universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Accelerating the Enumeration of Elementary Conversion Modes with ECMproject

verfasst von | submitted by

Marcus Holzer BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 862
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Chemie
Betreut von | Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Jürgen Zanghellini

Abstract

In the analysis of metabolic models the enumeration of Elementary Conversion Modes ECMs is a powerful tool, allowing the description of all stoichiometrically possible conversions that a cell can catalyze when transforming nutrients to products. When the goal is to only study the interactions of a cell with its environment, (ECMs) can be used to describe a cell's metabolic interaction with its environment without reference to the intracellular details. This reduction in complexity allows the study of larger metabolic models and therefore contributes to ridging the gap between a cell's genotype and phenotype. However, the calculation of such models is still limited. The newly developed ECMproject was tested against the current standard program ecmtool by comparing the runtimes on increasingly larger models. On the largest model compared, the runtime was reduced from 9 days to only 12 hours with the new program, therefore opening the way for the ECM enumeration on larger metabolic models.

Kurzfassung

Bei der Analyse von metabolischen Modellen ist die Berechnung ihrer "Elementary Conversion Modes" (ECMs) ein wichtiger schritt. ECMs ermöglichen die Beschreibung aller stöchiometrisch möglichen Umwandlungen, die eine Zelle katalysieren kann, wenn sie Nährstoffe in Produkte umwandelt. Wenn das Ziel darin besteht, nur die Wechselwirkungen einer Zelle mit ihrer Umgebung zu untersuchen, können ECMs verwendet werden, um die metabolische Interaktion einer Zelle mit ihrer Umgebung zu beschreiben, ohne auf die intrazellulären Details einzugehen. Diese Verringerung der Komplexität ermöglicht die Analyse größerer metabolischer Modelle und trägt somit dazu bei, die Lücke zwischen dem Genotyp und Phänotyp einer Zelle zu schließen. Allerdings ist die Berechnung solcher Modelle nach wie vor eingeschränkt. Das neu entwickelte Programm ECMproject wurde mit dem bisherigen Standard ecmtool verglichen. Für diesen zweck wurden die Laufzeiten bei immer größerwerdenen Modellen gegenübergestellt. Bei dem größten verglichenen Modell konnte eine Laufzeitreduktion von 9 Tagen auf 12 Stunden erzielt werden und ermöglicht somit die ECM Berechnung größerer Modelle als bisher.

Acknowledgements

First I would like to thank Univ.-Prof. DI Dr. Jürgen Zanghellini, for giving me the opportunity to explore such a unique and fascinating side of chemistry, in the form of metabolic network modelling. I would like to thank Christian Mayer who co-developed the program with me, for his insights and persistence. I would also like the whole Team of Biochemical Network Analysis for welcoming me with open arms and being such a friendly and entertaining group to work with. Lastly I would like to thank my wife, children and the rest of my family for always supporting me throughout my life.

Contents

1. Introduction

The study of an organisms metabolic system is an important undertaking on the path to a better understanding of its molecular mechanisms [12]. Such an understanding can be vital in several scientific fields, from understanding the mechanisms of cells in plants to the effects of drugs on specific cells. As is the case in many scientific fields, the ability to simulate real life systems on a computer has vastly improved the ability to study said systems. The simulation of the metabolic pathways of an organism happens through metabolic network modelling, which allows to represent an organism as a network of nodes and edges. In such a network metabolic genes are correlated with metabolic pathways. With rapidly available genome sequencing the construction of genome-scale metabolic models has increased in recent years [8]. Several databases have been created where metabolic models are available for simulation such as the BIGG database which was published in 2015 by King et al. [11]. Such databases contain genome-scale metabolic models which comprise the entirety of an organisms metabolic pathways. Additionally, specific models exist which focus on particular aspects of a metabolism and do not represent its entire pathways. Without the need for labor intensive cultivation and analysis of cells, the computer simulation of an organisms metabolic system can support traditional studies by saving time and resources. However, genome-scale metabolic models contain thousands of reactions and metabolites as shown in the example of *Escharicia coli* in figure 1.1. The size of such models makes a comprehensive study difficult. Different methods were developed with the goal to analyse metabolic models [18].

One method for the study of metabolic models is Flux Balance Analysis (FBA) which allows the analysis of specific questions such as cell growth trough maximizing the biomass production of an organism [15]. However, this method only offers one optimal solution and therefore gives only a narrow view of a metabolism. In order to get a broader insight into an organisms machinations other methods are required. The calculation of Elementary Flux Modes (EFM) of an organism gives a representation of the whole space of all possible solutions, compared to the single optimal solution of FBA [17] [20]. Calculating the EFMs of a metabolic network reduces the network to the pathways connecting the external reactions with each other. A pathway in such a network is a set of connected reactions which lead from one point in the network to another. EFMs allow the study of the pathways connecting the external reactions of a network, which enables for example the examination of changes in an organisms metabolite output when changes of the input metabolites occur. However, with increasingly larger models the number of EFMs grows, resulting in unfeasible computation times and making EFM analysis viable only in smaller models. Additionally, numerous EFMs show identical input to output conversions which makes their exhaustive enumeration not always necessary [6].

In many cases the interest when studying cells or other small organisms lies in their

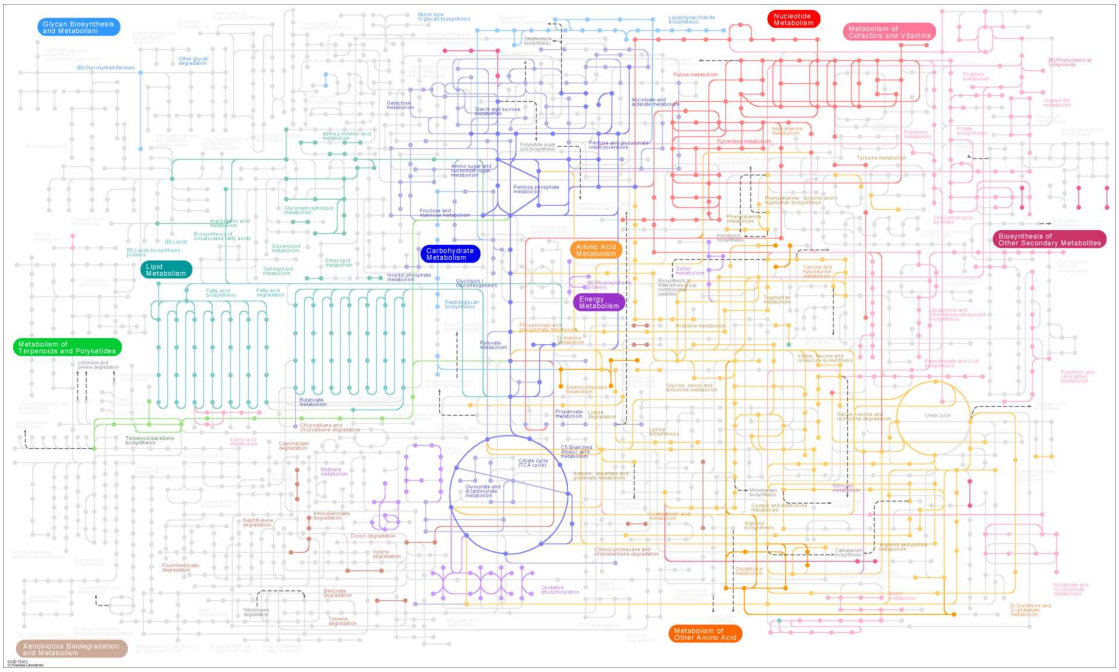


Figure 1.1 – Abstract depiction of the metabolic model of Escherichia Coli from the KEGG pathway database. The KEGG database exists since 1995 and was launched and maintained by Kanehisa Laboratories of the University of Kyoto and the Human Genome Center of the University of Tokio. It can be found under `https://www.genome.jp/kegg/`.

interaction with their surroundings and how they are affected by changes in their environment. In order to study the results of these changes it is often enough to consider only the substrate and products of a cell. This means that instead of looking at the entirety of possible pathways through an organism, it is often enough to consider all feasible overall conversions that the organism is capable of. Elementary Conversion Modes (ECMs) achieve exactly that, they represent a metabolism as the sum of its overall conversions, thereby reducing the complexity of the metabolism from the numerous possible pathways to only the inputs and outputs. Therefore ECMs characterize a cells metabolic interaction with its environment without reference to the intracellular details. This allows ECMs to provide a comprehensive way to study a cells complete metabolic capabilities and contribute to bridging the gap between a cells genotype and phenotype. Early studies showed that the number of ECMs of an organism is vastly lower than the number of EFMs, since ECMs operate in a lower-dimensional space of external metabolite changes rather than reaction rates. This allows ECMs to be computed for much larger models than EFMs. However, even new methods such as ECM decomposition are limited by larger metabolic models as the computation time grows exponentially with the model size.

The current standard for ECM calculation is ecmtool which was introduced in 2021 by Clement et al. [7]. In 2023 Buchner et al. [5] published an updated version of ecmtool

which used the same workflow but introduced parallelization to the calculation of ECMs. These changes vastly improved the calculation times of ECMs. However, even with these improvements the enumeration of large models is still not feasible. In this thesis a further improvement for ECM enumeration will be shown by changing the calculation method while keeping the advantages of parallelization. The resulting tool is called ECMproject and was developed together with Christian Mayer as part of our individual master's theses.

2. Theory

The theory section of this thesis is divided into four main parts. The first part defines Elementary Conversion Modes (ECMs), how they can be described mathematically and gives an example of a toy network for an easier introduction into the subject. After covering the fundamentals, in the second part the current tool of choice for the calculation of ECMs which is called ecmtool will be explained. This part will first go into detail on the first iteration of ecmtool, which took the theory of ECM enumeration and transformed it into a usable program. Afterwards the changes which took place when the second iteration of ecmtool was introduced will be described. In its second iteration mplrs was incorporated into the workflow of ecmtool and made the original version faster and more memory efficient. The last section deals with a new approach for calculating ECMs, which still uses mplrs, but utilizes a different method for the calculation of ECMs which leads to even shorter runtimes and therefore makes the application of ECMs much more tangible.

2.1. Background Information on Polyhedral Cones

Before defining ECMs some concepts from linear algebra will be explained, which will be important for the definition and later enumeration of ECMs. These concepts will be referenced in later sections as they become relevant.

2.1.1. General Information about Polyhedral Cones

The polyhedral cone is a key concept when discussing ECMs. It is constructed from the stoichiometric matrix of a metabolic model. The formation of such a cone will be described in section 2.2.1 where the concept of metabolic networks will be described in more detail. Figure 2.1 shows an example of a polyhedral cone. The blue planes show the half-spaces spanning the sides of the cone. The half-spaces are given by the constraints given to the cone. Where these half-spaces intersect, the edges of the cone are formed which are its extreme rays. Additionally, the intersections also give the vertices of the cone. In a metabolic model, such constraints would be given by the steady-state constraints and the upper and lower bounds of the reaction rates.

2.1.2. Hull and Vertex Representation of a Polyhedral Cone

A polyhedral cone can be described in two ways, a hull representation and a vertex representation. The hull representation, also known as the inequality-, half-space or H-representation defines the polytope as the intersection of a finite number of half-spaces. Each of these half-spaces is defined by a linear inequality. On the other hand, in the

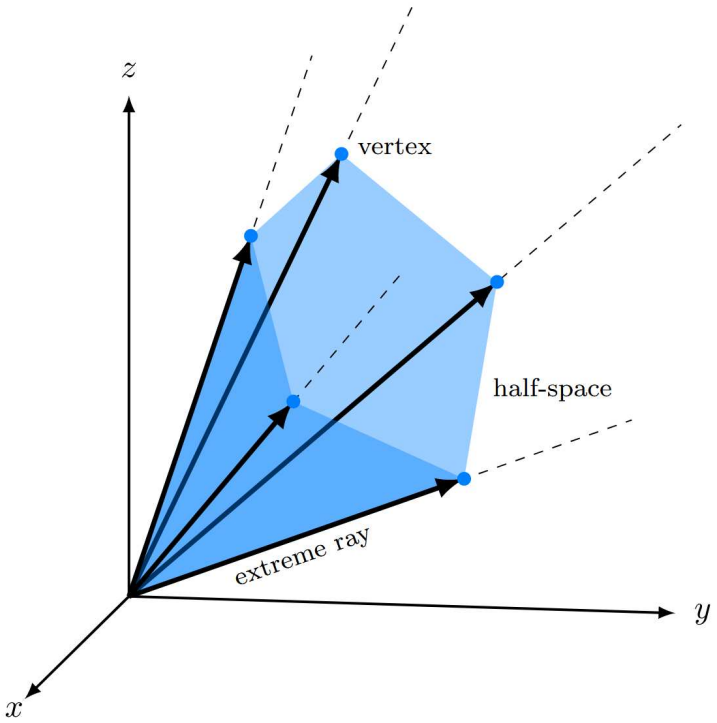


Figure 2.1 – Shows an example of a polyhedral cone. The blue planes depict the half-spaces. The intersections of the half-spaces give the extreme rays and constraints result in the vertices of the polyhedral cone.

generator-, vertex or V-representation a convex polytope is described as the convex hull of a finite set of vertices [4] [10].

An important part when dealing with polyhedral cones is the vertex enumeration problem. This problem stems from computational geometry and optimization and involves finding the vertices and extreme rays of a polyhedron described by a system of linear inequalities. This means the vertex enumeration problem transforms the hull representation of a convex polytope into its vertex representation.

Given an $m \times n$ matrix A and an m dimensional vector b, a convex cone C in its hull representation is defined as:

$$C = \{x \in \mathbb{R}^d \mid b + Ax \geq 0\}, \quad (\text{half} - \text{space representation}). \tag{2.1}$$

A point $x \in C$ is a vertex of C if and only if it is the unique solution to a subset of d or more inequalities from (2.1) solved as equations. Such a subset of inequalities is called a basis. As mentioned above, solving the vertex enumeration problem is to output all vertices of a polytope C, which gives the vertex representation:

$$C = \{x = R\lambda \mid \lambda_i \geq 0\}, \quad (\text{vertex representation}). \tag{2.2}$$

Another way of describing the vertex representation is by way of the Minkowski-Weyl

theorem. The theorem says, that every polyhedral cone can be generated by taking conical combinations of a finite set of vectors, $r_1, \ldots, r_n \in \mathbb{R}^d$. Collecting these vectors as the columns of a matrix R, this gives the vertex representation.

These representations of C are denoted by hull(C) and ver(C).

2.1.3. Dual Cones

One key method to enumerate vertices in a convex polyhedron is the Double Description method. An important quality to understand for this method is the dual of a cone. If C is a polyhedral cone in $\mathbb{R}^d$, then it's dual cone C is defined as:

$$C^* = \{u \in \mathbb{R}^d \mid u \cdot v \geq 0, v \in C\}. \tag{2.3}$$

The dual of the dual of a convex polyhedral cone is equal to itself if C is convex and bounded. For the calculation the important property of dual cones is that the generating set of C, ver(C), forms the hull representation of C^*, hull(C^*), and vice versa, [16]

$$\text{ver}(C) = \text{hull}(C^*), \tag{2.4}$$

$$\text{hull}(C) = \text{ver}(C^*). \tag{2.5}$$

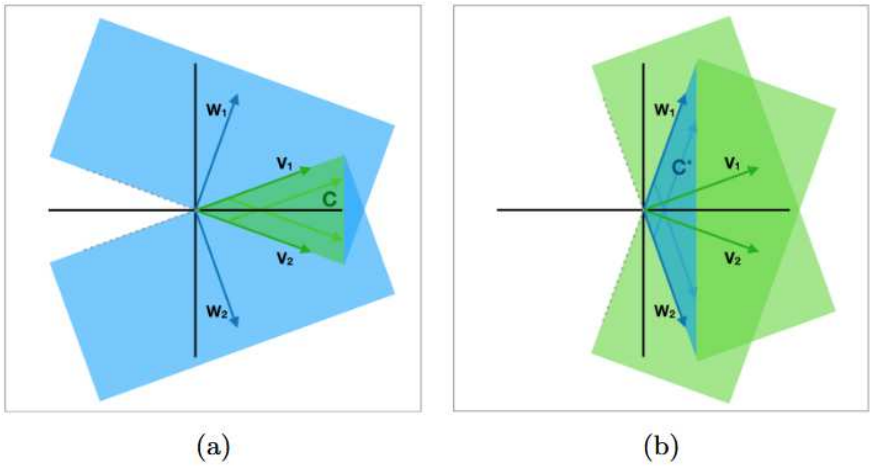


Figure 2.2 – Depiction of the dual cone properties of a polyhedral cone. (a) Shows a 2-dimensional cone C in green which is spanned by the vertices v_1 and v_2. Another way of describing C is via the intersection of the two half-spaces $w_1 \cdot x \geq 0$ and $w_2 \cdot x \geq 0$ shown in blue. Therefore, w_1 and w_2 form the hull representation of C. (b) Shows the dual of C depicted as C^* in form of the blue cone. Here the half-spaces of C become the vertices of C^*. The illustration was taken from the supplementary material of the 2021 paper of Clement et al. [7] in order to better illustrate the properties of the dual of a cone.

Figure 2.2 shows the properties of the dual of a cone in more detail. The 2-dimensional green cone C can be described in two ways. Either by the vertices v_1 and v_2 spanning the cone, or by the two overlapping half-spaces $w_1 \cdot x \geq 0$ and $w_2 \cdot x \geq 0$ shown in blue. These two descriptions of the cone give the vertex- and hull representation respectively.

The dual of the green cone C is the blue cone C^*. In C^* the previous vertices v_1 and v_2 give the half-spaces. This shows the connection between the two representations of a cone and their duals.

2.1.4. Pointedness

In order to describe a pointed polyhedral cone, the lineality space must first be defined. Linealities are vectors $v \in C$ for which it is also true that $-v \in C$. The space that encompasses all such vectors is called the lineality space. This means, that by the definition of C, $Av \geq 0$ and $A(-v) = -Av \geq 0$, which results in the description of the lineality space as the null-space of the constraint matrix that is used in the inequality representation of C, i.e.,

$$\mathrm{Lin}(C) = \{x \in \mathbb{R}^d \mid Ax = 0\}. \tag{2.6}$$

If such a lineality space contains only the zero vector, then a polyhedral cone is defined as pointed. The important characteristic of a pointed cone is that it has a unique vertex representation.

2.2. Defining Elementary Conversion Modes (ECMs)

Before getting into the details of ECMs a general description of metabolic models might be useful. Metabolic networks are a powerful tool for studying small organisms by representing it as a network of reactions and metabolites.

2.2.1. Metabolic Networks

A metabolic network describes the complete set of metabolic processes that occur in a cell. In such a network the nodes represent the metabolites, while the edges represent the reactions taking place inside the cell. An example for a metabolic network is given in figure 2.3 where a toy model is depicted which was used in early development of ECMproject.

In the field of metabolic networks, the stoichiometry matrix denoted as N serves as a useful representation of the different reactions performed by a cell and the metabolites used in the process. An example for such a stoichiometric matrix can be seen in table 2.1 for the toy model from figure 2.3. The columns of the matrix N correspond to the individual reactions r, while the rows represent the metabolites m. In essence, each column of N provides information about the metabolites involved in a particular reaction, including their consumption and production ratios. In order to account for reversibility, reversible reactions are divided into a separate forward and backward reaction, with all stoichiometric coefficients multiplied by -1 or with reactants and products changing their roles, resulting in an irreversible reaction set. In the give example reaction $R11$ is reversible and is therefore split into $R11$ and $R12$.

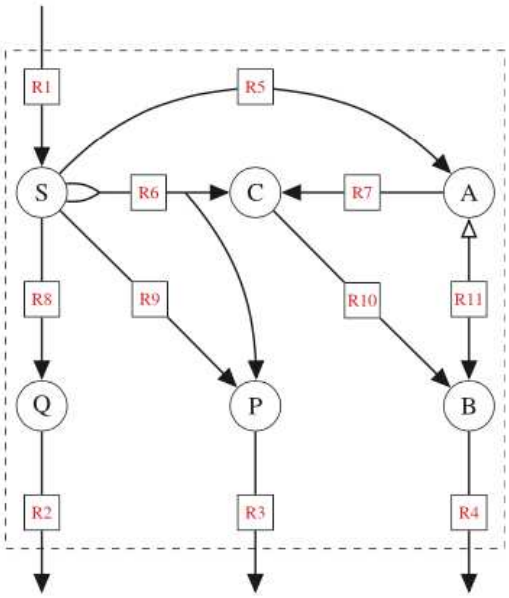


Figure 2.3 – Example of a metabolic network. This toy network was used for initial tests, since due to its small size, results can be checked manually.

	R01	R02	R03	R04	R05	R06	R07	R08	R09	R10	R11	R12
A	0	0	0	0	1	0	-1	0	0	0	-1	1
B	0	0	0	-1	0	0	0	0	0	1	1	-1
C	0	0	0	0	0	1	1	0	0	-1	0	0
P	0	0	-1	0	0	1	0	0	1	0	0	0
Q	0	-1	0	0	0	0	0	1	0	0	0	0
S	1	0	0	0	-1	-2	0	-1	-1	0	0	0

Table 2.1 – Stoichiometric matrix of the toy model, as seen in figure 2.3. The reversible reaction R11 was split into a forward and backward reaction, therefore reaction R12 was added to the stoichiometric matrix.

Metabolites within a system can exist as internal, or external, with internal metabolites specifically identified by the index set Int. The conversion process entails the multiplication of the stoichiometric matrix N by the vector of reaction rates v, resulting in the rates of change for all metabolite concentrations, denoted as $\dot{c} = Nv$. In a steady-state metabolism, where the overall system remains in equilibrium, all internal metabolite concentrations remain constant. Therefore, the condition $c_i = 0$ is true for all i which belong to the index set Int. With all the above information, the space of all steady-state conversions is given by

$$C = \left\{ \dot{c} = Nv \;\middle|\; \dot{c}_i = 0 \text{ if } i \in Int,\ v_j \geq 0 \text{ for all } j \right\}. \tag{2.7}$$

This space is called the conversion cone and describes all metabolic capabilities of a metabolism. In section 2.1 figure 2.1 was shown as an example of such a cone. In a conversion cone constructed from a metabolic model all possible conversions lie on the half-spaces or the hull of this cone. The properties of such a polyhedral cone were discussed in more detail in the previous section and will be referenced as they become relevant for the calculation of ECMs.

2.2.2. ECMs

In 2005, Urbanczik and Wagner introduced the concept of Elementary Conversion Modes (ECMs) as a key concept in the field of metabolic networks [19]. In increasingly larger organisms, analysis via metabolic networks becomes difficult, since due to their growing size, changes in metabolite uptake and output become almost impossible to calculate, therefore making EFM analysis infeasible. FBA still works on larger networks, but gives only a narrow view of a metabolism. ECMs offer a way to reduce the complexity of metabolic models by only characterizing an organisms interaction with its environment. Therefore, a metabolic model is reduced to the uptake and emission of metabolites by an organism.

From Urbanczik and Wagner's introduction of ECMs, a distinct definition of ECMs can be formulated, which was used by Clement et al. in their original presentation of ecmtool in 2021 [7].

Definition: *The set of ECMs is defined as the minimal collection of conversions, such that every steady-state conversion can be expressed as a positive sum of ECMs without the elimination of any external metabolite in that summation.*

To illustrate the key characteristics of ECMs, the toy network shown in Figure 2.4 is employed. This network depicts three external metabolites (A, B, and BM), indicated by underlines, and internal metabolites (C_1 to C_4). The black arrows indicate a stoichiometry of 1, therefore in order to increase BM by 1, 1 C_2 and 1 C_4 are needed ($C_2 + C_4 \rightarrow$ BM). In the toy network, when following along the black arrows, there are several possible pathways for the conversion between the external metabolites. ECMs, shown by the colored arrows, bypass the different internal pathways and connect the external metabolites directly. In a metabolic model, all steady-state conversions collectively form the conversion cone,

which can be visualized as a convex polyhedral cone. The possible conversions within the model reside within the extreme rays of this cone, and each steady-state conversion can be represented as a positive sum of these extreme rays. The extreme rays, denoted by the blue and yellow arrows, serve as ECMs. However, relying solely on the extreme rays as ECMs is not sufficient and fails to fully capture a comprehensive set of minimal building blocks for biologically realistic conversions. ECMs should encompass all conversions without canceling the production of any metabolite.

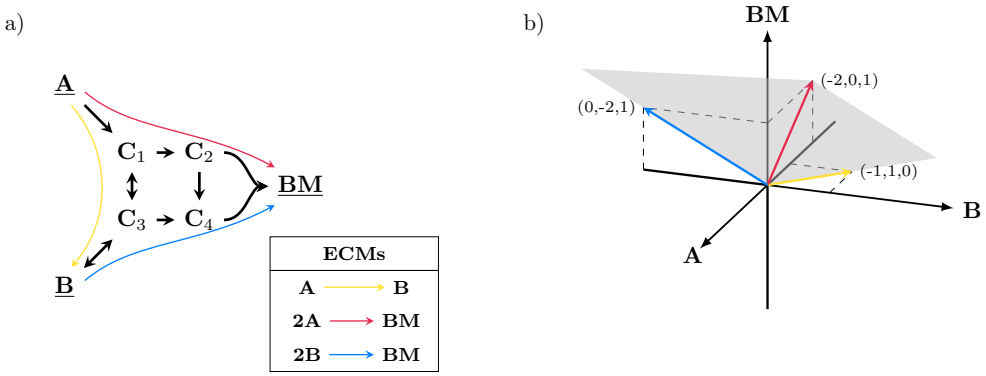


Figure 2.4 – a) Shows a toy metabolic network where the underlined metabolites (A, B and BM) represent the external metabolites. C_1 to C_4 are the internal metabolites. The black arrows indicate the reactions with a stoichiometry of 1, as an example in order to increase BM by 1, 1 C_2 and 1 C_4 are needed ($C_2 + C_4 \rightarrow$ BM). The colored lines show the possible ECMs through the network. b) Shows the steady state conversion cone in grey. The cone is spanned by the blue and yellow ECMs. The red ECM lies inside the cone on the plane where $B = 0$.

In the provided example, to describe the possibility of producing BM from A without simultaneously expressing and consuming B, an additional ECM is required. From figure 2.4 it can be seen that the conversion 2 A / BM (shown in red), can be written as a positive sum of the two other ECMs: 2(A/B) + (2B/BM) = (2A/BM). However, when summing these ECMs, the metabolite B is canceled out. The second definition ensures that this cancellation does not happen and the two ECMs in yellow and blue remain separate and an additional ECM is added to describe the possible combination of the two. Therefore in the end the example system has three ECMs.

In summary, ECMs represent a minimal set of steady-state conversions that comprise the entire range of feasible bio-transformations achievable by an organism. These conversions play a crucial role in helping to understand metabolic processes.

Now, with a full description of ECMs, the next step is to outline how they are calculated.

2.3. Calculation of ECMs with *ecmtool*

Before going into detail about the calculation method used in the initial iteration of ecmtool, it might be useful to reiterate the goal of ECM enumeration. The objective is to use the steady-state conversion cone as shown in equation (2.7), which represents the full

metabolic model and convert it into a state, where the ECMs can be found as a minimal set of generators of this cone.

In 2021 Clement et al. introduced the original version of ecmtool [7]. In this iteration a method for the calculation of ECMs was used which was introduced in 2005 by Urbanczik and Wagner [19]. This method can be summed up as twice solving a vertex enumeration (VE) problem of a convex polyhedron.

As described in the previous section, the convex polyhedron can be represented by the stoichiometric matrix and a vector for the reaction rates which is the H-, half-space- or hull representation of C as was shown in equation (2.1). Solving the vertex enumeration problem gives a list of vertices which is called the V- or vertex representation of C. The vertices of the vertex representation gained from the second vertex enumeration gives the ECMs. The two vertex enumerations are necessary since ecmtool starts with the vertex representation, where it is harder to implement steady-state constraints compared to the hull representation.

2.3.1. The Double Description Method

The Double Description method will be described only shortly since the current version of ecmtool uses a lexicographic reverse search algorithm, but in order to keep continuity ecmtool will be first introduced in its original form and subsequent changes will be presented afterward.

The algorithm for the Double Description method was first introduced by Motzkin et al[14]. for the purpose of translating between the two descriptions of a polyhedron which were introduced in section 2.1.2. A double description pair, or DD pair (hull(C), ver(C)) consists of the hull representation, hull(C) and the vertex representation, ver(C) of the same cone.

The algorithm given by the Double Description method provides a way to find the vertex representation based on a given hull representation. For the Double Description method to work, a proposition must hold true which says: (hull(C), ver(C)) is a DD pair, if and only if (ver(C^*), hull(C^*)) is also a DD pair. The relationship of the DD pairs was discussed in section 2.1.3 in the background information about polyhedral cones. The exact method how these DD pairs are calculated exceeds the scope of this thesis and is not relevant in order to understand the general workflow of ecmtool.

As was described in section 2.1, the cone represented by the inequalities in hull(C) has a hull representation of its dual cone which is ver(C^*). In order to get hull(C) from a starting ver(C), the dual cone of ver(C) is taken, which is hull(C^*) and treated as the hull representation of the cone. From there the double description method gives ver(C^*) which in the end leads to the vertex representation hull(C).

2.3.2. Workflow

The calculation process of ECMs is based on the fact that obtaining a vertex representation of a polyhedral cone is relatively straightforward when a hull representation is known. This task was accomplished using the Double Description method implemented by Polco

[13]. An essential part of the workflow involves establishing a connection between the vertex representation of a cone and the hull representation of its dual which was described in equations (2.4) and (2.5).

The original ecmtool followed the workflow shown below:

$$\mathrm{ver}(C_0) = \mathrm{hull}(C_0^*) \xrightarrow{\text{DD}} \mathrm{ver}(C_0^*) = \mathrm{hull}(C_0) \xrightarrow{\text{SS constraints}} \mathrm{hull}(C) \xrightarrow{\text{DD}} \mathrm{ver}(C) = \text{ECMs}. \tag{2.8}$$

The starting point is the vertex representation of the larger non-steady-state conversion cone,

$$C_0 = \{\dot{c} = Nv \,\big|\, v \geq 0\}, \tag{2.9}$$

which encompasses all conversions, including those that do not satisfy the steady-state constraint. In this state the cone resides in concentration space. In the metabolic network, all reactions are assumed to be irreversible, due to being split into forward and backward reactions.

To briefly summarize the workflow shown in 2.8, the starting point as mentioned is a vertex representation of a non-steady-state conversion cone $\mathrm{ver}(C_0)$. From there the hull representation of the dual cone $\mathrm{hull}(C_0^*)$ is taken and the Double Description method (DD) is applied which gives $\mathrm{ver}(C_0^*)$ and therefore $\mathrm{hull}(C_0^*)$. Next the internal metabolites are removed and steady state (SS) constraints are added, therefore giving $\mathrm{hull}(C)$. Finally, performing the Double Description method (DD) for a second time gives $\mathrm{ver}(C)$ which corresponds to the ECMs. In the first part where the polyhedral cone is in a non-steady-state it resides in the concentration space. After the steady-state constraints are applied the concentration cone is converted to the flux cone.

In the following sections the individual steps of the workflow will be described independently.

$\mathbf{ver(C_0) \rightarrow hull(C_0^*)}$ - preprocessing

With the columns of the stoichiometric matrix from a metabolic network a vertex representation of the non-steady-state conversion cone can be formed (equation (2.9)). As described in section 2.1.3, the vertex representation gives the hull representation of its dual cone.

$\mathbf{hull(C_0^*) \xrightarrow{DD} ver(C_0^*)}$ - first vertex enumeration

The next step is the first iteration of the Double Description method. After applying the method, a vertex representation of the non-steady-state conversion cone (C_0^*) is obtained. This vertex representation describes a unique cone.

$\mathbf{ver(C_0^*) = hull(C_0) \xrightarrow{SS\ constraints} hull(C)}$ - intermediate processing

The first part of this step is to get the hull representation of C_0. Collecting all these inequalities in a matrix H yields:

$$C_0 = \{c \mid Hc \geq 0\}. \tag{2.10}$$

From here on the steady-state constraints are applied. To incorporate the steady-state constraints effectively, two inequality constraints are added per internal metabolite, ensuring $\dot{c}_i = 0$ for $i \in Int$. Since the internal metabolites are not of interest they are dropped and only the external metabolites remain. At this point reversible reactions are split and redundant inequalities removed.

$\mathbf{hull}(C) \xrightarrow{\mathbf{DD}} \mathbf{ver}(C) = \mathbf{ECMs}$ - second vertex enumeration and postprocessing

The final step, begins with the obtained hull representation of the steady-state conversion cone. Since all reversible metabolites have been split in the previous step, the cone is confined to one orthant, implying its pointedness. Consequently, the Double Description method can be directly applied to identify a minimal generating set of the steady-state conversion cone which gives the ECMs.

2.4. An Alternative Path with *mplrs*

In its second iteration, introduced by Buchner et al. in 2023 [5], ecmtool replaced the Double Description method with a lexicographic reverse search (LRS) algorithm, which was implemented by using mplrs. In 2018 Avis and Jordan[3] published mplrs which offers a parallel implementation of the vertex enumeration algorithm LRS. Even though a new method was used for solving the vertex enumeration problem, the rest of the workflow of ecmtool remained the same. Consequently, LRS is another method to achieve the same goal as with the Double Description method. Using mplrs offers an advantage in faster calculation speed and lower memory usage since mplrs is able to make use of parallelization. As mentioned in the previous section, the goal of the vertex enumeration problem is to get all vertices of a convex polyhedron or polytope and therefore also of a polyhedral cone. The following sections will describe how mplrs works in detail, since ECMproject also uses mplrs in its workflow and relies on its parallel processing capabilities for a fast computation of the vertex enumeration problem.

2.4.1. Lexicographic Reverse Search - LRS

As stated before, the main function of LRS is finding the vertex representation of a polyhedron described by its hull representation. The way LRS achieves this is by reversing the simplex method [1] [2]. The simplex algorithm comes from linear programming and is used to traverse the edges of a polyhedron defined by linear inequalities. The algorithm begins at a starting vertex and travels along the edges of the polyhedron until the vertex of the optimal solution is reached. Reverse search traces all possible paths taken by the simplex algorithm in reverse and finds their starting points. Another way often used to describe reverse search is as the traversal of a spanning tree, denoted the reverse search tree T, of a graph $G = (V, E)$, with V being the vertices and E being the edges. During

the search, only edges that belong to the reverse search tree are traversed. The nodes of this graph are the objects which are generated during the search. Since each node can only be reached by a single path, there is no need to store additional nodes other than the current one.

The reverse search is defined by two functions, a finite local search f and an adjacency oracle *adj*. The local search function f generates a path in G from $v \in V$ to a target vertex v^* through repeated application. Together, the paths generated in by this function give the reverse search tree T. The adjacency oracle holds information about the edges connecting the individual vertices.

LRS has two key advantages that make it suitable for efficient parallel processing. Firstly, it is unnecessary to keep more than one node of the tree in memory at any given time, and there is no need for a database to keep track of visited nodes. Secondly, the enumeration process can be restarted from any specific node within the tree by only referring to the details of that individual node. This is in contrast to traditional depth-first search algorithms, where restarting requires a comprehensive database of visited nodes as well as a backtrack stack leading back to the root of the search tree.

For LRS to work, certain properties of a graph G must be fulfilled. G must be connected and the upper bound of the maximum vertex degree Δ must be known. Additionally, the adjacency oracle must be able to generate the adjacent vertices of a given vertex v sequentially and without repetition, which is achieved by function $\mathrm{Adj}(v, j)$, where v is a vertex of G and $j = 1, 2, ..., \Delta$. The values of this function are either a vertex adjacent to v, where each vertex adjacent to v appears only once since j ranges over its possible values, or null. The local search function $f(v)$ returns the tuple (u, j) for every $v \neq v^*$. Here the vertex $v = \mathrm{Adj}(u, j)$ in a way so that u is a parent of v in T. Algorithm 1 shows the pseudo code for a generic reverse search.

Dictionaries are used for the application of reverse search to the vertex enumeration problem. The starting point is the inequality or H representation which forms a convex polyhedron from an $m \times n$ dimensional matrix $A = (a_{ij})$. Equation (2.1) showed this representation in an earlier section. In order to create a dictionary for the inequality representation, one new non-negative variable, called slack variable, is added for each inequality:

$$x_{d+i} = b_i + \sum_{j=1}^{d} a_{ij} x_j, \quad x_{d+i} \geq 0 \quad i = 1, 2, ..., m. \tag{2.11}$$

The original variables are called decision variables. For any vertex to exist, the polyhedron must have $m \geq d$, which allows solving the equations for various sets of variables on the left hand side. The left hand side of such a dictionary is called basic and contains the slack variables. The right hand side is called non-basic, or co-basic and contains the decision variables. The notation $B = \{i : x_i \text{ is basic}\}$ and $N = \{j : x_j \text{ is co-basic}\}$ is used to describe the two sides. This N differs from that used for the

Algorithm 1: Generic Reverse Search

```
procedure RS(v*, Δ, Adj, f)
    v ← v*   j ← 0
    repeat
        while j < Δ do
            j ← j + 1
            if f(Adj(v, j)) = v then
                v ← Adj(v, j)
                j ← 0
                output v
            end
        end
        if v ≠ v* then
            (v, j) ← f(v)
        end
    until v = v* and j = Δ
end procedure
```

description of the matrices in previous sections, but will be used nonetheless, since this seems to be the standard notation used when describing the LRS algorithm.

A pivot interchange is performed, where one index from B is swapped with one from N. Afterwards, the equations for the new slack variables are solved. A basic solution from a dictionary can be obtained by setting $x_j = 0$ for all $j \in N$. A basic feasible solution (BFS) is found if $x_j \geq 0$ for every slack variable x_j. If a dictionary has a slack variable $x_j = 0$ it is called degenerate.

In the next step, the graph $G = (V, E)$ is defined, where each node in V corresponds to a BFS and is labelled with a cobasic set N. The edges in E each correspond to a pivot between two BFSs. With this graph the adjacency oracle is defined with B and N being index sets for the current directory. For $i \in B$ and $j \; in N$

$$\mathrm{Adj}(N, i, j) = \begin{cases} N \backslash \{j\} \cup \{i\} & \text{if this gives a feasible dictionary} \\ \emptyset & \text{otherwise.} \end{cases} \tag{2.12}$$

In order to find the target v^* for the reverse search, a linear program is solved over this dictionary with any objective function. Then, a new objective function is chosen so that the optimum dictionary is unique and represents v^*. For the selection of the variable which enters the basis, lrs uses Bland's least subscript rule and a lexicographic ratio text to select the leaving variable. The lexicographic rule simulates a simple polytope which greatly reduces the number of bases to be considered. Reverse search is initiated from the unique optimum dictionary.

2.4.2. Parallelization

One of the strengths of mplrs is the ability to make use of parallelization for solving the vertex enumeration problem. Algorithm 1 showed the pseudo code of the generic reverse search. In order to achieve parallelization of this algorithm, the lack of memory property is used, which allows the algorithm to be restarted at any node in the reverse search tree T. After a restart, all remaining nodes of T will be generated. This feature of the base algorithm is adapted to allow for a subtree to be enumerated from its given root. With this adaptation four additional parameters must be supplied when calling the reverse search procedure:

- The start vertex is vertex from which the reverse search should be initiated and replaces v^*
- depth is initially the depth in T of start vertex and will be updated to be the depth in T of the vertex v currently being considered in the search
- max depth is the depth at which forward steps are terminated
- min depth is the depth at which backtrack steps are terminated

These modifications can be seen in Algorithm 2 where the pseudo code for the extended reverse search is shown.

Algorithm 2: Extended Reverse Search

```
procedure RS2(start_vertex, Δ, Adj, f, depth, max_depth, min_depth)
    v ← start_vertex   j ← 0
    repeat
        while j < Δand depth < max_depth do
            j ← j + 1
            if f(Adj(v, j)) = v then
                v ← Adj(v, j)
                j ← 0
                depth ← depth − 1
                output v
            end
        end
        if depth > 0 then
            (v, j) ← f(v)
            depth ← depth + 1
        end
    until depth = min_depth and j = Δ
end procedure
```

The changes from Algorithm 1 to Algorithm 2 are only simple modifications, but they result in an extension of the functionality of the original Algorithm.

The extended reverse search algorithm allows for parallelization without further modification. There are three phases to the parallel process. First, the reverse search tree T is generated to a fixed depth *initdepth*. The nodes of the tree are stored in a list L. This list is then used in the second phase, where threads are scheduled in parallel from L using the subtree enumeration feature. This feature requires the parameter max threads which gives the maximum number of parallel threads which can run at the same time. Additionally, the output stream will be controlled. While in Phase 1, the output stream will be directed to list L. From there, all vertices that have depth less than init depth are then removed from L and output. In Phase 2 threads are scheduled from the nodes in L up to the number previously specified by max threads. Each thread then uses lrs to enumerate the subtree, which is rooted at the node that was removed from L for the enumeration. When L becomes empty the algorithm moves on to Phase 3, where the threads are terminated one by one until there are no more running and the procedure terminates. The output from the threads is concatenated and put into a single output stream. Algorithm 3 outlines the parallel procedure. The pseudo code shows, that the only interaction between the parallel threads is the common output collection process, and the only signalling which is required is when a thread terminates.

Algorithm 3: Parallel Reverse Search - PRS

```
procedure PRS(start_vertex, Δ, Adj, f, init_depth, max_threads)
    num_threads ← 0
    redirect output to a list L ;                                  // Phase 1
    RS2(stat_vertex, Δ, Adj, f, 0, init_depth, 0)
    redirect output to collection process
    remove all v ∈ L with depth(v) < init_depth and output(v)
    while num_threads < max_threads and L ≠ ∅ do
        remove any v ∈ L                                           // Phase 2
        RS2(v, Δ, Adj, f, depth(v), ∞, depth(v))
        num_threads ← num_threads + 1
    end
    while num_threads > 0 do
        wait until a termination signal is received
        if L≠ ∅ then
            remove any v ∈ L
            RS2(v, Δ, Adj, f, depth(v), ∞, depth(v))
        else
            num_threads ← num_threads − 1 ;                        // Phase 3
        end
    end
end procedure
```

The different Phases of Algorithm 3 have varying amounts of parallelization. Phase

1 has no parallelization, while in Phase 2 all designated cores are used, and in Phase 3 the amount of parallelization decreases as the individual threads terminate. Comparing the additional work required in contrast to Algorithm 1, it can be seen that most of it is due to restarting the reverse search process. Determining the value for the init_depth parameter can have diverse results. A high value results in only on thread working for an extended period. Additionally, the list L tends to be larger, which leads to more restarting overhead. However, a larger value for init_depth also results in less time spent in Phase 3. This effect of init_depth means, that the efficiency of the parallelization relies heavily on the structure of the tree T. In the worst-case scenario, the tree is a linear path, and Phase 2 cannot be parallelized effectively. The best-case scenario is a balanced tree T, resulting in a shorter list L, reduced overhead, and all threads finishing around the same time.

2.4.3. Problems with ecmtool

The key step in obtaining ECMs is performing a vertex enumeration on the hull representation of the reduced steady-state conversion cone. In order to achieve this ecmtool starts with the vertex representation in a non-steady-state in concentration space, then performs a preliminary vertex enumeration and only then adds the steady-state constraints to the resulting hull representation. This step converts the cone from concentration space to flux space. Additionally, the removal of the internal metabolites happens in the intermediate step between the vertex enumerations. In the end the complex computational task of performing the vertex enumeration has to be performed twice.

In the following section a new method will be introduced, which allows to start directly from the steady-state hull representation in flux space. With this change only a single vertex enumeration needs to be performed, which should allow for faster computation times.

2.5. ECMproject

Our new approach for ECM calculation uses mplrs for the calculation, but does so with a different method which changes the workflow. As was mentioned in the beginning of the previous section about the second iteration of ecmtool, the overall workflow of both versions of ecmtool stayed the same. With the introduction of ECMproject the workflow of the calculation changes, which can be seen in figure 2.5. The changes in the workflow stem mainly from the integration of the steady-state constrains at the initiation of the conversion cone. Therefore, ECMproject starts in flux space. As was shown in section 2.3.2 ecmtool starts with the vertex representation of a non-steady-state conversion cone in concentration space. The steady state constraints are only added after the first vertex enumeration which transforms the cone from concentration to flux space. In contrast ECMproject starts with the hull representation of a conversion cone with the steady state constraints already applied, which changes the whole workflow since the cone is in flux space from the beginning. After the initiation ECMproject performs its name

giving projection step, which reduces the dimension of the conversion cone until only external interactions remain. This is accomplished by projecting all the conversion cone onto its exchange reactions. Conveniently, mplrs possesses a function which allows for the projection to be performed. With the reduced conversion cone only a single vertex enumeration has to be performed in order to gain the ECMs. The hope of this approach is that the projection step is computationally easy and should be very fast. After the projection, the vertex enumeration needs to be performed only once and for a matrix with reduced complexity.

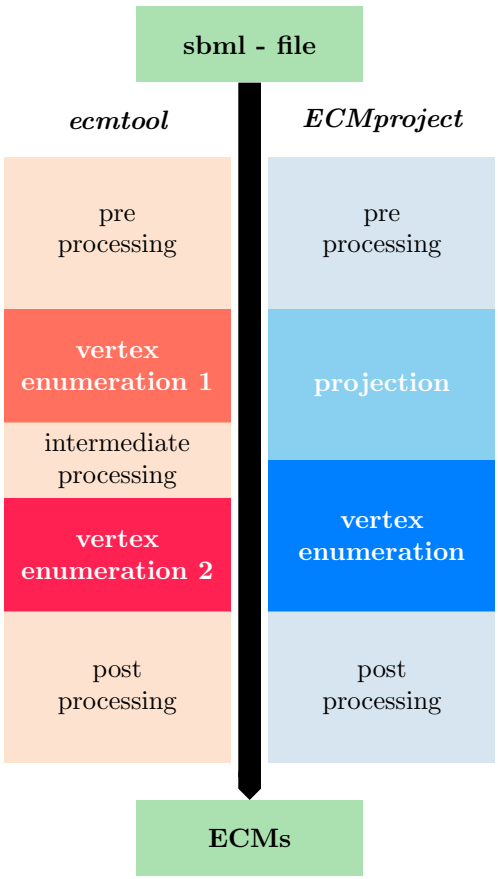


Figure 2.5 – Workflow comparison between ecmtool and ECMproject. It shows one step less for ECMproject, since there is no more need for intermediate processing. ECMproject has only one vertex enumeration, the first enumeration is replaced by the projection step.

The following sections will go into more detail on the individual steps of the workflow of ECMproject and how the program works in general.

2.5.1. Preprocessing

Metabolic networks are usually stored as sbml-files which stands for Systems Biology Markup Language [9]. This format is based on the xml format and is specifically tailored

for storing computational models of biological processes. The information from the sbml-file is imported into Python using the cobra package. With the information transformed into Python objects a function is applied in order to identify external reactions in the metabolic network which are than tagged with an exchange true or false attribute. In the sbml-file different criteria can be used to identify external reactions:

- SBO-Term:000062 which is a code identifying exchange reactions.
- Reaction IDs containing R_EX or EX.
- A one-sided reaction not functioning as a sink or demand reaction.

Therefore giving all exchange reactions a uniform tag makes it easy to target those reactions.

In order to obtain all ECMs the convex polyhedron has to reside on a single orthant, otherwise there would be multiple solutions to the vertex enumeration problem. This is accomplished by splitting all reversible exchange reactions into two irreversible exchange reactions, one forwards and one backwards.

Creating the H-Representation

In order to start the calculations the hull or H-representation of the metabolic network needs to be derived from the sbml-file. As was explained in section 2.1.2 the hull representation can be described as the inequalities of an $m \times n$ matrix A and an m dimensional vector b. These inequalities can be written as:

$$Ax \geq b \tag{2.13}$$

$$-b + Ax \geq 0 \tag{2.14}$$

Deriving the matrix A from the metabolic network, each row encompasses one reaction with the columns representing the metabolites. Therefore the elements of the matrix give information about which metabolites and in which amounts they are used for each reaction. The vector x of equations 2.13 and 2.14 describes the reaction rates corresponding to each row of the matrix and has therefore dimension m. Lastly, vector b holds the constraint variables.

For mplrs to perform the projection step, which will be described in the next part, an input file containing the H-representation must be created. Figure 2.2 shows such an H representation for the toy model seen in figure 2.3. The matrix here is comprised of three parts, the left most column which represents the b-vector which holds the steady-state constraints and irreversible reactions. As mentioned before the b-vector is set to 0 in a steady state environment. The external reversible reactions of the metabolic model are split during preprocessing and therefore the stoichiometric matrix in the upper part, donated s-matrix, becomes N with irreversible reactions. In this example there are no

reversible external reactions, therefore the splitting of reactions is skipped. In the end, the irreversible, steady state environment equation 2.13 becomes

$$Nx = 0. \tag{2.15}$$

Lastly the bottom part of the H-representation holds the irreversibility or unity matrix I_{Irr}. This H-representation is passed to mplrs for the projection in the next step.

H - Representation

0	0	0	0	0	1	0	-1	0	0	0	-1	
0	0	0	0	-1	0	0	0	0	0	1	1	s - matrix
0	0	0	0	0	0	1	1	0	0	-1	0	
0	0	0	-1	0	0	1	0	0	1	0	0	
0	0	-1	0	0	0	0	0	1	0	0	0	
0	1	0	0	0	-1	-2	0	-1	-1	0	0	
0	1	0	0	0	0	0	0	0	0	0	0	
0	0	1	0	0	0	0	0	0	0	0	0	
0	0	0	1	0	0	0	0	0	0	0	0	
0	0	0	0	1	0	0	0	0	0	0	0	
0	0	0	0	0	1	0	0	0	0	0	0	unity matrix
0	0	0	0	0	0	1	0	0	0	0	0	
0	0	0	0	0	0	0	1	0	0	0	0	
0	0	0	0	0	0	0	0	1	0	0	0	
0	0	0	0	0	0	0	0	0	1	0	0	
0	0	0	0	0	0	0	0	0	0	1	0	
0	0	0	0	0	0	0	0	0	0	0	1	

Table 2.2 – Stoichiometric matrix of the toy model, as seen in figure 2.3. The left most column represents the b-vector, the upper matrix is the stoichiometric matrix, donated s-matrix for short and the lower matrix is the irreversibility or unity matrix.

2.5.2. Projection

With the exchange reactions identified and the H-representation created the projection can be performed. In this phase of the workflow the conversion cone is projected onto the exchange reactions using the Fourier-Motzkin elimination method. With this step the high dimensional conversion cone of the whole metabolic network is projected onto a lower dimensional space consisting only of the exchange reactions and their corresponding metabolites.

Fourier-Motzkin Elimination

The Fourier-Motzkin elimination is a classical algorithm for the elimination of variables from systems of linear inequalities, therefore projecting a high dimensional polytope onto

a lower dimensional space. Given a system of linear inequalities

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_{ij}, \tag{2.16}$$

where $a_{ij}, b_i \in \mathbb{R}$ for $i = 1, ..., m$ and $j = 1, ..., n$. In order to eliminate x_k for some $k \in \{1, ..., n\}$ the following steps are performed:

1. For each $j \in \{1, ..., m\}$,
   - if $a_{jk} > 0$, then the jth inequality is multiplied by $\frac{1}{a_{jk}}$
   - if $a_{jk} < 0$, then the jth inequality is multiplied by $-\frac{1}{a_{jk}}$
2. A new system of inequalities is formed by,
   - copying down all inequalities in which the coefficient of x_k is 0, and
   - obtaining new inequalities by adding together each inequality in which x_k has a positive coefficient and each inequality in which x_k has a negative coefficient.

The projection step of the workflow eliminates one variable at a time after which the initial matrix N is reduced to N_{ex} consisting only of the exchange reactions and their corresponding metabolites. After each projection mplrs automatically removes all redundancies in order to speed up calculations. With the reduced matrix an input file containing the reduced H-representation is created which is then used for the vertex enumeration.

2.5.3. Vertex Enumeration

Since ECMproject already started with the steady-state constraints applied from the beginning there is no need for any intermediate processing of the data and the vertex enumeration can be started immediately after the last projection step. The theory of the vertex enumeration and the effect that mplrs has on it was described in the previous sections and will therefor not be reiterated at this point. Computationally there is no difference in the execution of the vertex enumeration of ecmtool and in ECMproject. The difference is that after performing the projection only one vertex enumeration has to be performed. At the end of this step, the H-representation of the conversion cone is turned into the V-representation.

2.5.4. Postprocessing

After the big calculation steps the reversible reactions which were split in the first step of the workflow need to be combined again. After that, the ECMs are calculated from the v-representation of the previous step and written into a result file. With bigger metabolic

networks the number of ECMs can be substantial and therefore those result files can become very big. In order to further reduce the running time of the program a parallel processing option was included for the post processing step using the multiprocessing package of python. The parallel post processing can be optionally turned of if desired. When parallel post processing is enabled, the matrix composed of the V-representation of the conversion cone is split into different chunks which are then processed separately by each individual core. In order to get the ECMs the V-representation is multiplied with the transposed, reduced stoichiometric matrix N^T, which only holds the external metabolites and external reactions.

$$\mathrm{ver}(\boldsymbol{C}) \cdot N^T = \mathrm{ECMs} \tag{2.17}$$

After the final calculation the ECMs are output into a result file.

3. Materials and Method

In the scope of this thesis two tools for the enumeration of ECMs were compared to each other. The currently established ecmtool which was originally developed by Clement et al. in 2021 and subsequently improved by Buchner et al. in 2023. The new tool is called ECMproject and was developed as part of this thesis. The testing and data collection was done on the Jucuda Server which is part of the network cluster of the University of Vienna.

Software of the Jucuda Server
Debian GNU/Linux 11
Linux 5.10.0-16-amd64 (Kernel)
Bash version 5.1.4(1)-release
ecmtool (Git version from 13.07.2023)
time-1.9 (gnutime)
mpirun (Open MPI) 4.1.0
mplrs:lrslib v.7.2 2022.3.6(hybrid arithmetic)

Table 3.1 – Software installed on the Jucuda Server and the versions used during development and data collection

Conda Environments

packages	tool	
	ECMproject	ecmtool
cbmpy	-	0.8.4
conda	4.12.0	4.12.0
cobra	0.26.3	-
cython	-	0.29.33
gmpy2	-	2.1.5
lrslib	70.a	70.a
matplotlib	3.7.2	-
numpy	1.23.5	1.24.2
pandas	1.5.3	-
pip	23.1.2	23.1.2
psutil	5.9.5	5.9.4
python	3.11.4	3.11.3
scipy	-	1.10.1
sympy	1.12	1.11.1

Table 3.2 – Software packages used on the individual conda environments during the calculations.

4. Results and Discussion

The goal of ECMproject is to provide a faster method for ECM calculation. For the purpose of testing and comparing the runtime as well as the memory consumption during use, a synthetic minimal cell was used and scaled to different complexity levels. With increasing complexities of the test model the behavior of ECMproject on increasingly bigger models was monitored. Additionally, the same test model and complexity levels were used on meassurements of ecmtool, in order to accurately compare the old method with the new one applied in ECMproject. With all data points taken, the two programs and their differing methods were compared with each other.

4.1. Development

As mentioned in the introduction, ECMproject was developed together with Christian Mayer as part of our individual master's theses. During early development each of us tried their own approach when creating the program. Results from our different versions were compared and individual parts identified which gave faster times. Further along the two versions were adapted until a single version of ECMproject was established. This was the single core version of ECMproject. With this joined version first test results in the comparison with ecmtool were gained. However a potential for saving runtime in the parallelization of the postprocessing step was seen. Therefore different approaches for the implementation of parallel processing were tried. Together, a working parallel post processing step was integrated and the, at the time of writing this thesis, final version of ECMproject was finished.

4.2. Test Model

In order to test the ECMproject and how it scales with an increasingly complex metabolic model, the synthetic minimal cell JCVI-syn3A was chosen. JCVI-syn3A was created by Breuer et al. in 2019 and contains 155 genes, 304 metabolites and 338 reactions of which 94 are external. The complexity of the model can be adapted by changing the number of excess nutrients of the minimal medium which changes the number of internal and external reactions. The complexity levels for the test performed in this thesis range from 0 to 22, where complexity 0 has only 253 of the total 338 reactions and complexity 22 has 316. For the calculations a model of each level of complexity was created, ranging from "mmsyn_sm00" to "mmsyn_sm22". These models are shown in table 4.1 and were provided by Buchner et al. (2023).

In table 4.1 the number of reactions and metabolites, as well as how many of them are external are shown for each level of complexity. Additionally, the last column shows the resulting number of ECMs when using ECMproject on the respective model. For each level of complexity two to three reactions and one to two metabolites were added to synthetic cell JCVI-syn3A. Of the added reactions and metabolites, always one reaction and one metabolite was external. With each increase in complexity, the combinatorial possibilities of conversions in the model increase which is reflected in the drastic increase in ECMs. Since the large number of ECMs is also reflected in the file size, a zip option was included in ECMproject, which writes the results directly into a zip file, therefore decreasing the file size 32-fold. However, enabling this option also increases the postprocessing time 18-fold, since parallel postprocessing is not possible with this option enabled which will be shown in section 4.5, where all the results of the parallel postprocessing step are depicted.

model	complexity	Reactions		Metabolites		ECMs
		total	external	total	external	
$e_c oli_c ore$	-	95	20	72	20	689
mmsyn_sm00	0	253	56	245	56	4132
mmsyn_sm01	1	255	57	246	57	6486
mmsyn_sm02	2	257	58	247	58	9702
mmsyn_sm03	3	260	59	249	59	19394
mmsyn_sm04	4	263	60	251	60	38778
mmsyn_sm05	5	266	61	253	61	77546
mmsyn_sm06	6	269	62	255	62	155082
mmsyn_sm07	7	272	63	257	63	310154
mmsyn_sm08	8	275	64	259	64	620298
mmsyn_sm09	9	278	65	261	65	1240586
mmsyn_sm10	10	281	66	263	66	2481162
mmsyn_sm11	11	284	67	265	67	4962314
mmsyn_sm12	12	287	68	267	68	9924618
mmsyn_sm13	13	290	69	269	69	17756170
mmsyn_sm14	14	293	70	271	70	35512330
mmsyn_sm15	15	296	71	273	71	71024650
mmsyn_sm16	16	299	72	275	72	142049290
mmsyn_sm17	17	302	73	277	73	284098570
mmsyn_sm18	18	305	74	279	74	514523146
mmsyn_sm19	19	308	75	281	75	869854722
mmsyn_sm20	20	311	76	283	76	1403221121
mmsyn_sm21	21	314	77	285	77	2821027740
mmsyn_sm22	22	316	78	286	78	4212839045

Table 4.1 – Models used for the tests of ECMproject. The first model, "$e_c oli_c ore$" *wasusedforinitialtestsandvalidationduetoitssmallersize.Theincreasinglylargermodelsnamed*" $mmsyn_s mXX$" *wereus*

4.3. Performance Meassurements

The performance of ECMproject and ecmtool was measured by monitoring the wall clock time as well as the RSS usage during each run. RSS is the resident set size and describes the portion of memory in a computer which is occupied by a process and held in physical RAM (random access memory). For each data point three runs were performed and the average of those data points was taken. The wall clock time was measured by utilizing the python package *time*, which monitors the execution time of the script. In order to measure the memory consumed by the program, the RSS usage was measured by using *gnutime*.

4.4. Validation

For the purpose of validating the accuracy of ECMproject, a comparison of the calculated ECMs was performed between ECMproject and ecmtool. for this purpose the "e_coli_core" model was used. The ECMs calculated with both programs were compared by using a python script. The comparison of the results showed that ECMproject reliably produces the same results as ecmtool.

4.5. ECMproject Parallel Postprocessing

During the development of ECMproject, the main method for lowering the overall time for the calculation of ECMs was by implementing the new method which uses the projection step. During testing the influence of the individual workflow steps were measured. As can be seen in figure 4.1, at low complexity levels the projection step is the strongest contributing step to the overall calculation time. However, at higher levels of complexity, the influence of the individual steps shifts. The preprocessing and projection step increase only marginally with higher complexity levels, while the vertex enumeration and the postprocessing become the dominant factors of the overall processing time. At complexity level 12 the vertex enumeration becomes a higher contributor than the projection and at complexity 16 the postprocessing becomes the second highest contributor.

The fact that the post processing is the second highest contributor to the overall time sparked the idea during development to implement parallel processing into the last step of the workflow. As mentioned in section 2.5.4 about the postprocessing in the workflow, this parallelization is achieved by using the multiprocessing package of python. The initial multi core version used the *"Pool()"* function to manage the parallel processes which resulted in a drastic increase of memory consumption compared to the single core version. This was caused by the last process sometimes finishing before earlier ones and therefore clogging up the RAM of the system. The final iteration of ECMproject uses a sleep function to wait for all ECMs to finish which solved the memory issue. Figure 4.2 shows the comparison of the total runtime between the single and multi core setting of ECMproject, as well as the memory usage of both versions. It can be seen, that the overall runtime is lower in all model sizes when using parallel postprocessing. Additionally, the

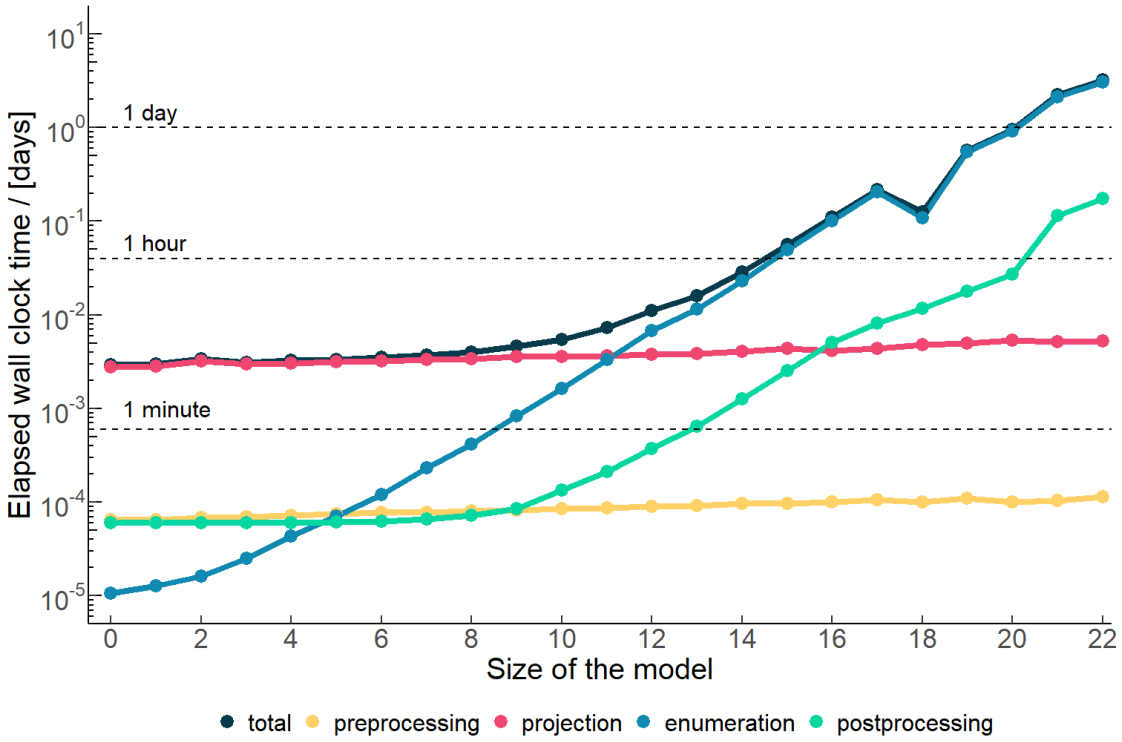
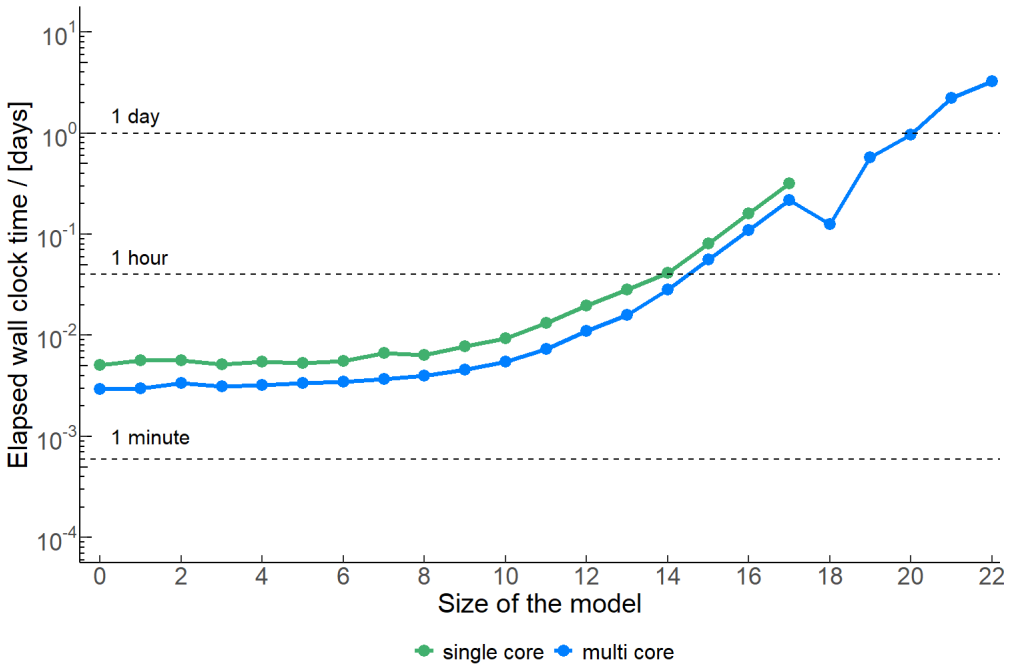


Figure 4.1 – Influence of the different workflow steps on the overall time over an increasingly complex system. The preprocessing and projection steps of the workflow increase only marginally with a higher complexity of the medium, while the vertex enumeration and postprocessing increase considerably.

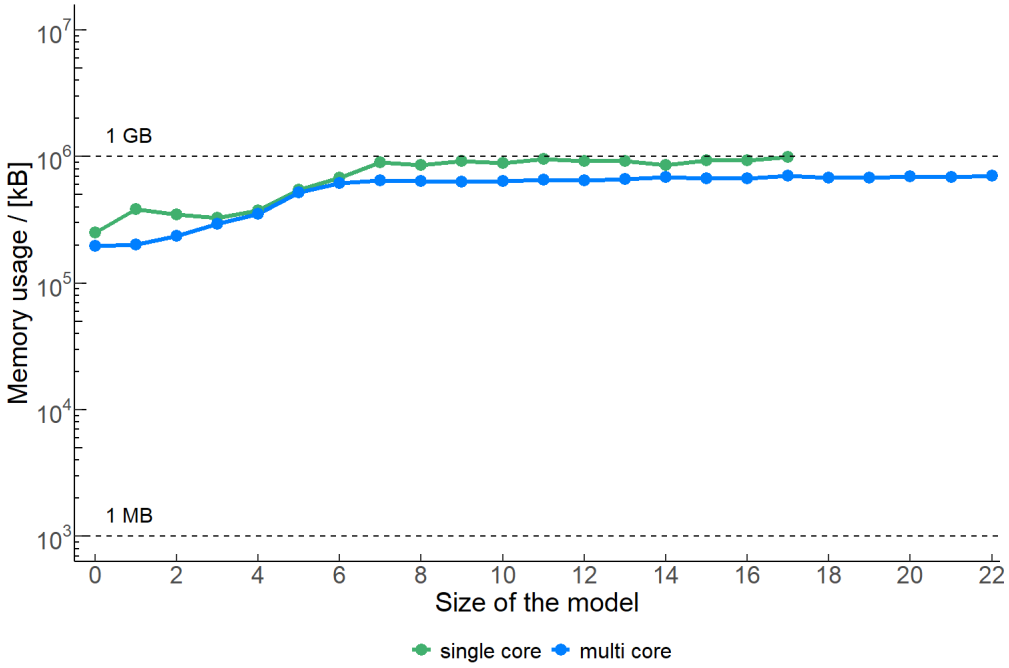
memory usage is slightly lower with the multi core option, although both variants do not exceed 1 GB of memory usage with the tested models.

In figure 4.3 the difference of the postprocessing time is shown separately. It illustrates the increasing gap between the single and multi core setting of ECMproject, resulting in larger time saves on bigger metabolic systems. On the other hand on very small models the single core setting is faster, but calculation times in such small models are rather short anyway.

All further results shown in the next sections were gained from using ECMproject in the multi core mode for postprocessing.



(a) Total runtime



(b) Memory usage

Figure 4.2 – (a) Comparison of the total runtime of ECMproject in the single core and multi core configuration. As expected the multi core mode shown in blue is slightly faster than single core. (b) Comparing the memory usage of the single and multi core mode shows a slightly higher RAM usage from the single core mode.

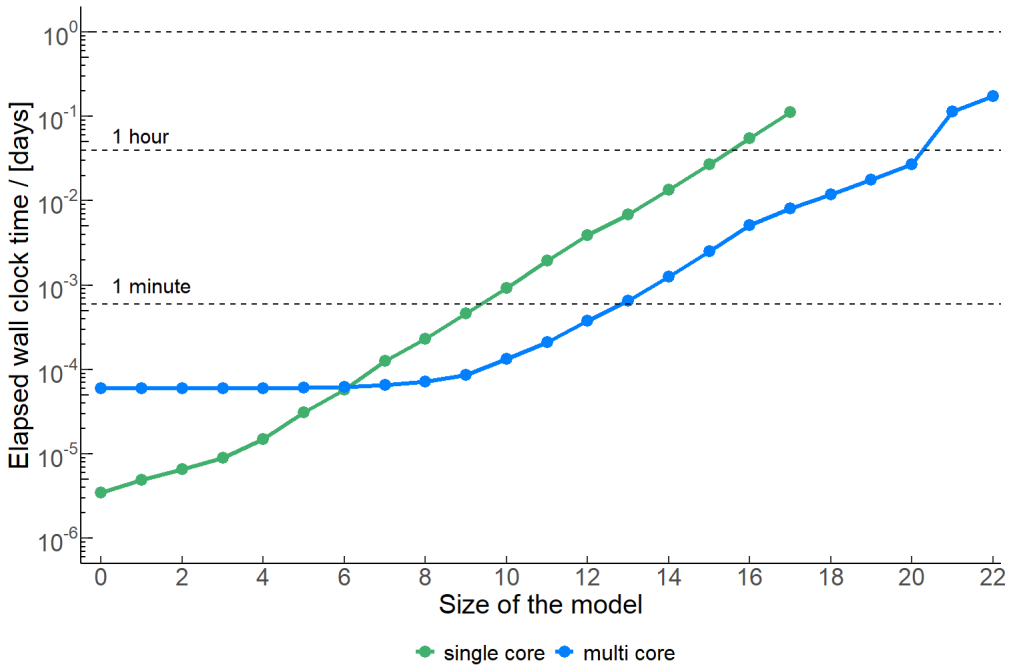
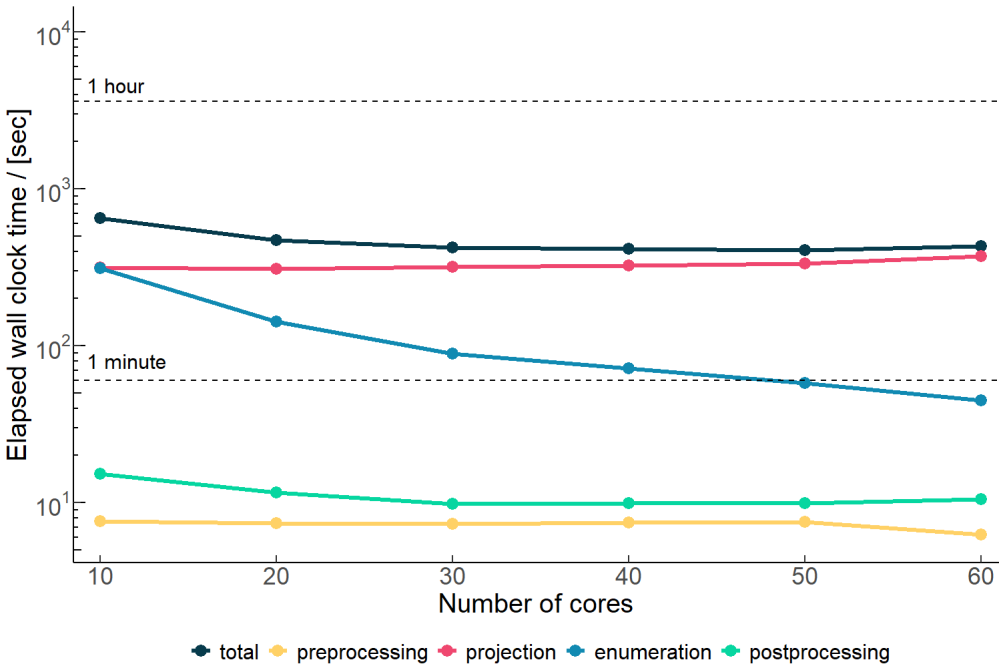


Figure 4.3 – Runtimes of the postprocessing step of ECMproject in single core (green) and multi core (blue) mode. In this step the parallel postprocessing takes effect. It shows for very small models the single core postprocessing is actually faster. However, the advantage of parallel postprocessing becomes apparent at model sizes bigger than 7 on the given scale.

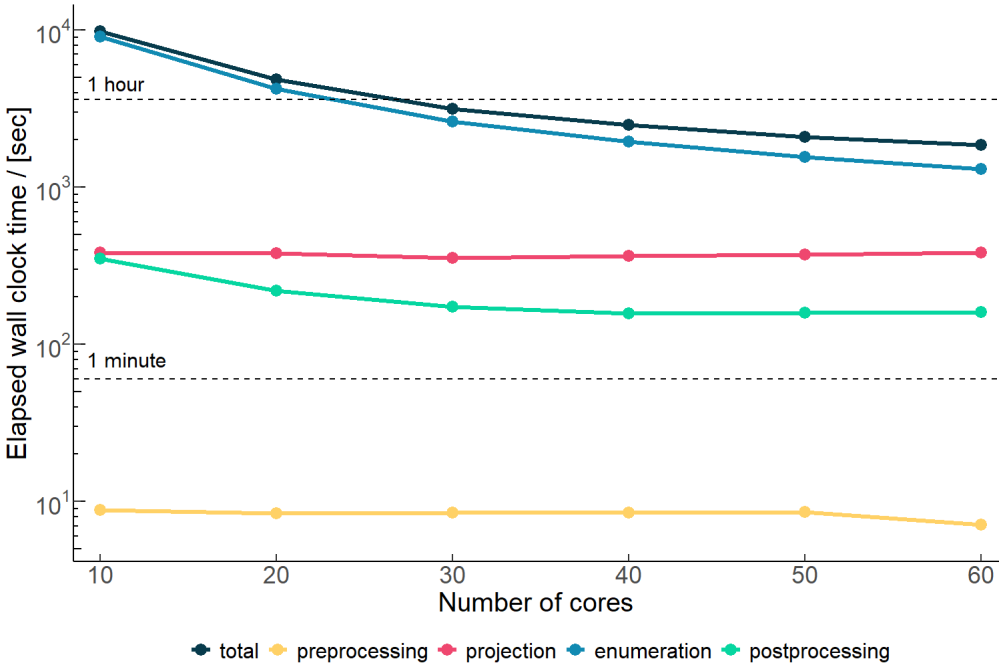
4.6. Number of Cores and Individual Steps

When using ECMproject, the number of cores used by the CPUs can be set by the user. The influence of the cores were studied by performing consecutive runs of the same model using a different number of cores for each run. These runs were performed on the models: *mmsyn_sm00*, *mmsyn_sm05*, *mmsyn_sm10* and *mmsyn_sm15*. In figure 4.4 the results of *mmsyn_sm10* and *mmsyn_sm15* are depicted for ECMproject since they showed the most change. The preprocessing and projection steps show no significant change in processing time with an increase in the number of cores used. This was expected, since these two steps make no use of parallel processing. The vertex enumeration and postprocessing grow in performance when increasing the number of processes. For models in this size range a core number of 30 is sufficiently fast, but with bigger models a larger number of cores is needed to gain results in a managable time.

In figure 4.5 the effect of the number of cores used is shown for the individual workflow steps of ecmtool for the models *mmsyn_sm10* and *mmsyn_sm15*. It can be seen that the parallelization of mplrs is used in the first and second vertex enumeration, since their respective runtimes decrease with a higher number of cores. It is also noticeable that all other steps make no use of parallel processing. As with ECMproject, in ecmtool the second largest contributor to the running time is the postprocessing.

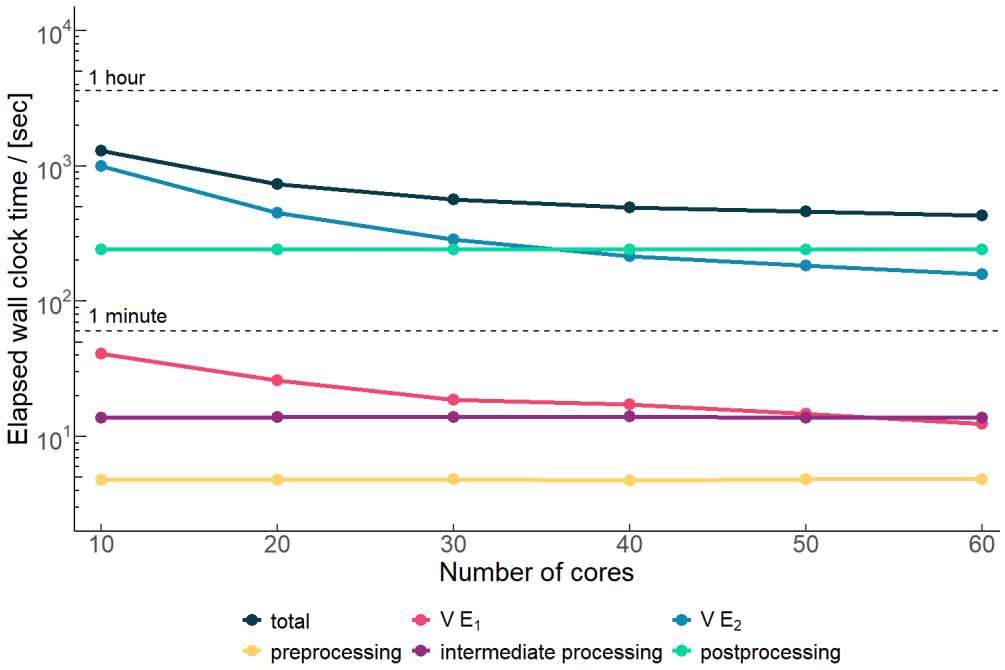


(a) mmsyn_sym10

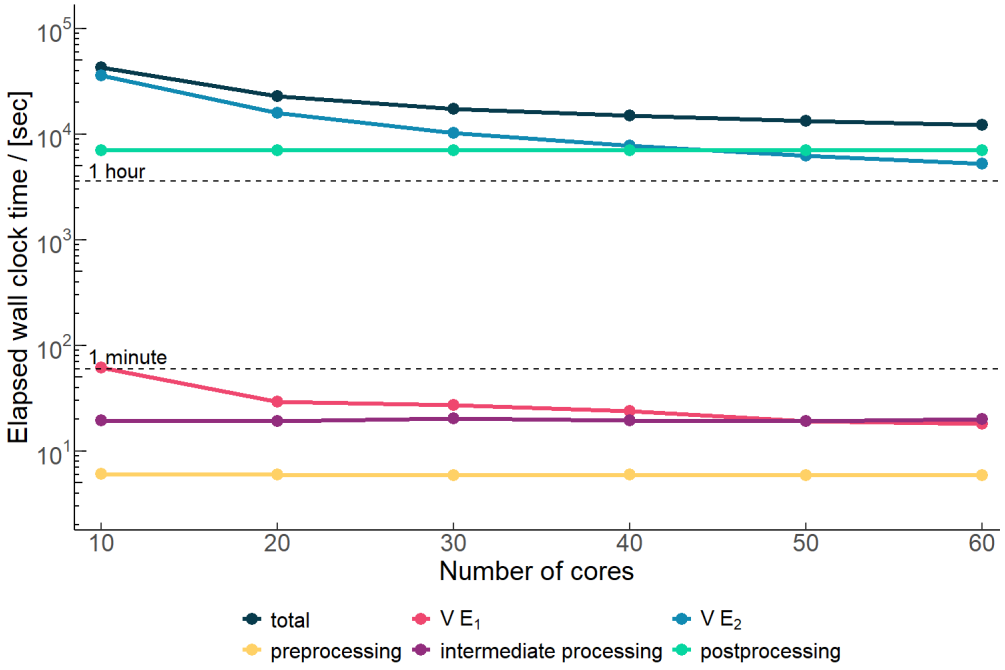


(b) mmsyn_sm15

Figure 4.4 – Comparison of the influence of the number of cores on the different workflow steps of ECMproject for *mmsyn_sm10* (a) and *mmsyn_sm15* (b). In these levels of complexity of the model the projection and vertex enumeration are shown as the strongest contributors to the overall time. The number of cores mainly influences the enumeration and postprocessing steps and the influence becomes stronger with an increasing complexity of the model.



(a) mmsyn_sym10



(b) mmsyn_sym15

Figure 4.5 – Comparison of the influence of the number of cores on the different workflow steps of ecmtool for *mmsyn_sm10* (a) and *mmsyn_sm15* (b). The largest contributors to the overall time are the second vertex enumeration and the postprocessing. The number of cores mainly change the effectiveness of the first and second vertex enumeration.

4.7. *ecmtool* vs ECMproject

For the final comparison of ecmtool and ECMproject all models from *mmsyn_sm00* to *mmsyn_sm22* were used for ECMproject. For ecmtool the tests were stopped at *mmsyn_sm19* since the running times were getting to long. Figure 4.6 illustrates that the goal to create a faster way to calculate ECMs was achieved. On smaller models in this case with a complexity of 9 or lower, ecmtool is faster than ECMproject. However, the running time of ECMproject in these cases lies still in the minutes and in most events a larger metabolic model is of interest. The advantaged of ecmtool with smaller models would only be useful in a situation where a lot of small models would need to be calculated in quick succession. With increasing complexity of the model the difference in running time becomes more apparent and the gap increases. For *mmsyn_sm19* the calculation of ecmtool took over 9, almost 10 days, while ECMproject needed only around 13 hours for the same calculation.

The RAM usage for both models is roughly the same, lying beneath 1 GB and keeping stable with increasing complexity as can be seen in figure 4.7.

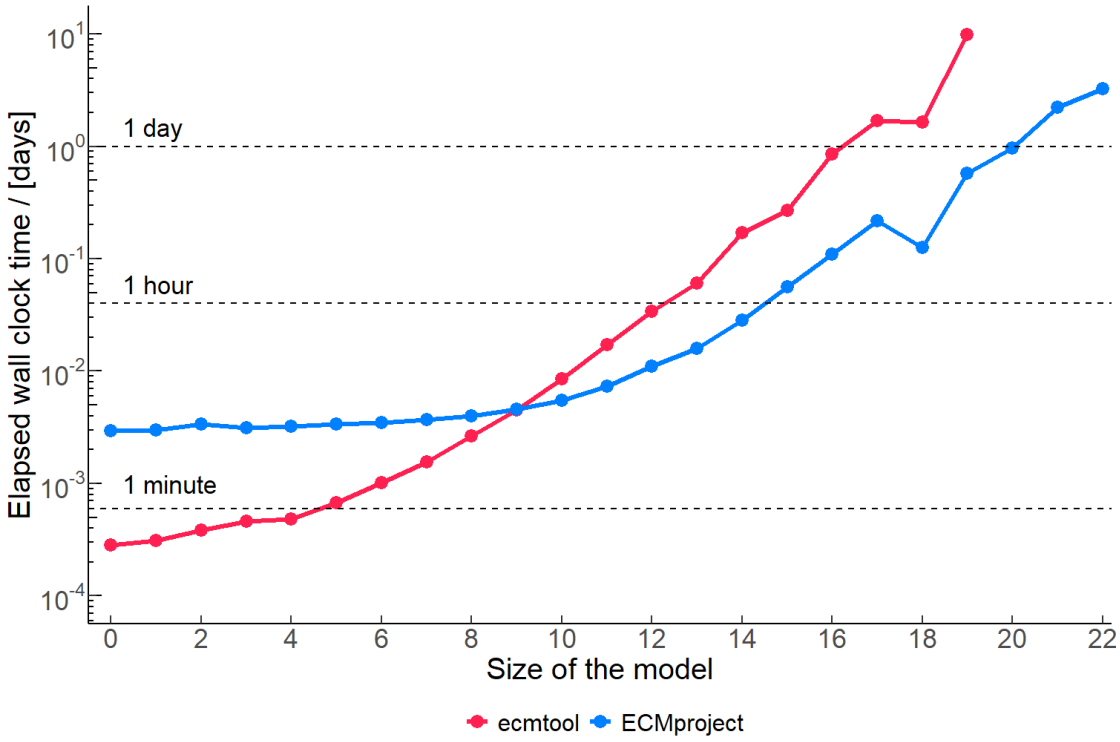


Figure 4.6 – Comparison of the running time of ecmtool and ECMproject for the different levels of complexity of JCVI-syn3A. ECMproject shows a faster running time on complexity levels 9 and higher, with increasing time savings on bigger models.

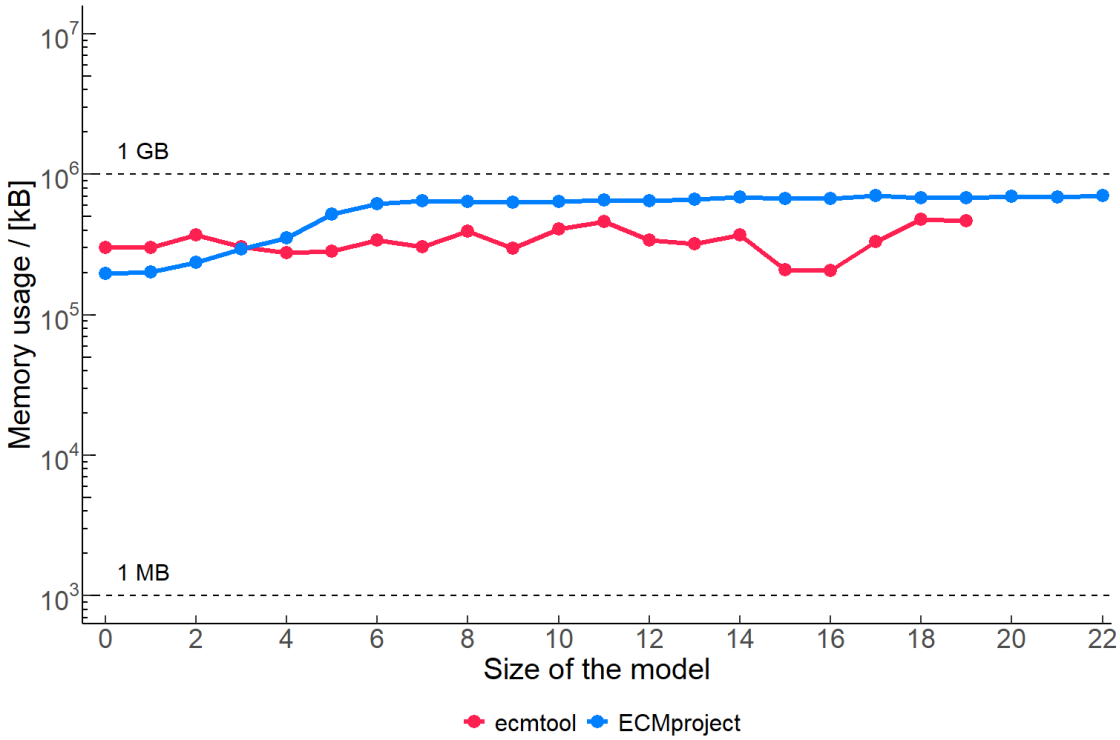
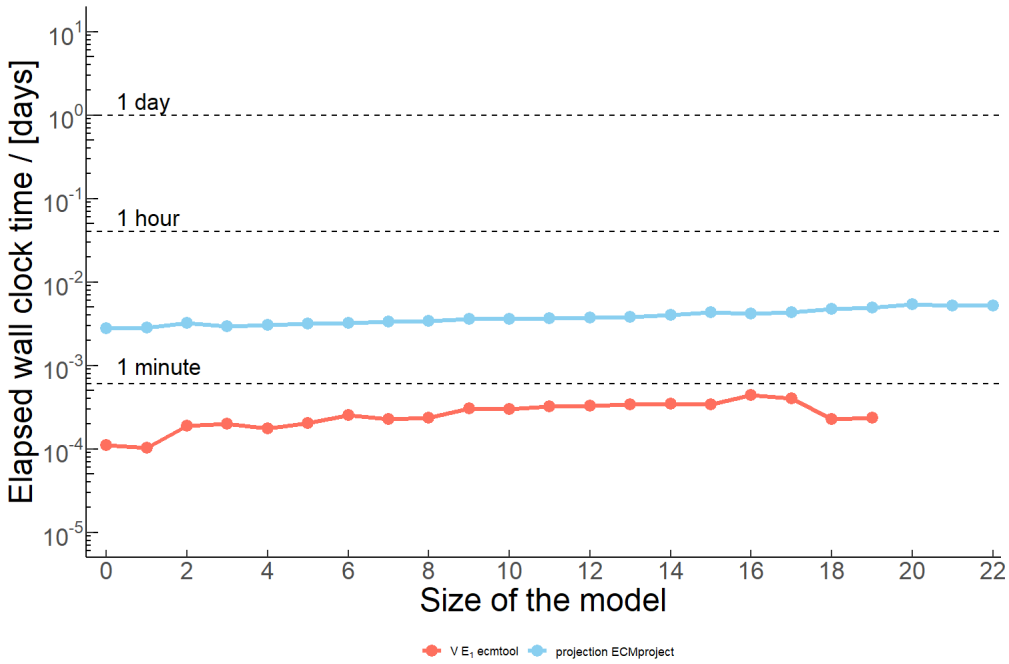


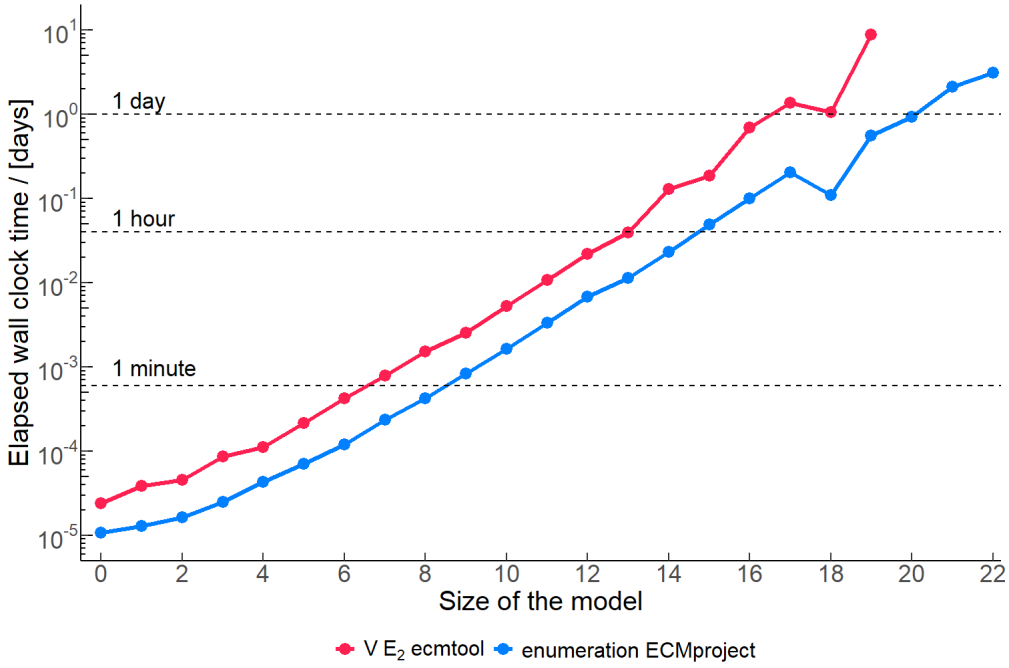
Figure 4.7 – Comparison of the memory usage of ecmtool and ECMproject for the different levels of complexity of JCVI-syn3A. The memory usage of both programs stays nearly constant at just under 1 GB of RAM usage.

4.7.1. Calculation Steps of *ecmtool* and ECMproject

Taking a closer look at the individual calculation steps in figure 4.8 gives more insight into the differences of the methods. The fact that ECMproject is slower for smaller models stems from the characteristic that the first vertex enumeration of ecmtool is faster than the projection step of ECMproject. The first calculation step of ecmtool takes under one minute even for bigger models, while the time for the projection ranges from four minutes to seven minutes. When looking at the second vertex enumeration of ecmtool compared to the single vertex enumeration of ECMproject, the time saving potential of the new method becomes apparent. The second calculation step of ECMproject is faster than the second vertex enumeration of ecmtool. The progression of the curves in figure 4.6 can be seen from the two separate calculation steps. At lower complexity models, the first calculation step has a stronger effect on the overall runtime. As the complexity increases, the influence of the second calculation step increases and becomes dominant. This results in ecmtool being faster for small models while ECMproject becomes faster in larger models. The slow speed of the projection step comes from mplrs not being able to perform the projection as a parallel process and each projection has to be performed individually.



(a) First calculation step



(b) Second calculation step

Figure 4.8 – Comparison of the running times of the first vertex enumeration of ecmtool and the projection of ECMproject (a), as well as the second vertex enumeration of ecmtool and the vertex enumeration of ECMproject (b). The first step (a) remains relatively constant ecen on bigger models with ecmtool showing faster running times. The running time of the second step (b) increases with bigger models. Here ECMproject has the faster times.

4.7.2. Number of Cores and Overall Runtime

In figure 4.9 a comparison of the influence that the number of used cores has on the overall runtime can be seen for the model *"mmsyn_sm15"*. When looking at the behavior of the two programs, ECMproject shows a better scalability and therefore a stronger reduction in runtime with more cores used. When comparing the scalability of individual steps as seen before in figures 4.4 and 4.5 it can be observed that the vertex enumeration of ECMproject shows a better scalability compared to the second vertex enumeration of ecmtool. Another influence on the performance improvement with a higher core count is the implementation of the parallel postprocessing, since at this level of model complexity it is the second largest contributor to overall runtime as was seen in figure 4.4. For ecmtool the total runtime from 10 cores to 60 cores is reduced by a factor of 3.5, while the runtime reduction of ECMproject lies at a factor of 5.4. This results in ECMproject benefiting more from larger computer clusters for its calculations.

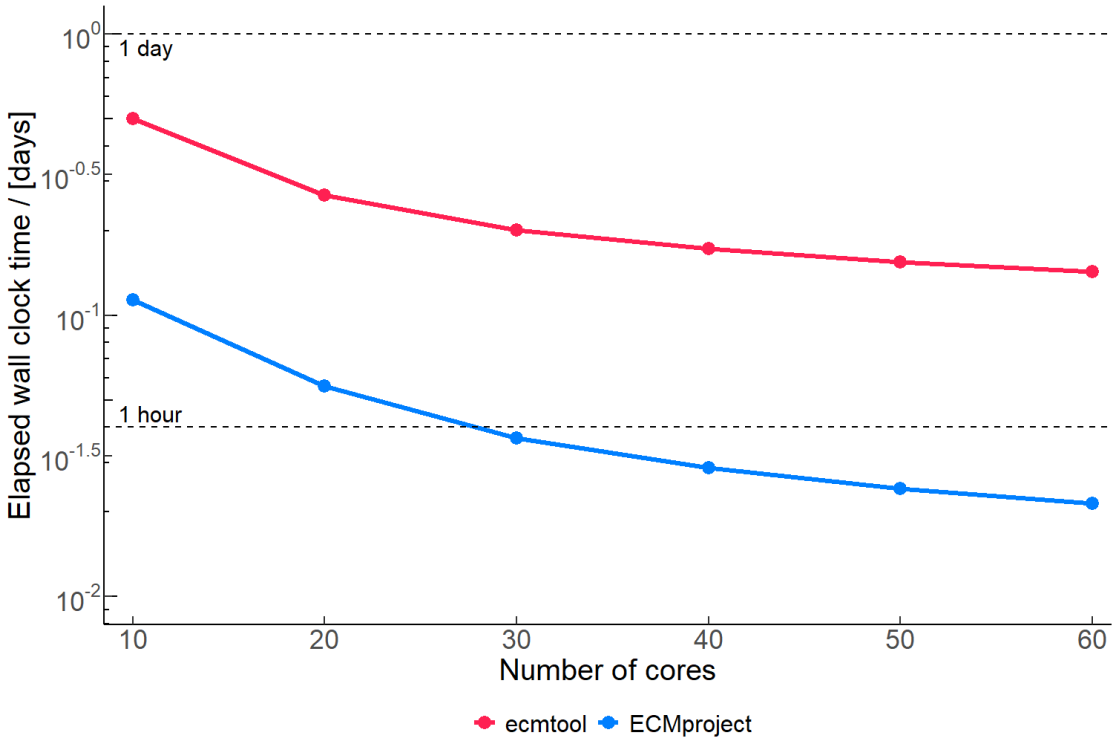


Figure 4.9 – Comparison of the influence that the number of used cores has on the total runtime of ecmtool and ECMproject on the model mmsyn_sm15. ECMproject shows a stronger reduction in running time with higher core numbers.

5. Conclusion

In the scope of this master thesis, a new method for the calculation of ECMs was developed and deployed in the program ECMproject. This new method uses the *projection* function of *mplrs* to achieve faster calculation times then the previous method deployed in ecmtool, which relied on twice solving the vertex enumeration problem. With ECMproject the first vertex enumeration is replaced by a projection step and only one vertex enumeration needs to be performed. These changes result in a significant decrease in runtime for the program, which allows for the enumeration of ECMs on much larger models than before in reasonable times. This allows for more possibilities when studying metabolic models and their interactions with each other.

During development and for gathering the final data during testing, the synthetic minimal cell JCVI-syn3A was used. Modified versions of this cell were assigned complexity levels ranging from 0 to 22, where 0 had 56 external metabolites and 22 had 78. With this synthetic cell the behavior of ECMproject over increasingly bigger models was monitored. Additionally, the tests for the comparison of runtime between ECMproject and ecmtool were performed using the differently scaled versions of JCVI-syn3A. The resulting ECMs were compared between ecmtool and ECMproject to validate the results of ECMproject.

In the course of development a potential in further reducing the running time of ECMproject was seen in the postprocessing step. Since this step is the second longest on larger models a parallel processing function was implemented.

Data was collected on the running time and memory usage of ECMproject for the different complexity levels of JCVI-syn3A. This data was compared to the performance of ecmtool on the same model. For both ECMproject and ecmtool the memory usage remained stable and below 1 GB throughout the tests. Comparing the overall runtime shows that for small models ecmtool remains faster, however on larger models ECMproject outperforms ecmtool. As an example, on complexity level 19 the ECM calculation for JCVI-syn3A takes over 9 days when using ecmtool, while it takes only 13 hours with ECMproject. Additionally to the overall runtime, the runtime of individual steps was compared. This comparison showed that the first vertex enumeration of ecmtool is faster than the projection step of ECMproject and both steps increase only marginally in runtime even on larger models. On the other hand, the second vertex enumeration of ecmtool is slower than the vertex enumeration of ECMproject. Since this step becomes the main contributor to the overall runtime the bigger the model gets, it makes ECMproject overall the faster program. Additionally to the runtime, the scalability was examined, by comparing the performance over different core numbers when using the two programs. Here ECMproject showed a better scalability through a larger reduction in runtime between using 10 cores and 60 cores compared to ecmtool. This improved scalability stems from the more efficient scaling of the vertex enumeration of ECMproject compared

to the second vertex enumeration of ecmtool. Another factor contributing is the parallel postprocessing which makes the last step of the workflow scalable with higher core usage.

Overall the improved running time of ECMproject over ecmtool allows for the study of larger scale models than was previously possible. However, when looking at databases for metabolic models shows that even the smaller genome-scale metabolic networks contain over 500 metabolites and reactions. Therefore, the sheer size of a single-cell organism demands further improvements in the field of metabolic pathway enumeration. One possible improvement would be the parallelization of the projection step. However the biggest contributor to the overall time still remains in the vertex enumeration. Advancements through improved algorithms or other ways for a more efficient solution to the vertex enumeration problem would provide the highest benefits for the goal of calculating ECMs of full single cell organisms.

Bibliography

[1] David Avis and Komei Fukuda. "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra". In: *Proceedings of the seventh annual symposium on Computational geometry*. 1991, pp. 98–104.

[2] David Avis and Komei Fukuda. "Reverse search for enumeration". In: *Discrete applied mathematics* 65.1-3 (1996), pp. 21–46.

[3] David Avis and Charles Jordan. "mplrs: A scalable parallel vertex/facet enumeration code". In: *Mathematical Programming Computation* 10.2 (2018), pp. 267–302.

[4] Arne Brondsted. *An introduction to convex polytopes*. Vol. 90. Springer Science & Business Media, 2012.

[5] Bianca Buchner et al. "ecmtool: fast and memory-efficient enumeration of elementary conversion modes". In: *Bioinformatics* 39.3 (Feb. 2023), btad095. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btad095. eprint: https://academic.oup.com/bioinformatics/article-pdf/39/3/btad095/49407400/btad095.pdf. URL: https://doi.org/10.1093/bioinformatics/btad095.

[6] Bianca A Buchner and Jürgen Zanghellini. "EFMlrs: a Python package for elementary flux mode enumeration via lexicographic reverse search". In: *BMC bioinformatics* 22 (2021), pp. 1–21.

[7] Tom J. Clement et al. "Unlocking Elementary Conversion Modes: ecmtool Unveils All Capabilities of Metabolic Networks". In: *Patterns* 2.1 (2021), p. 100177. ISSN: 2666-3899. DOI: https://doi.org/10.1016/j.patter.2020.100177. URL: https://www.sciencedirect.com/science/article/pii/S2666389920302415.

[8] Oliver Fiehn. "Combining genomics, metabolome analysis, and biochemical modelling to understand metabolic networks". In: *Comparative and functional genomics* 2.3 (2001), pp. 155–168.

[9] M. Hucka et al. "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models". In: *Bioinformatics* 19.4 (Mar. 2003), pp. 524–531. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btg015. eprint: https://academic.oup.com/bioinformatics/article-pdf/19/4/524/48903880/bioinformatics_19_4_524.pdf. URL: https://doi.org/10.1093/bioinformatics/btg015.

[10] Fukuda K. *Frequently asked questions in polyhedral computation*. https://www.cs.mcgill.ca/~fukuda/soft/polyfaq/polyfaq.html. [Online; accessed 2023 07 23]. 2004.

[11] Zachary A. King et al. "BiGG Models: A platform for integrating, standardizing and sharing genome-scale models". In: *Nucleic Acids Research* 44.D1 (Oct. 2015), pp. D515–D522. ISSN: 0305-1048. DOI: 10.1093/nar/gkv1049. eprint: https://academic.oup.com/nar/article-pdf/44/D1/D515/16661243/gkv1049.pdf. URL: https://doi.org/10.1093/nar/gkv1049.

[12] Vincent Lacroix et al. "An introduction to metabolic networks and their structural analysis". In: *IEEE/ACM transactions on computational biology and bioinformatics* 5.4 (2008), pp. 594–617.

[13] Terzer M. *Polco: A Java tool to compute extreme rays of polyhedral cones;* https://csb.ethz.ch/tools/software/polco.html. [Online; accessed 2024 06 25]. 2009.

[14] T. S. Motzkin et al. "3. The Double Description Method". In: *Contributions to the Theory of Games (AM-28), Volume II*. Ed. by Harold William Kuhn and Albert William Tucker. Princeton: Princeton University Press, 1953, pp. 51–74. ISBN: 9781400881970. DOI: doi:10.1515/9781400881970-004. URL: https://doi.org/10.1515/9781400881970-004.

[15] Jeffrey D Orth, Ines Thiele and Bernhard Ø Palsson. "What is flux balance analysis?" In: *Nature biotechnology* 28.3 (2010), pp. 245–248.

[16] Rockafellar RT. "Convex Analysis". In: *Convex Analysis*. Princeton: Princeton University Press, 1970.

[17] Marco Terzer and Jörg Stelling. "Large-scale computation of elementary flux modes with bit pattern trees". In: *Bioinformatics* 24.19 (2008), pp. 2229–2235.

[18] Namrata Tomar and Rajat K. De. "Comparing methods for metabolic network analysis and an application to metabolic engineering". In: *Gene* 521.1 (2013), pp. 1–14. ISSN: 0378-1119. DOI: https://doi.org/10.1016/j.gene.2013.03.017. URL: https://www.sciencedirect.com/science/article/pii/S0378111913002515.

[19] R. Urbanczik and C. Wagner. "Functional stoichiometric analysis of metabolic networks". In: *Bioinformatics* 21.22 (Sept. 2005), pp. 4176–4180. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bti674. eprint: https://academic.oup.com/bioinformatics/article-pdf/21/22/4176/16841065/bti674.pdf. URL: https://doi.org/10.1093/bioinformatics/bti674.

[20] Jürgen Zanghellini et al. "Elementary flux modes in a nutshell: properties, calculation and applications". In: *Biotechnology journal* 8.9 (2013), pp. 1009–1016.

A. Appendix

A.0.1. Data and Code Availability

Thr source code for ECMproject can be found on GitHub in Christian Mayers repository at https://github.com/c-mayer/ECMproject.

All the data used for the plotting of the figures can be found below. The runtimes in the tables are always given in seconds.

models	cores	total	pre-processing	projection	vertex enumeration	post-processing	ECMs	max_rss [GB]
mmsyn_sm00	20	253.5	5.6	241.9	0.9	5.2	4132	0.196
mmsyn_sm01	20	256.3	5.6	244.3	1.1	5.2	6486	0.201
mmsyn_sm02	20	292.3	5.9	279.9	1.4	5.2	9702	0.236
mmsyn_sm03	20	269.4	6.0	256.0	2.1	5.2	19394	0.295
mmsyn_sm04	20	280.1	6.2	264.9	3.7	5.2	38778	0.352
mmsyn_sm05	20	289.9	6.5	272.1	6.1	5.3	77546	0.523
mmsyn_sm06	20	301.7	6.7	279.3	10.3	5.4	155082	0.618
mmsyn_sm07	20	320.1	6.7	287.5	20.2	5.7	310154	0.649
mmsyn_sm08	20	344.7	7.0	295.3	36.3	6.2	620298	0.638
mmsyn_sm09	20	396.8	7.1	310.5	71.7	7.4	1240586	0.632
mmsyn_sm10	20	470.4	7.4	309.3	142.1	11.6	2481162	0.639
mmsyn_sm11	20	630.8	7.5	317.0	287.9	18.4	4962314	0.656
mmsyn_sm12	20	954.8	7.8	324.8	589.7	32.5	9924618	0.645
mmsyn_sm13	20	1382.1	8.0	332.2	985.7	56.2	17756170	0.666
mmsyn_sm14	20	2454.4	8.3	348.5	1988.1	109.3	35512330	0.689
mmsyn_sm15	20	4850.4	8.4	378.2	4245.0	218.8	71024650	0.671
mmsyn_sm16	20	9492.8	8.6	358.7	8685.0	440.5	142049290	0.671
mmsyn_sm17	20	18720.3	9.2	375.2	17629.7	706.1	284098570	0.708
mmsyn_sm18	60	10832.1	8.7	412.4	9392.8	1018.2	514523146	0.683
mmsyn_sm19	60	49735.8	9.5	429.0	47762.1	1535.2	869854722	0.682
mmsyn_sm20	60	82744.2	8.7	465.7	79924.6	2345.2	1403221121	0.694
mmsyn_sm21	60	192268.0	9.0	447.4	181994.6	9816.9	2821027740	0.687
mmsyn_sm22	60	281329.7	9.8	452.1	265919.3	14948.6	4212839045	0.702

Table A.1 – Results of the increasing model size for ECMproject in its default setting with parallel postprocessing activated.

models	cores	total	pre-processing	projection	vertex enumeration	post-processing	max_rss [GB]
mmsyn_sm00	3	381.5	17.8	354.8	3.8	5.1	0.268
	5	279.9	8.4	265.0	1.4	5.1	0.227
	10	246.1	5.6	234.4	1.0	5.1	0.272
	20	253.5	5.6	241.9	0.9	5.2	0.196
	30	262.2	5.7	250.4	0.9	5.2	0.257
	40	268.9	5.7	257.1	0.9	5.2	0.243
	50	285.8	4.6	275.1	0.9	5.2	0.268
	60	284.3	4.7	273.4	0.9	5.2	0.273
mmsyn_sm05	3	525.3	23.4	430.8	66.0	5.2	0.521
	5	297.0	10.1	258.9	22.7	5.3	0.522
	10	289.0	6.3	267.1	10.2	5.3	0.526
	20	289.9	6.5	272.1	6.1	5.3	0.523
	30	305.9	6.5	289.2	5.0	5.3	0.508
	40	302.9	6.5	287.1	3.9	5.3	0.522
	50	310.3	5.3	295.8	3.8	5.3	0.511
	60	319.1	5.6	304.7	3.4	5.4	0.525
mmsyn_sm10	3	3113.4	30.3	561.4	2487.3	34.3	0.633
	5	1248.1	13.1	383.3	830.0	21.7	0.652
	10	650.4	7.6	314.0	313.5	15.3	0.640
	20	470.4	7.4	309.3	142.1	11.6	0.639
	30	422.8	7.4	316.8	88.8	9.9	0.628
	40	414.0	7.4	324.5	72.0	10.0	0.639
	50	408.4	7.6	333.1	57.8	9.9	0.628
	60	432.7	6.3	371.1	44.8	10.5	0.649
mmsyn_sm15	10	9816.3	8.8	383.1	9071.9	352.4	0.661
	20	4850.4	8.4	378.2	4245.0	218.8	0.671
	30	3155.2	8.5	355.7	2616.8	174.2	0.670
	40	2480.4	8.5	364.4	1950.6	156.9	0.665
	50	2095.9	8.6	372.7	1556.2	158.4	0.651
	60	1849.8	7.1	383.0	1299.8	159.8	0.652

Table A.2 – Results for the models where different numbers of cores were tested for ECMproject in its default setting with parallel postprocessing activated.

models	cores	total	pre-processing	projection	vertex enumeration	post-processing	ECMs	max_rss [GB]
mmsyn_sm00	20	438.6	6.8	429.8	1.6	0.3	4132	0.250
mmsyn_sm01	20	487.1	7.5	477.1	2.0	0.4	6486	0.384
mmsyn_sm02	20	491.6	7.3	481.3	2.5	0.6	9702	0.347
mmsyn_sm03	20	447.1	7.1	436.0	3.2	0.8	19394	0.329
mmsyn_sm04	20	471.0	7.3	456.5	5.8	1.3	38778	0.377
mmsyn_sm05	20	461.6	7.5	440.7	10.7	2.7	77546	0.550
mmsyn_sm06	20	477.7	7.7	450.4	14.6	5.0	155082	0.684
mmsyn_sm07	20	578.9	10.1	532.1	25.7	11.0	310154	0.896
mmsyn_sm08	20	550.5	8.2	473.1	49.0	20.1	620298	0.861
mmsyn_sm09	20	667.0	8.2	520.9	98.1	39.8	1240586	0.920
mmsyn_sm10	20	804.7	8.4	524.4	191.8	80.1	2481162	0.887
mmsyn_sm11	20	1140.7	9.1	561.0	400.4	170.2	4962314	0.963
mmsyn_sm12	20	1704.5	9.9	565.3	794.6	334.7	9924618	0.919
mmsyn_sm13	20	2440.6	9.3	559.5	1274.1	597.7	17756170	0.921
mmsyn_sm14	20	3570.4	8.4	409.1	1988.2	1164.8	35512330	0.860
mmsyn_sm15	20	6966.5	8.4	378.0	4244.0	2336.2	71024650	0.936
mmsyn_sm16	20	13896.9	8.6	406.1	8687.1	4795.1	142049290	0.937
mmsyn_sm17	20	27605.9	9.1	365.9	17614.8	9616.1	284098570	0.989

Table A.3 – Results of the increasing model size for ECMproject in its single core setting for postprocessing.

models	cores	total	pre-processing	VE 1	intermediate processing	VE 2	post-processing	max_rss [GB]
mmsyn_sm00	20	24.3	3.8	9.5	8.5	2.1	0.4	0.300
mmsyn_sm01	20	26.7	3.9	9.0	9.8	3.3	0.6	0.302
mmsyn_sm02	20	33.3	4.0	16.5	7.9	3.9	1.0	0.371
mmsyn_sm03	20	39.6	4.2	17.3	8.8	7.4	1.9	0.304
mmsyn_sm04	20	41.5	4.3	15.3	8.5	9.6	3.8	0.277
mmsyn_sm05	20	57.7	4.6	17.6	9.6	18.5	7.5	0.283
mmsyn_sm06	20	88.3	4.8	21.9	10.0	36.5	15.0	0.342
mmsyn_sm07	20	133.6	5.2	19.9	10.9	67.6	30.0	0.304
mmsyn_sm08	20	227.4	4.5	20.4	11.6	130.4	60.5	0.395
mmsyn_sm09	20	386.0	4.7	26.5	12.4	220.7	121.7	0.296
mmsyn_sm10	20	736.4	4.8	26.1	14.0	450.5	241.1	0.407
mmsyn_sm11	20	1472.2	5.0	28.2	15.2	933.5	490.4	0.463
mmsyn_sm12	20	2926.8	5.4	28.2	15.7	1905.3	972.2	0.340
mmsyn_sm13	20	5207.3	5.5	29.7	18.1	3405.9	1748.1	0.319
mmsyn_sm14	20	14739.9	5.6	29.9	18.1	11199.3	3486.9	0.371
mmsyn_sm15	20	23066.7	6.0	29.3	19.3	16026.4	6985.7	0.210
mmsyn_sm16	20	73923.5	6.0	38.2	24.4	59553.5	14301.4	0.206
mmsyn_sm17	20	146722.4	6.3	34.8	23.7	118075.1	28582.5	0.333
mmsyn_sm18	60	141957.4	6.6	19.6	27.2	90425.2	51478.7	0.478
mmsyn_sm19	60	851414.2	6.9	20.5	25.6	762573.1	88788.0	0.468

Table A.4 – Results of the increasing model size for ecmtool.

models	cores	total	pre-processing	VE 1	intermediate processing	VE 2	post-processing	max_rss [GB]
mmsyn_sm00	3	122.5	3.7	99.9	8.5	9.9	0.4	0.373
	5	49.2	3.7	31.8	8.5	4.8	0.4	0.215
	10	33.1	3.7	17.1	8.5	3.3	0.4	0.374
	20	24.3	3.8	9.5	8.5	2.1	0.4	0.300
	30	23.4	3.7	9.0	8.5	1.7	0.4	0.382
	40	22.1	3.7	7.9	8.4	1.6	0.4	0.309
	50	21.7	3.8	7.4	8.6	1.5	0.4	0.371
	60	20.9	3.7	6.8	8.5	1.5	0.4	0.379
mmsyn_sm05	3	463.7	4.7	202.5	9.7	239.3	7.5	0.436
	5	162.8	4.8	62.5	9.7	78.3	7.5	0.330
	10	88.9	4.7	33.6	9.6	33.6	7.5	0.318
	20	57.7	4.6	17.6	9.6	18.5	7.5	0.283
	30	49.6	4.6	15.1	9.7	12.7	7.5	0.347
	40	48.1	4.6	14.3	9.6	12.0	7.6	0.224
	50	45.1	4.7	11.3	9.6	11.9	7.5	0.375
	60	44.5	4.6	11.2	9.6	11.6	7.5	0.371
mmsyn_sm10	3	8188.4	4.7	255.5	13.9	7673.6	240.8	0.360
	5	2920.3	4.8	78.1	14.0	2582.1	241.4	0.274
	10	1291.6	4.8	41.1	13.8	991.6	240.3	0.332
	20	736.4	4.8	26.1	14.0	450.5	241.1	0.407
	30	565.0	4.8	18.7	14.0	285.6	241.8	0.382
	40	491.7	4.7	17.3	14.0	214.5	241.1	0.371
	50	459.7	4.9	14.7	13.8	184.1	242.3	0.338
	60	430.1	4.9	12.4	13.8	158.7	240.3	0.401
mmsyn_sm15	10	43161.3	6.0	61.2	19.3	36036.9	7037.9	0.216
	20	23066.7	6.0	29.3	19.3	16026.4	6985.7	0.210
	30	17377.3	5.9	27.4	20.5	10333.6	6990.0	0.209
	40	14909.5	6.0	23.9	19.4	7815.7	7044.4	0.414
	50	13329.2	5.9	19.2	19.3	6231.0	7053.8	0.204
	60	12326.8	5.9	18.0	20.0	5240.3	7042.5	0.210

Table A.5 – Results for the models where different numbers of cores were tested for ecmtool.