}$ and an infeasible elementary part $\lambda_i \boldsymbol{e}_i$. Then*

$$\exists \boldsymbol{e} \not\propto \boldsymbol{e}_i \,:\, \phi(\boldsymbol{e}) \subseteq \phi(\boldsymbol{v}) \tag{6.13}$$

Proof. At least one flux value of a reversible reaction in $\boldsymbol{e}_i$ has been cancelled out or reverted in $\boldsymbol{v}$, otherwise, $\phi(\boldsymbol{v}) \supseteq \phi(\boldsymbol{e}_i)$, and $\boldsymbol{v}$ would be infeasible according to eq. (6.7). Due to eq. (6.12), also $\zeta(\boldsymbol{e}_i') \not\supseteq \zeta(\boldsymbol{v}')$ holds for the projections to the EM cone (denoted by $'$). Note that elementary modes are not minimal, hence $\boldsymbol{v}$ can be an EM and still composite, and we have

$$\text{either} \quad \boldsymbol{v} \text{ is an EM} \tag{6.14}$$
$$\text{or} \quad \exists \boldsymbol{e} \not\propto \boldsymbol{e}_i \,:\, \phi(\boldsymbol{e}) \subset \phi(\boldsymbol{v}) \tag{6.15}$$

This follows from the combinatorial test for extreme rays (Lemma 2.4): since $\zeta(\boldsymbol{e}_i')$ is not a superset of $\zeta(\boldsymbol{v}')$, either another EM $\boldsymbol{e}$ distinct from $\boldsymbol{e}_i$ must exist with $\zeta(\boldsymbol{e}') \supset \zeta(\boldsymbol{v}')$ and eq. (6.15) holds, or $\boldsymbol{v}$ is itself an EM, as stated by eq. (6.14). Eq. (6.13) is clear if (6.15) holds. On the other hand, if $\boldsymbol{v}$ is an EM and since $\boldsymbol{v} \not\propto \boldsymbol{e}_i$, we can set $\boldsymbol{e} = \boldsymbol{v}$ with $\phi(\boldsymbol{e}) = \phi(\boldsymbol{v}) \subseteq \phi(\boldsymbol{v})$. □

Proposition 6.3. *Let $\boldsymbol{v} = \boldsymbol{q} + \lambda_i \boldsymbol{e}_i$ be a feasible ray composed of a (not necessarily feasible) ray $\boldsymbol{q}$ and an infeasible elementary part $\lambda_i \boldsymbol{e}_i$, and $\boldsymbol{v}$ itself is not an EM, i.e. eq. (6.15) holds. Then*

$$\exists \boldsymbol{e}_1 \not\propto \boldsymbol{e}_i, \boldsymbol{e} \,:\, \phi(\boldsymbol{e}_1) \subset \phi(\boldsymbol{v}) \tag{6.16}$$

Proof. Let us introduce a new vector $\boldsymbol{v}_1 = \boldsymbol{v} - \lambda \boldsymbol{e}$. If we choose a minimal $\lambda = \min(\frac{r_j}{e_j})$ from all j with $e_j \neq 0$, at least one flux value of $\boldsymbol{v}$ and $\boldsymbol{e}$ is cancelled in $\boldsymbol{v}_1$. Note that $\lambda > 0$ since $\phi(\boldsymbol{v}) \supset \phi(\boldsymbol{e})$, i.e. all flux value pairs (r_j, e_j) have equal sign for $e_j \neq 0$, and $\boldsymbol{v}_1$ has no inverted sign compared to $\boldsymbol{v}$ due to the minimal choice of λ. Hence, $\phi(\boldsymbol{v}_1) \subset \phi(\boldsymbol{v})$, and consequently, all non-negativity constraints still hold and $\boldsymbol{v}_1$ is a ray of the flux cone.
Furthermore, we know that $\phi(\boldsymbol{v}) \not\supseteq \phi(\boldsymbol{e}_i)$, hence clearly also $\phi(\boldsymbol{v}_1) \not\supseteq \phi(\boldsymbol{e}_i)$, and since we have cancelled out at least one value of $\boldsymbol{e}$, also $\phi(\boldsymbol{v}_1) \not\supseteq \phi(\boldsymbol{e})$. Then,

$$\text{either} \quad \boldsymbol{v}_1 \text{ is an EM} \tag{6.17}$$
$$\text{or} \quad \exists \boldsymbol{e}_1 \not\propto \boldsymbol{e}_i, \boldsymbol{e} \,:\, \phi(\boldsymbol{e}_1) \subset \phi(\boldsymbol{v}_1) \tag{6.18}$$

The reasoning is the same as for (6.14, 6.15): either $\boldsymbol{v}_1$ is an EM, or its elementarity is disproved by existence of $\boldsymbol{e}_1$. Since $\phi(\boldsymbol{v}_1) \subset \phi(\boldsymbol{v})$, (6.16) follows. □

Lemma 6.1 (Elimination Lemma for Infeasible Elementary Terms)**.** *Let $\boldsymbol{v} = \boldsymbol{q} + \lambda_i \boldsymbol{e}_i$ be a feasible ray composed of a (not necessarily feasible) ray $\boldsymbol{q}$ and an infeasible elementary part $\lambda_i \boldsymbol{e}_i$. Then, there exists some $\boldsymbol{\lambda} \geq \boldsymbol{0}$ such that $\boldsymbol{v} = \sum_{j=1}^{n} \lambda_j \boldsymbol{e}_j$ with $\lambda_j = 0$ if $j = i$, i.e. $\boldsymbol{v}$ can be decomposed into elementary modes without using $\boldsymbol{e}_i$.*

Proof. Using Prop. 6.2, either $\boldsymbol{v} \not\propto \boldsymbol{e}_i$ is an EM according to (6.14) and we're done, or we can apply Prop. 6.3 and $\boldsymbol{v} = \boldsymbol{v}_1 + \lambda \boldsymbol{e}$. If $\boldsymbol{v}_1$ is an EM (eq. 6.17), we're done since $\boldsymbol{e}, \boldsymbol{v}_1 \not\propto \boldsymbol{e}_i$. Otherwise, we reapply the decomposition technique used in Prop. 6.3 for $\boldsymbol{v}_1$ and $\boldsymbol{e}_1$, i.e. we

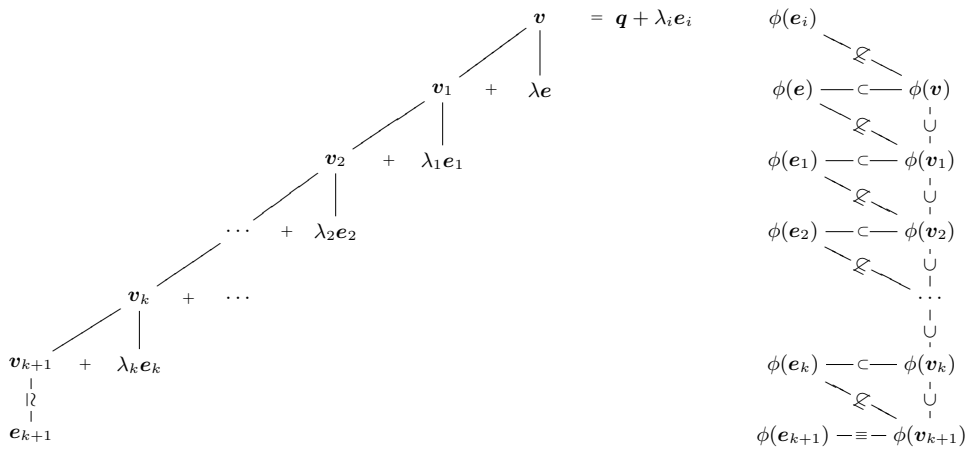


Figure 6.3: Elimination of infeasible elementary term $\lambda_i \boldsymbol{e}_i$: decomposition of $\boldsymbol{v}$ into elementary modes $\boldsymbol{e}, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_{k+1}$ (left) and depencencies of flux patterns (right).

find a ray $\boldsymbol{v}_2$ such that $\boldsymbol{v}_1 = \boldsymbol{v}_2 + \lambda_1 \boldsymbol{e}_1$.
We continue with the decomposition technique $\boldsymbol{v}_k = \boldsymbol{v}_{k+1} + \lambda_k \boldsymbol{e}_k$ until $\boldsymbol{v}_{k+1}$ is an EM. Note that the newly found elementary modes $\boldsymbol{e}_k \not\simeq \boldsymbol{e}_i$, nor are they equivalent to previously found elementary modes, since $\phi(\boldsymbol{v}) \not\supseteq \phi(\boldsymbol{e}_i)$, and $\phi(\boldsymbol{v}_{k+1}) \subset \phi(\boldsymbol{v}_k) \subset \ldots \subset \phi(\boldsymbol{v}_1) \subset \phi(\boldsymbol{v})$. We can thus rewrite $\boldsymbol{v}$ without using $\boldsymbol{e}_i$, namely $\boldsymbol{v} = \lambda \boldsymbol{e} + \lambda_1 \boldsymbol{e}_1 + \ldots + \lambda_k \boldsymbol{e}_k + \lambda_{k+1} \boldsymbol{e}_{k+1}$ stopping at the elementary $\boldsymbol{v}_{k+1} \simeq \boldsymbol{e}_{k+1}$. □

Decomposition technique and dependencies of flux patterns are visualized in Fig. 6.3. An example elimination is given in Appendix A4.2.

Now, we are ready to prove Theorem 6.1.

Proof. Assume that some $\boldsymbol{e}_k$ in eq. (6.8) is infeasible, and let us denoted it by $\boldsymbol{e}_i$. Using Lemma 6.1, we can eliminate $\boldsymbol{e}_i$ for every feasible ray $\boldsymbol{v}$ of the flux cone, i.e. we find some $\lambda_k \geq 0$ such that

$$\boldsymbol{v} = \sum_{k \in \{1,\ldots,r\} \setminus \{i\}} \lambda_k \boldsymbol{e}_k \tag{6.19}$$

Since we can still generate all feasible EMs without $\boldsymbol{e}_i$, we can remove it and consider the reduced flux cone generated by $(n-1)$ elementary modes $[\boldsymbol{e}_k], k \in \{1, \ldots, r\} \setminus \{i\}$. Continuing with the reduced flux cone, we eliminate another infeasible elementary term until no infeasible terms are left. □

6.2.4 Results

We have applied the technique to a large network of *S.cerevisiae* based on the `iND750` model published in (Duarte et al., 2004). To make EM computation possible for a network of this size, two model versions for growth on glucose and ethanol were derived, describing the central carbon metabolism and containing the compartments cytosol and mitochondria.

In the original application, Kümmel (2008) computed 480,858 EMs for growth on glucose and 287,262 EMs for growth on ethanol, a relatively small number for a genome scale model with over 200 reactions. In the glucose case, approximately 50% of the EMs where thermodynamically feasible due to the NET analysis, and only 5% for growth on ethanol. From the reduced feasible EM sets, Kümmel (2008) derives active and inactive reaction sets, but we are not discussing the results here, since we have refined both the computation procedure and the model in the meantime. This yields much larger EM sets, but also enables for a more recent interpretation of the results.

In the current ongoing collaboration with S. J. Jol and M. Heinemann of the Institute of Molecular Systems Biology at ETH Zurich, the model was revised several times, yielding two separate configurations for growth on glucose and ethanol. Separating the models is mainly necessary due to the complexity of computing EMs for large scale models: the glucose data set consists of 20,931,654 elementary modes, for ethanol, 5,657,286 EMs where computed. The workflow for the overall procedure has also been refined. For instance, we perform feasibility analysis *before* the EM computation, first using LP (see Section 3.1.2), then performing NET analysis (Kümmel et al., 2006) for single reactions. This already reduces the reaction reversibilities significantly, and most of the inactive reactions found in (Kümmel, 2008) could be detected without the need to perform the expensive EM computation.

Hence, the current EM computation is applied to a network with severely pre-constrained reaction directionalities. Interestingly, the feasibility space still shrinks significantly when we apply NET analysis to the EM set: only approximately 30% of the elementary modes are classified feasible (glucose and ethanol), and over 40% (glucose) and 60% (ethanol) of the EMs are considered infeasible (remaining EMs have not been classified yet). In Fig. 6.4, we compare the importance of individual reactions (measured in terms their frequency) between the different EM sets. We have also analyzed the pathway lengths in the different EM sets: the average lengths are comparable in the feasible and infeasible set, even if feasible EMs are somewhat shorter for both glucose and ethanol (data not shown). It seems that tightened thermodynamic constraints—here reflected by the NET feasibility classification—put a downward pressure on pathway lengths. This is not surprising since larger pathways have more active reactions, competing for favorable Gibbs formation energies of the involved reactants.

6.2.5 Discussion

It is important to note that Theorem 6.1 only applies to elementary modes (Def. 2.40) and is not applicable to extreme pathways or minimal generators of the flux cone (see Section 2.4). Using the EM cone in Prop. 6.1 is crucial, since we can only map flux patterns to zero sets in a one-to-one fashion for EMs. On the other hand, the results are quite general since we can use Theorem 6.1 for any classifier satisfying Def. 6.2.

We have also shown that the method is indeed powerful: the yeast model for growth on

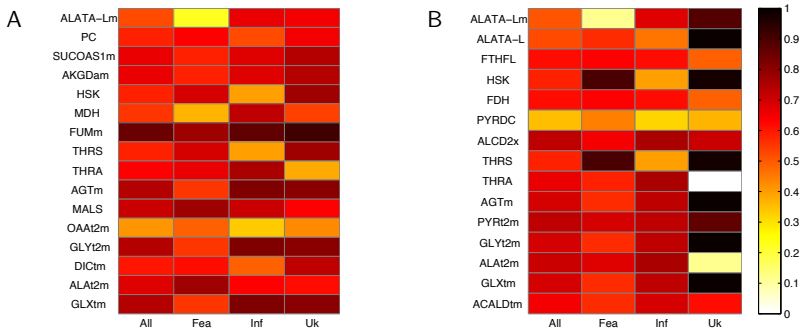


Figure 6.4: Frequencies of most differing reactions considering all (All), only feasible (Fea) or only infeasible (Inf) elementary modes (EMs) of a metabolic network for *S.cerevisiae*. Feasibility classification was not possible for some EMs listed as unknown (Uk) **A:** Growth on glucose with compartmentalized network consisting of 217 metabolites and 229 reactions (22 reversible). The total set contains 20,931,654 EMs, 6,661,116 are classified feasible, 8,423,395 infeasible. **B:** Network for growth on ethanol with 217 metabolites and 220 reactions (24 reversible). 1,755,472 of the 5,657,286 EMs are classified feasible, 3,606,401 infeasible.

glucose and ethanol needs approximately one third of the elementary modes to generate only the feasible space. In depth analysis of the differences in the feasible and infeasible set is still ongoing, and it has still to be shown that we can issue significant statements or unveil new insights with the new method. However, we have already shown that certain reactions show a significant change of frequency in the EM sets if only feasible modes are considered. As an alternative approach, one could also try to apply feasibility constraints to the flux cone directly, either avoiding the computation of EMs completely, or in the hope of speeding it up considerably due to the additional restrictions. The NET tool tightens irreversibility constraints considering Gibbs free energies of the reactant, applied to single reactions *before* and to reaction sets *after* EM computation. However, it would be favorable to incorparate all constraints beforehand. The NET constraints involve integer variables, yielding a non-convex optimization space. Reformulations or relaxations of the constraints might be possible, however, that preseve convexity and hopefully do not distort the solution space significantly. Such simplifications could improve applicability and performance of optimization approaches, but also facilitate the computation of generators for a more strongly restricted solution space.

6.3 Generators for Genome Scale Models

We have already computed elementary modes on a large scale in (Terzer and Stelling, 2008). Also the yeast model used in Section 6.2 yields large EM sets. Here, we also present some of our newest applications, where we compute *minimal generators* using different genome scale models. The resulting sets contain 16, 51 and 392 million generators, respectively.

6.3.1 Introduction

The number of elementary modes grows exponentially with the size of the metabolic network. For large networks covering metabolism at a genome scale level, it is thus usually not possible to compute elementary modes. Possible ways to work around the combinatorial explosion exist but have also some severe drawbacks: one could split the computation into separate tasks, each subtask considering only restricted network configurations. For instance, if an organism can grow on glucose, lactate and malate, EMs for each uptake configuration could be computed separately. With this approach, all elementary modes are missed that need at least two of the substrate metabolites. In (Terzer and Stelling, 2008), we show that the fraction of missed EMs can indeed be drastic: around 90% of all EMs were missed with separate computations.

Another way to reduce the number of pathways is to use a different set of generators for the flux cone: as mentioned in Section 2.4, *minimal generators* constitute the smallest generating set for the flux cone. We have computed minimal generators for different genome scale metabolic networks, yielding sets of 16, 51 and 392 million generators, respectively. It is the first time that genome scale networks can be described constructively in this way—however, it is still not possible to compute generators for all networks. Hence, further algorithmic improvements are necessary, or other concepts needed, to deal with the tremendous complexity of structural analysis for genome scale metabolic networks.

6.3.2 Results

6.3.2.1 EMs for E.coli central metabolism

Our first large scale application was presented in (Terzer and Stelling, 2008). We used different configurations of the central metabolism network of *E.coli* (Stelling et al., 2002) with growth on glucose. Besides uptake of glucose, six central amino acids from different biosynthesis families are available, only one at a time in a first experiment. This results in six sets of elementary modes (EMs) for the amino acids *alanine*, *aspartate*, *glutamate*, *histidine*, *phenylalanine* and *serine*. The sets contain between 321,431 (alanine) and 858,648 (glutamate) EMs. In a second evaluation, we computed the set of EMs for combined uptake that consists of 26,381,168 modes.

Fig. 6.5A shows the number of elementary modes per amino acid uptake configuration– in terms of the number of simultaneously enabled uptake reactions. The total number of EMs with at most one amino acid uptake is $1,714,691$, only 6.5% of the EM set enabling simultaneous ingestion of up to six amino acids. Conversely, by computing EM sets for one amino acid uptake per simulation, and combining the resulting sets afterwards, adding up to

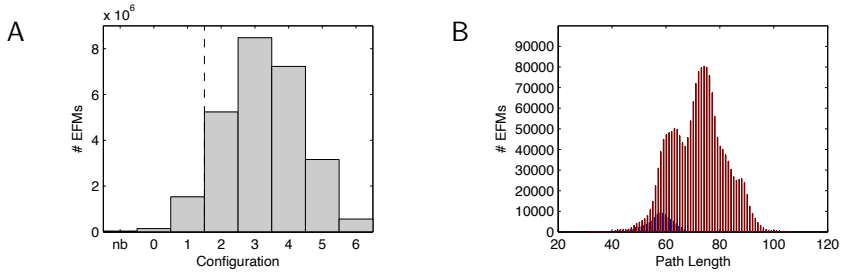


Figure 6.5: **A:** Number of EMs for different amino acid uptake configurations, namely modes without biomass production (nb) and modes with 0-6 concurrently enabled amino acid uptake reactions. Only EMs on the left of the dashed line can be computed from single amino acid uptake configurations. **B:** Path length distributions of EMs for glycine production in *H. pylori* when glycine is produced alone (dark bars) or possibly jointly with other amino acids (light bars).

1.7M EMs, simultaneous uptake cases are missed. These correspond to $> 90\%$ of the metabolic pathways.

6.3.2.2 EMs for H.pylori amino acid production

Our first application to a genome scale model was presented in (Terzer and Stelling, 2008), where we applied our algorithm to a metabolic network of *H.pylori*. In their study, (Price et al., 2002) computed the much smaller set of extreme pathways (EPs, see Section 2.4.2) for the formation of all nonessential amino acids (AAs) and ribonucleotides. Here, we focus on the amino acids and compute EMs for all nonessential AAs simultaneously. Allowable inputs are D- and L-alanine, arginine, adenine, sulfate, urea and oxygen; outputs are ammonia, carbon dioxide and the carbon sinks succinate, acetate, formate and lactate (in correspondence to case 4 in Price et al. (2002)), together with all non-essential amino acids. Focusing on the production of a specific amino acid, only a small fraction of EMs is found with typical small-scale simulations, preventing the simultaneous production of other amino acids (between 3% for proline and 16% for asparagine, respectively). Altogether, only 815,576 of 5,785,975 EMs are found with single amino acid simulations—as for the *E.coli* case, over 85% of the modes are missed (see Appendix A3.2 for details). Furthermore, allowing for simultaneous production of all amino acids yields EMs with larger path lengths as shown in Fig. 6.5B for glycine production. Path length distributions are similar for the other amino acids (see Appendix A3.2) and the more complex pathways are not reflected in simplified single amino acid experiments.

6.3.2.3 Minimal Generators for M.barkeri, H.pylori and S.aureus

One possibility to deal with the vast number of elementary modes is to use one of the smaller generating sets for the flux cone (see Section 2.4). However, as shown in Klamt and Stelling

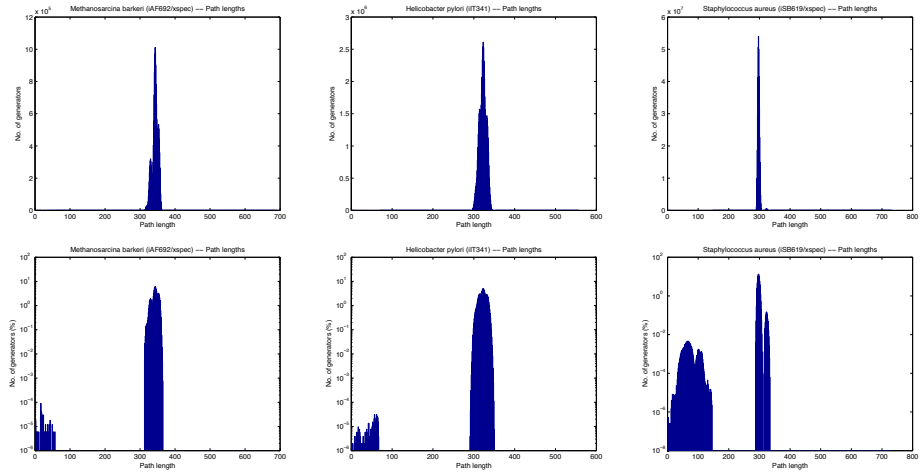


Figure 6.6: Pathway lengths for minimal generators with pathway counts on a linear scale (top) and in percentage of the total number of pathways on a log scale. The networks are (left to right): *M.barkeri* (iAF692, 631 metabolites × 691 reactions), *H.pylori* (iIT341, 486 × 555) and *S.aureus* (iSB619, 645 × 729).

(2003), this might also have certain disadvantages, since the smaller sets sometimes miss to find genetically independent pathways. We have computed *minimal generators* for three metabolism networks at a genome scale. Not surprisingly, the *M.barkeri* network (Feist et al., 2006) yields the smallest set with 16,515,151 generators, since this is the only archaea of the investigated bacteria. M. barkeri belongs to the methanogens family, producing methane as a metabolic byproduct under anoxic conditions. Unlike most methanogens, however, it cannot only ferment carbon dioxide, but also a variety of other carbon sources including methanol, methylamines and acetate. The computation was performed on a doubly dual-core system with 30G memory. Computing the binary extreme rays took only 4 minutes; however, the post-processing step to reconstruct the numeric values (see Section 3.3.2) needed almost 15 hours, mainly because the whole computation was performed with exact arithmetic. The pathlength histogram of the generators is shown in Fig. 6.6; the reaction participation table is available in Appendix A3.3.

The second network of *H.pylori* by Thiele et al. (2005) is an extended version of the network used to compute elementary modes in Section 6.3.2.2. We computed 51,149,062 generators on a machine with 4 quad-core CPUs and 110G memory. It took almost 5h to compute the binary vectors, and another 13h to compute the numeric vectors. The computation was again executed with exact arithmetic, since the large networks are usually very ill conditioned after compression (see Section 3.1).

Our last application concerns a network of *Staphylococcus aureus* with 645 metabolites and 729 reactions (Becker and Palsson, 2005). We could not compute the generators in main

memory, even not on a machine with 128G RAM. Hence, the computation was performed out-of core as described in Section 5.1. On a system with 30G memory and two quad-core CPUs, computing the binary generators took 47h, uncompressing the 392,742,819 generators into numeric form another 170 hours, again performed with exact arithmetic.

Pathlength distributions for generators of all three networks are shown in Fig. 6.6; reaction participation tables can be found in Appendix A3.

6.3.3 Discussion

It was one of the declared goals of this thesis to make pathway analysis methods applicable for genome scale models. It is clear that the generating sets for large networks may be huge due to the combinatorial nature of the problem. We have shown that we can indeed compute minimal generators for several models, even if we could not apply our algorithm to all the genome scale networks available today. Also for the presented networks, the sets can consist of hundreds of millions of generators, and we expect over a billion for larger networks. It is definitely appropriate to ask whether such large sets are of great help—they are definitely difficult to handle, our largest network needed sophisticated processing scripts and machines with lots of memory to process the huge data set, even for the simple analyses presented here. It is not clear whether such large data sets contain more or "better" information, which cannot be found with other, more efficient methods like optimization techniques. We will not answer this question here, since this was not the goal of this work. However, we showed in the first part of this section that certain simplifications may cause loss of huge parts of the solution space, possibly leading to deceitful interpretations. For instance, we have shown that many pathways for simultaneous substrate uptake are missed if the simulation is split into single-uptake configurations – noteworthy *elementary pathways*, that is, they cannot be reconstructed from combinations of single-uptake pathways. A possible way to deal with the huge number of pathways could be that smaller generating sets are computed first. An interesting approach has been proposed by Urbanczik and Wagner (2005): instead of projecting the flux cone into higher dimensional spaces (see Section 2.4), they propose a projection to the *conversion cone*, describing the exchange of metabolism with the environment. Note that the same algorithms can be used, hence our achievements also prove valuable for the computation of *metabolic conversions*. Urbanczik and Wagner (2005) also show that individual pathways—such as elementary modes– can still be computed, for instance only for the conversions of special interest. We think that our technical improvements—together with such new ideas to deal with the combinatorial nature of structural network analysis—paves the way for novel and thorough studies. We feel confident that this also reveals new biological insights in the near future.

7
Conclusions & Outlook

The main goal of this thesis was to implement an efficient algorithm for the computation of elementary modes for biochemical reaction networks. Since most problem cases are degenerate, the algorithm of choice was the double description method (DD method), an algorithm known to perform well for such cases. The complexity of the DD method is not well understood, and many aspects are critical for an efficient implementation. Hence, the focus of this thesis spans a large diversity of technical improvements, and we started with the analysis and comparison of various approaches proposed in literature. Our initial attention was directed to constraint and row ordering techniques, and we have tested several static and dynamic sorting strategies, with the somewhat disappointing result that we only confirmed what was already known before: the algorithm reacts extremely sensitive to row ordering, but we could—like others before us—not give any theoretical explanations. Based on empirical results, some sorting strategies are favored above others, but contradictory cases exist and the best sorting depends heavily on the concrete problem instance.

We were luckier with the next topic: elementarity or adjacency testing of extreme rays. Two techniques were know from theory, and different implementations existed. Some authors prefer pre-computation and storage of the adjacencies, others compute them on-the-fly, as in our implementation. The main drawback of storing the adjacency matrix is the memory demand, but we could also not improve performance by precomputing and updating the adjacency information. We introduced *bit pattern trees* and could improve adjacency testing significantly, without requiring much more memory. Both the combinatorial and the algebraic adjacency test benefit from bit pattern trees, but the most important improvement was the *candidate narrowing* approach based on a *tree all-against-all* pairing procedure. We also showed how rank computations can be improved with an update strategy, and we use residue arithmetic to avoid numerical instability.

We should also mention that our implementation was now based on the *nullspace approach*

introduced by Wagner (2004), and we also incorporated the *binary approach* (Gagneur and Klamt, 2004) saving a lot of memory since intermediary extreme rays can be stored in binary form—at least to large parts. We have also tested a lot of compression and consistency techniques, which is critical if the algorithm is applied to real biological networks. Our compression is very similar to that proposed in (Gagneur and Klamt, 2004). An important part of the implementation was also the choice of the appropriate number type: we decided to leave the choice to the user, and implemented the whole program with multiple number types. This seems trivial, but sometimes complicates algorithms significantly since it is a recurrent theme from preprosession to postprocessing. The number type is involved in almost every algorithm part, and selectable options like different adjacency test methods or nullspace versus canonical approach must all follow a generic implementation approach to support the different types. In the present report, we have demostrated the generic number type support in our implementation of basic linear algebra functions.

Another important aspect of the thesis was applicability of the program to large scale problems. Of course, this involves again performance, but also memory becomes a significant issue. We have implemented *out-of-core* strategies storing the intermediary rays on disk. This slows down the computation considerably, but at least enables the computation of large networks. We have used out-of-core computation for our largest genome-scale application, since we could not store intermediary rays in memory, even not on a machine with 128G memory. The main complexity of the out-of-core functionality is again the combination with other aspects: if only double precision numbers were used, we could easily compute a byte offset for numbers stored in a file. But since we also deal with arbitrary precision numbers, the procedure is more complicated. We could have used standard databases for this purpose, but we are sure that they would not have matched our performance requirements. We do not need transaction support offered by databases, but our implementation reads and writes millions of numbers within seconds. To further improve performance, we have included multi-level caches and support for multi-threaded access.

The next step towards large scale computation deals with parallelization: it seems natural to make use of modern multi core CPUs, and it turned out that our candidate narrowing approach with two bit pattern trees was not very difficult to parallelize. We have also implemented a second parallelization approach, where we try to break the sequential operation of the iterative double description method. The scale-up behavior of both approaches is nice, at least up to around 12 threads. For tests on a machine with 4 quad-core CPUs, we could not improve performance further by adding more than 12 threads, even if 16 cores are available. This aspect needs more investigation, and we have also planned to do so, since this has also been requested by the reviewers of our PPAM09 (Wyrzykowski and (tba), 2009) paper. Our weakest point at the moment is the distributed computation part. We have implemented a simple manually distributed version of the algorithm, but neither does it outperform the serial version, nor does it scale well with the number of nodes in the cluster. Here, we think that a portation to an MPI based architecture would be beneficial.

We have also some results on the application side: our strongest finding is what we call *optimal metabolic operation point*, a point in the solution space—2% below optimal biomass yield—where the cell gains most flexibility paying only a small price of suboptimality. We hypothesize that the cell should optimally operate near this point, and we strengthen our thesis with a sample flux prediction based on this assumption. Furthermore, there is recent

evidence in literature (Schuetz et al., 2009) that cells might indeed be operating close to a surface of pareto optimality, considering three instead of only one objective function. We think that it is worth to pursue this approach and apply it to other networks, also using alternative yield measures and possibly multiple objectives.

In a second application, we prove that *feasibility classifiers* can be applied to elementary modes (EMs), and that infeasible EMs can be removed safely without loosing valid solutions. We expect more results from our continued collaboration with the Institute of Molecular Systems Biology at ETH Zurich within the next months.

In a last application section, we have computed elementary modes and generators for different large scale networks. We show that simplified approaches mostly miss the majority of the pathways, for instance if data sets are computed separately for different nutrient metabolites. The largest data sets have been computed for genome scale networks, even if we did not succeed for all available networks so far. Still, we have shown that it might indeed be possible to compute a generating basis for the networks, even if the analysis and interpretation of the results is for the greater part ahead of us. We also discussed the application of clustering techniques to the generating sets, but haven't found an appropriate method so far. In a semester project, Thomas Liphardt implemented a biclustering algorithm for EM datasets, but some rework is necessary to make it applicable to larger matrices. The handling of the large generating sets is anyway not easy, our largest case consisting of 390 million generators is stored in a (compressed) file of 100G! The techniques and tools to handle such large sets have to be defined and found yet, and the utility of the huge amount of data still be proven.

Appendix

A1 Mathematical Fundamentals

A1.1 Linear Algebra

Subspaces are vector subspaces of $\mathbb{R}^d$, and every subspace is *finitely generated* (eq. A.1) and *finitely constrained* (eq. A.2):

$$\mathcal{S} = \mathcal{R}(\boldsymbol{A}) = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x} = \boldsymbol{A}\,\boldsymbol{\lambda}\} \qquad \text{with} \quad \boldsymbol{A} \in \mathbb{R}^{d\times r}, \boldsymbol{\lambda} \in \mathbb{R}^{r\times 1} \tag{A.1}$$

$$= \mathcal{N}(\boldsymbol{B}) = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{B}\,\boldsymbol{x} = \boldsymbol{0}\} \qquad \text{with} \quad \boldsymbol{B} \in \mathbb{R}^{k\times d} \tag{A.2}$$

The set $\mathcal{R}(\boldsymbol{A})$ is the *column space* of $\boldsymbol{A}$, that is, all vectors that can be written as linear combinations of the columns of $\boldsymbol{A}$. We say that the column vectors $\boldsymbol{A}_{\star 1}, \boldsymbol{A}_{\star 2}, \ldots, \boldsymbol{A}_{\star r}$ *span* the subspace, that is, $\mathcal{R}(\boldsymbol{A}) = \mathbf{span}\,(\boldsymbol{A}_{\star 1}, \boldsymbol{A}_{\star 2}, \ldots, \boldsymbol{A}_{\star r})$. We call $\mathcal{R}(\boldsymbol{A})$ *range* or *image* of $\boldsymbol{A}$, and we write $\mathcal{R}(\boldsymbol{A}) = \mathbf{im}\ \boldsymbol{A}$.

The set $\mathcal{N}(\boldsymbol{B})$ are the vectors $\boldsymbol{x}$ mapped into zero by $\boldsymbol{B}$, and we call $\mathcal{N}(\boldsymbol{B})$ *nullspace* or *kernel* of the matrix $\boldsymbol{B}$, written as $\mathcal{N}(\boldsymbol{B}) = \mathbf{ker}\ \boldsymbol{B}$.

The two representations A.1 and A.2 are mathematically equivalent, and we can convert between them using *Gaussian elimination*. The dimension of the subspace $\mathcal{R}(\boldsymbol{A})$ is called $\mathbf{rank}\ \boldsymbol{A}$, and it corresponds to the number of linearly independant columns in $\boldsymbol{A}$. Note that $\mathbf{rank}\ \boldsymbol{A} \leq \min(d, r)$, and we say that $\boldsymbol{A}$ has *full rank* if $\mathbf{rank}\ \boldsymbol{A} = \min(d, r)$, and call $\boldsymbol{A}$ *rank deficient* otherwise. The dimension $\mathbf{rank}\ \boldsymbol{B}$ of the nullspace $\mathcal{N}(\boldsymbol{B})$ is called *nullity*, and by the *rank-nullity theorem*, we know that $\mathbf{rank}\ \boldsymbol{A} + \mathbf{rank}\ \boldsymbol{B} = d$.

Definition A.1. *The set* $\mathcal{S}^\perp = \{\boldsymbol{y} \in \mathbb{R}^d : \boldsymbol{y}^T\boldsymbol{x} = \boldsymbol{0} \quad \forall \boldsymbol{x} \in \mathcal{S}\}$ *of a linear subspace* $\mathcal{S} \in \mathbb{R}^d$ *is called* dual *of* $\mathcal{S}$ *or* orthogonal complement *of* $\mathcal{S}$.

The orthogonal complement of a subspace is again a subspace, and we have:

$$\mathcal{S}^{\perp\perp} = \mathcal{S} \tag{A.3}$$

$$\mathcal{R}(\boldsymbol{A})^\perp = \mathcal{N}(\boldsymbol{A}^T) \tag{A.4}$$

A2 Specifications & Computation Results

A2.1 E.coli Central Metabolism Network

Computation times for different configurations of the central metabolism network of Stelling et al. (2002) used in Terzer and Stelling (2008).

Configuration	S2	S3	L1
Substrates	Glc, Succ, Glyc, Ac	Glc, Succ, Glyc, Ac, Asp	Glc, Ala, Asp, Glu, His, Phe, Ser
Products	Ac, Form, Eth, Lac, CO2	Ac, Form, Eth, Lac, CO2, Succ	Ac, Form, Eth, Lac, CO2, Succ
# EFMs	507,632	2,450,787	26,381,168

Method	Computation time[1]		
Combinatorial			
–1 thread	1m 30s	19m 39s	7h 43m 17s
–2 threads	57s	12m 33s	4h 41m 46s
–4 threads	40s	8m 08s	3h 05m 29s
Standard rank			
–1 thread	1m 47s	30m 23s	11h 03m 14s
–2 threads	1m 35s	16m 29s	5h 58m 42s
–4 threads	57s	10m 08s	4h 23m 13s
Rank up, residue			
–1 thread	2m 24s	30m 32s	12h 56m 38s
–2 threads	1m 25s	16m 53s	6h 59m 13s
–4 threads	54s	10m 54s	4h 12m 40s
Rank up, exact			
–1 thread	6m 20s	69m 30s	21h 46m 41s
–2 threads	3m 28s	36m 28s	10h 37m 03s
–4 threads	1m 57s	20m 00s	6h 12m 44s
CellNetAnalyzer/Metatool (5.0.4)[2]			
Our timing	2m 57s	92m 09s	$\gg$ 5d[3]
As stated in[4]	3m 19s	87m 08s	n/a

[1] Only iteration phase, w/o pre– and postprocessing

[2] Note that computation times are not directly comparable, because (i) Metatool uses only one thread, (ii) different compression techniques are applied, and (iii) the initial kernel matrix is not identical, due to different pivoting and row sorting strategies. Relevant are in particular (ii), influencing the number of iteration steps, and (iii), affecting the order of constraint processing. Both (ii) and (iii) have high impact on the number of intermediate modes.

[3] The computation was stopped at iteration 75 of 80 after more than 5 days.

[4] Klamt et al. (2005)

A3 Data & Figures

A3.1 E.coli Reaction Variation for Suboptimal Yield

This section contains data and figures related to Section 6.1. It has been published in Terzer and Stelling (2008).

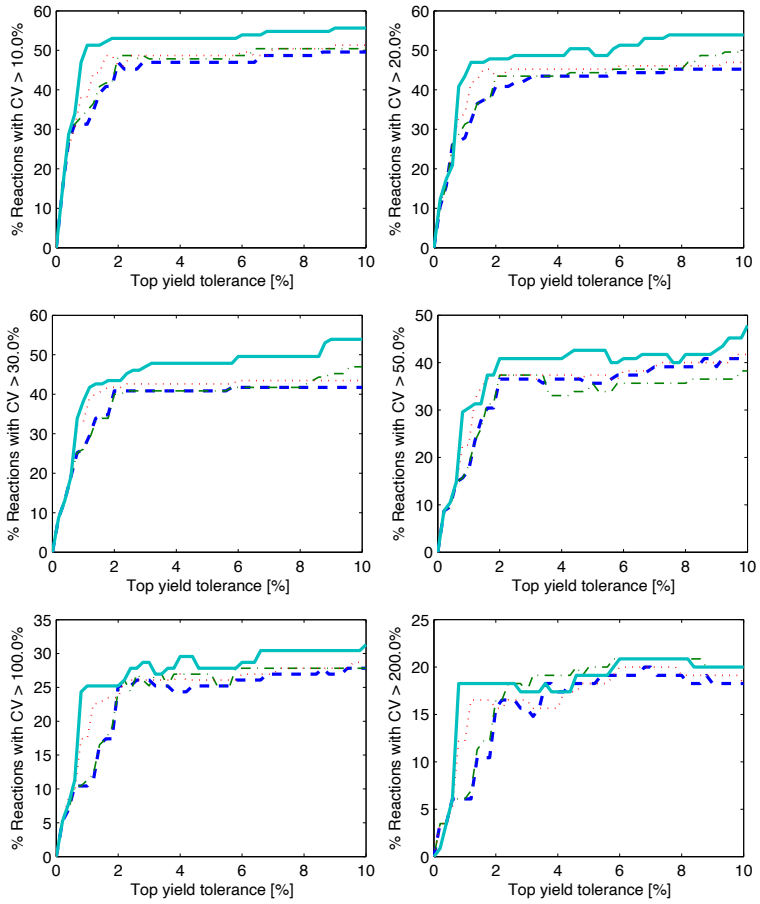


Figure A.1: Flux variability for different uptake configurations of *E.coli*, considering only top yield EFMs. Number of reactions with coefficient of variation (CV) above different threshold levels. The configurations are: glucose without amino acid uptake (dashed line), with phenylalanine (dotted), with glutamate (dash-dotted) and with all selected amino acids (solid line).

A3.2 Analysis of Non-essential Amino Acid Production in H.pylori

This section contains data and figures related to Section 6.3.2.2. It has been published in Terzer and Stelling (2008).

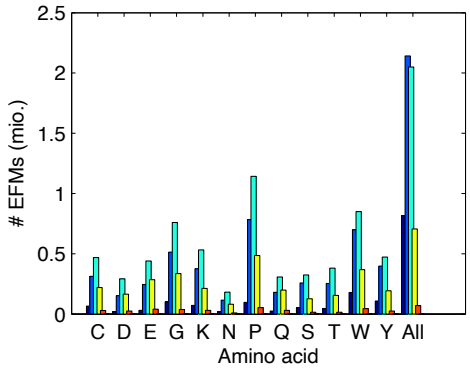


Figure A.2: Number of EFMs for the production of non-essential amino acids (AAs). The bars (left to right) stand for production of a single AA, the simultaneous production of two AAs, ..., to the simultaneous production of up to six amino acids. Each bar group (except for the right most) focuses on one amino acid, meaning that it is always produced within this group. The letters C, D, E, G, K, N, P, Q, S, T, W and Y stand for cysteine, aspartic acid, glutamic acid, glycine, lysine, asparagine, proline, glutamine, serine, threonine, tryptophan and tyrosine respectively.

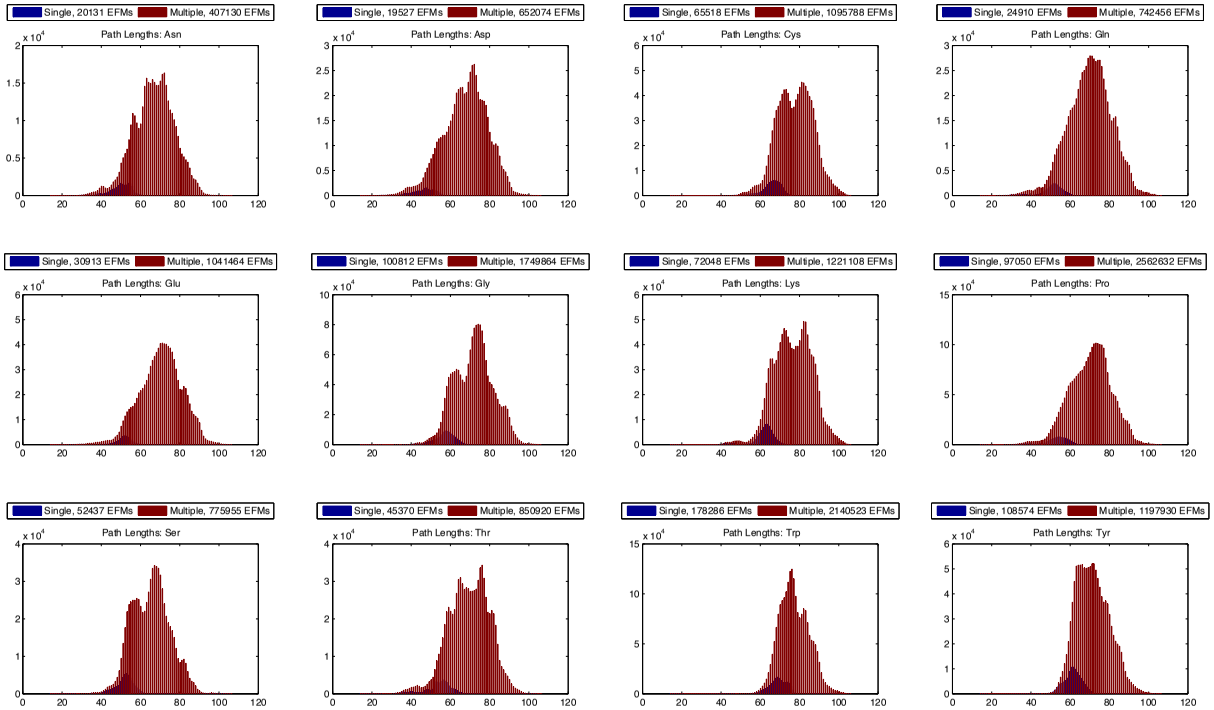


Figure A.3: Path length distribution of the EFMs for the production of all amino acids. The smaller (dark blue) set covers EFMs solely producing the amino acid under focus, the larger (burgundy) set covers EFMs producing the selected amino acid possibly jointly with other amino acids.

A3.3 M.barkeri Reaction Frequencies

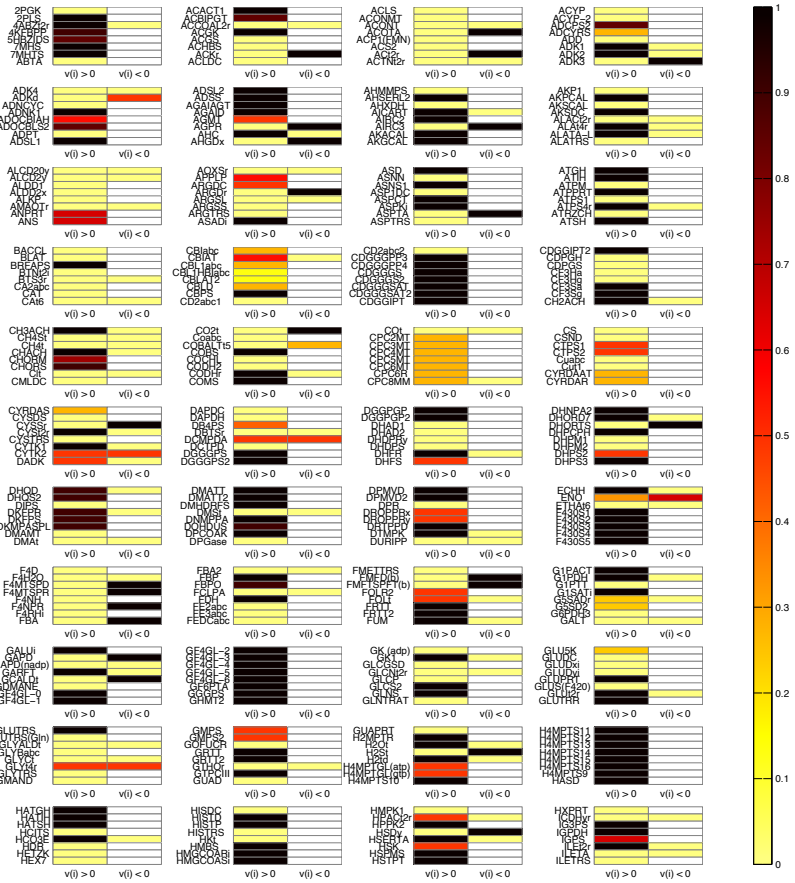


Figure A.4: Reaction frequencies for *M.barkeri* network iAF692 (Feist et al., 2006) with 631 metabolites and 691 reactions, based on 16,515,151 minimal generators. Colored cells correspond to reaction frequencies for forward (left columns) and backward flux (right). White cells indicate that the reaction is not reversible.

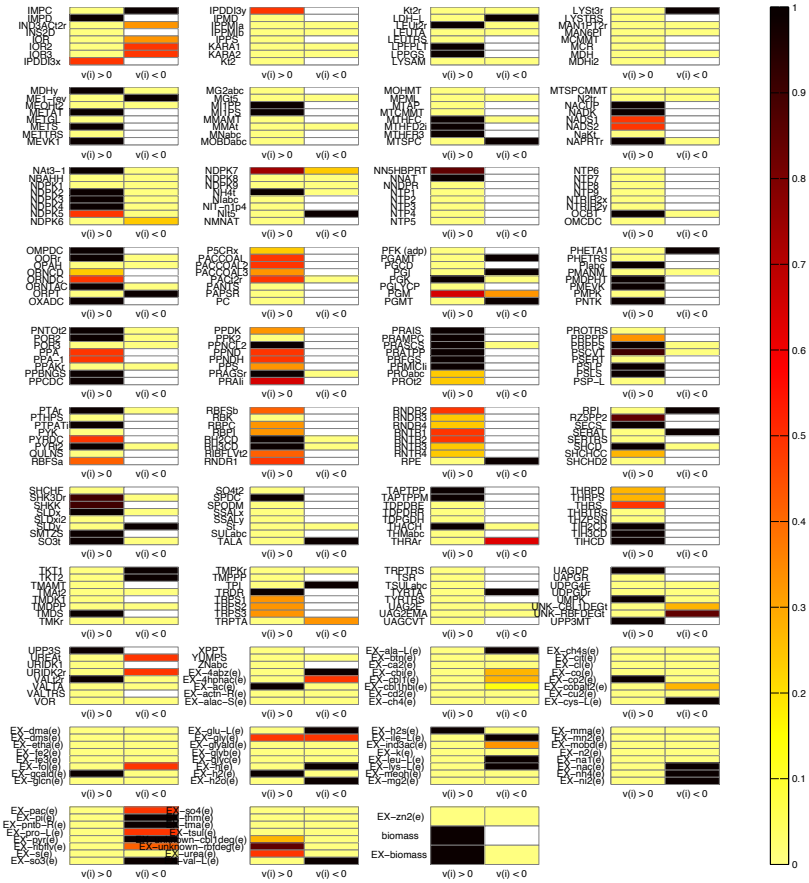


Figure A.5: Reaction frequencies for *M.barkeri* network iAF692 (Feist et al., 2006) with 631 metabolites and 691 reactions, based on 16,515,151 minimal generators. Colored cells correspond to reaction frequencies for forward (left columns) and backward flux (right). White cells indicate that the reaction is not reversible.

A3.4 H.pylori Reaction Frequencies

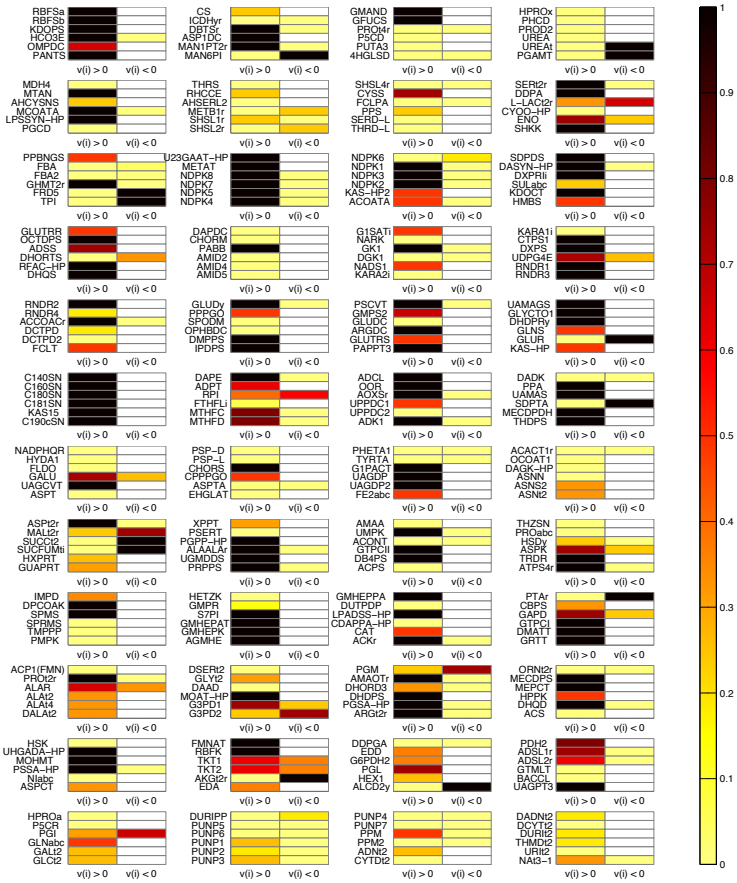


Figure A.6: Reaction frequencies for *H.pylori* network iIT341 (Thiele et al., 2005) with 486 metabolites and 555 reactions, based on 51,149,062 minimal generators. Colored cells correspond to reaction frequencies for forward (left columns) and backward flux (right). White cells indicate that the reaction is not reversible.

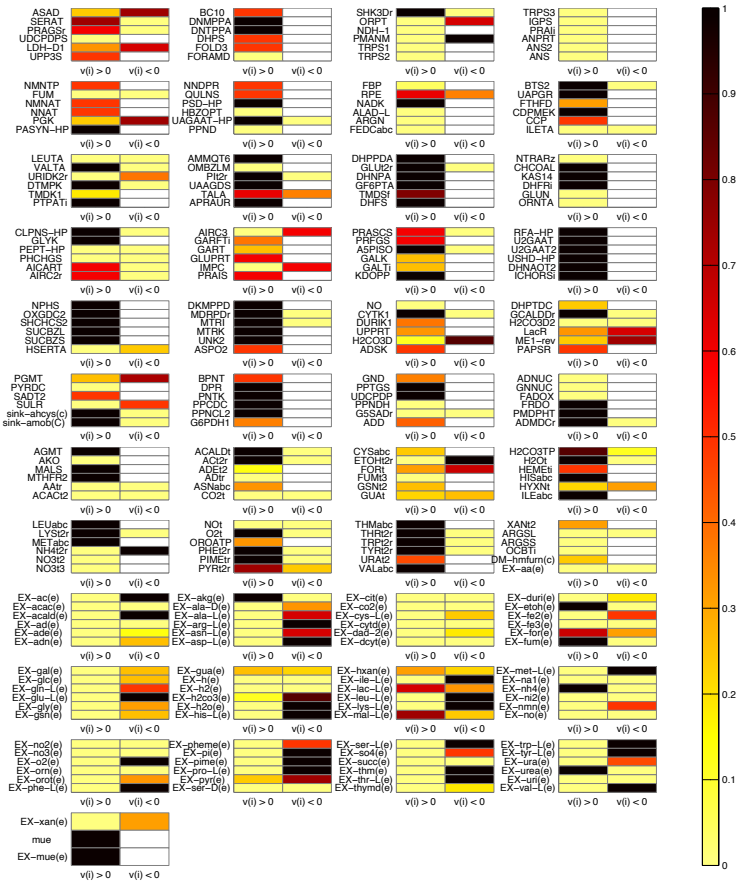


Figure A.7: Reaction frequencies for *H.pylori* network iIT341 (Thiele et al., 2005) with 486 metabolites and 555 reactions, based on 51,149,062 minimal generators. Colored cells correspond to reaction frequencies for forward (left columns) and backward flux (right). White cells indicate that the reaction is not reversible.

A3.5 S.aureus Reaction Frequencies

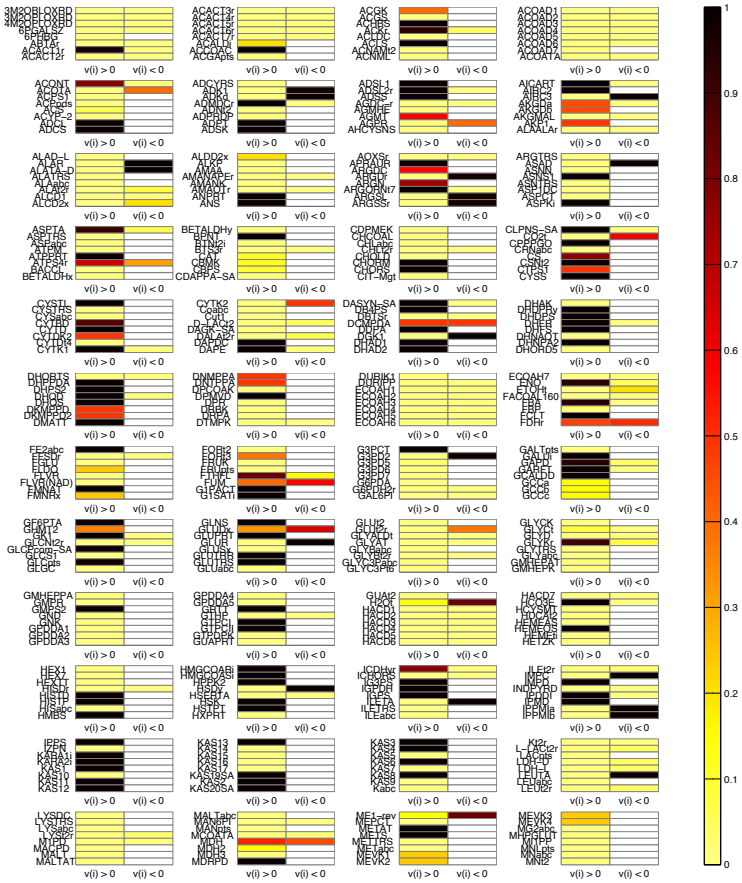


Figure A.8: Reaction frequencies for *S.aureus* network iSB619 (Becker and Palsson, 2005) with 645 metabolites and 729 reactions, based on 392,742,819 minimal generators. Colored cells correspond to reaction frequencies for forward (left columns) and backward flux (right). White cells indicate that the reaction is not reversible.

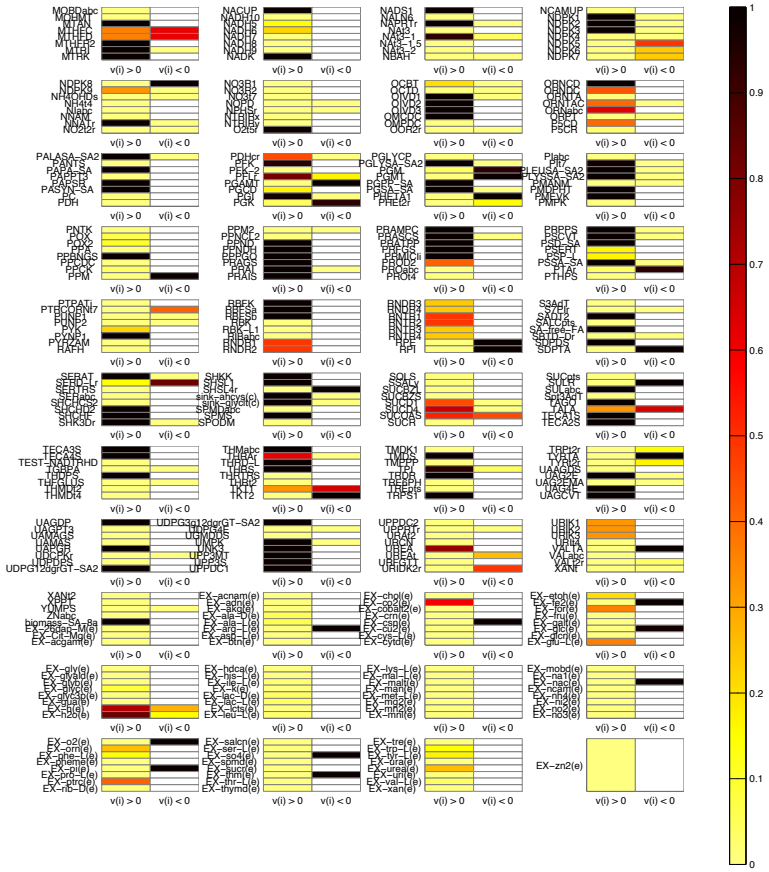


Figure A.9: Reaction frequencies for *S.aureus* network iSB619 (Becker and Palsson, 2005) with 645 metabolites and 729 reactions, based on 392,742,819 minimal generators. Colored cells correspond to reaction frequencies for forward (left columns) and backward flux (right). White cells indicate that the reaction is not reversible.

A4 Examples

A4.1 Candidate Narrowing with Bit Pattern Trees

The following example has been taken from (Klamt and Stelling, 2006). Step by step, we construct the tableau containing semi-binary vectors. Since the first iteration steps are rather simple, a bit pattern tree is only constructed for the last step. To keep it simple, we do not compress the network.

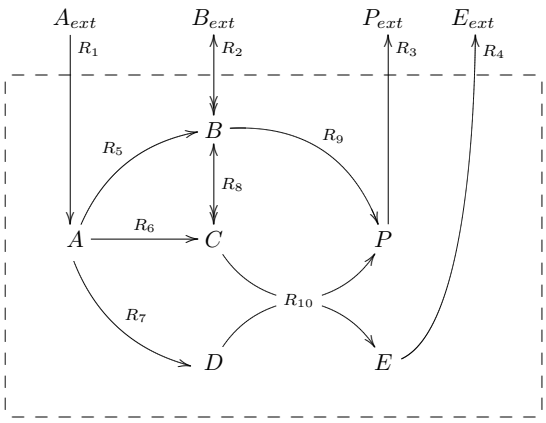


Figure A.10: Metabolic network with metabolites A, B, C, D, E, P and reactions R_1 to R_{10}. Double arrow heads indicate default (forward) flux direction for reversible reactions R_2 and R_8. See also (Klamt and Stelling, 2006)

The stoichometric matrix of this network is

$$\boldsymbol{N} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{A.5}$$

A4.1.1 Initialization phase

To get the expanded stoichiometric matrix, columns corresponding to reversible reactions (i.e. column 2 and 8) are negated and appended after the original column. Equivalently, each

reversible reaction is split up into a forward and a backward irreversible reaction.

$$\boldsymbol{N}^{exp} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{A.6}$$

We compute the kernel matrix

$$\boldsymbol{K}' = \begin{bmatrix} 0 & 1 & -1 & 1 & 0 & 2 \\ 1 & -1 & 1 & -1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.7}$$

Note that $\boldsymbol{K}'$ is in principle already in echelon form, we simply permute some rows in $\boldsymbol{K}'$ (corresponding to columns in $\boldsymbol{N}$, i.e. we renumber the reactions). Using the rows $(3, 6, 9, 10, 11, 5, 12, 8, 4, 7, 1, 2)$ of $\boldsymbol{K}'$, we get

$$\boldsymbol{K} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 1 & 0 & 1 \\ 0 & 1 & -1 & 1 & 0 & 2 \\ 1 & -1 & 1 & -1 & 1 & 0 \end{bmatrix} \tag{A.8}$$

Now, we convert the upper part to binary. Note that we have *true* values if a flux is zero, i.e. the complementary notation compared to (Gagneur and Klamt, 2004). To distinguish between binary and numeric, we use $\star$ for *true* and $\cdot$ for $false$:

$$\boldsymbol{R}^6 = \left[\begin{array}{cccccc} \cdot & \star & \star & \star & \star & \star \\ \star & \cdot & \star & \star & \star & \star \\ \star & \star & \cdot & \star & \star & \star \\ \star & \star & \star & \cdot & \star & \star \\ \star & \star & \star & \star & \cdot & \star \\ \star & \star & \star & \star & \star & \cdot \\ \hline 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 1 & 0 & 1 \\ 0 & 1 & -1 & 1 & 0 & 2 \\ 1 & -1 & 1 & -1 & 1 & 0 \end{array}\right] \tag{A.9}$$

A4.1.2 Iteration phase

By chosing the echelon form of the kernel matrix, the first 6 rows already contain nonnegative values. Thus, we can immediately start with iteration step 7.

Simple steps without bit pattern trees

The next 3 iterations are very easy, since no modes have a negative value, i.e. no modes are removed and no new ones created. We simply convert from numeric to binary:

$$\boldsymbol{R}^9 = \left[\begin{array}{cccccc} \cdot & \star & \star & \star & \star & \star \\ \star & \cdot & \star & \star & \star & \star \\ \star & \star & \cdot & \star & \star & \star \\ \star & \star & \star & \cdot & \star & \star \\ \star & \star & \star & \star & \cdot & \star \\ \star & \star & \star & \star & \star & \cdot \\ \star & \star & \star & \star & \star & \cdot \\ \star & \star & \star & \star & \star & \cdot \\ \star & \star & \star & \star & \cdot & \cdot \\ \hline 0 & 0 & -1 & 1 & 0 & 1 \\ 0 & 1 & -1 & 1 & 0 & 2 \\ 1 & -1 & 1 & -1 & 1 & 0 \end{array}\right] \tag{A.10}$$

Now, it gets interesting since the mode in column 3 becomes invalid due to the negative flux value at the top most numeric row. Column 3 is removed, and new modes are created by combining it with all modes with a positive flux value at this row, i.e. with columns 4 and 6.

Combining columns 3 and 4, we get $(\star\star\cdot\cdot\star\star\star\star\,|\,0,0,0)^T$, and 3 and 5 yields $(\star\star\cdot\star\ \star\cdot\cdot\cdot\cdot\,|\,0,1,1)^T$. Note that the binary part is received by applying the AND function, since zero flux values are only preserved if both modes have zero flux, and since the new mode is a nonnegative combination of both modes. Both newly created modes are elementary, which can be verified by looking at the binary part: no mode, according to the combinatorial test, is a superset of the new modes (other than their ancestors). Finally, we remove mode 3, append the newly created modes and convert the processed row to binary:

$$\boldsymbol{R}^{10} = \left[\begin{array}{ccccccc} \cdot & \star & \star & \star & \star & \star & \star \\ \star & \cdot & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star & \cdot & \cdot \\ \star & \star & \cdot & \star & \star & \cdot & \star \\ \star & \star & \star & \cdot & \star & \star & \star \\ \star & \star & \star & \star & \cdot & \star & \cdot \\ \star & \star & \star & \star & \cdot & \star & \cdot \\ \star & \star & \star & \star & \cdot & \star & \cdot \\ \star & \star & \star & \cdot & \cdot & \star & \cdot \\ \star & \star & \cdot & \star & \cdot & \star & \star \\ \hline 0 & 1 & 1 & 0 & 2 & 0 & 1 \\ 1 & -1 & -1 & 1 & 0 & 0 & 1 \end{array}\right] \tag{A.11}$$

The next step is again trivial:

$$\boldsymbol{R}^{11} = \left[\begin{array}{ccccccc} \cdot & \star & \star & \star & \star & \star & \star \\ \star & \cdot & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star & \cdot & \cdot \\ \star & \star & \cdot & \star & \star & \cdot & \star \\ \star & \star & \star & \cdot & \star & \star & \star \\ \star & \star & \star & \star & \cdot & \star & \cdot \\ \star & \star & \star & \star & \cdot & \star & \cdot \\ \star & \star & \star & \star & \cdot & \star & \cdot \\ \star & \star & \star & \cdot & \cdot & \star & \cdot \\ \star & \star & \cdot & \star & \cdot & \star & \star \\ \star & \cdot & \cdot & \star & \cdot & \star & \cdot \\ \hline 1 & -1 & -1 & 1 & 0 & 0 & 1 \end{array}\right] \tag{A.12}$$

Constructing the bit pattern trees

For the last iteration step, we finally create bit pattern trees, namely the trees T^+, T^- and T^0 for modes with positive, negative and zero flux for the remaining reaction. For simplicity, we use the binary rows 1, 2 and 3 to separate the sets of the subtrees. Any rows could be chosen, e.g. to balance the trees. However, it has proven advantageous to use the same rows at least for T^+ and T^-. In the present example, we get rather small and degenerated trees.

Let's construct T^+ with columns 1, 4 and 7. The first binary row separates mode 1 from the others. Mode 4 and 7 are still not separated by the second bit, but the third finally yields the tree with unary leaf nodes. In practical implementations, the leaf nodes can also contain multiple modes to reduce the tree size (e.g. up to 4 as in the current implementation). Furthermore, we would remove the node $Z_U^{+\star}$ and shrink the linear path in the tree.

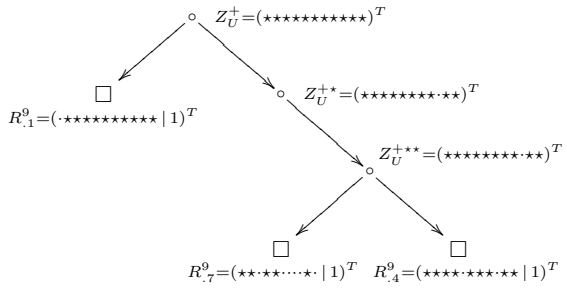


Figure A.11: Bit pattern tree T^+ with columns 1, 4 and 7 as leaf nodes, and union patterns Z_U^+, $Z_U^{+\star}$ and $Z_U^{+\star\star}$ at intermediary nodes, unifying the zero sets contained in the subtree (i.e. applying the OR operation to the binary parts of the modes).

Using columns 2 and 3, we get T^{-}. The two modes are split by the second binary row.

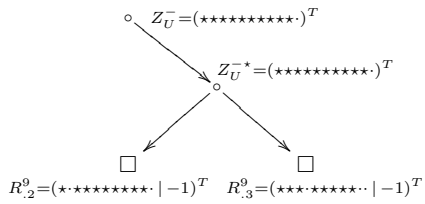


Figure A.12: Bit pattern tree T^{-} with columns 2 and 3 as leaf nodes, and union patterns Z_U^- and $Z_U^{-\star}$ at intermediary nodes.

From the remaining columns 5 and 6, we get T^0. The two modes are not split before the third binary row. Again, this linear pathway would be shortened in practice.

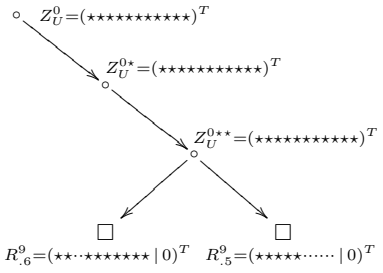


Figure A.13: Bit pattern tree T^0 with columns 5 and 6 as leaf nodes, and union patterns Z_U^0, $Z_U^{0\star}$ and $Z_U^{0\star\star}$ at intermediary nodes.

Recursive enumeration of elementary mode candidates

To enumerate all ancestor candidates for new modes $(r^+, r^-) \in (T^+, T^-)$, we recursively pair the subtrees of T^+ and T^-. But before recursing, we perform an initial *prerequirement-check*[1]. Any necessary precondition could be used, but as recommended in the main text, we only check whether the minimum set size is met. The set to be tested is the *cut pattern* $Z_C = Z_U^+ \cap Z_U^-$, and the minimum set size is

$$card_{min} = q - 2 - rank(\boldsymbol{N}) = 12 - 2 - 6 = 4 \tag{A.13}$$

Obviously, this condition is met, and we can pair the subtrees now. Usually, this leads to 4 subtree recursions, but here, T^- has only one child tree on the right, represented by the intermediary node $T^{-\star}$ with associated union pattern $Z_U^{-\star}$. The T^+ tree has two child nodes, hence we have two recursions on the first tree level[2]. We will start with the left child tree of T^+, which is actually a leaf node containing the mode $R^9_{.1}$.

Again, before continuing the recursion in a depth-first manner, we primarily check if the minimum set size is met. Here, the set to be tested is the $Z_C^{(\cdot,\star)} = Z_U^{+\cdot} \cap Z_U^{-\star}$. Note that the union pattern $Z_U^{+\cdot}$ coincides with the binary part of the mode $R^9_{.1}$, since we have unary leaves. Thus, $Z_C^{(\cdot,\star)} = (\cdot\star\star\star\star\star\star\star\star\star\star\cdot)^T$, which obviously also passes the test.

Now, we have already reached the bottom of the tree T^+, and recursing continues only within the other tree[3]. First, we take the left subtree $T^{-\star\cdot}$, a leaf node with mode $R^9_{.2}$. For leaves with multiple modes, we would again test the cut pattern first[4], but here, it coincides with the binary part of the new mode candidate $R^9_{(1,2)} = (\cdot\cdot\star\star\star\star\star\star\star\star\cdot|\,0)^T$, and we immediately test the candidate. Again, the minimum set size is met, and we have to perform the final *adjacency-test*. We defer this test to the next subsection, stating only that it passes the test, and continue here with our recursion.

The *deepest pending recursion* occurred at node $T^{-\star}$, where we only traversed the left child tree. Traversing the right child tree, we get again a new mode candidate $R^9_{(1,3)} = (\cdot\star\star\cdot\star\star\star\star\cdot\cdot|\,0)^T$, and also this mode fulfills the minimum set size and passes the adjacency test.

[1] Compare with line 2 in Alg. 6: `AddAdjacent`.
[2] In Alg. 6: `AddAdjacent`, missing subtrees of intermediary nodes are not handled specifically, since one can simply connect the single child tree to the parent node.
[3] Until now, the recursions correspond to line 5 in Alg. 6: `AddAdjacent`. Now, we recurse according to line 25.
[4] Again line 2 in Alg. 6: `AddAdjacent`.

We have now completed the recursion of the top left subtree of T^+ with the whole tree T^-. The recursion steps are summarized and visualized in Fig. A.14.

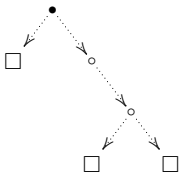
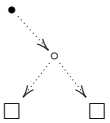
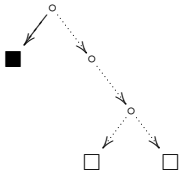
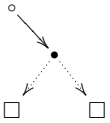
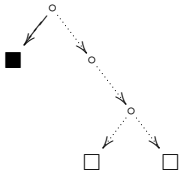
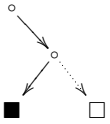
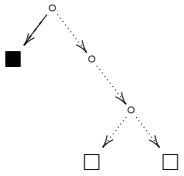
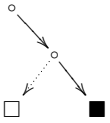
Step	T^+	T^-	Z_C / $Z_\cap$	*Description*
1			$Z_C = Z_U^+ \cap Z_U^-$ $(\star\star\star\star\star\star\star\star\star\star\cdot)^T$	Perform a set size test. Passed since set contains $10 \geq 4$ elements.
2			$Z_C^{(\cdot,\star)} = Z_U^{+\cdot} \cap Z_U^{-\star}$ $= R_{\cdot 1}^9 \cap Z_U^{-\star}$ $(\cdot\star\star\star\star\star\star\star\star\star\cdot)^T$	Perform a set size test. Passed, $9 \geq 4$ elements.
3			$Z_C^{(\cdot,\star\cdot)} = Z_U^{+\cdot} \cap Z_U^{-\star\cdot}$ $= Z_\cap$ $= R_{\cdot 1}^9 \cap R_{\cdot 2}^9$ $(\cdot\cdot\star\star\star\star\star\star\star\star\cdot)^T$	Perform a set size test. Passed, $8 \geq 4$ elements. Two leaves, perform adjacency test. Also passed.
4			$Z_C^{(\cdot,\star\star)} = Z_U^{+\cdot} \cap Z_U^{-\star\star}$ $= Z_\cap$ $= R_{\cdot 1}^9 \cap R_{\cdot 3}^9$ $(\cdot\star\star\cdot\star\star\star\star\star\cdot\cdot)^T$	Perform a set size test. Passed, $7 \geq 4$ elements. Two leaves, perform adjacency test. Also passed.

Figure A.14: Recursive enumeration of new mode candidates with bit pattern trees T^+ and T^-. Here, the recursion of the top left subtree of T^+ and the whole tree T^- is shown.

The next recursion pairs the top right subtree of T^+ with the top left subtree of T^-. All remaining recursion steps are given in Fig. A.15.

Five of six candidates for new elementary modes pass all adjacency tests (see Fig. A.14 and A.15, recursion steps 3, 4, 7, 8 and 10). One candidate fails the test, since the new mode would have too few zero fluxes (see Fig. A.15, recursion step 9).

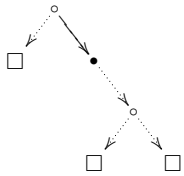
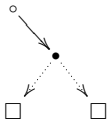
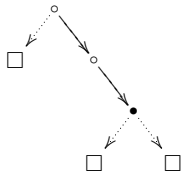
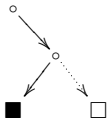
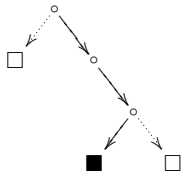
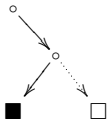
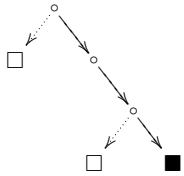
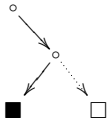
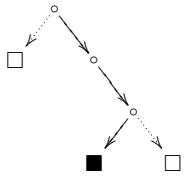
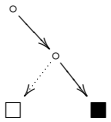
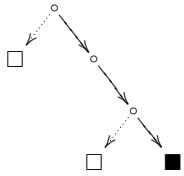
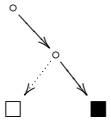
Step	T^+	T^-	Z_C / $Z_\cap$	*Description*
5			$Z_C^{(\star,\star)} = Z_U^{+\star} \cap Z_U^{-\star}$ $(\star\star\star\star\star\star\star\star\cdot\star\cdot)^T$	Perform a set size test. Passed, $9 \geq 4$ elements.
6			$Z_C^{(\star\star,\star\cdot)} = Z_U^{+\star\star} \cap Z_U^{-\star\cdot}$ $= Z_U^{+\star\star} \cap R^9_{\cdot 2}$ $(\star\cdot\star\star\star\star\star\star\cdot\star\cdot)^T$	Perform a set size test. Passed, $8 \geq 4$ elements.
7			$Z_C^{(\star\star\cdot,\star\cdot)} = Z_U^{+\star\star\cdot} \cap Z_U^{-\star\cdot}$ $= Z_\cap$ $= R^9_{\cdot 7} \quad \cap R^9_{\cdot 2}$ $(\star\cdot\cdot\star\star\cdot\cdot\cdot\cdot\star\cdot)^T$	Perform a set size test. Passed, $4 \geq 4$ elements. Two leaves, perform adjacency test. Also passed.
8			$Z_C^{(\star\star\star,\star\cdot)} = Z_U^{+\star\star\star} \cap Z_U^{-\star\cdot}$ $= Z_\cap$ $= R^9_{\cdot 4} \quad \cap R^9_{\cdot 2}$ $(\star\cdot\star\star\cdot\star\star\star\cdot\star\cdot)^T$	Perform a set size test. Passed, $7 \geq 4$ elements. Two leaves, perform adjacency test. Also passed.
9			$Z_C^{(\star\star\cdot,\star\star)} = Z_U^{+\star\star\cdot} \cap Z_U^{-\star\star}$ $= Z_\cap$ $= R^9_{\cdot 7} \quad \cap R^9_{\cdot 3}$ $(\star\star\cdot\cdot\star\cdot\cdot\cdot\cdot\cdot\cdot)^T$	Perform a set size test. **Not passed**, $3 \not\geq 4$ elements. No adjacency test needed.
10			$Z_C^{(\star\star\star,\star\star)} = Z_U^{+\star\star\star} \cap Z_U^{-\star\star}$ $= Z_\cap$ $= R^9_{\cdot 4} \quad \cap R^9_{\cdot 3}$ $(\star\star\star\cdot\cdot\star\star\star\cdot\cdot\cdot)^T$	Perform a set size test. Passed, $6 \geq 4$ elements. Two leaves, perform adjacency test. Also passed.

Figure A.15: Recursive enumeration of new mode candidates with bit pattern trees T^+ and T^-. Here, the recursion of the top right subtrees of T^+ and T^- is shown.

After removing modes 2 and 3 and adding the 5 new modes, we convert the last row from numeric to binary:

$$\boldsymbol{R}^{12} = \begin{bmatrix} \cdot & \star & \star & \star & \star & \cdot & \cdot & \star & \star & \star \\ \star & \star & \star & \star & \star & \cdot & \star & \cdot & \cdot & \star \\ \star & \star & \star & \cdot & \cdot & \star & \star & \cdot & \star & \star \\ \star & \star & \star & \cdot & \star & \star & \cdot & \star & \star & \cdot \\ \star & \cdot & \star & \star & \star & \star & \star & \star & \cdot & \cdot \\ \star & \star & \cdot & \star & \cdot & \star & \star & \cdot & \star & \star \\ \star & \star & \cdot & \star & \cdot & \star & \star & \cdot & \star & \star \\ \star & \star & \cdot & \star & \cdot & \star & \star & \cdot & \star & \star \\ \star & \cdot & \cdot & \star & \cdot & \star & \star & \cdot & \cdot & \cdot \\ \star & \star & \cdot & \star & \star & \star & \cdot & \star & \star & \cdot \\ \star & \star & \cdot & \star & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \star & \star & \cdot & \star & \star & \star & \star & \star \end{bmatrix} \tag{A.14}$$

Remark: In this example, we have performed 10 set size tests and 5 adjacency tests using pattern trees. The direct approach (i.e. testing all six candidates) results in only 6 set size tests and 5 adjacency tests. In this small example, no *real* cut pattern Z_C (one which does not coincide with a candidate intersection set $Z_\cap$) fails the test. For realistic problems, many subtree combinations cannot yield new elementary modes, which can be detected early by performing a cut pattern tests. In such a case, we can save many tests at the price of very little additional effort.

Adjacency testing with bit pattern trees

Combinatorial test

In this section, we exemplify the use of bit pattern trees to perform an adjacency test. The combinatorial test requires that no mode exists with at least all zero fluxes of the candidate mode to be tested (different from the ancestor modes of the candidate, see Prop. 2.6d). Given $Z_\cap$, the zero fluxes for the new mode candidate, we check if we find a superset of $Z_\cap$ in the binary part of the modes (excluding the ancestor modes).

Pseudo code for the combinatorial adjacency test with bit pattern trees is given in Fct. `hasSuperSet()`, p. 62 for intermediate and leaf tree nodes. For our example, we use the first candidate arising from the ancestor modes 1 and 2 (see recursion step 3 in Fig. A.14). We test the candidate intersection set $Z_\cap = R^9_{\cdot 1} \cap R^9_{\cdot 2} = (\cdot\cdot\star\star\star\star\star\star\star\star\cdot)^T$ against all trees T^+, T^- and T^0 (see Fig. A.16).

Starting with T^+ (see Fig. A.11), we first test whether $Z_\cap \subseteq Z^+_U$, which is the case. Then, we take the left branch, ending up in the leaf node containing mode 1, one of our ancestors, and we omit the test. Next, we take the right branch of the tree, and we test whether $Z_\cap \subseteq Z^{+\star}_U$. The expression evaluates to *false* due to the third element from last, and this test fails. Hence, we can abort traversing this tree branch, and we already know that tree T^+ does not contain a super set of $Z_\cap$.

Next, we traverse T^- (see Fig. A.12), and after passing the tests $Z_\cap \subseteq Z_U^-$ and $Z_\cap \subseteq Z_U^{-\star}$, we end up in the left leaf node containing mode 2, again an ancestor of our candidate. The right leaf node contains mode 3, and one verifies that its binary part is not a superset of $Z_\cap$.

Traversing the last tree T^0 (see Fig. A.13) works in a similar way: the first three tests are passed, since $Z_\cap \subseteq Z_U^0, Z_U^{0\star}, Z_U^{0\star\star}$. Then, neither the binary part of mode 6 in the left nor that of mode 5 in the right leaf node is a superset of $Z_\cap$, and hence, our candidate passes the adjacency test.

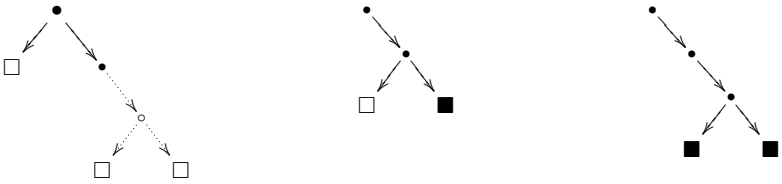


Figure A.16: Traversing bit pattern trees T^+, T^- and T^0 to perform the adjacency test with candidate intersection set $Z_\cap = (\cdots\star\star\star\star\star\star\star\star\cdot)^T$. Straight arrows indicate traversed paths, dotted arrows are not traversed. At filled intermediate nodes (•) and leaf nodes (■), a subset test was performed.

Remark: In this example, we have performed 10 subset tests to verify the elementarity of the selected candidate mode with bit pattern trees. The straightforward approach to iterate over all 7 modes leads to 5 subset tests, omitting the ancestor modes of our candidate. However, for realistic problems, the bit pattern trees can become quite large, and we can abort the traversion process early in many cases. In such cases, we can save many tests (i.e. all tests of the subtree, like in the example with T^+) at the price of a few additional tests with union patterns of intermediary nodes.

Rank updating

According to Prop. 2.6c, adjacency can also be tested by computing the rank of a submatrix of the stoichiometric matrix. More precisely, we take the columns of $\boldsymbol{N^{ext}}$ corresponding to non-zero flux values.

With recursive enumeration of candidates, we traverse the two trees T^+ and T^-, and non-zero flux values common to the whole subtrees are reflected by the cut pattern Z_C.

Consider for instance recursion step 5 in Fig. A.15. Inspecting cut pattern $Z_C^{(\star,\star)} = (\star\star\star\star\ \star\star\star\star\cdot\star\cdot)^T$, we already know at this early stage of traversing that all submatrices will contain columns 10 and 12. Hence, we use them as pivot columns and start the triangularization process. In step 6, column 2 is added and triangularized. The partly triangularized matrices look like this:

$$\left[\begin{array}{cc|cccc} \overbrace{\times}^{10} & \overbrace{\otimes}^{12} & \otimes & \otimes & \otimes & \otimes \\ 0 & \times & \otimes & \otimes & \otimes & \otimes \\ \hline 0 & 0 & \otimes & \otimes & \otimes & \otimes \\ 0 & 0 & \otimes & \otimes & \otimes & \otimes \\ 0 & 0 & \otimes & \otimes & \otimes & \otimes \\ \vdots & \vdots & \vdots & \vdots & \ddots & \end{array}\right] \qquad \left[\begin{array}{ccc|ccc} \overbrace{\times}^{10} & \overbrace{\otimes}^{12} & \overbrace{\otimes}^{2} & \otimes & \otimes & \otimes \\ 0 & \times & \otimes & \otimes & \otimes & \otimes \\ 0 & 0 & \times & \otimes & \otimes & \otimes \\ \hline 0 & 0 & 0 & \otimes & \otimes & \otimes \\ 0 & 0 & 0 & \otimes & \otimes & \otimes \\ \vdots & \vdots & \vdots & \vdots & \ddots & \end{array}\right]$$

$$\boldsymbol{N}^{ext}_{\triangle 10,12} \qquad\qquad \boldsymbol{N}^{ext}_{\triangle 10,12,2}$$

Figure A.17: Rank updating: partially triangularized matrix after adding columns 10 and 12 to the triangularized part (left). Continued triangularization after additionally adding column 2 (right). $\times$ stands for non-zero pivot elements, $\otimes$ for any value.

When we continue the recursion process, we finally perform the actual rank computation to test for adjacency (steps 7, 8 and 10). All rank computations can now benefit from the partial triangularization. The rank computations in step 7 and 8 can use the right matrix $\boldsymbol{N}^{ext}_{\triangle 10,12,2}$ in Fig. A.17. In step 9, no rank is computed, and in step 10, we can still benefit from the partial triangularization by basing the rank computation on the matrix $\boldsymbol{N}^{ext}_{\triangle 10,12}$.

A4.1.3 Postprocessing phase

First, we unmap the rows in our final binary matrix $\boldsymbol{R}^{12}$. The rows $1 \dots 12$ correspond to the reactions $(3, 6, 9, 10, 11, 5, 12, 8, 4, 7, 1, 2)$, put back to their original position, we get

$$\boldsymbol{R}^{binary} = \begin{bmatrix} \star & \star & \cdot & \star & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \star & \star & \cdot & \star & \star & \star & \star & \star \\ \cdot & \star & \star & \star & \star & \cdot & \cdot & \star & \star & \star \\ \star & \cdot & \cdot & \star & \cdot & \star & \star & \cdot & \cdot & \cdot \\ \star & \star & \cdot & \star & \cdot & \star & \star & \cdot & \star & \star \\ \star & \star & \star & \star & \star & \cdot & \star & \cdot & \cdot & \star \\ \star & \star & \cdot & \star & \star & \star & \cdot & \star & \star & \cdot \\ \star & \star & \cdot & \star & \cdot & \star & \star & \cdot & \star & \star \\ \star & \star & \star & \cdot & \cdot & \star & \star & \cdot & \star & \star \\ \star & \star & \star & \cdot & \star & \star & \cdot & \star & \star & \cdot \\ \star & \cdot & \star & \star & \star & \star & \star & \star & \cdot & \cdot \\ \star & \star & \cdot & \star & \cdot & \star & \star & \cdot & \star & \star \end{bmatrix} \tag{A.15}$$

Now, each binary mode is converted back to numeric. The $\star$ elements in the mode correspond to zero flux values, for the remaining ones, the equation

$$\boldsymbol{N}_{\cdot \overline{Z}(\boldsymbol{r})} \boldsymbol{r}_{\overline{Z}(\boldsymbol{r})} = \boldsymbol{0} \tag{A.16}$$

must hold and is unique up to a constant scaling factor (see Prop. 3.2).

For the first mode, only row 2 and 3 are non-zero in $\boldsymbol{R}^{binary}$, thus, we take the columns 2 and 3 of $\boldsymbol{N}^{exp}$, and from

$$\begin{gathered} \boldsymbol{N}_{\cdot(2,3)} \qquad \boldsymbol{r}^{\,1}_{(2,3)} \quad = \quad \boldsymbol{0} \\ \begin{bmatrix} 0 & 0 \\ 1 & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} r_2 \\ r_3 \end{bmatrix} \quad = \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{gathered} \tag{A.17}$$

we get

$$\boldsymbol{r}^{\,1}_{(2,3)} = null\big(\boldsymbol{N}_{\cdot(2,3)}\big) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{A.18}$$

and thus $\boldsymbol{r}^{\,1} = (0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T$.

Repeating this for all columns in $\boldsymbol{R}^{binary}$, we get the (still expanded) numeric modes as columns in

$$\boldsymbol{R}^{exp} = \begin{bmatrix} 0 & 0 & 2 & 0 & 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{A.19}$$

Now, we recombine the rows (2,3) and (9,10) corresponding to the reversible reactions R2 and R8, respectively (see eq. (2.69)). Note that for every reversible reaction, a futile cycle mode must exist, and we primarily remove them (here, modes 1 and 4).

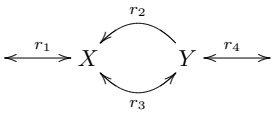
The resulting 8 elementary modes in the columns of $\boldsymbol{R}$ are

$$\boldsymbol{R} = \begin{bmatrix} 0 & 2 & 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 0 & 1 & -1 & -1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 1 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{A.20}$$

A4.2 Feasibility of Elementary Modes

This section contains examples related to Section 6.2.

Example A.1. *Let us consider the following network (default direction is left to right for reversible reactions):*



$$\boldsymbol{S} = \begin{bmatrix} 1 & 1 & -1 & 0 \\ 0 & -1 & 1 & -1 \end{bmatrix} \quad \boldsymbol{E} = \begin{bmatrix} 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & -1 & 0 \end{bmatrix}$$

Figure A.18: Exemplary metabolic network with stoichiometric matrix $\boldsymbol{S}$ and elementary modes (EMs, columns of $\boldsymbol{E}$).

Flux patterns *(see Def. 6.1) consider only the direction of the fluxes, but not the concrete* flux value. *As an illustration for a* feasibility classifier *(see Def. 6.2), we can simply think of a set of forbidden* flux patterns. *Note that sub-patterns—flux patterns that contain the forbidden pattern plus additional fluxes—are also forbidden by definition. For instance,*

a) Disallowing the flux patterns $\{-3\}$ *and* $\{-4\}$ *disables all flux vectors containing a right to left flux for reaction* 3 OR *reaction* 4.

b) Disallowing flux pattern $\{-3, -4\}$ *prohibits flux into the same direction for both reactions at the same time (*AND*), still allowing a flux in this direction for only one reaction.*

Sample fluxes:

$$\begin{aligned} \boldsymbol{v}' &= [\ 1 \ \ 1 \ \ 2 \ \ 1\ ] \\ \boldsymbol{v}'' &= [\ -2 \ \ 1 \ \ -1 \ \ -2\ ] \end{aligned}$$

The patterns for the EMs and the flux vectors according to Def. 6.1 are

$$\begin{array}{lll} \phi(\boldsymbol{e}_1): \{1,\ 3,\ 4\} & \phi(\boldsymbol{e}_3): \{-1,\ 2,\ -4\} & \phi(\boldsymbol{v}'): \{1,\ 2,\ 3,\ 4\} \\ \phi(\boldsymbol{e}_2): \{-1,\ -3,\ -4\} & \phi(\boldsymbol{e}_4): \{2,\ 3\} & \phi(\boldsymbol{v}''): \{-1,\ 2,\ -3,\ -4\} \end{array}$$

Condition a) disallows $\boldsymbol{e}_2$, $\boldsymbol{e}_3$ *and* $\boldsymbol{v}_B$, *condition b)* $\boldsymbol{e}_2$ *and* $\boldsymbol{v}_B$ *respectively.*

Example A.2. *Let us consider a dissallowed flux pattern, denoted by* ϕ^{inf} *for* infeasible pattern*:* $\phi^{inf} = \{-3, -4\}$. *Furthermore, we are interested in the following two feasible flux vectors:*

$$\begin{aligned} \boldsymbol{v}_A &= 2\,\boldsymbol{e}_1 + \boldsymbol{e}_2 = [\ 1 \ \ 0 \ \ 1 \ \ 1\ ] = \boldsymbol{e}_1 \\ \boldsymbol{v}_B &= 2\,\boldsymbol{e}_2 + 2\,\boldsymbol{e}_4 = [\ -2 \ \ 2 \ \ 0 \ \ -2\ ] \simeq \boldsymbol{e}_3 \end{aligned}$$

The elementary mode $\boldsymbol{e}_2$ *is classified infeasible by* ϕ^{inf}. *The feasible vectors* $\boldsymbol{v}_A$ *and* $\boldsymbol{v}_B$ *can be composed without using the infeasible EM* $\boldsymbol{e}_2$, *since but both vectors are themselves simply scaled feasible EMs.*

Example A.3. *Let us now assume an infeasibility pattern* $\phi^{inf} = \{3\}$ *classifying* $\boldsymbol{e}_1$ *and* $\boldsymbol{e}_4$ *infeasible. We are considering the following feasible flux vectors:*

$$\begin{array}{rcrcrcl} \boldsymbol{r}_C & = & 2\,\boldsymbol{e}_2 & + & \boldsymbol{e}_4 & = & [\quad -2 \quad 1 \quad -1 \quad -2 \quad] \\ \boldsymbol{r}_D & = & 1.5\,\boldsymbol{e}_2 & + & \boldsymbol{e}_4 & = & [\quad -1.5 \quad 1 \quad -0.5 \quad -1.5 \quad] \end{array}$$

Using Elimination Lemma 6.1, we find alternative compositions for $\boldsymbol{r}_C$ *and* $\boldsymbol{r}_D$*:*

1. *Note that* $\phi(\boldsymbol{e}_2) \subset \phi(\boldsymbol{r}_C)$ *and* $\phi(\boldsymbol{e}_2) \subset \phi(\boldsymbol{r}_D)$*. Hence* $\boldsymbol{r}_C$ *and* $\boldsymbol{r}_D$ *are not EMs, since we have* $\zeta(\boldsymbol{e}'_2) \supset \zeta(\boldsymbol{r}'_C)$ *and* $\zeta(\boldsymbol{e}'_2) \supset \zeta(\boldsymbol{r}'_D)$ *for the zero sets (eq. 2.39) in the EM space (Def. 2.39).*

2. *We construct the vectors*

$$\begin{array}{rcllcl} \boldsymbol{r}_{1C} & = & \boldsymbol{r}_C - \lambda_C \cdot \boldsymbol{e}_2, & \lambda_C & = & \min(\ {}^{-2}/_{-1},\ n/a,\ {}^{-1}/_{-1},\ {}^{-2}/_{-1}\ ) = 1 \\ \boldsymbol{r}_{1D} & = & \boldsymbol{r}_D - \lambda_D \cdot \boldsymbol{e}_2, & \lambda_D & = & \min(\ {}^{-1.5}/_{-1},\ n/a,\ {}^{-0.5}/_{-1},\ {}^{-1.5}/_{-1}\ ) = 0.5 \end{array}$$

3. *Since* $\boldsymbol{r}_{1C}, \boldsymbol{r}_{1D} \simeq \boldsymbol{e}_3$ *are EMs, we find*

$$\begin{array}{rcrcl} \boldsymbol{r}_C & = & \boldsymbol{e}_2 & + & \boldsymbol{e}_3 \\ \boldsymbol{r}_D & = & 0.5\,\boldsymbol{e}_2 & + & \boldsymbol{e}_3 \end{array}$$

Bibliography

4ti2 team. 2006. 4ti2—A software package for algebraic, geometric and combinatorial problems on linear spaces. Available at www.4ti2.de.

Acuña, V., F. Chierichetti, V. Lacroix, A. Marchetti-Spaccamela, M.-F. Sagot, and L. Stougie. 2009. Modes and cuts in metabolic networks: complexity and algorithms. Biosystems **95**:51–60.

Akesson, M., J. Förster, and J. Nielsen. 2004. Integration of gene expression data into genome-scale metabolic models. Metab Eng **6**:285–293.

Apweiler, R., A. Bairoch, C. H. Wu, et al. 2004. UniProt: the Universal Protein Knowledgebase. Nucleic Acids Res **32**:115–119.

Arakawa, K., Y. Yamada, K. Shinoda, Y. Nakayama, and M. Tomita. 2006. GEM System: automatic prototyping of cell-wide metabolic pathway models from genomes. BMC Bioinformatics **7**:168.

Avis, D., and D. Bremner. 1995. How Good are Convex Hull Algorithms? Computational Geometry: Theory and Applications **7**:265–301.

Avis, D., and K. Fukuda. 1991. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. In: SCG '91: Proceedings of the seventh annual symposium on Computational geometry. New York, NY, USA: ACM, 98–104.

Bairoch, A. 2000. The ENZYME database in 2000. Nucl. Acids Res. **28**:304–305.

Becker, S. A., and B. Ø. Palsson. 2005. Genome-scale reconstruction of the metabolic network in Staphylococcus aureus N315: an initial draft to the two-dimensional annotation. BMC Microbiol **5**:8.

Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. Commun. ACM **18**:509–517.

Bilu, Y., T. Shlomi, N. Barkai, and E. Ruppin. 2006. Conservation of Expression and Sequence of Metabolic Genes Is Reflected by Activity Across Metabolic States. PLoS Comput Biol **2**.

Bonneau, R., M. T. Facciotti, D. J. Reiss, et al. 2007. A predictive model for transcriptional control of physiology in a free living cell. Cell **131**:1354–1365.

Borgwardt, K. H. 2007. Average-Case Analysis of the Double Description Method and the Beneath-Beyond Algorithm. Discrete Comput. Geom. **37**:175–204.

Burgard, A. P., and C. D. Maranas. 2003. Optimization-based framework for inferring and testing hypothesized metabolic objective functions. Biotechnol Bioeng **82**:670–677.

Burgard, A. P., E. V. Nikolaev, C. H. Schilling, and C. D. Maranas. 2004. Flux coupling analysis of genome-scale metabolic network reconstructions. Genome Res **14**:301–312.

Burgard, A. P., P. Pharkya, and C. D. Maranas. 2003. Optknock: a bilevel programming framework for identifying gene knockout strategies for microbial strain optimization. Biotechnol Bioeng **84**:647–657.

Caspi, R., H. Foerster, C. A. Fulcher, et al. 2006. MetaCyc: a multiorganism database of metabolic pathways and enzymes. Nucleic Acids Res **34**:D511–D516.

Chazelle, B. 1993. An Optimal Convex Hull Algorithm in Any Fixed Dimension. Discrete & Computational Geometry **10**:377–409.

Chernikova, N. V. 1965. Algorithm for Finding a General Formula for the Non-Negative Solutions of System of Linear Inequalities. U.S.S.R. Computational Mathematics and Mathematical Physics **5**:228–233.

Clauset, A., C. Moore, and M. E. J. Newman. 2008. Hierarchical structure and the prediction of missing links in networks. Nature **453**:98–101.

Conradi, C., D. Flockerzi, J. Raisch, and J. Stelling. 2007. Subnetwork analysis reveals dynamic features of complex (bio)chemical networks. Proc Natl Acad Sci U S A **104**:19175–19180.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein. 2001. Introduction to Algorithms. The MIT Press, 2nd edition.

Covert, M. W., E. M. Knight, J. L. Reed, M. J. Herrgard, and B. O. Palsson. 2004. Integrating high-throughput and computational data elucidates bacterial networks. Nature **429**:92–96.

Covert, M. W., and B. Ø. Palsson. 2002. Transcriptional regulation in constraints-based metabolic models of Escherichia coli. J Biol Chem **277**:28058–28064.

Covert, M. W., and B. O. Palsson. 2003. Constraints-based models: regulation of gene expression reduces the steady-state solution space. J Theor Biol **221**:309–325.

Covert, M. W., C. H. Schilling, I. Famili, J. S. Edwards, I. I. Goryanin, E. Selkov, and B. O. Palsson. 2001a. Metabolic modeling of microbial strains in silico. Trends Biochem Sci **26**:179–186.

Covert, M. W., C. H. Schilling, and B. Palsson. 2001b. Regulation of gene expression in flux balance models of metabolism. J Theor Biol **213**:73–88.

Covert, M. W., N. Xiao, T. J. Chen, and J. R. Karr. 2008. Integrating metabolic, transcriptional regulatory and signal transduction models in Escherichia coli. Bioinformatics **24**:2044–2050.

Craciun, G., Y. Tang, and M. Feinberg. 2006. Understanding bistability in complex enzyme-driven reaction networks. Proc Natl Acad Sci U S A **103**:8697–8702.

Dauner, M., M. Sonderegger, M. Hochuli, T. Szyperski, K. Wüthrich, H.-P. Hohmann, U. Sauer, and J. E. Bailey. 2002. Intracellular Carbon Fluxes in Riboflavin-Producing *Bacillus subtilis* during Growth on Two-Carbon Substrate Mixtures. Applied and Environmental Microbiology **68**.

DeJongh, M., K. Formsma, P. Boillot, J. Gould, M. Rycenga, and A. Best. 2007. Toward the automated generation of genome-scale metabolic networks in the SEED. BMC Bioinformatics **8**:139.

Duarte, N. C., M. J. Herrgård, and B. Ø. Palsson. 2004. Reconstruction and validation of Saccharomyces cerevisiae iND750, a fully compartmentalized genome-scale metabolic model. Genome Res **14**:1298–1309.

Edelsbrunner, H. 1987. Algorithms in combinatorial geometry. New York, NY, USA: Springer-Verlag New York, Inc.

Edwards, J. S., R. U. Ibarra, and B. O. Palsson. 2001. In silico predictions of Escherichia coli metabolic capabilities are consistent with experimental data. Nat Biotechnol **19**:125–130.

Edwards, J. S., and B. O. Palsson. 2000. The Escherichia coli MG1655 in silico metabolic genotype: its definition, characteristics, and capabilities. Proc Natl Acad Sci U S A **97**:5528–5533.

Famili, I., and B. Ø. Palsson. 2003. The convex basis of the left null space of the stoichiometric matrix leads to the definition of metabolically meaningful pools. Biophys J **85**:16–26.

Feist, A. M., C. S. Henry, J. L. Reed, M. Krummenacker, A. R. Joyce, P. D. Karp, L. J. Broadbelt, V. Hatzimanikatis, and B. O. Palsson. 2007. A genome-scale metabolic reconstruction for Escherichia coli K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information. Mol Syst Biol **3**.

Feist, A. M., and B. Ø. Palsson. 2008. The growing scope of applications of genome-scale metabolic reconstructions using Escherichia coli. Nat Biotechnol **26**:659–667.

Feist, A. M., J. C. M. Scholten, B. Ø. Palsson, F. J. Brockman, and T. Ideker. 2006. Modeling methanogenesis with a genome-scale metabolic reconstruction of Methanosarcina barkeri. Mol Syst Biol **2**:2006.0004.

Fischer, E., and U. Sauer. 2003. A novel metabolic cycle catalyzes glucose oxidation and anaplerosis in hungry *Escherichia coli*. J Biol Chem **278**:46446–46451.

———. 2005. Large-scale in vivo flux analysis shows rigidity and suboptimal performance of Bacillus subtilis metabolism. Nat Genet **37**:636–640.

Fong, S. S., A. R. Joyce, and B. Ø. P. of Escherichia coli evolve to computationally predicted growth phenotypes. 2005. Parallel adaptive evolution cultures of Escherichia coli lead to convergent growth phenotypes with different gene expression states. Genome Res **15**:1365–1372.

Fong, S. S., and B. Ø. Palsson. 2004. Metabolic gene-deletion strains of Escherichia coli evolve to computationally predicted growth phenotypes. Nat Genet **36**:1056–1058.

Fukuda, K. 1996. cddlib-094, C–library for polyhedral computations. Available at http://www.ifor.math.ethz.ch/~fukuda/cdd_home/index.html.

Fukuda, K., and A. Prodon. 1995. Double Description Method Revisited. In: Combinatorics and Computer Science. 91–111.

Gagneur, J., and S. Klamt. 2004. Computation of elementary modes: A unifying framework and the new binary approach. BMC Bioinformatics **5**:175.

Gevorgyan, A., M. G. Poolman, and D. A. Fell. 2008. Detection of stoichiometric inconsistencies in biomolecular models. Bioinformatics **24**:2245–2251.

Gianchandani, E. P., M. A. Oberhardt, A. P. Burgard, C. D. Maranas, and J. A. Papin. 2008. Predicting biological system objectives de novo from internal state measurements. BMC bioinformatics **9**.

Golub, G. H., and C. F. Van Loan. 1996. Matrix computations (3rd ed.). Baltimore, MD, USA: Johns Hopkins University Press.

Henry, C. S., L. J. Broadbelt, and V. Hatzimanikatis. 2007. Thermodynamics-based metabolic flux analysis. Biophys J **92**:1792–1805.

Herrgård, M. J., B.-S. Lee, V. Portnoy, and B. Ø. Palsson. 2006. Integrated analysis of regulatory and metabolic networks reveals novel regulatory mechanisms in Saccharomyces cerevisiae. Genome Res **16**:627–635.

Herrgård, M. J., N. Swainston, P. Dobson, et al. 2008. A consensus yeast metabolic network reconstruction obtained from a community approach to systems biology. Nat Biotechnol **26**:1155–1160.

Hoppe, A., S. Hoffmann, and H.-G. Holzhütter. 2007. Including metabolite concentrations into flux balance analysis: thermodynamic realizability as a constraint on flux distributions in metabolic networks. BMC Syst Biol **1**:23.

Hucka, M., A. Finney, H. M. Sauro, et al. 2003. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics **19**:524–531.

Ibarra, R. U., J. S. Edwards, and B. O. Palsson. 2002. Escherichia coli K-12 undergoes adaptive evolution to achieve in silico predicted optimal growth. Nature **420**:186–189.

Izallalen, M., R. Mahadevan, A. Burgard, B. Postier, R. Didonato, J. Sun, C. H. Schilling, and D. R. Lovley. 2008. Geobacter sulfurreducens strain engineered for increased rates of respiration. Metab Eng **10**:267–275.

Jamshidi, N., and B. O. Palsson. 2008. Formulating genome-scale kinetic models in the post-genome era. Mol Syst Biol **4**.

Jevremovic, D., C. Trinh, F. Srienc, and D. Boley. 2008. A Simple Rank Test to Distinguish Extreme Pathways from Elementary Modes in Metabolic Networks. Computer Science and Engineering Technical Report **08-029**.

Kanehisa, M., S. Goto, S. Kawashima, and A. Nakaya. 2002. The KEGG databases at GenomeNet. Nucleic Acids Res **30**:42–46.

Karp, P. D., C. A. Ouzounis, C. M. Kochlacs, L. Goldovsky, P. Kaipa, D. Ahren, S. Tsoka, N. Darzentas, V. Kunin, and N. L. Bigas. 2005. Expansion of the BioCyc collection of pathway/genome databases to 160 genomes. Nucleic Acids Research **33**:6083–6089.

Karp, P. D., S. Paley, and P. Romero. 2002. The Pathway Tools software. Bioinformatics **18 Suppl 1**:S225–S232.

Khachiyan, L., E. Boros, K. Borys, K. Elbassioni, and V. Gurvich. 2006. Generating all vertices of a polyhedron is hard. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm. New York, NY, USA: ACM, 758–765.

Kharchenko, P., L. Chen, Y. Freund, D. Vitkup, and G. M. Church. 2006. Identifying metabolic enzymes with multiple types of association evidence. BMC Bioinformatics **7**:177.

Kharchenko, P., D. Vitkup, and G. M. Church. 2004. Filling gaps in a metabolic network using expression information. Bioinformatics **20 Suppl 1**:i178–i185.

Klamt, S. 2006. Generalized concept of minimal cut sets in biochemical networks. Biosystems **83**:233–247.

Klamt, S., J. Gagneur, and A. von Kamp. 2005. Algorithmic approaches for computing elementary modes in large biochemical reaction networks. IEE Proc. Systems Biol. **152**:249–55.

Klamt, S., and E. D. Gilles. 2004. Minimal cut sets in biochemical reaction networks. Bioinformatics **20**:226–234.

Klamt, S., and J. Stelling. 2002. Combinatorial complexity of pathway analysis in metabolic networks. Mol Biol Rep **29**:233–6.

———. 2003. Two approaches for metabolic pathway analysis? Trends Biotechnol **21**:64–69.

———. 2006. Stoichiometric and constraint-based modeling. In: Szallasi, Z., J. Stelling, and V. Periwal, editors, System Modeling in Cellular Biology. MIT Press (Cambridge / MA), 73–96.

Knuth, D. E. 1997. Seminumerical Algorithms, volume 2 of *The Art of Computer Programming*. Reading MA: Addison-Wesley, 3rd edition.

Kompala, D. S., D. Ramkrishna, and G. T. Tsao. 1984. Cybernetic modeling of microbial growth on multiple substrates. Biotechnol Bioeng **26**:1272–1281.

Kremling, A., K. Bettenbrock, and E. D. Gilles. 2007. Analysis of global control of Escherichia coli carbohydrate uptake. BMC Syst Biol **1**:42.

Kuepfer, L., U. Sauer, and L. M. Blank. 2005. Metabolic functions of duplicate genes in Saccharomyces cerevisiae. Genome Res **15**:1421–1430.

Kumar, V. S., M. S. Dasika, and C. D. Maranas. 2007. Optimization based automated curation of metabolic reconstructions. BMC Bioinformatics **8**:212.

Kümmel, A., S. Panke, and M. Heinemann. 2006. Putative regulatory sites unraveled by network-embedded thermodynamic analysis of metabolome data. Mol Syst Biol **2**:2006.0034.

Kümmel, A. M. R. 2008. Integrating thermodynamics-based modeling and quantitative experimental data for studying microbial metabolism. Doctor of sciences – Dis. no. 17812, ETH Zürich.

Lee, J. M., E. P. Gianchandani, J. A. Eddy, and J. A. Papin. 2008. Dynamic analysis of integrated signaling, metabolic, and regulatory networks. PLoS Comput Biol **4**:e1000086.

Lee, S., C. Palakornkule, M. M. Domach, and I. E. Grossmann. 2000. Recursive MILP model for finding all the alternate optima in LP models for metabolic networks. Computers & Chemical Engineering **24**:711 – 716.

Liebermeister, W., and E. Klipp. 2006a. Bringing metabolic networks to life: convenience rate law and thermodynamic constraints. Theor Biol Med Model **3**:41.

———. 2006b. Bringing metabolic networks to life: integration of kinetic, metabolic, and proteomic data. Theor Biol Med Model **3**:42.

Lovley, D. R. 2003. Cleaning up with genomics: applying molecular biology to bioremediation. Nat Rev Microbiol **1**:35–44.

Mahadevan, R., D. R. Bond, J. E. Butler, A. Esteve-Nuñez, M. V. Coppi, B. O. Palsson, C. H. Schilling, and D. R. Lovley. 2006. Characterization of metabolism in the Fe(III)-reducing organism Geobacter sulfurreducens by constraint-based modeling. Appl Environ Microbiol **72**:1558–1568.

Mahadevan, R., J. S. Edwards, and F. J. Doyle. 2002. Dynamic flux balance analysis of diauxic growth in Escherichia coli. Biophys J **83**:1331–1340.

Mahadevan, R., and C. H. Schilling. 2003. The effects of alternate optimal solutions in constraint-based genome-scale metabolic models. Metab Eng **5**:264–276.

Malkin, P. N. 2007. Computing Markov bases, Gröbner bases, and extreme rays. Doctorat en sciences appliquées, Département d'ingénierie mathématique – Université catholique de Louvain.

Maltsev, N., E. Glass, D. Sulakhe, A. Rodriguez, M. H. Syed, T. Bompada, Y. Zhang, and M. D'Souza. 2006. PUMA2–grid-based high-throughput analysis of genomes and metabolic pathways. Nucleic Acids Res **34**:D369–D372.

McMullen, P. 1970. The maximum numbers of faces of a convex polytope. Mathematika **17**:179–184.

Motzkin, T. S., H. Raiffa, G. Thompson, and R. M. Thrall. 1953. The double description method. In: Kuhn, H., and A. Tucker, editors, Contributions to the Theory of Games II, volume 8 of *Annals of Math. Studies*. Princeton University Press (Princeton / RI), 51–73.

Mulmuley, K. 1993. Computational Geometry: An Introduction Through Randomized Algorithms. Englewood Cliffs, NJ: Prentice Hall.

Nikoloski, Z., S. Grimbs, P. May, and J. Selbig. 2008. Metabolic networks are NP-hard to reconstruct. J Theor Biol **254**:807–816.

Notebaart, R. A., B. Teusink, R. J. Siezen, and B. Papp. 2008. Co-regulation of metabolic genes is better explained by flux coupling than by network distance. PLoS Comput Biol **4**.

Ogata, H., S. Goto, K. Sato, W. Fujibuchi, H. Bono, and M. Kanehisa. 1999. KEGG: Kyoto Encyclopedia of Genes and Genomes. Nucleic Acids Research **27**:29–34.

Ott, M. A., and G. Vriend. 2006. Correcting ligands, metabolites, and pathways. BMC Bioinformatics **7**:517.

Papin, J. A., and B. Ø. Palsson. 2004. The JAK-STAT signaling network in the human B-cell: an extreme signaling pathway analysis. Biophys J **87**:37–46.

Park, J. H., K. H. Lee, T. Y. Kim, and S. Y. Lee. 2007. Metabolic engineering of Escherichia coli for the production of L-valine based on transcriptome analysis and in silico gene knockout simulation. Proc Natl Acad Sci U S A **104**:7797–7802.

Patil, K. R., I. Rocha, J. Förster, and J. Nielsen. 2005. Evolutionary programming as a platform for in silico metabolic engineering. BMC Bioinformatics **6**:308.

Phalakornkule, C., S. Lee, T. Zhu, R. Koepsel, M. M. Ataai, I. E. Grossmann, and M. M. Domach. 2001. A MILP-based flux alternative generation and NMR experimental design strategy for metabolic engineering. Metab Eng **3**:124–137.

Pharkya, P., A. P. Burgard, and C. D. Maranas. 2004. OptStrain: a computational framework for redesign of microbial production systems. Genome Res **14**:2367–2376.

Pramanik, J., and J. D. Keasling. 1998. Effect of Escherichia coli biomass composition on central metabolic fluxes predicted by a stoichiometric model. Biotechnol Bioeng **60**:230–238.

Price, N., J. Papin, and B. Palsson. 2002. Determination of Redundancy and Systems Properties of the Metabolic Network of *Helicobacter Pylori* Using Genome-Scale Extreme Pathway Analysis. Genetics Research **12**:760–769.

Price, N. D., J. A. Papin, C. H. Schilling, and B. O. Palsson. 2003. Genome-scale microbial in silico models: the constraints-based approach. Trends Biotechnol **21**:162–169.

Provan, J. S. 1994. Efficient enumeration of the vertices of polyhedra associated with network LP's. Math. Program. **63**:47–64.

Reed, J. L., I. Famili, I. Thiele, and B. O. Palsson. 2006a. Towards multidimensional genome annotation. Nature Reviews Genetics **7**:130–141.

Reed, J. L., and B. Ø. Palsson. 2003. Thirteen years of building constraint-based in silico models of Escherichia coli. J Bacteriol **185**:2692–2699.

Reed, J. L., T. R. Patel, K. H. Chen, A. R. Joyce, M. K. Applebee, C. D. Herring, O. T. Bui, E. M. Knight, S. S. Fong, and B. O. Palsson. 2006b. Systems approach to refining genome annotation. Proc Natl Acad Sci U S A **103**:17480–17484.

Rockafellar, T. R. 1970. Convex Analysis. Princeton Landmarks in Mathematics. Princeton University Press, New Jersey.

Sauro, H. M., and B. Ingalls. 2004. Conservation analysis in biochemical networks: computational issues for software writers. Biophys Chem **109**:1–15.

Savinell, J. M., and B. O. Palsson. 1992. Network analysis of intermediary metabolism using linear optimization. I. Development of mathematical formalism. J Theor Biol **154**:421–454.

Schomburg, I., A. Chang, and D. Schomburg. 2002. BRENDA, enzyme data and metabolic information. Nucleic Acids Res **30**:47–49.

Schuetz, R., L. Kuepfer, and U. Sauer. 2007. Systematic evaluation of objective functions for predicting intracellular fluxes in *Escherichia coli*. Mol Syst Biol **3**.

Schuetz, R., N. Zamboni, M. Heinemann, and U. Sauer. 2009. Biologically realized flux distributions are near-optimal with minimal adjustment to changes. Personal communication **(tba)**.

Schuster, S., and C. Hilgetag. 1994. On elementary flux modes in biochemical reaction systems at steady state. J. Biol. Syst. **2**:165–182.

Segré, D., D. Vitkup, and G. M. Church. 2002. Analysis of optimality in natural and perturbed metabolic networks. Proc Natl Acad Sci U S A **99**:15112–15117.

Shlomi, T., O. Berkman, and E. Ruppin. 2005. Regulatory on/off minimization of metabolic flux changes after genetic perturbations. Proc Natl Acad Sci U S A **102**:7695–7700.

Smallbone, K., E. Simeonidis, D. S. Broomhead, and D. B. Kell. 2007. Something from nothing: bridging the gap between constraint-based and kinetic modelling. FEBS J **274**:5576–5585.

Stelling, J., S. Klamt, K. Bettenbrock, S. Schuster, and E. Gilles. 2002. Metabolic network structure determines key aspects of functionality and regulation. Nature **420**:190–193.

Terzer, M., N. Maynard, M. Covert, and J. Stelling. 2009. Genome-scale metabolic networks. Wiley Interdisciplinary Reviews: Systems Biology and Medicine .

Terzer, M., and J. Stelling. 2006. Accelerating the Computation of Elementary Modes Using Pattern Trees. In: Bucher, P., and B. M. E. Moret, editors, WABI, volume 4175 of *Lecture Notes in Computer Science*. Springer, 333–343.

———. 2008. Large scale computation of elementary flux modes with bit pattern trees. Bioinformatics **24**:2229–2235.

———. 2009. Parallel Extreme Ray and Pathway Computation. In: (Wyrzykowski and (tba), 2009), accepted.

Thiele, I., T. D. Vo, N. D. Price, and B. Ø. Palsson. 2005. Expanded metabolic reconstruction of Helicobacter pylori (iIT341 GSM/GPR): an in silico genome-scale characterization of single- and double-deletion mutants. J Bacteriol **187**:5818–5830.

Tran, L. M., M. L. Rizk, and J. C. Liao. 2008. Ensemble modeling of metabolic networks. Biophys J **95**:5606–5617.

Urbanczik, R., and C. Wagner. 2005. Functional stoichiometric analysis of metabolic networks. Bioinformatics **21**:4176–4180.

Vallabhajosyula, R. R., V. Chickarmane, and H. M. Sauro. 2006. Conservation analysis of large biochemical networks. Bioinformatics **22**:346–353.

Varma, A., and B. O. Palsson. 1994. Stoichiometric flux balance models quantitatively predict growth and metabolic by-product secretion in wild-type Escherichia coli W3110. Appl Environ Microbiol **60**:3724–3731.

Wagner, C. 2004. Nullspace approach to determine the elementary modes of chemical reaction systems. J. Phys. Chem. B **108**:2425–2431.

Wagner, C., and R. Urbanczik. 2005. The Geometry of the Flux Cone of a Metabolic Network. Biophys. J. **89**:3837–3845.

Wiback, S. J., R. Mahadevan, and B. Ø. Palsson. 2003. Reconstructing metabolic flux vectors from extreme pathways: defining the alpha-spectrum. J Theor Biol **224**:313–324.

Wiechert, W. 2001. 13C Metabolic Flux Analysis. Metabolic Engineering **3**:195 – 206.

Wittmann, C. 2002. Metabolic flux analysis using mass spectrometry. Adv Biochem Eng Biotechnol **74**:39–64.

Wyrzykowski, R., and (tba), editors. 2009. 8th International Conference on Parallel Processing and Applied Mathematics, PPAM 2009, Wroclaw, Poland, September 13-16, 2009, Proceedings, Lecture Notes in Computer Science. Springer.

Young, J. D., K. L. Henne, J. A. Morgan, A. E. Konopka, and D. Ramkrishna. 2008. Integrating cybernetic modeling with pathway analysis provides a dynamic, systems-level description of metabolic control. Biotechnol Bioeng **100**:542–559.

Yugi, K., Y. Nakayama, A. Kinoshita, and M. Tomita. 2005. Hybrid dynamic/static method for large-scale simulation of metabolism. Theor Biol Med Model **2**:42.

Index

Curriculum Vitae

Marco Terzer

date of birth:	February 2, 1973
place of birth:	Münsterlingen, Switzerland
citizenship:	Ebnat-Kappel, Switzerland

Education

2005 – 2009	PhD student at Institute of Computational Science ETH Zürich, Switzerland advisor: Prof. Dr. Jörg Stelling academic title: Dr. sc.
2008 – ⋯	candidate in CFA Study and Examination Program CFA Institute academic title: passed level I exam
1993 – 2000	studies at ETH Zürich major: Computer Science minor: Robotics academic title: M. sc. (Computer Science), ETH
1986 – 1993	high school in Davos, Switzerland degree: Matura C (Swiss high school diploma)
1980 – 1986	primary school in Davos, Switzerland

International experience

Dec 2002 – Mar 2003	language & culture sabbatical Lima, Peru
Jul 1998 – Sep 1998	internship Heidelberger Druckmaschinen AG 2 months in Atlanta, USA 1 month in Heidelberg, Germany

Work experience

2004 – 2005	software architect & technical project manager Infonic AG, Thalwil, Switzerland
2002 – 2004	software engineer & technical project leader AdNovum AG, Zürich, Switzerland
2000 – 2002	lead developer & development team leader Infonic AG, Thalwil, Switzerland
1996 – 1999	software developer, part time (∼50%) Infonic AG, Thalwil, Switzerland

Publications

- Terzer, M., and J. Stelling. 2009.
  Parallel Extreme Ray and Pathway Computation.
  PPAM 2009 Conference, Wroclaw, Poland (accepted).
- Terzer, M., N. Maynard, M. Covert, and J. Stelling. 2009.
  Genome-scale metabolic networks.
  Wiley Interdisciplinary Reviews: Systems Biology and Medicine.
- Terzer, M., and J. Stelling. 2008.
  Large scale computation of elementary flux modes with bit pattern trees.
  Bioinformatics 24:2229–2235.
- Terzer, M., M. Jovanovic, A. Choutko, O. Nikolayeva, A. Korn,
  D. Brockhoff, F. Zürcher, M. Friedmann, R. Schütz, E. Zitzler,
  J. Stelling and S. Panke. 2007.
  Design of a biological half adder.
  IET Synthetic Biology 1:53–58.
- Terzer, M., and J. Stelling. 2006.
  Accelerating the Computation of Elementary Modes Using Pattern Trees.
  In: Bucher, P., and B. M. E. Moret, editors, WABI,
  volume 4175 of Lecture Notes in Computer Science. Springer, 333–343.