MSc Mathematics

Master thesis

Elementary Conversion Mode Computation

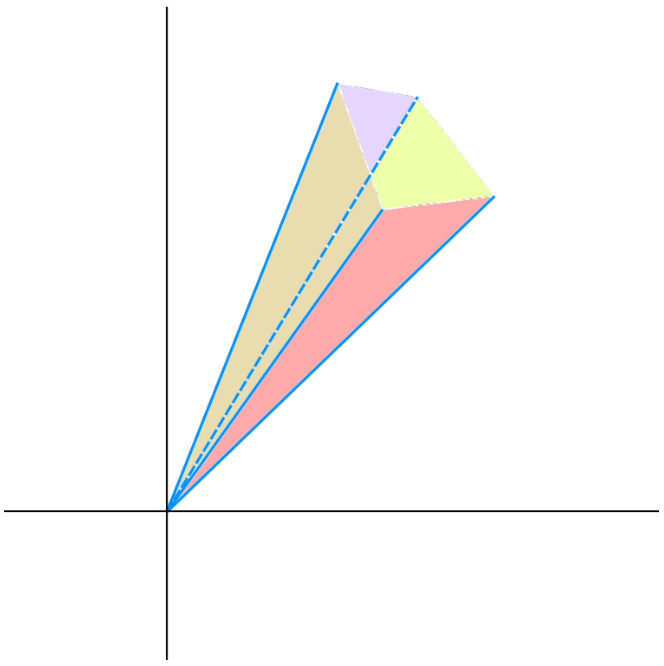
by

Erik Baalhuis

October 30, 2019

Supervisor: Dr. Bob Planqué

Second examiner: (placeholder)



Department of Mathematics
Faculty of Sciences

Abstract

The metabolic network of a cell consists of thousands of metabolites and thousands of reactions. An interesting way to gain insight from these models is the set of Elementary Flux Modes. They describe the possible modes of operation of the cell under a steady state assumption.

However, the computation of all Elementary Flux Modes is not feasible. Therefore a more limited perspective was suggested, where Elementary Conversion Modes are considered instead. These serve the same purpose of analyzing the metabolic capacities of the network, but ignore the inner workings of the cell in favor of only looking at external metabolites.

This thesis provides an algorithm to compute the full set of Elementary Conversion Modes for a given metabolic network. Specifically we use an adjacency test for extreme rays in a cone based on a linear program. Solving a large number of these linear programs is highly parallelizable.

Title: Elementary Conversion Mode Computation
Author: Erik Baalhuis, ebbaalhuis@gmail.com, 2545727
Supervisor: Dr. Bob Planqué
Second examiner: (placeholder)
Date: October 30, 2019

Department of Mathematics
VU University Amsterdam
de Boelelaan 1081, 1081 HV Amsterdam
`http://www.math.vu.nl/`

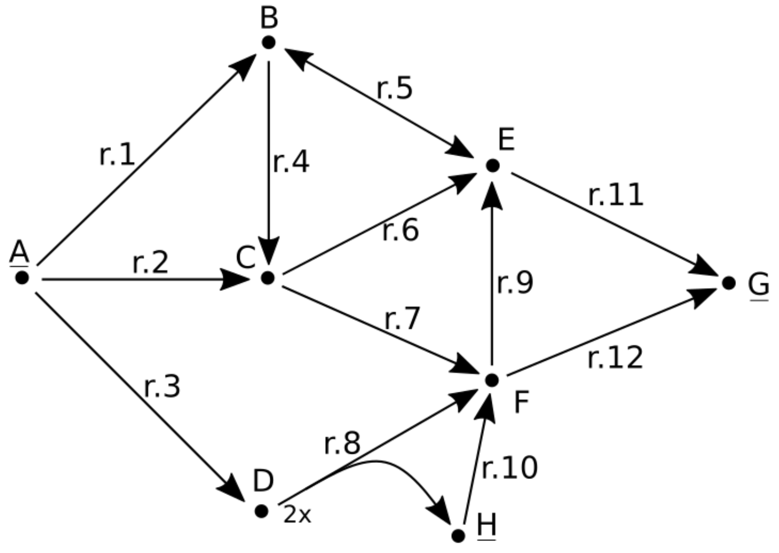
Contents

1 Introduction

In cell biology, the metabolic network consists of all the chemical reactions that allow the cell to function, grow, and duplicate itself. Large molecules like proteins and ribosome are not included, and neither is gene expression. Through the complete sequencing of genomes, it is now possible to reconstruct these networks entirely. Many mapped networks are available publicly[1].

The metabolites in the network (i.e. the molecules consumed or produced by metabolic reactions) can be classified as either internal or external [1]. Internal metabolites are those compounds that are physically restricted to remain inside the cell, while external metabolites can pass the cell membrane. We assume that the cell operates in a steady state: the concentration of internal metabolites does not change over time. External metabolites, on the other hand, can show net production or consumption.

Figure 1.1: Network `example`. The external metabolites (A, G and H) are underlined.



[1]`http://bigg.ucsd.edu/models`

1.1 Elementary Flux Modes

Mathematically, the metabolic network is described by a stoichiometry matrix N. Each row corresponds to a metabolite and each column to a reaction. Hence the dimensions of N are $m \times n$, where m is the number of metabolites and n the number of reactions. The entry N_{ij} is the molar amount of metabolite i that is produced (if N_{ij} is positive) or consumed (if N_{ij} is negative) in reaction j.

A priori the reactions that correspond to columns of N could take place in either direction. The reversibility of each reaction is extra information that the modeller should provide along with N. Note that having a reversible reaction r is equivalent to having two irreversible reactions r and $-r$. Hence by splitting each column in N that corresponds to a reversible reaction, we get an adjusted stoichiometry matrix $\overline{N}$ that has only irreversible reactions.

Table 1.1: Adjusted stoichiometry matrix of network `example`.

	r.1	**r.2**	**r.3**	**r.4**	**r.5**$^+$	**r.5**$^-$	**r.6**	**r.7**	**r.8**	**r.9**	**r.10**	**r.11**	**r.12**
$\underline{\mathbf{A}}$	-1	-1	-1	0	0	0	0	0	0	0	0	0	0
B	1	0	0	-1	-1	1	0	0	0	0	0	0	0
C	0	1	0	1	0	0	-1	-1	0	0	0	0	0
D	0	0	1	0	0	0	0	0	-2	0	0	0	0
E	0	0	0	0	1	-1	1	0	0	1	0	-1	0
F	0	0	0	0	0	0	0	1	1	-1	1	0	-1
$\underline{\mathbf{G}}$	0	0	0	0	0	0	0	0	0	0	0	1	1
$\underline{\mathbf{H}}$	0	0	0	0	0	0	0	0	1	0	-1	0	0

Note that it is possible for more than two metabolites to be involved in a single reaction, for example reaction 8 of `example` consumes metabolite D and produces F and $\underline{H}$ for a total of three metabolites involved.

The numbers in the stoichiometry matrix are relative molar amounts of each molecule. Though most numbers in the example are 1 or -1, note that other numbers can also occur; for example, reaction 8 takes two moles of metabolite D to make one mole of F and one mole of H.

A **flux mode $\boldsymbol{v}$** is a vector of n coefficients that each correspond to one reaction. By multiplying the stoichiometry matrix N with the flux mode $\boldsymbol{v}$ we get the rate of change of each metabolite under that flux mode:

$$\boldsymbol{x'} = N\boldsymbol{v}. \tag{1.1}$$

The steady state requirement is that the concentration of internal metabolites should not change over time. We assume that all flux modes satisfy this. So we require

$$\boldsymbol{x'}_{\text{int}} = (N\boldsymbol{v})_{\text{int}} = 0. \tag{1.2}$$

For example, $\boldsymbol{v}^* = (1,0,0,0,1,0,0,0,0,0,0,1,0)^T$ is a valid flux mode, since

$$\boldsymbol{x}'^* = N\boldsymbol{v}^* = (-1,0,0,0,0,0,1,0)^T \tag{1.3}$$

only has non-zero elements for the external metabolites $\underline{A}$ and $\underline{G}$.

After adjusting the stoichiometry matrix all the reactions are irreversible, so we also want

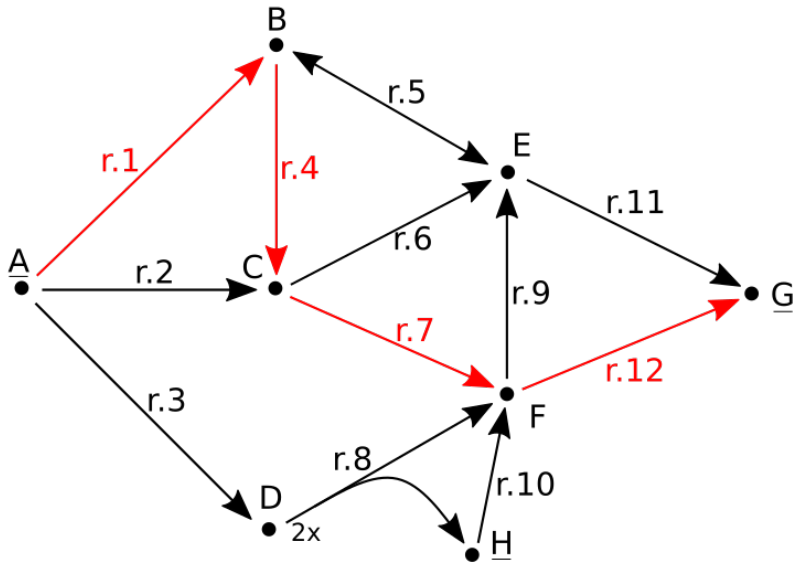
$$\boldsymbol{v} \geq \boldsymbol{0}. \tag{1.4}$$

We can then write the set of permissible flux modes as

$$C_{\text{flux}} = \{\boldsymbol{v} \in \mathbb{R}^n : (\overline{N}\boldsymbol{v})_{\text{int}} = \boldsymbol{0}, \boldsymbol{v} \geq \boldsymbol{0}\}. \tag{1.5}$$

We call a flux mode an **elementary flux mode (EFM)** if there is no other non-zero flux mode that uses a strict subset of the reactions used by the original. In this sense an EFM is a minimal functional unit [1].

Figure 1.2: Illustration of an EFM (id 7 in Table 1.2).



The set described in (1.5) is geometrically a convex cone. A detailed description of convex cones is available in Chapter 2. The EFMs are exactly the extreme rays of this cone [2]. For relatively small networks, the EFMs can be enumerated by `efmtool`[2] [3].

[2]`https://csb.ethz.ch/tools/software/efmtool.html`

The network `example` has 16 EFMs, as described in Table 1.2. Note that for some EFMs $\boldsymbol{x}' = N\boldsymbol{v} = \boldsymbol{0}$ (namely EFMs 10, 12, and 16). This means the reactions corresponding to those EFMs form a cycle that does not have any net production or consumption. This is not entirely possible thermodynamically, but it is nonetheless an important situation to consider because it might arise from modelling reductions or algorithmic steps.

Table 1.2: EFMs of network `example`.

id	**Flux mode**	**Reactions**	**Illustration**
1	$(0,0,0,0,0,0,0,0,0,1,1,1,0)$	9,10,11	$\underline{\text{H}} \to \text{F} \to \text{E} \to \underline{\text{G}}$
2	$(0,0,2,0,0,0,0,0,1,0,1,0,2)$	3,8,10,12	$2\underline{\text{A}} \to 2\text{D} \to \text{F}+\underline{\text{H}} \to 2\text{F} \to 2\underline{\text{G}}$
3	$(0,0,2,0,0,0,0,0,1,0,0,0,1)$	3,8,12	$2\underline{\text{A}} \to 2\text{D} \to \text{F}+\underline{\text{H}} \to \underline{\text{G}}+\underline{\text{H}}$
4	$(0,0,2,0,0,0,0,0,1,1,0,1,0)$	3,8,9,11	$2\underline{\text{A}} \to 2\text{D} \to \text{F}+\underline{\text{H}} \to \text{E}+\underline{\text{H}} \to \underline{\text{G}}+\underline{\text{H}}$
5	$(1,0,0,0,1,0,0,0,0,0,0,1,0)$	1,5^+,11	$\underline{\text{A}} \to \text{B} \to \text{E} \to \underline{\text{G}}$
6	$(0,1,0,0,0,0,1,0,0,0,0,1,0)$	2,6,11	$\underline{\text{A}} \to \text{C} \to \text{E} \to \underline{\text{G}}$
7	$(1,0,0,1,0,0,0,1,0,0,0,0,1)$	1,4,7,12	$\underline{\text{A}} \to \text{B} \to \text{C} \to \text{F} \to \underline{\text{G}}$
8	$(0,1,0,0,0,0,0,1,0,1,0,1,0)$	2,7,9,11	$\underline{\text{A}} \to \text{C} \to \text{F} \to \text{E} \to \underline{\text{G}}$
9	$(0,1,0,0,0,0,0,1,0,0,0,0,1)$	2,7,12	$\underline{\text{A}} \to \text{C} \to \text{F} \to \underline{\text{G}}$
10	$(0,0,0,1,0,1,0,1,0,1,0,0,0)$	4,5^-,7,9	$\text{B} \to \text{C} \to \text{F} \to \text{E} \to \text{B}$
11	$(1,0,0,1,0,0,0,1,0,1,0,1,0)$	1,4,7,9,11	$\underline{\text{A}} \to \text{B} \to \text{C} \to \text{F} \to \text{E} \to \underline{\text{G}}$
12	$(0,0,0,1,0,1,1,0,0,0,0,0,0)$	4,5^-,6	$\text{B} \to \text{C} \to \text{E} \to \text{B}$
13	$(1,0,0,1,0,0,1,0,0,0,0,1,0)$	1,4,6,11	$\underline{\text{A}} \to \text{B} \to \text{C} \to \text{E} \to \underline{\text{G}}$
14	$(0,0,2,0,0,0,0,0,1,2,1,2,0)$	3,8,9,10,11	$2\underline{\text{A}} \to 2\text{D} \to \text{F}+\underline{\text{H}} \to 2\text{F} \to 2\text{E} \to 2\underline{\text{G}}$
15	$(0,0,0,0,0,0,0,0,0,0,1,0,1)$	10,12	$\underline{\text{H}} \to \text{F} \to \underline{\text{G}}$
16	$(0,0,0,0,1,1,0,0,0,0,0,0,0)$	5^-,5^+	$\text{B} \to \text{E} \to \text{B}$

The most important property of EFMs is that any flux mode can be written as the sum of positive multiples of EFMs. Because any flux mode can only use a reaction a non-negative number of times, there is no cancellation in this sum. So if a flux mode does not use a particular reaction, then each contributing EFM will also not use that reaction. Unfortunately, the decomposition into EFMs is not unique: it is possible that a flux mode can be decomposed in different ways as the sum of positive multiples of EFMs.

Knowledge of the full set of EFMs of a metabolic network is useful to many applications. By analyzing how many different EFMs can be used to produce a certain compound, we can quantify cellular robustness [4, 5]. It is also a powerful tool for metabolic engineering [6, 7]. Melzer et al. [8] developed an EFM-based method, called FluxDesign, which identifies deletion, knockdown and over-expression targets. More applications can be found in [1].

However, EFM analysis relies on the ability to enumerate all EFMs for a given network. The problem is that the number of EFMs in a network suffers from combinatorial explosion. Already in relatively small models such as the core metabolism of *E. coli* (about 100 reactions) [9] there are around 272 million EFMs [10]. In genome-scale metabolic models there are commonly thousands of reactions, and the number of EFMs becomes so large we can never hope to enumerate it [2, 12].

1.2 Elementary Conversion Modes

Because the EFMs of life-sized networks are not computable, Urbanczik and Wagner [11] proposed in 2005 to instead consider *Elementary Conversion Modes* (ECMs).

The idea is to only look at the rate of change of (external) metabolites $\boldsymbol{x'}$, instead of the flux mode $\boldsymbol{v}$. This way we obtain the conversion cone (as opposed to the flux cone from Equation 1.5):

$$C_{\text{conv}} = \{\boldsymbol{x'} \in \mathbb{R}^m : \boldsymbol{x'} = \bar{N}\boldsymbol{v} \text{ for some } \boldsymbol{v} \geq \mathbf{0}, \boldsymbol{x'}_{\text{int}} = \mathbf{0}\} \tag{1.6}$$

A conversion mode is an ECM if it is an extreme ray of C_{conv} intersected with an orthant $\mathcal{O}$. The intersection is to make sure an important property carries over from EFMs: this way, any conversion mode is a sum of positive multiples of ECMs. We will avoid having to deal with orthants in the computation of ECMs by transforming the cone in such a way that it lies entirely inside one orthant (section 3.2). For an example of why orthant intersection is necessary, see subsection 1.2.2.

ECMs result from identifying those EFMs that have identical metabolite rate of change: that is, two EFMs are equivalent if they produce and consume the same molecules overall, without taking into account the reactions used inside the cell or intermediary products.

For ECM calculation, it is no longer necessary to keep track of exactly which reactions are used. In the end only the stoichiometry of external metabolites identifies an ECM.

As an example of the reduction in numbers obtained by considering ECMs instead of EFMs, the *E. coli* 'core' network that has around 272 million EFMs has only 689 ECMs (see Section 5.1).

Table 1.3: ECMs of network `example`.

id	**Conversion mode**	**Illustration**
1	$(-2, 0, 0, 0, 0, 0, 2, 0)$	$2\underline{\text{A}} \to 2\underline{\text{G}}$
2	$(-2, 0, 0, 0, 0, 0, 1, 1)$	$2\underline{\text{A}} \to \underline{\text{G}} + \underline{\text{H}}$
3	$(0, 0, 0, 0, 0, 0, 1, -1)$	$\underline{\text{H}} \to \underline{\text{G}}$

The `example` network has 3 ECMs, that can be seen in Table 1.3. There are a few interesting observations to make about them. Firstly, note that all the entries corresponding to *internal* metabolites are zero in all ECMs: in this case all except the first and last two numbers.

Secondly, if you look at the EFMs of the same network, in Table 1.2, you see that most of them correspond to an ECM. For example, the net effect of EFMs 1 and 15 is $\underline{\text{H}} \to \underline{\text{G}}$, so they correspond to ECM 3. Others correspond to a multiple of an ECM, for example EFM 14 corresponds to ECM 1 multiplied by two; this multiple is not important, because EFMs and ECMs are defined in the setting of cones (see also Chapter 2). The EFMs that are cyclic (10, 12, and 16) have no net metabolic effect at all, so they do not correspond to any ECM.

1.2.1 Applications of ECMs

The conversion cone is an interesting object in its own right, enabling the determination of minimal media (minimal sets of input metabolites that allow the cell to grow) [11]. Conversion analysis can also be linked up with Flux Balance Analysis [13]. For example, the conversion cone can be used to study the degeneracy of the optimization problems considered in Flux Balance Analysis [11].

If you are interested only in the flows through a few reactions, the network can be modified by adding a new virtual external metabolite as a tag for each reaction that should be tracked, and then calculating the ECMs of the modified network.

Another interesting option is combining the strengths of EFM analysis and ECM analysis while investigating a network. After selecting a suitably small subnetwork, ECMs can be used to calculate the conversion cone of the rest of the network. These can then be used to augment the subnetwork. Then any EFM of the augmented subnetwork corresponds to at least one EFM of the whole network. This allows us to obtain descriptions of the operation of the entire network, which are detailed with respect to reactions in the subnetwork (we know the flux through each reaction) but less detailed about the operation of the other reactions (we only know the ECMs of the rest of the network, so only the total input and output) [11].

1.2.2 Example of orthant intersection

The ECMs of a network are not only the extreme rays of its conversion cone, but also the extreme rays of the intersection of the conversion cone with each orthant. Each orthant is defined by a system of m inequalities:

$$\epsilon_1 x_1 \geq 0 \qquad \epsilon_2 x_2 \geq 0 \qquad \ldots \qquad \epsilon_m x_m \geq 0 \tag{1.7}$$

where each ϵ_i is either 1 or -1. In other words, within an orthant each metabolite is restricted to being only produced or only consumed.

As an example, consider the network in Figure 1.3. Its conversion cone is illustrated in Figure 1.4. The network has no internal metabolites, and only 2 reactions. Both of them are ECMs; they are the extreme rays of the conversion cone. But there is one additional ECM: it is obtained by intersecting the conversion cone with the orthant $(A \geq 0; B \geq 0)$ or with the orthant $(A \geq 0; B \leq 0)$. The 'extra' ECM just produces B, and it can be seen as a combination of the two reactions of the network. This is its own ECM because it does not use metabolite A in any way. Defining ECMs in this manner ensures that all conversions can be written as a *non-cancelling* combination of ECMs.

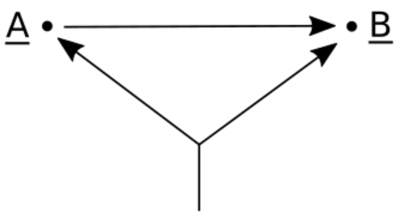


Figure 1.3: Network.

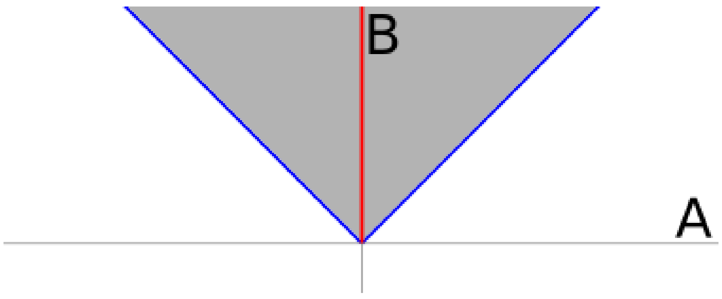


Figure 1.4: Conversion cone (grey) with extreme rays (blue) and additional ECM (red).

How do we obtain these ECMs algorithmically? One way would be to first calculate the conversion cone, and then find its intersection with each orthant. However, if there are m' external metabolites then this means intersecting with $2^{m'}$ orthants. Since the number of external metabolites can be over 100 in genome-scale networks, this is not feasible.

Instead, we split each external metabolite that can be both produced and consumed before calculating the conversion cone (see also section 3.2). After this modification to the network, the entire conversion cone lies inside a single orthant, since each external metabolite can either be only produced or only consumed. This splitting is illustrated in Figure 1.5. Note that A becomes an internal metabolite. The entire conversion cone now lies in the orthant $(A_{in} \leq 0; A_{out} \geq 0; B \geq 0)$.

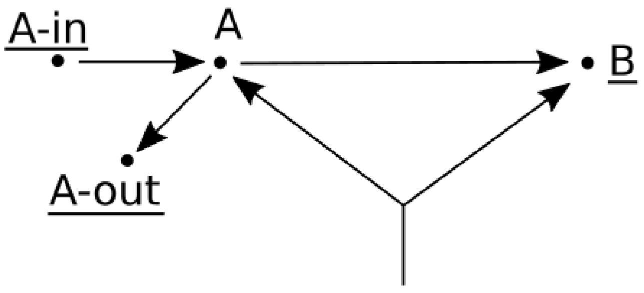


Figure 1.5: Network after splitting A.

2 Mathematical background

First some notation: a vector $\boldsymbol{v} \in \mathbb{R}^d$ is in bold to differentiate it from a scalar. A bold $\mathbf{0}$ means a zero vector, i.e. a vector of all zeroes. For a matrix A, its i-th row is indicated by $\boldsymbol{a_{i,*}}$ and the j-th column by $\boldsymbol{a_{*,j}}$.

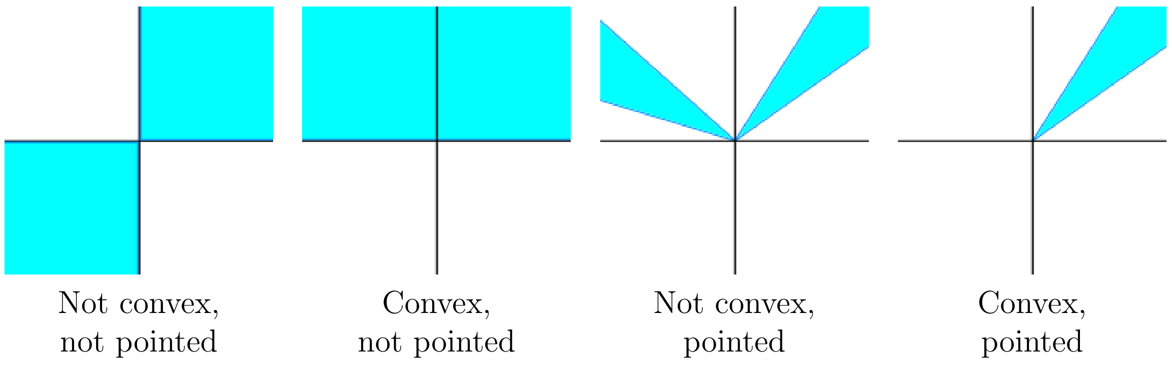
2.1 Cones

A set $C \subset \mathbb{R}^n$ is a **cone** if for each $\boldsymbol{x} \in C$ and positive $\alpha \in \mathbb{R}_{\geq 0}$, the product $\alpha\boldsymbol{x}$ is in C. So a cone consists of half-lines starting in the origin. Topologically a cone can be open or closed; in this thesis all cones will be closed (that means they contain their boundary).

A cone C is called a **convex cone** if $\alpha\boldsymbol{x} + \beta\boldsymbol{y}$ belongs to C for any positive $\alpha, \beta \in \mathbb{R}_{\geq 0}$ and $\boldsymbol{x}, \boldsymbol{y} \in C$.

A convex cone is **pointed** if it does not contain any line. In other words, a convex cone C is pointed if $C \cap -C = \{\mathbf{0}\}$. So for any point $\boldsymbol{x} \in C$ except the origin, $-\boldsymbol{x}$ should not be in C.

Figure 2.1: Different types of cones in $\mathbb{R}^2$.



2.1.1 Extreme rays

A vector $\boldsymbol{r}$ is called a **ray** of the cone C if $\boldsymbol{r} \neq \mathbf{0}$ and $\alpha\boldsymbol{r} \in C$ for all positive α. Two rays $\boldsymbol{r}$ and $\boldsymbol{r'}$ are identified if $\boldsymbol{r} = \alpha\boldsymbol{r'}$ for some positive α. This equivalence is denoted by $\boldsymbol{r} \sim \boldsymbol{r'}$.

A **conic combination** K of vectors $\{\boldsymbol{r_1}, \boldsymbol{r_2}, \ldots, \boldsymbol{r_k}\}$ is a linear combination with positive coefficients:

$$K = \{\lambda_1 \boldsymbol{r_1} + \lambda_2 \boldsymbol{r_2} + \cdots + \lambda_k \boldsymbol{r_k} : \lambda_i \geq 0, \boldsymbol{r_i} \in \mathbb{R}^n\}. \tag{2.1}$$

An **extreme ray** is a ray that cannot be written as a conic combination of other rays.

Proposition 2.1. *Any ray* $\boldsymbol{r}$ *of* C *is a conic combination of extreme rays of* C.

For a proof, see Proposition 5 in [15].

2.1.2 Cone duality

For any set $A \in \mathbb{R}^n$, its **dual cone** is defined as

$$A^* = \{\boldsymbol{x} \in \mathbb{R}^n : \langle \boldsymbol{a}, \boldsymbol{x} \rangle \geq 0 \text{ for all } \boldsymbol{a} \in A\},$$

where $\langle \boldsymbol{.}, \boldsymbol{.} \rangle$ is the standard dot product. The dual cone of any set is always a convex cone.

Proposition If C is a convex cone, then $C^{**} = C$.

2.1.3 Two representations

If there exists a matrix A such that $C = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \geq \boldsymbol{0}\}$, then C is an **H-cone** or **polyhedral cone**. Note that $A\boldsymbol{x} \geq \boldsymbol{0}$ means that $\boldsymbol{a_{i,*}x} \geq 0$ for every row $\boldsymbol{a_{i,*}}$ of A; so C is the intersection of halfspaces, hence the term H-cone. The matrix A is called a **halfspace representation** of C.

A cone C is a **V-cone** if it is the conic combination of finitely many vectors. The conic combination of vectors $\{\boldsymbol{r_1}, \boldsymbol{r_2}, \ldots, \boldsymbol{r_k}\}$ can be written in matrix form, where R is formed by using the vectors as columns:

$$C = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = R\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq \boldsymbol{0}\}.$$

The matrix R is called a **ray representation** of C.

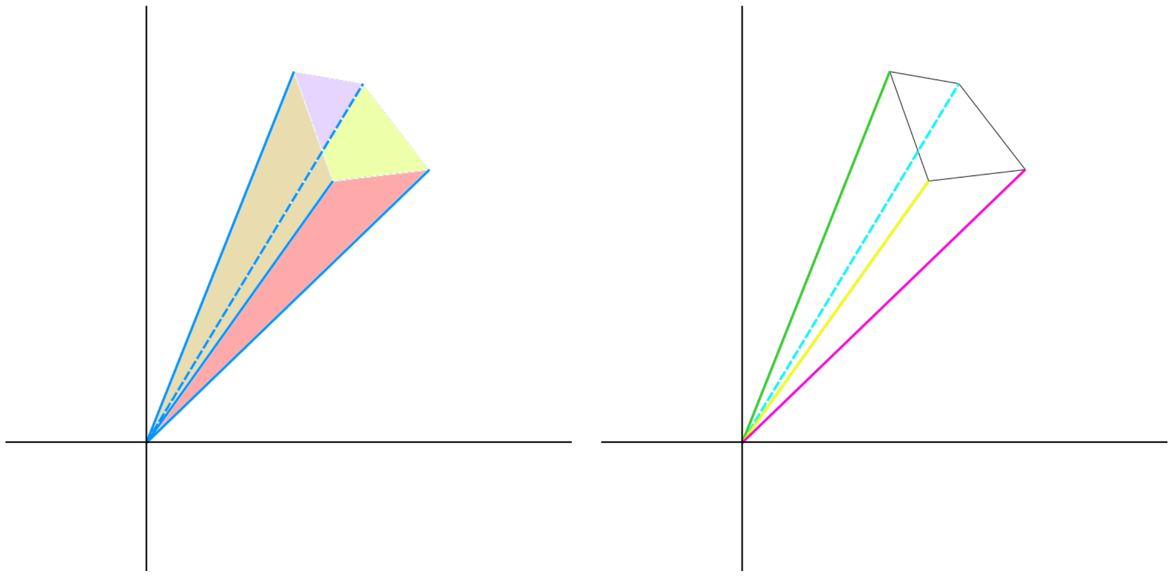
Proposition The set of extreme rays of a pointed cone C are a generating set for that cone, and any generating set includes the extreme rays.

Corollary The set of extreme rays is the unique minimal generating set.

Theorem *(Minkowski-Weyl Theorem for cones)*
A set $P \in \mathbb{R}^n$ is an H-cone if and only if it is a V-cone.

Proposition If a matrix A is a half-space representation of the cone C, then the rows of A form a generating set for the dual cone C^*.

Figure 2.2: Two representations of the same cone in $\mathbb{R}^3$.



2.1.4 Ray adjacency

Given an $m \times n$ matrix A with polyhedral cone

$$C = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \geq \boldsymbol{0}\}$$

and corresponding ray representation R:

$$C = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = R\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq \boldsymbol{0}\}.$$

For each extreme ray $\boldsymbol{r_{*,j}}$ of the cone (i.e. the j-th column of R), define the **zero set**

$$Z(\boldsymbol{r_{*,j}}) = \{i : \boldsymbol{a_{i,*}} \boldsymbol{r_{*,j}} = 0\},$$

where $\boldsymbol{a_{i,*}}$ is the i-th row of A.

Two distinct extreme rays $\boldsymbol{r_{*,a}}$ and $\boldsymbol{r_{*,b}}$ are **adjacent** if

$$Z(\boldsymbol{r_{*,a}}) \cap Z(\boldsymbol{r_{*,b}}) \subset Z(\boldsymbol{r_{*,c}}) \implies \boldsymbol{r_{*,c}} \sim \boldsymbol{r_{*,a}} \text{ or } \boldsymbol{r_{*,c}} \sim \boldsymbol{r_{*,b}}$$

In other words, there are no rays other than $\boldsymbol{r_{*,a}}$ and $\boldsymbol{r_{*,b}}$ that satisfy all the constraints from A with equality, that are satisfied with equality by both $\boldsymbol{r_{*,a}}$ and $\boldsymbol{r_{*,b}}$.

Geometrically, two distinct extreme rays are adjacent if the minimal face of C containing both contains no other extreme rays. For more details, see Proposition 7 in [15]. For example, in Figure 2.2 the green and blue extreme rays are adjacent; the minimal face containing both is the plane indicated in purple in the left half of the figure. The green and pink rays are not adjacent, since the minimal face containing both is the entire cone (which also contains other extreme rays).

Here we find the reason why we need to work with a pointed cone. Suppose the cone is not pointed, then there is a $\boldsymbol{r} \in P(A)$ with $-\boldsymbol{r} \in P(A)$ as well. Hence $A\boldsymbol{r} \geq \mathbf{0}$ and $A(-\boldsymbol{r}) \geq \mathbf{0}$, so $A\boldsymbol{r} = \mathbf{0}$. This means $\boldsymbol{r}$ is perpendicular to all rows of A, so the zero set of $\boldsymbol{r}$ is the largest possible zero set, consisting of all row indices: $Z(\boldsymbol{r}) = \{1, 2, \ldots, m\}$. Because $Z(\boldsymbol{r}_{*,a}) \cap Z(\boldsymbol{r}_{*,b}) \subset Z(\boldsymbol{r})$ for any two extreme rays $\boldsymbol{r}_{*,a}$ and $\boldsymbol{r}_{*,b}$, all pairs are adjacent (except possibly pairs containing $\boldsymbol{r}$ or $-\boldsymbol{r}$ itself).

2.2 Double Description method

The Double Description method is an algorithm originally suggested by Motzkin et al. [14] to translate between the two descriptions of a cone. For a more comprehensive description and proofs of the propositions, see [15].

A pair (A, R) is called a **double description pair** or DD pair if A is a halfspace representation and R is a ray representation of the same cone. That is,

$$\{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \geq \mathbf{0}\} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = R\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq \mathbf{0}\}.$$

The following proposition shows that an algorithm that can translate in one direction automatically also solves the inverse direction.

Proposition (A, R) is a DD pair if and only if (R^T, A^T) is a DD pair.

The double description method provides an algorithm to find an R based on a given A such that (A, R) is a DD pair. In the proposition above, R^T is a halfspace representation for the dual cone of $P(A)$. If we want to find a DD pair starting from a given R, we can take R^T and treat it as the halfspace representation of a cone, then apply the double description method to find A^T (hence giving (A, R).

The core of the double description algorithm is an incremental procedure. The input is an $m \times n$ matrix A, a halfspace representation of the cone $P(A) = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \geq \mathbf{0}\}$. Let $K \subset \{1, \ldots, m\}$ be a subset of the row indices of A and A_K the submatrix consisting of the corresponding rows. Suppose we already have a ray representation for the cone $P(A_K)$, i.e. we have a DD pair (A_K, R_K). To perform the incremental step, select any index i not in K; we will add that row to A_K to form $A_{K\cup\{i\}}$ and construct a DD pair $(A_{K\cup\{i\}}, R_{K\cup\{i\}})$ with that using R_K.

First, we partition the column index set J of R_K into three parts:
$J^+ = \{j \in J : \boldsymbol{a}_{i,*}\boldsymbol{r}_{*,j} > 0\}$,
$J^0 = \{j \in J : \boldsymbol{a}_{i,*}\boldsymbol{r}_{*,j} = 0\}$,
$J^- = \{j \in J : \boldsymbol{a}_{i,*}\boldsymbol{r}_{*,j} < 0\}$.

To make the matrix $R_{K\cup\{i\}}$ we keep all the columns $\boldsymbol{r_{*,j}}$ from R_K with $j \in J^+$ or $j \in J^0$. These rays satisfy $\boldsymbol{a_{i,*}r_{*,j}} \geq 0$ and so they are in $P(A_{K\cup\{i\}})$ unchanged. The columns corresponding to J^- are not in the new cone, but they give rise to new generating rays in the following way.

Let $\boldsymbol{r_{jj',*}} = (\boldsymbol{a_{i,*}r_{*,j}})\boldsymbol{r_{*,j'}} - (\boldsymbol{a_{i,*}r_{*,j'}})\boldsymbol{r_{*,j}}$ for each $(j, j') \in J^+ \times J^-$. Note that $\boldsymbol{a_{i,*}r_{jj'}} = 0$ for each choice of (j, j'). These columns are also added to R'.

To sum up, R' is the $d \times |J'|$ matrix with columns $\boldsymbol{r_{*,j}}$ $(j \in J')$ such that
$J' = J^+ \cup J^0 \cup (J^+ \times J^-)$, and
$\boldsymbol{r_{jj'}} = (\boldsymbol{a_{i,*}r_{*,j}})\boldsymbol{r_{*,j'}} - (\boldsymbol{a_{i,*}r_{*,j'}})\boldsymbol{r_{*,j}}$.

Proposition (A_{K+i}, R') is a DD pair.

Repeating this step, eventually all the rows of A will be included, giving a DD pair (A, R) as intended. What is left is to find is a starting point. A good option is starting with d linearly independent rows of A to form $A_{k,*}$; in this case $A_{k,*}$ is invertible, and $A_{k,*}x = \boldsymbol{\lambda} \geq \boldsymbol{0}$ implies that $A^{-1}_{k,*}\boldsymbol{x} = \boldsymbol{\lambda}$, so $(A_{k,*}, A^{-1}_{k,*})$ is a DD pair.

2.2.1 Redundant rays

Although the Double Description method as described above gives a ray representation R, it is not necessarily a *minimal* representation. That is to say, there can be redundant columns in R that are not extreme rays of the cone. In fact, the number of unnecessary rays in practice increases very fast and goes beyond any tractable limit [15].

One possible solution is to discard all redundant rays, preferably between each step of the Double Description method. This can be done, for example, with the program `redund` from *lrslib* [16], which looks for redundant rays by trying to find matching conic combinations of other rays. However, in our experience this became computationally infeasible when dealing with over roughly 10,000 rays.

Alternatively, for each $\boldsymbol{r_{jj'}}$ that we created with $(j, j') \in J^+ \times J^-$ we can test the originating rays for adjacency. It turns out the new ray $\boldsymbol{r_{jj'}}$ is redundant if and only if its parent rays were not adjacent (lemma 8 in [15]).

3 Direct intersection approach

3.1 Overview

In this section we give an overview of our algorithm to compute the Elementary Conversion Modes (ECMs) of a given network. We will start with the cone consisting of all conic combinations of reactions from the network in section 3.2.

The procedure we use to refine the cone step by step only works if the cone is pointed to begin with. So before we proceed, some modifications might be required to make sure the cone is pointed. These are in section 3.3. It is then possible to use `redund` [16] to remove redundant rays.

At this point the matrix R_0 likely still has many non-zero rows that correspond to internal metabolites. In the final ECMs, all coefficients in these rows must be zero. We will 'eliminate' these internal metabolites one by one, each time updating R_i to R_{i+1}.

For each internal metabolite j, we essentially intersect the previous cone with the constraint $x'_j = 0$. The procedure is similar to the iterative step in the Double Description method described in section 2.2, with the difference that we always intersect with an equality instead of an inequality. This means we only keep those rays that have a zero at position j, and add a ray for each adjacent $(+, -)$ pair. We use a specialized adjacency test described in section 3.4 and section 3.5.

Each of the steps above will result in a new R_i, that has a row of zeroes corresponding to the internal metabolite that was just eliminated. The columns of R_i are the extreme rays of a new cone. After doing this for each internal metabolite, the resulting cone satisfies the steady state requirement and its extreme rays (the columns of R_{final} are the ECMs.

3.2 Input

The input is a stoichiometric matrix R of size $m \times n$, where each row represents a metabolite and each column a reaction. Hence the network has m metabolites and n reactions.

Each of the metabolites is marked as either *internal* or *external.* Internal metabolites are required to be in steady state in the ECMs. The output will be an $m \times k$ matrix

R_{final} in which every column is an ECM. The rows of this output matrix still correspond to metabolites, and those rows whose metabolites are internal should be all zeroes; this is the steady state requirement.

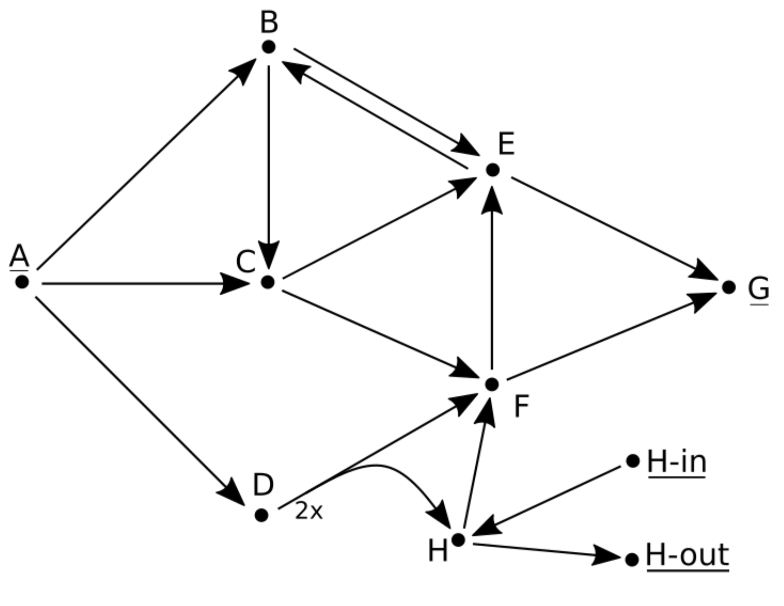
Furthermore, for each external metabolite it is indicated in the input whether it is an *input* (can only be consumed), an *output* (can only be produced) or *bidirectional* (can be both consumed and produced). In order to avoid intersection with orthants as required by the definition of ECMs, we want to remove any bidirectional external metabolites. This way, the entire conversion cone will lie in a single orthant, thus making orthant intersections unnecessary. We achieve this by making two virtual external metabolites m_{in} and m_{out} for each bidirectional metabolite m. Then reactions are added that turn m_{in} into m and m into m_{out}. Finally m is marked internal, and the two new external metabolites m_{in} and m_{out} are input and output, respectively (so no bidirectional metabolites are left). See subsection 3.2.1 for an example.

Each reaction is marked as *reversible* or *irreversible*. It is important to know which reactions are reversible, since we normally only allow each reaction to be used a positive amount. A reversible reaction r will be split into two irreversible reactions r and $-r$.

3.2.1 Example

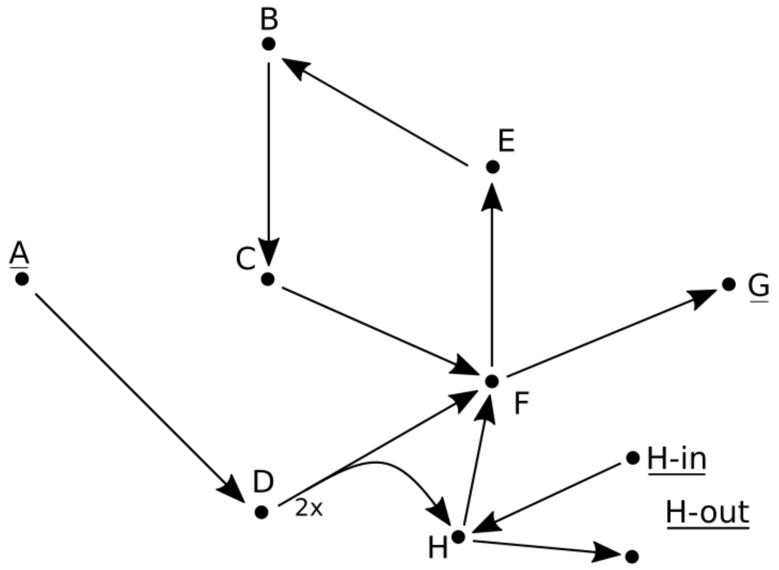
As an example we will use the network from Figure 1.1. Its stoichiometry matrix is in Table 1.1, with the reversible reaction already split into two columns. There is one bidirectional external metabolite: H. We make H internal by adding H_{in} and H_{out} as illustrated in Figure 3.1.

Figure 3.1: Metabolite H is split.



The columns of R form a set of generating rays for the cone we are working with. We can now use `redund` [16] to obtain a *minimal* set of generating rays. This is done by discarding those rays that can be written as a conic combination of other rays. The result can be seen in Figure 3.2.

Figure 3.2: Remaining rays (reactions) after discarding redundant ones.



3.3 Making the cone pointed

When we start out with the matrix R obtained from the stoichiometry of the network, there is no guarantee whether the cone generated by the columns of R is pointed or not. Making the cone pointed has several advantages: pointed cones have a unique set of extreme rays, and the adjacency test in the next section only works well for pointed cones.

Proposition 3.1. *The cone generated by the columns of R is pointed if and only if no non-zero conic combination of columns of R sums to zero.*

Proof of 3.1
If the cone C is not pointed, then by definition there are $\boldsymbol{r}, -\boldsymbol{r} \in C$. These $\boldsymbol{r}$ and $-\boldsymbol{r}$ can be written as a conic combination of columns of R, since C is spanned by those columns. Adding these two conic combinations gives a combination that sums to zero.

On the other hand, suppose there is a non-zero conic combination of columns of R that sums to zero: $\sum_{i=1}^{n} \alpha_i \boldsymbol{r}_{*,i} = 0$ and $\alpha_i \geq 0$ for all i. There must be at least one α

that is strictly positive, say α_x. Consider the conic combination without $\boldsymbol{r_{*,x}}$:

$$\sum_{i=1,\ldots,n;i\neq x} \alpha_i \boldsymbol{r_{*,i}} = -\alpha_x \boldsymbol{r_{*,x}}. \tag{3.1}$$

Hence both $\alpha_x \boldsymbol{r_{*,x}}$ and $-\alpha_x \boldsymbol{r_{*,x}}$ are in the cone, showing that it is not pointed. □

We will call such a non-zero conic combination of columns of R that sums to zero a *cycle*. The plan is to find and eliminate the cycles one by one, leaving a pointed cone after they are all removed.

Note that the pointed cone we end up with depends on which metabolites get removed in subsection 3.3.2. However, for the end result it does not matter which internal metabolites were removed as part of a cycle and which were removed through the equality intersection that is done after the cone is made pointed.

3.3.1 Finding a cycle

Given a matrix R, we will use a linear program to find a cycle in the cone spanned by the columns of R. We will try to find a $\boldsymbol{\lambda} \geq \mathbf{0}$ that satisfies $R\boldsymbol{\lambda} = \mathbf{0}$ and $\boldsymbol{\lambda} \neq \mathbf{0}$; if such a $\boldsymbol{\lambda}$ does not exist, there are no cycles.

The cycle finding LP is as follows:

$$\text{maximize} \sum_i \lambda_i$$

$$\text{subject to}$$

$$R\boldsymbol{\lambda} = \mathbf{0},$$

$$\boldsymbol{\lambda} \geq 0,$$

$$\lambda_i \leq 1 \text{ for all } i.$$

The LP is always feasible, since $\boldsymbol{\lambda} = \mathbf{0}$ is a solution with objective value 0. If there are no cycles, then this is the only feasible solution, and the cycle removal is done.

If there are cycles, then the optimal solution $\boldsymbol{\lambda}^*$ will have at least one position equal to 1, say $\lambda^*_x = 1$. This is because any $\boldsymbol{\lambda}$ that induces a cycle satisfies $R\boldsymbol{\lambda} = 0$, so multiples of λ will satisfy that as well. Hence the constraint $\lambda_i \leq 1$ (for all i) is the only thing keeping λ (and with that the optimal value) bounded. The corresponding $\boldsymbol{r_{*,x}}$ is a ray that is involved in a cycle. This gives us sufficient information to remove (part of) the cycle.

3.3.2 Eliminating part of a cycle

Using the method of the previous section, we find a ray $\boldsymbol{r_{*,x}}$ that is involved in a cycle.

Fix any metabolite m_i that has a non-zero value in $\boldsymbol{r_{*,x}}$. Because we split each external metabolite into being only input or only output in Section 3.2, it is impossible for external metabolites to have non-zero coefficients in any ray that is part of a cycle, as there can be no circular flow through such external metabolites. So we can be assured that m_i is an internal metabolite. Our strategy is to eliminate this metabolite along with $\boldsymbol{r_{*,x}}$, hence reducing the total number of metabolites involved in cycles.

Because $\boldsymbol{r_{*,x}}$ is part of a cycle, we know that there is some conic combination of rays that sums to $-\boldsymbol{r_{*,x}}$. So the network is capable of performing both $\boldsymbol{r_{*,x}}$ and $-\boldsymbol{r_{*,x}}$. One of these will produce m_i and the other will consume it. This means we can use a (positive or negative) multiple of $\boldsymbol{r_{*,x}}$ to cancel out m_i in all rays. In case the multiple is negative, we could have used the conic combination of rays that sums to $-\boldsymbol{r_{*,x}}$ instead (in positive multiple); so this cancellation can be done with only conic combinations, and therefore does not change the metabolic capabilities of the cone.

Whenever a ray has a non-zero coefficient $r_{i,y}$ corresponding to m_i, we change that ray by adding a multiple of $\boldsymbol{r_{*,x}}$ so that $r_{i,y}$ becomes exactly 0:

$$\boldsymbol{r_{*,y}} \mapsto \boldsymbol{r_{*,y}} - r_{i,y}\boldsymbol{r_{*,x}}. \tag{3.2}$$

Note that the new network still has the exact same metabolic capacities in steady state; because m_i is internal, its net production had to be zero in all ECMs already.

The cycle removal procedure now goes back to find a new cycle (Section 3.3.1) and repeats until there are no cycles left. Since each elimination step removes an internal metabolite, this will terminate after finitely many steps.

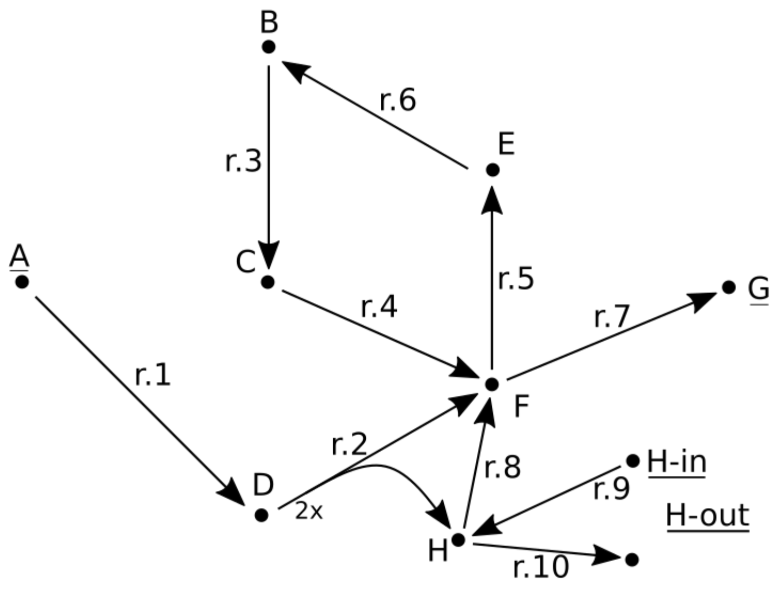
3.3.3 Example

We continue with the example network as we left it in Figure 3.2. From the picture it is clear that there is one cycle: (B $\to$ C $\to$ F $\to$ E $\to$ B). Let us see how the cycle detection procedure from subsection 3.3.1 would find it.
In this case the cycle finding LP will find the optimal solution $\boldsymbol{\lambda}^* = (0, 0, 1, 1, 1, 1, 0, 0, 0, 0)$. So we learn that rays 3, 4, 5 and 6 are involved in a cycle.

For the elimination part, a selection is made of one of these rays (say ray 3), and then a metabolite is chosen that has a non-zero coefficient in that ray (say metabolite B). The only other ray that uses B is ray 6. Hence the column under r.6 is transformed as follows:

$$\boldsymbol{r_{*,6}} \mapsto \boldsymbol{r_{*,6}} - r_{2,6}\boldsymbol{r_{*,3}}. \tag{3.3}$$

Figure 3.3: Remaining rays (reactions) after discarding redundant ones.



Here $r_{2,6}$ is the coefficient of metabolite B in reaction 3, so $r_{2,6} = -1$. So in the end

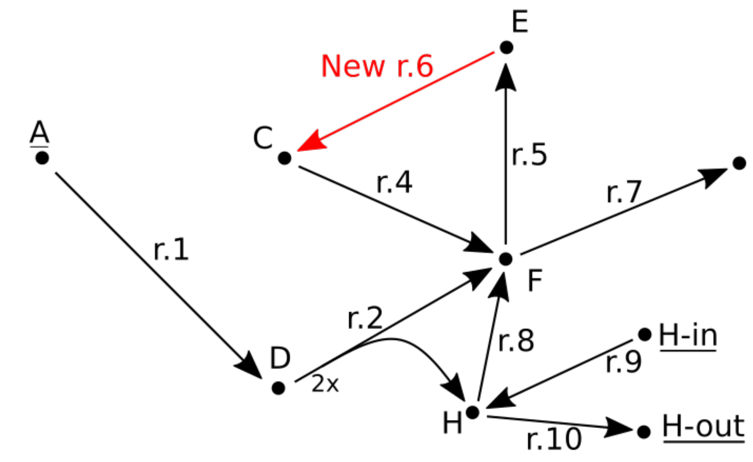
$$\boldsymbol{r_{*,6}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \mapsto \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + 1 \cdot \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{3.4}$$

Essentially ray 3 was added to ray 6 to create a new ray that makes C out of E. Ray 3 is then removed. The resulting network can be seen in Figure 3.4. After three more steps, the network will be cycle-free (Figure 3.5).

Table 3.1: Matrix R corresponding to the example in Figure 3.3.

	r.1	**r.2**	**r.3**	**r.4**	**r.5**	**r.6**	**r.7**	**r.8**	**r.9**	**r.10**
A	-1	0	0	0	0	0	0	0	0	0
B	0	0	-1	0	0	1	0	0	0	0
C	0	0	1	-1	0	0	0	0	0	0
D	1	-2	0	0	0	0	0	0	0	0
E	0	0	0	0	1	-1	0	0	0	0
F	0	1	0	1	-1	0	-1	1	0	0
G	0	0	0	0	0	0	1	0	0	0
H	0	1	0	0	0	0	0	-1	1	-1
$\mathbf{H}_{in}$	0	0	0	0	0	0	0	0	-1	0
$\mathbf{H}_{out}$	0	0	0	0	0	0	0	0	0	1

Figure 3.4: Network after one cycle removing step.



3.4 Geometric adjacency test

Given is an $m \times n$ matrix A with polyhedral cone

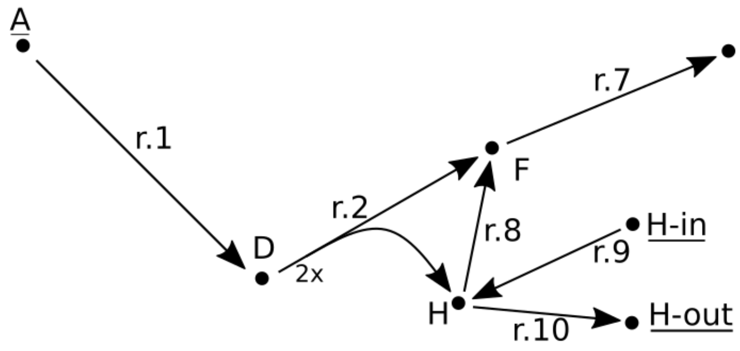
$$C = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \geq \mathbf{0}\}$$

and corresponding ray representation R:

$$C = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = R\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq \mathbf{0}\}.$$

Recall the idea of ray adjacency from subsection 2.1.4. Note that the definition of adjacency relies on the halfspace representation A. We would like to find an alternative way to determine the adjacency of two rays, using only the ray representation R. This is essential because our algorithm never computes A; doing so would require using the Double Description method, which turns out to be too inefficient in terms of computation time as well as memory. This is because a cone that has few extreme rays can

Figure 3.5: Network after all cycles are removed.



still require many halfspaces for the halfspace representation. To be exact, the number of halfspaces can grow exponentially with respect to the number of extreme rays (McMullen's upper bound theorem, see for example 2.9 in [18]).

The next subsection provides this alternative adjacency test. The idea is to test whether two rays are adjacent by considering a point in between them. If this point can also be formed by a conic combination that includes other rays than the original two, then they are not adjacent.

3.4.1 The Linear Program

Let

$$\boldsymbol{x} = \frac{1}{4}\boldsymbol{r_{*,a}} + \frac{3}{4}\boldsymbol{r_{*,b}}$$

and consider the linear programming problem LP($\boldsymbol{r_{*,a}}, \boldsymbol{r_{*,b}}$):

$$\text{maximize} \sum_i \lambda_i - \lambda_a - \lambda_b,$$

$$\text{subject to}$$

$$R\boldsymbol{\lambda} = \boldsymbol{x},$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}.$$

Note that the LP is always feasible, and has optimal value at least 0, since $\bar{\boldsymbol{\lambda}}$ with $\bar{\lambda}_a = 1/4$, $\bar{\lambda}_b = 3/4$ and zero in the other coordinates is feasible with objective value zero. The reason for choosing 1/4 and 3/4 will become clear in the proof of Theorem 3.2.

Theorem 3.1 (geometric adjacency test)**.** *The following are equivalent for extreme rays* $\boldsymbol{r_{*,a}}, \boldsymbol{r_{*,b}}$*:*
(1) $\boldsymbol{r_{*,a}}$ *and* $\boldsymbol{r_{*,b}}$ *are adjacent.*
(2) $Z(\boldsymbol{r_{*,a}}) \cap Z(\boldsymbol{r_{*,b}}) \subset Z(\boldsymbol{r_{*,c}}) \implies \boldsymbol{r_{*,c}} \sim \boldsymbol{r_{*,a}}$ *or* $\boldsymbol{r_{*,c}} \sim \boldsymbol{r_{*,b}}$.
(3) LP($\boldsymbol{r_{*,a}}, \boldsymbol{r_{*,b}}$*) has optimal value 0.*

Proof of 3.1
The equivalence of (1) and (2) is just the definition of adjacency.

(2 $\implies$ 3) Suppose (2) holds, so $Z(\boldsymbol{r_{*,a}}) \cap Z(\boldsymbol{r_{*,b}}) \subset Z(\boldsymbol{r_{*,c}}) \implies \boldsymbol{r_{*,c}} \sim \boldsymbol{r_{*,a}}$ or $\boldsymbol{r_{*,c}} \sim \boldsymbol{r_{*,b}}$.

Consider any extreme ray $\boldsymbol{r_{*,c}}$ that is not equivalent to $\boldsymbol{r_{*,a}}$ or $\boldsymbol{r_{*,b}}$. We know that $Z(\boldsymbol{r_{*,c}})$ does not contain $Z(\boldsymbol{r_{*,a}}) \cap Z(\boldsymbol{r_{*,b}})$, so there is some index $i \in Z(\boldsymbol{r_{*,a}}) \cap Z(\boldsymbol{r_{*,b}})$ that is not in $Z(\boldsymbol{r_{*,c}})$. This means $\boldsymbol{a_{i,*}}\boldsymbol{r_{*,c}} > 0$.

Suppose $\boldsymbol{\lambda}$ is a feasible solution for LP$(\boldsymbol{r_{*,a}}, \boldsymbol{r_{*,b}})$, so $R\boldsymbol{\lambda} = \boldsymbol{x}$. Now consider

$$\boldsymbol{a_{i,*}}R\boldsymbol{\lambda} = \sum_j \boldsymbol{a_{i,*}}\boldsymbol{r_{*,j}}\lambda_j \geq \boldsymbol{a_{i,*}}\boldsymbol{r_{*,c}}\lambda_c. \tag{3.5}$$

The last inequality holds because each ray $\boldsymbol{r_{*,j}}$ is in the cone; hence $\boldsymbol{a_{i,*}}\boldsymbol{r_{*,j}} \geq 0$, and also $\lambda_j \geq 0$ (one of the LP constraints). At the same time

$$\boldsymbol{a_{i,*}}R\boldsymbol{\lambda} = \boldsymbol{a_{i,*}}\boldsymbol{x} = \boldsymbol{a_{i,*}}(\frac{1}{4}\boldsymbol{r_{*,a}} + \frac{3}{4}\boldsymbol{r_{*,b}}) = 0, \tag{3.6}$$

since $i \in Z(\boldsymbol{r_{*,a}})$ and $i \in Z(\boldsymbol{r_{*,b}})$. Combining (3.5) and (3.6) with $\boldsymbol{a_{i,*}}\boldsymbol{r_{*,c}} > 0$ gives $\lambda_c = 0$. Because $\boldsymbol{r_{*,c}}$ was any extreme ray not equivalent to $\boldsymbol{r_{*,a}}$ or $\boldsymbol{r_{*,b}}$, this means $\sum_i \lambda_{i,*} - \lambda_a - \lambda_b = 0$, hence (3) holds.

(3 $\implies$ 2) For a contrapositive proof, assume (2) does not hold. Then there is some ray $\boldsymbol{r_{*,c}}$ such that $Z(\boldsymbol{r_{*,a}}) \cap Z(\boldsymbol{r_{*,b}}) \subset Z(\boldsymbol{r_{*,c}})$.

By definition $\boldsymbol{x} = 1/4\boldsymbol{r_{*,a}} + 3/4\boldsymbol{r_{*,b}}$. Therefore, for any row $\boldsymbol{a_{i,*}}$,

$$\boldsymbol{a_{i,*}}\boldsymbol{x} = \boldsymbol{a_{i,*}}(\frac{1}{4}\boldsymbol{r_{*,a}} + \frac{3}{4}\boldsymbol{r_{*,b}}) = \frac{1}{4}\boldsymbol{a_{i,*}}\boldsymbol{r_{*,a}} + \frac{3}{4}\boldsymbol{a_{i,*}}\boldsymbol{r_{*,b}}. \tag{3.7}$$

Because $\boldsymbol{r_{*,a}}$ and $\boldsymbol{r_{*,b}}$ are in the cone, $\boldsymbol{a_{i,*}}\boldsymbol{r_{*,a}} \geq 0$ and $\boldsymbol{a_{i,*}}\boldsymbol{r_{*,b}} \geq 0$. Thus $\boldsymbol{a_{i,*}}x = 0$ if and only if $\boldsymbol{a_{i,*}}\boldsymbol{r_{*,a}} = \boldsymbol{a_{i,*}}\boldsymbol{r_{*,b}} = 0$, hence

$$Z(\boldsymbol{x}) = Z(\boldsymbol{r_{*,a}}) \cap Z(\boldsymbol{r_{*,b}}). \tag{3.8}$$

Consider the line segment

$$L(t) = t\boldsymbol{x} + (1-t)\boldsymbol{r_{*,c}} \tag{3.9}$$

for $t \in [0, 1]$. This is the line segment from $\boldsymbol{r_{*,c}}$ to $\boldsymbol{x}$. See also Figure 3.6.

Take any index k not in $Z(\boldsymbol{x})$. Since $\boldsymbol{a_{k,*}}\boldsymbol{x} > 0$ and $\boldsymbol{a_{k,*}}\boldsymbol{r_{*,c}} \geq 0$, we have, for any $t \in (0, 1]$,

$$\boldsymbol{a_{k,*}}L(t) = \boldsymbol{a_{k,*}}t\boldsymbol{x} + \boldsymbol{a_{k,*}}(1-t)\boldsymbol{r_{*,c}} > 0. \tag{3.10}$$

Because $\boldsymbol{a_{k,*}}L(t)$ is continuous with respect to t, there is an $\varepsilon_k > 0$ such that $\boldsymbol{a_{k,*}}L(1+\varepsilon_k) > 0$. Define ε as the minimum of all the ε_k, then

$$\boldsymbol{a_{k,*}}L(1+\varepsilon) > 0 \text{ for all } k \text{ not in } Z(\boldsymbol{x}) \tag{3.11}$$

On the other hand, any index $m \in Z(\boldsymbol{x})$ is also in $Z(\boldsymbol{r_{*,c}})$ (this follows from Equation 3.8). We have

$$\boldsymbol{a_{m,*}}L(1+\varepsilon) = \boldsymbol{a_{m,*}}(1+\varepsilon)\boldsymbol{x} - \boldsymbol{a_{m,*}}\varepsilon\boldsymbol{r_{*,c}} = 0 \text{ for all } m \in Z(\boldsymbol{x}). \tag{3.12}$$

Denote $\boldsymbol{\alpha} = L(1+\varepsilon)$; it follows that $\boldsymbol{\alpha} \in P(A)$ and $Z(\boldsymbol{\alpha}) = Z(\boldsymbol{x})$ from Equation 3.11 and Equation 3.12.

We now have

$$\boldsymbol{\alpha} = L(1+\varepsilon) = (1+\varepsilon)\boldsymbol{x} - \varepsilon\boldsymbol{r_{*,c}}, \tag{3.13}$$

therefore

$$\boldsymbol{\alpha} + \varepsilon\boldsymbol{r_{*,c}} = (1+\varepsilon)\boldsymbol{x}, \tag{3.14}$$

and so

$$\frac{\boldsymbol{\alpha}}{1+\varepsilon} + \frac{\varepsilon\boldsymbol{r_{*,c}}}{1+\varepsilon} = \boldsymbol{x}. \tag{3.15}$$

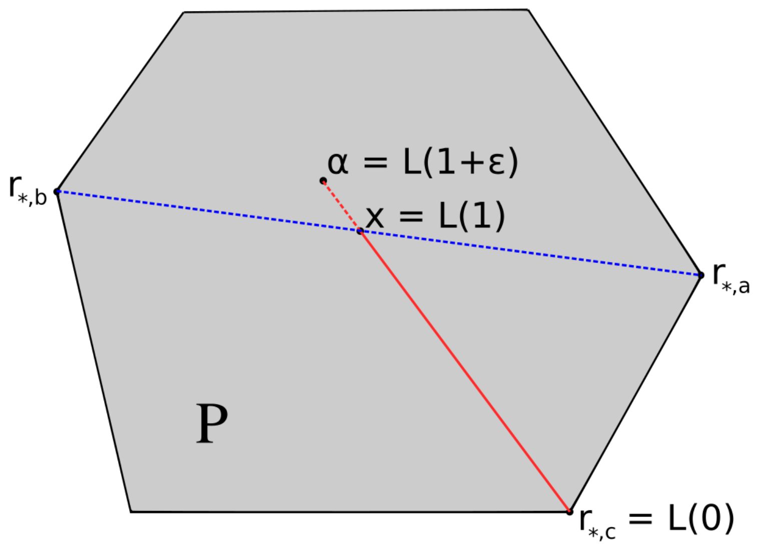


Figure 3.6: Illustration for the proof of ($3 \implies 2$). The point $\boldsymbol{\alpha}$ will always be inside P, indicating a feasible solution to the LP with objective strictly higher than 0. The grey area might not look like a cone at first, but consider it as a cross section of one.

Because $\boldsymbol{\alpha}$ is a point in $P(A)$, it can be written as a conic combination of rays. This shows that the left-hand side of Equation 3.15 gives a $\boldsymbol{\lambda} \geq \mathbf{0}$ such that $R\boldsymbol{\lambda} = \boldsymbol{x}$ and $\lambda_c > 0$, giving a feasible solution to LP($\boldsymbol{r}_{*,a}, \boldsymbol{r}_{*,b}$) with objective value strictly higher than 0. $\square$

This shows that solving the right LP can serve as a test for the adjacency of extreme rays.

Moreover, the following theorem guarantees that the optimal value of LP($\boldsymbol{r}_{*,a}, \boldsymbol{r}_{*,b}$) is bounded away from zero if the rays are not adjacent. This is important for a robust numerical way to distinguish between the adjacent and non-adjacent case.

The assumption in the theorem that the columns of R are normalized is easily satisfied in the algorithm; after scaling a column still represents the same ray. The assumption that LP($\boldsymbol{r}_{*,a}, \boldsymbol{r}_{*,b}$) is bounded is satisfied because the cone is pointed after applying the procedure in section 3.3.

Theorem 3.2 (large optimal value)**.** *Assume that each column of R has L^1-norm 1, and denote the L^1-norm by $||\cdot||$. Assume LP($\boldsymbol{r}_{*,a}, \boldsymbol{r}_{*,b}$) is bounded. If $\boldsymbol{r}_{*,a}$ and $\boldsymbol{r}_{*,b}$ are not adjacent, then LP($\boldsymbol{r}_{*,a}, \boldsymbol{r}_{*,b}$) has optimal value at least*

$$\min(||\boldsymbol{x} - (\boldsymbol{x} \cdot \boldsymbol{r}_{*,a})\boldsymbol{r}_{*,a}||, ||\boldsymbol{x} - (\boldsymbol{x} \cdot \boldsymbol{r}_{*,b})\boldsymbol{r}_{*,b}||).$$

Proof of 3.2
Recall that $\boldsymbol{x} = 1/4\boldsymbol{r}_{*,a} + 3/4\boldsymbol{r}_{*,b}$. The reverse triangle inequality states that

$$||y - z|| \geq \Big| ||y|| - ||z|| \Big|. \tag{3.16}$$

Taking $y = 1/4\boldsymbol{r}_{*,a}$ and $z = -3/4\boldsymbol{r}_{*,b}$, and using that by assumption $||\boldsymbol{r}_{*,a}|| = ||\boldsymbol{r}_{*,b}|| = 1$, this gives

$$||\boldsymbol{x}|| = ||1/4\boldsymbol{r}_{*,a} + 3/4\boldsymbol{r}_{*,b}|| \geq \Big| ||1/4\boldsymbol{r}_{*,a}|| - ||3/4\boldsymbol{r}_{*,b}|| \Big| = \Big| -1/2 \Big| = 1/2, \tag{3.17}$$

so

$$||\boldsymbol{x}|| \geq 1/2. \tag{3.18}$$

Let $\boldsymbol{\mu}$ be the optimal solution to LP($\boldsymbol{r}_{*,a}, \boldsymbol{r}_{*,b}$). Write $\boldsymbol{\mu'}$ to indicate $\boldsymbol{\mu}$ with positions a and b removed, and R' to indicate R with columns $\boldsymbol{r}_{*,a}$ and $\boldsymbol{r}_{*,b}$ removed. So we have

$$\mu_a\boldsymbol{r}_{*,a} + \mu_b\boldsymbol{r}_{*,b} + R'\boldsymbol{\mu'} = \boldsymbol{x}. \tag{3.19}$$

From the LP constraints we also know

$$\mu_i \geq 0 \text{ for all } i. \tag{3.20}$$

The objective was to maximize $(\sum_i \mu_i) - \mu_a - \mu_b = \sum_i \mu'_i$. We assumed that $\boldsymbol{r}_{*,a}$ and $\boldsymbol{r}_{*,b}$ are not adjacent, so according to Theorem 3.1 the objective value of $\boldsymbol{\mu}$ is strictly greater than zero. Hence

$$\mu'_i > 0 \text{ for at least one } i. \tag{3.21}$$

Suppose that both $\mu_a > 0$ and $\mu_b > 0$. We will show that this contradicts the optimality of $\boldsymbol{\mu}$. Beside $\boldsymbol{\mu}$ we know one more feasible solution: the vector $\boldsymbol{\nu}$ with all zeroes except that $\nu_a = 1/4$ and $\nu_b = 3/4$. Since both $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are feasible solutions, we get

$$R(\boldsymbol{\mu} - \boldsymbol{\nu}) = 0. \tag{3.22}$$

Now consider $\bar{\boldsymbol{\mu}} = \boldsymbol{\mu} + \delta(\boldsymbol{\mu} - \boldsymbol{\nu})$ for some small $\delta > 0$. This $\bar{\boldsymbol{\mu}}$ satisfies

$$R\bar{\boldsymbol{\mu}} = R(\boldsymbol{\mu} + \delta(\boldsymbol{\mu} - \boldsymbol{\nu})) = R\boldsymbol{\mu} + \delta R(\boldsymbol{\mu} - \boldsymbol{\nu}) = R\boldsymbol{\mu} + 0 = \boldsymbol{x}. \tag{3.23}$$

The only positions in $(\boldsymbol{\mu} - \boldsymbol{\nu})$ that could be negative are a and b, since $\boldsymbol{\nu}$ is zero everywhere else. But $\mu_a > 0$ and $\mu_b > 0$, so if we pick δ small enough then $\bar{\mu}_a \geq 0$ and $\bar{\mu}_b \geq 0$. Hence

$$\bar{\boldsymbol{\mu}} \geq \mathbf{0}. \tag{3.24}$$

Together (3.23) and (3.24) show that $\bar{\boldsymbol{\mu}}$ is a feasible solution. Additionally, it has a better objective value than $\boldsymbol{\mu}$, since (3.21) implies that adding $\delta(\boldsymbol{\mu} - \boldsymbol{\nu})$ increases the objective value. This contradicts the optimality of $\boldsymbol{\mu}$, so it must be that $\mu_a = 0$ or $\mu_b = 0$. Assume that $\mu_a = 0$ without loss of generality (even though a and b play different roles in $\boldsymbol{x} = 1/4\boldsymbol{r_{*,a}} + 3/4\boldsymbol{r_{*,b}}$, swapping the 1/4 and 3/4 does not make a difference for this proof).

We now want to find a lower bound for the distance between $\boldsymbol{r_{*,b}}$ and $\boldsymbol{x}$. We have assumed that $\mu_a = 0$ but we do not know anything about μ_b except that $\mu_b \geq 0$. To find the minimal distance between $\boldsymbol{r_{*,b}}$ and $\boldsymbol{x}$ for any possible $\mu_b \geq 0$, consider the orthogonal projection of $\boldsymbol{x}$ onto the line spanned by $\boldsymbol{r_{*,b}}$:

$$\boldsymbol{x}_{||\boldsymbol{r_{*,b}}} = (\boldsymbol{x} \cdot \boldsymbol{r_{*,b}})\frac{\boldsymbol{r_{*,b}}}{||\boldsymbol{r_{*,b}}||} = (\boldsymbol{x} \cdot \boldsymbol{r_{*,b}})\boldsymbol{r_{*,b}}, \tag{3.25}$$

since $||\boldsymbol{r_{*,b}}|| = 1$. This gives a lower bound for the distance between $\boldsymbol{x}$ and $\mu_b\boldsymbol{r_{*,b}}$:

$$||\boldsymbol{x} - \mu_b\boldsymbol{r_{*,b}}|| \geq ||\boldsymbol{x}_{\perp\boldsymbol{r_{*,b}}}|| = ||\boldsymbol{x} - \boldsymbol{x}_{||\boldsymbol{r_{*,b}}}|| = ||\boldsymbol{x} - (\boldsymbol{x} \cdot \boldsymbol{r_{*,b}})\boldsymbol{r_{*,b}}||. \tag{3.26}$$

Since we know that

$$\mu_b\boldsymbol{r_{*,b}} + R'\boldsymbol{\mu'} = \boldsymbol{x}, \tag{3.27}$$

we have

$$R'\boldsymbol{\mu'} = \boldsymbol{x} - \mu_b\boldsymbol{r_{*,b}}. \tag{3.28}$$

By assumption every column of R' has L^1-norm 1, so applying the triangle inequality gives

$$||R'\boldsymbol{\mu'}|| = ||\sum_i \mu_i \boldsymbol{R'_{*,i}}|| \leq \sum_i |\mu_i|.||\boldsymbol{R'_{*,i}}|| = \sum_i |\mu_i| = ||\boldsymbol{\mu'}||. \tag{3.29}$$

Let opt be the optimal value for LP$(\boldsymbol{r_{*,a}}, \boldsymbol{r_{*,b}})$. Then combining (3.29) with (3.28) and (3.26) shows that

$$\text{opt} = ||\boldsymbol{\mu'}|| \geq ||\boldsymbol{x} - (\boldsymbol{x} \cdot \boldsymbol{r_{*,b}})\boldsymbol{r_{*,b}}||. \tag{3.30}$$

Finally, considering the case where $\mu_b = 0$ instead of $\mu_a = 0$, instead of the previous bound we get

$$\text{opt} \geq \min(||\boldsymbol{x} - (\boldsymbol{x} \cdot \boldsymbol{r}_{*,a})\boldsymbol{r}_{*,a}||, ||\boldsymbol{x} - (\boldsymbol{x} \cdot \boldsymbol{r}_{*,b})\boldsymbol{r}_{*,b}||). \tag{3.31}$$

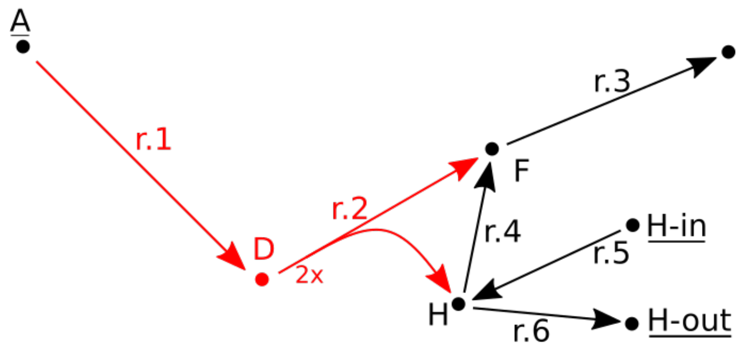
$\square$

3.4.2 Example

We return to the example network as we left it in Figure 3.5. The next step is removing the internal metabolites one by one. When a certain internal metabolite is selected, each ray that produces that metabolite (i.e. has a positive entry in the row corresponding to that metabolite) forms a pair with each ray that consumes that metabolite. For each pair we then run the geometric adjacency test.

As a heuristic, the internal metabolite that results in the least number of pairs is removed first. In case of the example, this is metabolite D; it has only one producing ray and one consuming ray, so one pair. It is easy to read the number of pairs from the row in R corresponding to a metabolite, since it is just the number of positive entries multiplied by the number of negative entries. Hence this heuristic selection is inexpensive.

Figure 3.7: Network at the start of internal metabolite elimination.



Only 1 LP is required, since there is only one pair of rays. We have LP($\boldsymbol{r}_{*,1}, \boldsymbol{r}_{*,2}$):

$$\text{maximize} \sum_i \lambda_i - \lambda_1 - \lambda_2,$$

subject to

$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 0 & 0 & 0 & 0 \\ 0 & 1 & -3 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{\lambda} = \begin{bmatrix} -1/2 \\ -1/2 \\ 1/2 \\ 0 \\ 1/2 \\ 0 \\ 0 \end{bmatrix},$$

Table 3.2: Matrix R corresponding to the example in Figure 3.7.

	r.1	**r.2**	**r.3**	**r.4**	**r.5**	**r.6**
A	-1	0	0	0	0	0
D	**1**	**-2**	0	0	0	0
F	0	1	-3	1	0	0
G	0	0	1	0	0	0
H	0	1	0	-1	1	-1
$\mathbf{H}_{in}$	0	0	0	0	-1	0
$\mathbf{H}_{out}$	0	0	0	0	0	1

$$\boldsymbol{\lambda} \geq \mathbf{0}.$$

The LP has optimal value 0. This can be seen, for example, by noting that the first two rows enforce $\lambda_1 = \lambda_2 = 1/2$, after which all other elements of $\boldsymbol{\lambda}$ have to be 0. Hence the first and second ray are adjacent, so they will give rise to a new extreme ray in the next iteration of R.

The exact combination of **r.1** and **r.2** that will be added can be seen from the numbers 1 and -2 in the row corresponding to D. As we are eliminating metabolite D, we want to create a 0 in this row. The only way to do that (up to scaling) is by adding the new ray

$$\mathbf{r.(1{,}2)} = 2 \cdot \mathbf{r.1} + 1 \cdot \mathbf{r.2} = (-2, 0, 1, 0, 1, 0, 0)^T. \tag{3.32}$$

The result is illustrated in Figure 3.8.

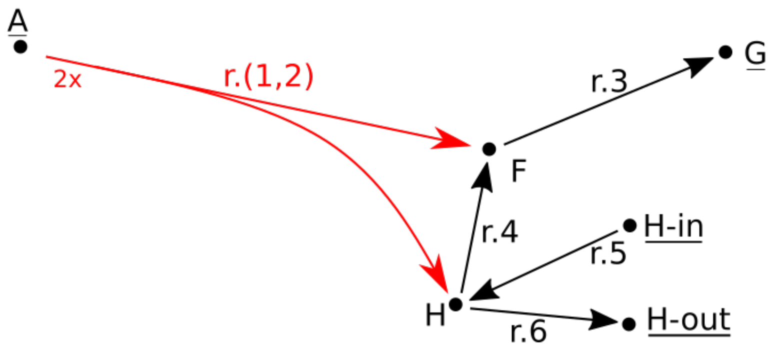
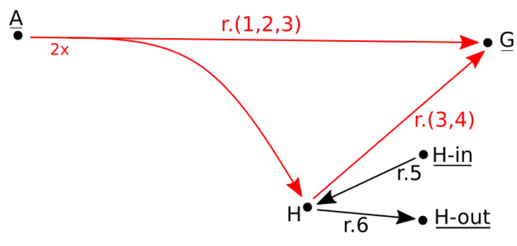
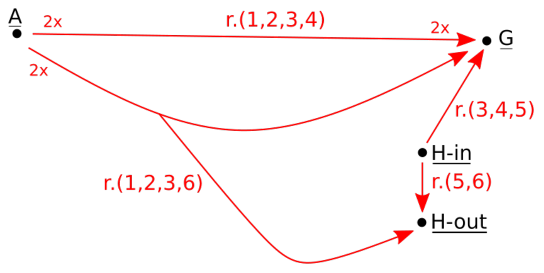


Figure 3.8: After eliminating metabolite D, there are four old rays and one new one.

Finally, four rays are left after eliminating all the internal metabolites. Three of them correspond exactly to the three ECMs of the network, and one futile ray turns H_{in} into H_{out}. The last step is recombining H_{in} and H_{out} back into one metabolite H. The futile ray then becomes all zeroes and is removed, and we are left with the 3 ECMs (one produces H, one consumes H).



Eliminating F.



Eliminating H.

Figure 3.9: Progression of rays as internal metabolites are eliminated.

3.5 Solving the LP

To perform adjacency tests, we will have to solve many Linear Programming problems of the kind introduced in the previous section:

$$\text{maximize} \sum_i \lambda_i - \lambda_a - \lambda_b,$$

subject to

$$R\boldsymbol{\lambda} = \boldsymbol{x},$$

$$\boldsymbol{\lambda} \geq \mathbf{0},$$

where

$$\boldsymbol{x} = 1/4\boldsymbol{r_{*,a}} + 3/4\boldsymbol{r_{*,b}}.$$

Let $m \times n$ be the dimensions of R. It is safe to assume $n \geq m$; if this is not the case, there are more linear constraints than variables, hence some of them will be redundant. If we do encounter an R with more rows than columns, we can discard linearly dependent rows from R.

A **basis** for the LP is any set B of m indices from $\{1, \ldots n\}$ such that the corresponding columns of R form a basis of the column space of R. In other words, a basis is a B such that the $m \times m$ submatrix R_B of R is non-singular.

We say that a feasible solution $\boldsymbol{\lambda}$ is a **basic feasible solution (BFS)** with basis B if all the non-zero elements of $\boldsymbol{\lambda}$ are indexed by B. Note that any basis B has at most one corresponding BFS: since R_B is nonsingular, $R_B\boldsymbol{\lambda} = \boldsymbol{x}$ has a unique solution $\boldsymbol{\lambda_B}$. This $\boldsymbol{\lambda_B}$ is a BFS if and only if it satisfies $\boldsymbol{\lambda_B} \geq \mathbf{0}$.

The converse is not true: one BFS can have multiple bases. This occurs when there is a BFS $\boldsymbol{\lambda}$ that has fewer than m non-zero elements. This is called a **degenerate** solution.

The LP at the start of this section has one obvious degenerate solution. Let $\boldsymbol{\lambda_{ab}}$ be the vector of length n with all zeroes, except that $\lambda_a = 1/4$ and $\lambda_b = 3/4$. By construction $R\boldsymbol{\lambda_{ab}} = \boldsymbol{x}$, and we also have $\boldsymbol{\lambda_{ab}} \geq \boldsymbol{0}$. So $\boldsymbol{\lambda_{ab}}$ is a feasible solution with only two non-zero elements. This means $\boldsymbol{\lambda_{ab}}$ is a degenerate solution as long as $m > 2$.

Degenerate points can cause *cycling*, a situation where iterative solution methods such as the (revised) simplex method visit the exact same point more than once. When this happens, the method will visit the same point ad nauseam and will never terminate. We overcome this by perturbation, see Section 3.5.4.

3.5.1 Revised simplex method

The revised simplex method uses the idea of a basis for a linear program to find an optimal solution. Essentially the method is based on the Karush-Kuhn-Tucker (KKT) conditions, which are a necessary and sufficient condition for optimality. More information is available in the book *Numerical Optimization* [20], specifically chapter 12 and 13.

Suppose we have an LP problem in standard form:

$$\text{minimize } \boldsymbol{c}^T\boldsymbol{x}$$

$$\text{subject to}$$

$$A\boldsymbol{x} = \boldsymbol{b},$$

$$\boldsymbol{x} \geq \boldsymbol{0}.$$

Let A be size $m \times n$ and have rank m. Then the KKT conditions are:

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{3.33}$$

$$A^T\boldsymbol{\pi} + \boldsymbol{s} = \boldsymbol{c}, \tag{3.34}$$

$$\boldsymbol{x} \geq 0, \tag{3.35}$$

$$\boldsymbol{s} \geq 0, \tag{3.36}$$

$$\boldsymbol{s}^T\boldsymbol{x} = 0. \tag{3.37}$$

Here $\boldsymbol{\pi}$ is the Lagrange multiplier associated with the constraint $A\boldsymbol{x} = \boldsymbol{b}$, and $\boldsymbol{s}$ is the Lagrange multiplier associated with $\boldsymbol{x} \geq \boldsymbol{0}$.

In the revised simplex method, we start in a vertex $\boldsymbol{x}$ of the feasible polytope. To check whether the KKT conditions hold, we find a basis B (of size m) such that A_B is nonsingular and all the non-zero positions in $\boldsymbol{x}$ are in B. Note that this is easy when $\boldsymbol{x}$

is a non-degenerate vertex, since in that case there will be exactly m non-zero elements in $\boldsymbol{x}$. However, if $\boldsymbol{x}$ is degenerate, then there are fewer non-zero elements and we have to find some way to complete B. The way to do this is described in subsection 3.5.2. For now, assume we have a basis B of size m corresponding to $\boldsymbol{x}$.

Denote by N the indices not in B, and split $\boldsymbol{x}$, $\boldsymbol{c}$ and $\boldsymbol{s}$ according to B:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_B} \\ \boldsymbol{x_N} \end{bmatrix} = \begin{bmatrix} A_B^{-1}\boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix}, \qquad \boldsymbol{c} = \begin{bmatrix} \boldsymbol{c_B} \\ \boldsymbol{c_N} \end{bmatrix}, \qquad \boldsymbol{s} = \begin{bmatrix} \boldsymbol{s_B} \\ \boldsymbol{s_N} \end{bmatrix}.$$

Additionally, split the second KKT constraint,

$$A^T\boldsymbol{\pi} + \boldsymbol{s} = \boldsymbol{c},$$

also into two parts according to B:

$$A_B^T\boldsymbol{\pi} + \boldsymbol{s_B} = \boldsymbol{c_B}, \tag{3.38}$$

$$A_N^T\boldsymbol{\pi} + \boldsymbol{s_N} = \boldsymbol{c_N}. \tag{3.39}$$

In order to satisfy the last KKT condition, let $\boldsymbol{s_B} = \boldsymbol{0}$. Then from (3.38) we can deduce that

$$\boldsymbol{\pi} = (A_B^T)^{-1}\boldsymbol{c_B}. \tag{3.40}$$

Since A_B and $\boldsymbol{c_B}$ are known, this allows us to calculate $\boldsymbol{\pi}$. Next, from (3.39) we get

$$\boldsymbol{s_N} = \boldsymbol{c_N} - A_N^T\boldsymbol{\pi}. \tag{3.41}$$

Now $\boldsymbol{x}$ satisfies the KKT conditions if and only if $\boldsymbol{s_N} \geq \boldsymbol{0}$, so we can tell whether or not the vertex $\boldsymbol{x}$ is optimal. If $\boldsymbol{x}$ is indeed optimal, we are done and the LP is solved. Otherwise, we do a pivot as described in subsection 3.5.3.

3.5.2 Finding an initial basis

When the revised simplex method encounters a degenerate basic feasible solution $\boldsymbol{x}$, we need a way to find a basis B corresponding to $\boldsymbol{x}$. That means finding a set of m columns from A such that the submatrix of A consisting of those columns is non-singular. The basis also needs to include all columns for which $\boldsymbol{x}$ has a non-zero entry. Note that this basis is still required when we are dealing with degeneracy with a perturbation as described in subsection 3.5.4.

We start by taking the indices in which $\boldsymbol{x}$ is non-zero. The columns corresponding to these indices are guaranteed to be linearly independent, since $\boldsymbol{x}$ is a BFS. Then, we try adding each column from A one by one and check to see if the resulting matrix is still of maximal rank. When this is not the case, the most recently added column is dropped again. After trying this for all columns of A, we always find m total columns that are linearly independent, since A has rank m.

3.5.3 Pivot

In case $\boldsymbol{x}$ is optimal, the revised simplex method is done. Otherwise, a pivot operation is performed by replacing one column index in B by one in N.

To select the entering index, consider $\boldsymbol{s_N}$. At least one element is negative, otherwise the previous $\boldsymbol{x}$ would have been optimal. Choose any q with $s_q < 0$ as the entering index.

The procedure for altering B and changing $\boldsymbol{x}$ and $\boldsymbol{s}$ accordingly is as follows:

- increase x_q from zero;
- keep all other components of $\boldsymbol{x_N}$ at zero;
- change the current basic vector $\boldsymbol{x_B}$ in such a way that $A\boldsymbol{x} = \boldsymbol{b}$ remains satisfied;
- keep increasing x_q until one of the components of $\boldsymbol{x_B}$ (say x_p) reaches zero, or determine that no such component exists (then the LP is unbounded);
- remove p from B and replace it with q.

Call the new vertex $\boldsymbol{x}^+$. Since both $A\boldsymbol{x} = \boldsymbol{b}$ and $A\boldsymbol{x}^+ = \boldsymbol{b}$, and since $\boldsymbol{x_N} = \boldsymbol{0}$ and $x_i^+ = 0$ for $i \in N \setminus \{q\}$, we have

$$A\boldsymbol{x}^+ = A_B\boldsymbol{x}_{\boldsymbol{B}}^+ + A_q x_q^+ = A_B\boldsymbol{x_B} = A\boldsymbol{x}. \tag{3.42}$$

Since A_B is nonsingular we can apply A_B^{-1} and obtain

$$\boldsymbol{x}_{\boldsymbol{B}}^+ = \boldsymbol{x_B} - A_B^{-1}A_q x_q^+. \tag{3.43}$$

This shows how to change $\boldsymbol{x_B}$ so that $A\boldsymbol{x} = \boldsymbol{b}$ remains satisfied.

Now let us verify that this pivot leads to a decrease in the objective $\boldsymbol{c}^T\boldsymbol{x}$. We know that

$$\boldsymbol{x}_{\boldsymbol{N}}^+ = (0, \dots, 0, x_q^+, 0, \dots, 0)^T, \tag{3.44}$$

so

$$\begin{aligned}\boldsymbol{c}^T\boldsymbol{x}^+ &= \boldsymbol{c_B}^T\boldsymbol{x}_{\boldsymbol{B}}^+ + \boldsymbol{c_N}^T\boldsymbol{x}_{\boldsymbol{N}}^+ \\ &= \boldsymbol{c_B}^T\boldsymbol{x}_{\boldsymbol{B}}^+ + c_q x_q^+ \\ &= \boldsymbol{c_B}^T\boldsymbol{x_B} - \boldsymbol{c_B}^T A_B^{-1}A_q x_q^+ + c_q x_q^+.\end{aligned} \tag{3.45}$$

From (3.40) we have $\boldsymbol{c_B}^T A_B^{-1} = \boldsymbol{\pi}^T$, and from (3.39) we get $A_q\boldsymbol{\pi} = c_q - s_q$ since A_q is a column of A_N. Therefore,

$$\boldsymbol{c_B}^T A_B^{-1}A_q x_q^+ = \boldsymbol{\pi}^T A_q x_q^+ = (c_q - s_q)x_q^+, \tag{3.46}$$

so by substituting in (3.45) we obtain

$$\boldsymbol{c}^T\boldsymbol{x}^+ = \boldsymbol{c_B}^T\boldsymbol{x_B} - (c_q - s_q)x_q^+ + c_q x_q^+ = \boldsymbol{c_B}^T\boldsymbol{x_B} - s_q x_q^+. \tag{3.47}$$

Since $\boldsymbol{x_N} = 0$, we have $\boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{c_B}^T\boldsymbol{x_B}$ and therefore

$$\boldsymbol{c}^T\boldsymbol{x}^+ = \boldsymbol{c}^T\boldsymbol{x} - s_q x_q^+. \tag{3.48}$$

We chose q such that $s_q < 0$, and since $x_q^+ > 0$ if we are able to increase it at all, it follows that the step produces a decrease in the objective function $\boldsymbol{c}^T\boldsymbol{x}$. Note that in our setting of performing adjacency tests, we only need to know whether the optimal value is zero or not. So as soon as a pivot causes a large enough change in $\boldsymbol{c}^T\boldsymbol{x}$ (large enough that it is not just a rounding error away from zero), we can stop the revised simplex method and return that the rays are not adjacent.

If we are not able to increase x_q at all, then $\boldsymbol{x}$ must be a degenerate solution, since one of the components of $\boldsymbol{x_B}$ must already be zero before the pivot. In that case the pivot operation only changes the basis, but the objective value stays the same. This is problematic because we could later change the basis back to the one we started at, whereas in the non-degenerate case the strict decrease of $\boldsymbol{c}^T\boldsymbol{x}$ prevents this. The next section presents a solution to this cycling problem.

3.5.4 Perturbation to prevent cycling

Degenerate vertices (i.e. vertices $\boldsymbol{x}$ that have fewer than m non-zero components) are problematic for the (revised) simplex method. When $\boldsymbol{x}$ is degenerate, the pivot operation described above might not cause any change in $\boldsymbol{x}$ at all. It is possible to make a number of degenerate pivots resulting in the same basis we had before. In this case the algorithm would never terminate.

We use a perturbation strategy to circumvent this problem. Call the original instance LP and the perturbed one LP'. The perturbation is only applied when we reach a degenerate vertex $\bar{\boldsymbol{x}}$ in the revised simplex method. At that point we also have a corresponding basis $\bar{B}$. Instead of the original constraint

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{3.49}$$

we change the right-hand-side vector to

$$A\boldsymbol{x} = \boldsymbol{b'} = \boldsymbol{b} + A_{\bar{B}}\boldsymbol{\varepsilon}. \tag{3.50}$$

Here $\boldsymbol{\varepsilon}$ is a vector of length m with elements chosen uniformly at random from $[\delta/2, \delta]$, where $\delta > 0$ is a small constant to be used by the algorithm. Note that $\boldsymbol{\varepsilon} > \boldsymbol{0}$. The solution corresponding to this new constraint is

$$\bar{\boldsymbol{x}}' = A_{\bar{B}}^{-1}\boldsymbol{b'} = A_{\bar{B}}^{-1}(\boldsymbol{b} + A_{\bar{B}}\boldsymbol{\varepsilon}) = \bar{\boldsymbol{x}} + \boldsymbol{\varepsilon}. \tag{3.51}$$

Since we choose $\boldsymbol{\varepsilon} > \boldsymbol{0}$, this new solution still satisfies $\bar{\boldsymbol{x}}' \geq \boldsymbol{0}$. This means $\bar{B}$ is also a feasible basis for the perturbed problem LP'.

Theorem 3.3 (properties of perturbed LP). *The following hold for LP':*
(a) LP' is non-degenerate.
(b) If B is a feasible basis of LP', then B is also a feasible basis of LP.
(c) If B is an optimal basis of LP', then B is also an optimal basis of LP.
(d) If x_q can leave and x_p can enter in a pivot corresponding to B in LP', then the same holds in LP.

For a proof, see for example [19]. This theorem shows that we can use the perturbed version LP' to do all the pivots, and once we find an optimal basis it is guaranteed that this basis is also optimal for the original problem.

4 Implementation in ecmtool

An open source implementation of the direct intersection approach is available in *ecmtool*. It is written in Python, with some parts using Cython[1]. The implementation of the revised simplex method is based on code from SciPy[2].

The input required by ecmtool is a metabolic network stored as an SBML model[3].

4.1 Compression

To make computation faster, the network obtained from an SBML file first undergoes several compression steps. By compression we mean the network is changed to reduce the number of metabolites or reactions (or both) without affecting the ECMs. Some of the compression methods we use were suggested by [11]. Compression can be disabled with the argument `--compress false`.

4.1.1 Compression by reversible reaction

This compression uses a reversible reaction to eliminate an internal metabolite involved in that reaction.

For each reversible reaction, we find the internal metabolites that take part in that reaction. If there are none (the reaction involves only external metabolites), we skip this reaction. Out of these internal metabolites, we select one of them involved in the least number of reactions for elimination.

We now eliminate the selected metabolite m by adding a multiple of the reversible reaction to each other reaction that involves m. The multiple is chosen so that the sum has a 0 in the row corresponding to m. Finally the reversible reaction itself can be dropped, because it is the only reaction with a non-zero coefficient for m; and since m is internal, the steady state requirement now excludes this reaction from being used at all.

As an example, consider the same network we have used before (Figure 4.1). There is one reversible reaction, **r.5**. Metabolites B and E are involved in this reaction, and

[1]https://cython.org/
[2]https://www.scipy.org/
[3]http://sbml.org

they are both internal. Because metabolite B is in 3 reactions while metabolite E is in 4, metabolite B is selected for elimination. We modify the other 2 reactions that use B:

$$\mathbf{r.1} = (-1,1,0,0,0,0,0,0)^T \mapsto \mathbf{r.1} + \mathbf{r.5} = (-1,0,0,0,1,0,0,0)^T \tag{4.1}$$

and

$$\mathbf{r.4} = (0,-1,1,0,0,0,0,0)^T \mapsto \mathbf{r.4} - \mathbf{r.5} = (0,0,1,0,-1,0,0,0)^T. \tag{4.2}$$

Finally **r.5** is dropped. The result is on the right in Figure 4.1.

Figure 4.1: Example of compression by reversible reaction.

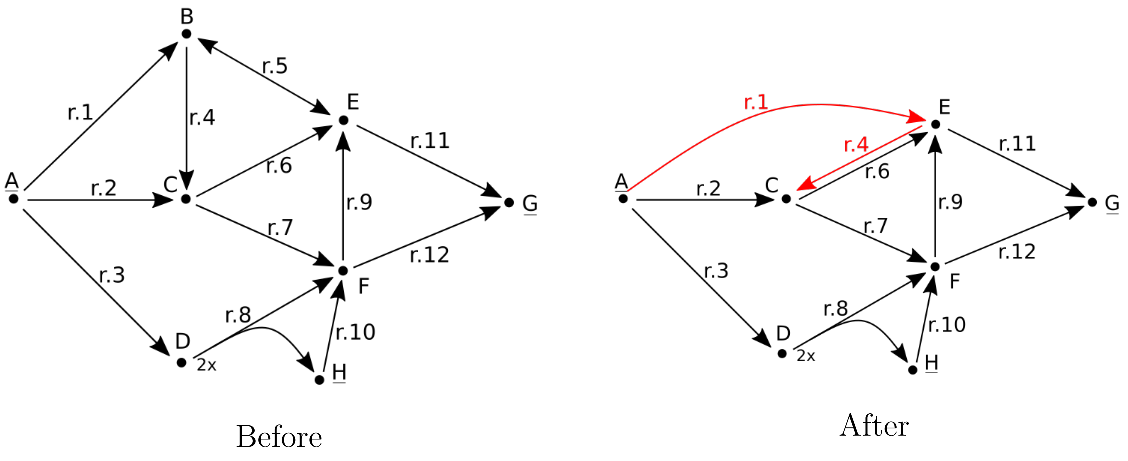


Table 4.1: Stoichiometry matrix of example network before compression.

	r.1	**r.2**	**r.3**	**r.4**	**r.5**	**r.6**	**r.7**	**r.8**	**r.9**	**r.10**	**r.11**	**r.12**
A	-1	-1	-1	0	0	0	0	0	0	0	0	0
B	1	0	0	-1	-1	0	0	0	0	0	0	0
C	0	1	0	1	0	-1	-1	0	0	0	0	0
D	0	0	1	0	0	0	0	-2	0	0	0	0
E	0	0	0	0	1	1	0	0	1	0	-1	0
F	0	0	0	0	0	0	1	1	-1	1	0	-1
G	0	0	0	0	0	0	0	0	0	0	1	1
H	0	0	0	0	0	0	0	1	0	-1	0	0
Reversible	no	no	no	no	yes	no	no	no	no	no	no	no

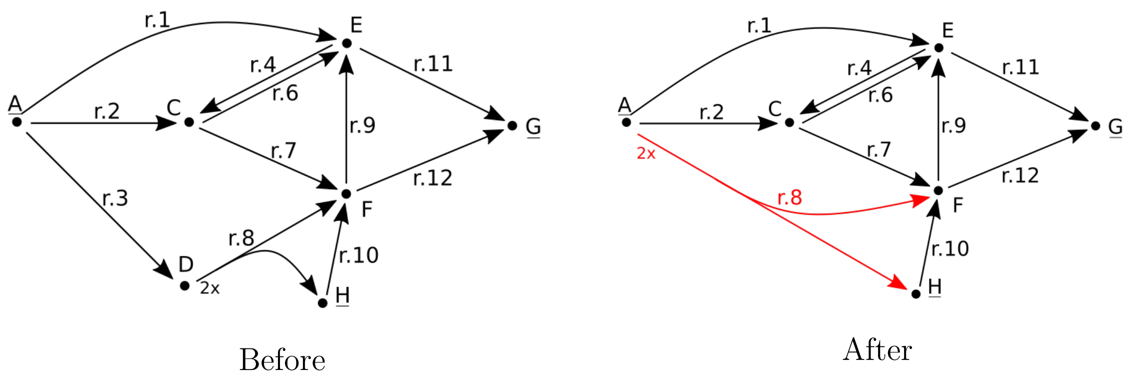
4.1.2 Compression by singly produced/consumed metabolite

This compression finds internal metabolites that are produced in only a single reaction, or consumed in only a single reaction. In this case the usage of the single producing (consuming) reaction is completely determined by the usage of the consuming (producing) reactions. This enables us to eliminate the singly produced/consumed internal metabolite, by adding multiple of the single reaction to all other reactions involving this metabolite.

We could also have achieved this later by doing an equality intersection (section 4.4) but compressing it beforehand avoids the work of the adjacency tests. The price to pay for this is that redundant (i.e. non-extreme) rays might be added, but this does not incur an extra cost since the network still needs to be processed by `redund` at this point anyway.

For example, consider the network as we left it in the previous subsection (Figure 4.2 left side). Internal metabolite D is produced exclusively by **r.3**. So we can add the appropriate multiple (in this case 2) of **r.3** to all reactions that consume D (in this case only **r.8**). Then, metabolite D and **r.3** can be dropped. The end result is on the right side of Figure 4.2.

Figure 4.2: Example of compression by singly produced/consumed metabolite (D).



4.2 External metabolite directions

As already mentioned in (3.2), each external metabolite is an *input* (can only be consumed), an *output* (can only be produced) or *bidirectional* (can be both consumed and produced).

By default, the program will determine the direction of each external metabolite by checking if that metabolite is produced by any reaction and consumed by any reaction. It is possible to specify beforehand which metabolites should only be consumed with `--inputs` and which should be only produced with `--outputs`.

4.3 Hiding metabolites (optional)

In many applications, not all external metabolites are of interest. With the `--hide` argument, the user can give a list of external metabolites that are not important to the question at hand.

A hidden metabolite is marked internal, and an exchange reaction is added in one or both reactions depending on the directionality (input/output/both) of the metabolite. This means the metabolite can be freely produced and/or consumed.

Geometrically, hiding metabolite m is equivalent to projecting the output cone onto the hyperplane $m = 0$.

The output consists of the ECMs of the modified network. In these ECMs the hidden metabolites are not present. If this causes an ECM to be all zero, it is removed. It is also possible that hiding metabolites causes rays that were extreme rays (i.e. ECMs) initially, to no longer be extreme rays (so they are now a conic combination of other rays). Such rays are not present in the output.

The advantage of hiding metabolites is a reduction in computation time that can be very significant. This reduction is possible because hidden metabolites can be treated differently by the algorithm throughout the computation, rather than modifying the ECMs at the end.

4.3.1 Example

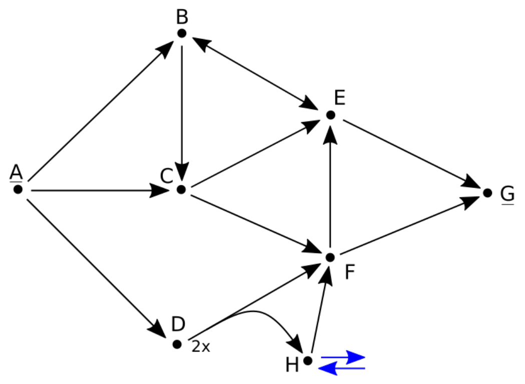


Figure 4.3: Metabolite H is hidden.

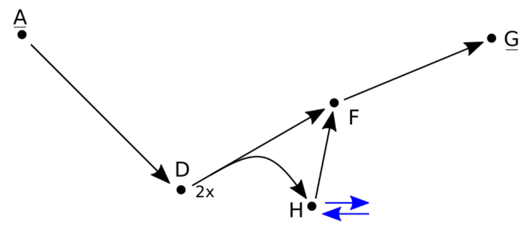


Figure 4.4: Redundant rays and one cycle are removed.

We use the same example from before, which is the network in Figure 1.1. This time we use `--hide` to hide metabolite H. The hiding occurs before anything else, so at this point the network looks like Figure 4.3. Note that H is a bidirectional external metabolite in the original network: it can be both produced and consumed.

Then `redund` is used to get rid of superfluous rays, and part of the cycle reduction is already done to remove metabolites B, C, and E. The situation is then as in Figure 4.4. Note that there is still one cycle left; it is formed by the two exchange reactions that were added to metabolite H. Removing this cycle by the cycle elimination procedure (as described in subsection 3.3.2) results in the network in Figure 4.5. Note that there is now a ray that produces F from nothing.

Finally, the internal metabolites D and F are eliminated. In the end, there is one ray that turns 2A into G, and one that produces G from nothing (Figure 4.6). These are the ECMs of the modified network.

	Original ECM	**Projection**	**Notes**
1.	$2\underline{A} \rightarrow 2\underline{G}$	$2\underline{A} \rightarrow 2\underline{G}$	Not extreme (1. = 2. + 3.)
2.	$2\underline{A} \rightarrow \underline{G} + \underline{H}$	$2\underline{A} \rightarrow \underline{G}$	
3.	$\underline{H} \rightarrow \underline{G}$	$\rightarrow \underline{G}$	

Table 4.2: Comparison of ECMs with and without hiding H.

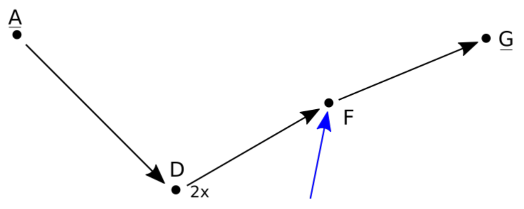


Figure 4.5: The last cycle is removed.

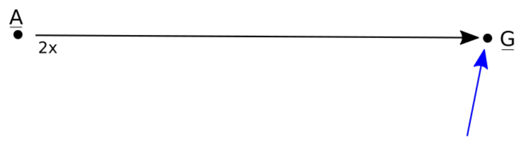


Figure 4.6: Final result, showing 2 ECMs.

4.4 Equality intersection

At this stage all cycles are removed as detailed in section 3.3.

4.4.1 Ordering of internal metabolites

The internal metabolites need to be eliminated one by one. For metabolite i, we intersect the cone R with the equality $R_{i,*} = 0$.

The order in which the internal metabolites are eliminated has a large effect on the total computation time. This is similar to the sensitivity of the Double Description method to changing the order in which inequalities get added [15].

If we were to remove metabolite i, we would need to do $n_i^+ \cdot n_i^-$ adjacency tests, where n_i^+ is the number of strictly positive numbers in row i of R and n_i^- is the number of strictly negative numbers in that row. We use $n_i^+ \cdot n_i^-$ as a heuristic to decide which metabolite to remove next, selecting the i with smallest $n_i^+ \cdot n_i^-$.

This choice is not necessarily optimal, since removing an 'easy' metabolite now might cause many future metabolites to take significantly more adjacency tests. However, the total number of possible orderings of internal metabolites is very large ($m_{int}!$ where m_{int} is the number of internal metabolites), and there is no clear way to find an optimal ordering. In our experience this heuristic gives a reasonably good performance while also being easy and computationally cheap to compute.

4.4.2 Removing an internal metabolite

Once the internal metabolite i that will be removed in this iteration is selected, we can compute the sets

$$J^+ = \{j \in \{1, \dots, n\} : R_{i,j} > 0\}, \tag{4.3}$$

$$J^- = \{j \in \{1, \dots, n\} : R_{i,j} < 0\}, \tag{4.4}$$

and

$$J^0 = \{j \in \{1, \dots, n\} : R_{i,j} = 0\}. \tag{4.5}$$

Then we need to compute the adjacency of each pair (r_{*,j_+}, r_{*,j_-}) with $(j_+, j_-) \in J^+ \times J^-$. This is the part of the algorithm that takes the most time by far. Using the geometric adjacency test as described in (3.4), we need to solve $|J^+ \times J^-|$ linear programming problems. In fact, the next section will show that we need to do slightly less work than 'solving' each of the LPs.

4.4.3 LP solving

Recall LP($\boldsymbol{r_{*,a}}, \boldsymbol{r_{*,b}}$) from (3.4.1):

$$\text{maximize} \sum_i \lambda_{i,*} - \lambda_a - \lambda_b,$$

$$\text{subject to}$$

$$R\boldsymbol{\lambda} = \boldsymbol{x},$$

$$\boldsymbol{\lambda} \geq \mathbf{0},$$

where

$$\boldsymbol{x} = 1/4\boldsymbol{r_{*,a}} + 3/4\boldsymbol{r_{*,b}}.$$

As shown in Theorem 3.1, $\boldsymbol{r_{*,a}}$ and $\boldsymbol{r_{*,b}}$ are adjacent if and only if the optimal value of this LP is zero. We are using the LP exclusively to test adjacency, so instead of solving it to optimality we only need to know whether the optimum is zero or not. This allows us to terminate the revised simplex solver as soon as we find a feasible solution with objective value greater than zero.

Before the revised simplex process itself, we normalize the columns of R with respect to the L1 norm for numerical stability. Then, we keep only a maximal linearly independent set of rows from R; rows that are dependent correspond to redundant constraints in the LP that only slow down the solver.

An additional starting requirement for the revised simplex method is an initial basic feasible solution $\boldsymbol{\mu}$, along with a corresponding basis. The basic solution we want to use is the one used to determine $\boldsymbol{x}$; so $\mu_a = 1/4$, $\mu_b = 3/4$ and the other elements are zero. However, since this $\boldsymbol{\mu}$ has only two non-zero elements, we do not immediately have a corresponding basis unless R has only two rows.

To retrieve an initial basis corresponding to $\boldsymbol{\mu}$, we need to find an $m \times m$ non-singular submatrix of R that contains columns $\boldsymbol{r_{*,a}}$ and $\boldsymbol{r_{*,b}}$. We do this by starting with the two required columns, and adding columns one by one so long as they are not linearly dependent on the columns already in use. It is always possible to find a basis, since $\boldsymbol{\mu}$ is a basic feasible solution and therefore has at least one basis. A more advanced method of finding initial bases is presented in subsection 4.4.4.

Now we are ready to use the revised simplex method. The starting vertex is always degenerate, so we apply the perturbation in (3.5.4). Then we do pivot steps one by one, until either the objective value is significantly far away from zero (in this case the rays are not adjacent) or we find that a solution with objective close to zero is optimal (in this case the rays are adjacent).

4.4.4 Finding initial bases

We need to compute the adjacency of each pair $(\boldsymbol{r}_{*,j_+}, \boldsymbol{r}_{*,j_-})$ with $(j_+, j_-) \in J^+ \times J^-$. The number of such pairs can be in the millions for genome scale networks. In subsection 4.4.3 we find an initial basis for each LP independently. However, there is some relation between the LPs that we can take advantage of in order to reduce the amount of work needed.

Recall that for each pair $(\boldsymbol{r}_{*,j_+}, \boldsymbol{r}_{*,j_-})$ we need to find a set $B(j_+, j_-)$ of m linearly independent columns of R that must contain $\boldsymbol{r}_{*,j_+}$ and $\boldsymbol{r}_{*,j_-}$. Since the pairs come from $J^+ \times J^-$, a significant number of them share the same $\boldsymbol{r}_{*,j_+}$. Denote $n^+ = |J^+|$ and $n^- = |J^-|$. The idea is to first find a basis for each $\boldsymbol{r}_{*,j_+}$ and then use that basis for each of the n^- pairs that contain that specific $\boldsymbol{r}_{*,j_+}$.

First, we find any basis B_0 of the columns space of R, i.e. any set of m linearly independent columns from R. We then find an LU decomposition of B_0.

For each $j_+ \in J^+$ we find $B(j_+)$ (a basis containing $\boldsymbol{r}_{*,j_+}$) by replacing one column from B_0 with $\boldsymbol{r}_{*,j_+}$. To do that, we use the LU decomposition of B_0 to solve

$$B_0 \boldsymbol{x} = \boldsymbol{r}_{*,j_+}. \tag{4.6}$$

According to Cramer's rule, since B_0 is square and invertible:

$$x_i = \frac{\det(B_0^i)}{\det(B_0)}, \tag{4.7}$$

where B_0^i is the matrix formed by replacing the i-th column of B_0 with $\boldsymbol{r}_{*,j_+}$. We know that $\boldsymbol{x} \neq \boldsymbol{0}$ since $\boldsymbol{r}_{*,j_+} \neq \boldsymbol{0}$ (because $\boldsymbol{r}_{*,j_+}$ is a ray). Therefore there is an i^* with $x_{i^*} \neq 0$, and we can replace the i^*-th column of B_0 with $\boldsymbol{r}_{*,j_+}$ to form $B(j_+)$. Equation 4.7 guarantees that $B(j_+)$ is nonsingular. We compute the LU decomposition of each $B(j_+)$.

Next we can construct $B(j_+, j_-)$ for each pair $(j_+, j_-) \in J^+ \times J^-$. We use the LU decomposition of $B(j_+)$ to solve

$$B(j_+)\boldsymbol{x} = \boldsymbol{r}_{*,j_-}. \tag{4.8}$$

Again $\boldsymbol{x} \neq \boldsymbol{0}$. It is also impossible that the only non-zero x_i is the one corresponding to $\boldsymbol{r}_{*,j_+}$, since $\boldsymbol{r}_{*,j_+}$ and $\boldsymbol{r}_{*,j_-}$ are distinct rays. So there is an $x_{i^*} \neq 0$ that is not the one corresponding to $\boldsymbol{r}_{*,j_+}$. This shows that the i^*-th column of $B(j_+)$ can be replaced by $\boldsymbol{r}_{*,j_-}$; we made sure not to replace $\boldsymbol{r}_{*,j_+}$, so this gives a basis $B(j_+, j_-)$ containing $\boldsymbol{r}_{*,j_+}$ and $\boldsymbol{r}_{*,j_-}$ that is guaranteed to be nonsingular by Equation 4.7.

In total, this procedure for finding bases for each pair (j_+, j_-) requires the LU decomposition of $n^+ + 1$ matrices as well as $n^+ + n^+n^-$ matrix-vector multiplications. LU decomposition has complexity $\mathcal{O}(m^3)$ in practice (optimized CW-like algorithms have complexity $\mathcal{O}(m^{2.373})$ [21], but they are not useful in practice). Matrix-vector multiplication has complexity $\mathcal{O}(m^2)$. So we can expect this approach to have complexity

$$\mathcal{O}(n^+ \cdot m^3) + \mathcal{O}(n^+ \cdot n^- \cdot m^2). \tag{4.9}$$

On the other hand, finding each basis individually (as in subsection 4.4.3) requires, in the worst case, n computations of the rank of an $m \times m$ matrix per (j_+, j_-) pair. A rank computation in practice[4] uses a singular value decomposition, which has complexity $\mathcal{O}(m^3)$. So the total complexity for the naive approach is

$$\mathcal{O}(n \cdot n^+ \cdot n^- \cdot m^3), \tag{4.10}$$

markedly slower than the procedure presented in this subsection.

4.5 Parallelization

Since the adjacency tests are independent of each other, and we commonly need to perform millions of them to eliminate a single metabolite (for genome scale networks) this algorithm is highly suitable for parallel processing. We have implemented this with the standard Python library `multiprocessing`[5].

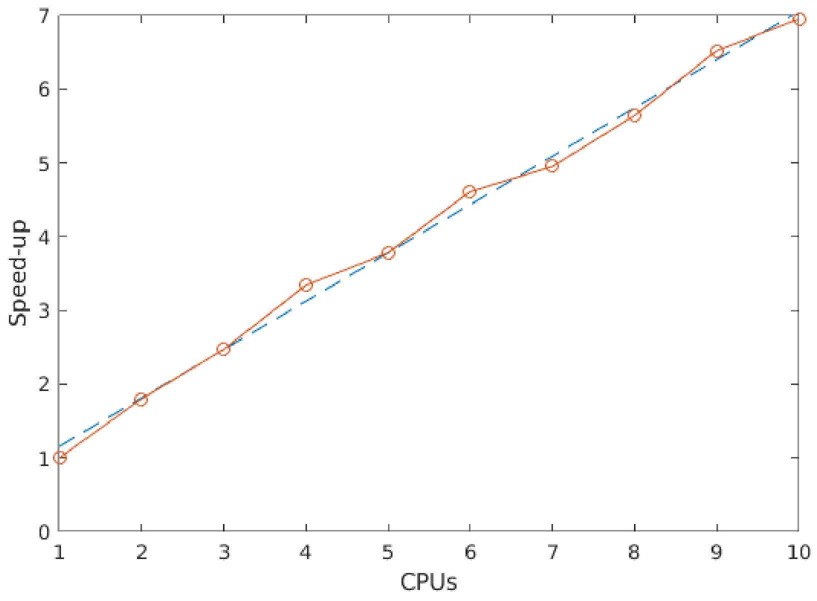
In table 4.3 we show the relative speed-up compared to using a single CPU for the *E. coli* core model. It shows close to linear gains. This is a smaller network, where the most LPs done in a single step is around $10e5$, so for genome scale networks we can expect the scaling to continue up to hundreds or more CPUs.

Table 4.3: Speed-up with different processor counts for *E. coli* core

CPUs	2	3	4	5	6	7	8	9	10
Speed-up rel. to 1 CPU	1.80	2.47	3.34	3.78	4.61	4.95	5.64	6.52	6.95

[4]`http://mathworks.com/help/matlab/ref/rank.html`
[5]`https://docs.python.org/3/library/multiprocessing.html`

Figure 4.7: CPU count vs speed-up (red) and linear least-squares fit (blue, dashed). The fit is $y = 0.655x + 0.503$.



4.6 Numerical precision

During the computation, there are two options for storing numbers: either as fractions (infinite precision) or as floating point numbers (finite precision). Generally, fractions are more accurate while floats are faster.

In principle all stoichiometric coefficients found in the initial matrix R are integers. This is because the numbers originate from molar amounts of molecules involved in a single reaction. However, the writers of metabolic models often include a so-called *biomass reaction* that models cell growth by consuming and producing some number of metabolites (that can be internal or external) [23]. This virtual reaction is not based on a single real chemical reaction, and its coefficients are not integers. In fact, the numbers in the biomass reaction are based on empirical data and are thus always measurements that are rounded based on numerical significance somehow.

Despite the biomass reaction, we found it preferable to store the matrix R with infinite precision as fractions. The downside is that the numerators and denominators, as well as the fractions themselves, grow larger as more internal metabolites are eliminated through equality intersection. However, since we identify ECMs by being extreme rays, it is important that we have an exact numerical representation of each ECM. Otherwise, the program could output many more ECMs than really exist, as differences in rounding would make them appear distinct.

During the solving of LPs we use a floating point version of R. This is because linear programming inherently requires the variables and constraints to be specified in terms of real numbers. Solving the adjacency tests in terms of fractions would amount to *integer* linear programming, which is computationally much harder (in fact, it is NP-complete [24]).

5 Computational results

5.1 *E. coli* core

The first network for which we have computed the ECMs is the E. coli *core* network[9]. It is available on the BiGG database[1]. This model contains only a limited part of the metabolism of *E. coli*. It has 72 metabolites and 95 reactions.

After compression, 37 metabolites and 48 rays are left (rays are columns of R, which are conic combinations of reactions). Then the intersection procedure begins, which eliminates all the internal metabolites; in this case there are 13. Each intersection requires a number of adjacency tests as indicated in the table below:

Table 5.1: *E. coli* core iterations

Iteration	**Metabolite**	**Tests required**	**Rays after**
1	akg	20	50
2	f6p	24	55
3	pep	33	51
4	mal__L	76	60
5	pyr	114	81
6	nadp	312	115
7	nadph	0	115
8	nad	880	151
9	nadh	0	151
10	h2o	2,006	178
11	co2	5,693	289
12	h_e	19,470	872
13	h	104,536	696

Some iterations require zero tests. This occurs when the internal metabolite is always used in conjunction with another internal metabolite, which has already been eliminated. In this case the entire row is already zero, so the intersection step has no work to do. The number of rays does not change in such a step.

Note that, for *E. coli* core, the number of tests increases monotonically if we ignore

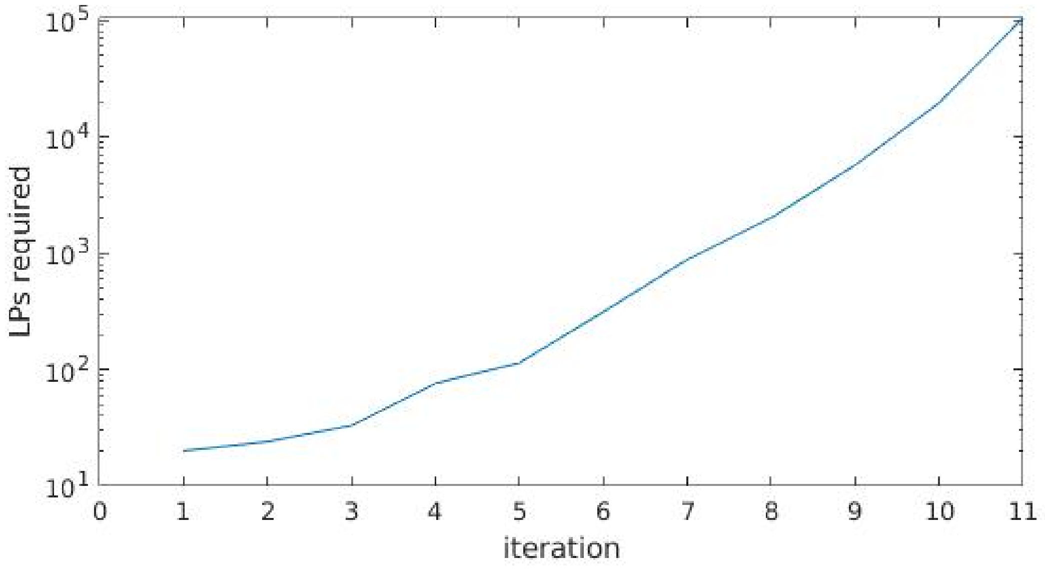
[1]`http://bigg.ucsd.edu/models/e_coli_core`

the special case where no tests are required. This indicates that the intersection steps later in the algorithm are generally more computationally demanding than those at the start. It appears that the increase in number of tests required is roughly exponential.

The number of rays does not quite increase monotonically (it goes from 55 to 51, and later from 872 to 696), but it is not far off. We see that the number of rays increases much more slowly than the number of adjacency tests required; this is because the matrix R becomes more dense over time.

Note that 696 is not the final number of ECMs, because splitting bidirectional external metabolites causes some artificial rays that are zero after un-splitting those metabolites at the end.

Figure 5.1: Log scale plot of the number of LPs required in iterations for *E. coli* core



5.2 iAB_RBC_283

This is genome scale model of a human red blood cell that was published in 2011[17]. It is available on the BiGG database[2].

Initially there are 342 metabolites and 469 reactions. After compression we have 135 metabolites and 125 rays, with 26 metabolites being internal.

Unfortunately we have not yet been able to enumerate the ECMs of this network, because the required number of LPs becomes prohibitively large after some time. In table

[2]`http://bigg.ucsd.edu/models/iAB_RBC_283`

5.2 the internal metabolites that still need to be eliminated after h2o are atp, adp, and r1p.

Table 5.2: iAB_RBC_283 iterations

Iteration	**Metabolite**	**Tests required**	**Rays after**
1	arg__L	0	125
2	nh4	3	124
3	o2	12	127
4	co2	14	132
5	adn	18	139
6	pchol_hs_18_1_18_2	20	138
7	pchol_hs_18_2_16_0	20	138
8	pchol_hs_18_2_18_1	20	137
9	glyc3p	21	145
10	chol	0	144
11	coa	0	130
12	h2o_e	28	135
13	nadh	30	146
14	nad	2	144
15	ppi	43	141
16	odecoa	49	153
17	pmtcoa	0	153
18	f6p	52	184
19	nadph	416	303
20	nadp	0	302
21	pi	1,617	1,089
22	h	92,336	8,318
23	h2o	11,146,575	?
24	?	?	?
25	?	?	?
26	?	?	?

5.2.1 Hiding two metabolites

One of the options we have for speeding up computation is hiding external metabolites (4.3). This comes at the cost of losing those metabolites in the output ECMs. For this model, for example, hiding adenine and adenosine gives a significant speed-up.

Since adenine is one of the four nucleobases used in forming nucleotides of the nucleic acids (DNA and RNA), and adenosine has derivatives that play an important role in biochemical energy transfer such as adenosine triphosphate (ATP) and adenosine diphosphate (ADP), we can expect to lose a lot of important information by ignoring them. However, enumerating all of the ECMs except for the amount of adenine and adenosine in them is still useful for many applications.

Table 5.3: iAB_RBC_283 iterations, after hiding adenine and adenosine

Iteration	**Metabolite**	**Tests required**	**Rays after**
1	arg__L	0	121
2	nh4	3	120
3	o2	12	123
4	co2	14	128
5	pchol_hs_18_1_18_2	20	127
6	pchol_hs_18_2_16_0	20	126
7	pchol_hs_18_2_18_1	20	125
8	glyc3p	21	133
9	chol	0	132
10	coa	0	118
11	h2o2	28	123
12	nadh	30	134
13	nad	2	132
14	ppi	41	130
15	odecoa	49	142
16	pmtcoa	0	142
17	f6p	52	173
18	nadph	416	292
19	nadp	0	291
20	adp	1,568	718
21	atp	1,572	494
22	pi	150	537
23	h	9,807	2,786
24	h2o	1,729,512	16,580

Hiding these two metabolites enables us to compute 16,580 ECMs. We expect this to be close to the number of ECMs of the full network, since the only reduction would come from ECMs that are identical except on adenine and adenosine.

6 Conclusion

We have shown that computation of the Elementary Conversion Modes of metabolic networks is possible by using an adjacency test that consists of solving a linear program. The information about the adjacency of rays is used to efficiently intersect an intermediary cone, so that one of the internal metabolites is eliminated. After eliminating all internal metabolites in this way, we are left with a cone whose extreme rays are precisely the ECMs.

The number of LP instances that has to be solved in order to find all ECMs of genome scale networks is very large; at least 10 million for all networks we tried. This means additional work is required, in order to either reduce this number or increase the speed with which a linear program can be solved.

It might be possible to simultaneously solve thousands of LPs by making use of GPU programming. Alternatively, a supercomputer with many cores could enumerate all ECMs for large networks if given enough time. In either case, the direct intersection approach is highly suitable for parallelisation.

In fact we are not interested in the exact solution of each LP, but rather in the question whether it has optimal value zero. This occurs only when the feasible region is a single point. There might be ways to exploit this feature and speed up computation.

An option that is implemented already in ecmtool is hiding certain metabolites. This gives a reduced output, that can nonetheless provide all the information required for applications in which not all external metabolites are of importance. We have provided an example where hiding two metabolites makes ECM computation of a genome scale network feasible. However, the speed-up obtained by hiding metabolites is dependent on the network and on which metabolites are hidden, so there is no way to predict performance before trying the entire computation.

Bibliography

[1] J. Zanghellini, D. E. Ruckerbauer, M. Hanscho, C. Jungreuthmayer, *Elementary flux modes in a nutshell: Properties, calculation and applications.*, Biotechnology Journal, **8**: 1009-1016, doi: 10.1002/biot.201200269, 2013.

[2] J. Gagneur, S. Klamt, *Computation of elementary modes: a unifying framework and the new binary approach.* BMC Bioinformatics **5**:175, doi: 10.1186/1471-2105-5-175, 2004.

[3] M. Terzer, *Large scale methods to enumerate extreme rays and elementary modes.* ETH PhD Thesis (available at `http://e-collection.ethbib.ethz.ch/view/eth:534`), 2009.

[4] J. Stelling, S. Klamt, K. Bettenbrock, S. Schuster et al., *Metabolic network structure determines key aspects of functionality and regulation.*, Nature 2002, *420*, pp. 190-193.

[5] J. Behre, T. Wilhelm, A. von Kamp, E. Ruppin et al., *Structural robustness of metabolic networks with respect to multiple knock-outs.*, J. Theor. Biol. 2008, *252*, pp. 433-441.

[6] C. T. Trinh, R. A. Thompson, *Elementary mode analysis: A useful metabolic pathway analysys tool for reprogramming microbial metabolic pathways.* in *Repgrogramming Microbial Metabolic Pathways*, volume 64, Springer Netherlands, Dordrecht 2012, pp. 21-42.

[7] C. T. Trinh, A. Wlaschin, F. Srienc, *Elementary mode analysis: A useful metabolic pathway analysis tool for characterizing cellular metabolism.* Appl. Microbiol. Biotechnol. 2009, *81*, pp. 813-826.

[8] G. Melzer, M. E. Esfanbadabi, E. Franco-Lara, C. Wittmann, *Flux Design: In silico design of cell factories based on correlation of pathway fluxes to desired properties.* BMC Syst. Biol. 2009, *3*, pp. 120.

[9] J. D. Orth, R. M. T. Fleming, B. Ø. Palsson, *Reconstruction and use of microbial metabolic networks: the core Escherichia coli metabolic model as an educational guide.* Chapter 10.2.1 in: EcoSal - Escherichia coli and Salmonella Cellular and Molecular Biology, ASM Press, 2009.

[10] J. Zanghellini, C. Jungreuthmayer, *Designing optimal cell factories: Integer programming couples elementary mode analysis with regulation.* BMC Syst. Biol. **6**:103, 2012.

[11] R. Urbanczik, C. Wagner, *Functional stoichiometric analysis of metabolic networks.* Bioinformatics Journal, **21**: 4176-4180, doi: 10.1093/bioinformatics/bti674, 2005.

[12] S. Klamt, J. Stelling, *Combinatorial complexity of pathway analysis in metabolic networks.* Mol Bio Rep, **29**: 233-236, 2002.

[13] J. D. Orth, I. Thiele, B. Ø. Palsson, *What is flux balance analysis?* Nat Biotechnol. 2010 Mar; 28(3): pp. 245-248, doi: 10.1038/nbt.1614

[14] T.S. Motzkin, H. Raiffa, G.L. Thompson, R.M. Thrall, *The double description method.* Contributions to theory of games, volume 2, 1953.

[15] K. Fukuda, A. Prodon, *Double Description Method Revisited.* 1996.

[16] D. Avis, *lrs: A Revised Implementation of the Reverse Search Vertex Enumeration Algorithm.* See also: lrs home page (http://cgm.cs.mcgill.ca/ avis/C/lrs.html), 1996.

[17] A. Bordbar, N. Jamshidi, B. Ø. Palsson, *iAB-RBC-283: A proteomically derived knowledge-base of erythrocyte metabolism that can be used to simulate its physiological and patho-physiological states.* BMC Syst Biology, doi: 10.1186/1752-0509-5-110, 2011.

[18] K. Fukuda, *Polyhedral computation FAQ.* Swiss Federal Institute of Technology, Lausanne and Zurich, Switzerland. Available at `https://www.cs.mcgill.ca/~fukuda/soft/polyfaq/polyfaq.html`

[19] D. Bertsimas, J. N. Tsitsiklis, *Introduction to Linear Optimization.* Published by Athena Scientific, 1997.

[20] J. Nocedal, S. J. Wright, *Numerical Optimization.* Springer, second edition 2006.

[21] A. M. Davie, A. J. Stothers, *Improved bound for complexity of matrix multiplication.* Proceedings of the Royal Society of Edinburgh, **143A** (2), 2013, pp. 351-370, doi: 10.1017/S0308210511001648

[22] R. H. Bartels, G. H. Golub, *The simplex method of linear programming using the LU decomposition.* Communications of the ACM, 12 (1969), pp. 266-268.

[23] S. H. J. Chan, J. Cai, L. Wang, M. N. Simons-Senftle, C. D. Maranas, *Standardizing biomass reactions and ensuring complete mass balance in genome-scale metabolic models.* Bioinformatics Journal, Volume 33, Issue 22: pp. 3603-3609, doi: 10.1093/bioinformatics/btx453, 2017.

[24] A. Schrijver, *Theory of linear and integer programming.* Wiley and Sons, 1998.