ECMproject, accelerating Elementary conversion mode computation via Flux Cone Projection

Master Thesis

Submitted by:	**Christian Mayer**
Personal identification:	**01340318**
at	**Master Degree Program** **Bio Data Science**
Completed at:	**University of Vienna**
Supervisor:	**Univ.-Prof. DI Dr. Jürgen Zanghellini**
Assessor:	**Dr. Dmytro Iurashev** **Mag. Dmitrij Turaev** **MSc. PhD. Milica Krunic**

Tulln, 20.12.2023

DECLARATION OF AUTHORSHIP

I hereby affirm that this research work was written solely by me and that I have not previously submitted this work on another educational institution for the purpose of receiving an academic degree. In particular, contributions by other persons in this work have been appropriately cited and the data gathered through the methods described have been accurately reproduced.

Tulln, 20.12.2023 Christian Mayer

Date Signature

Kurzzusammenfassung:

Die Entschlüsselung aller metabolischer Möglichkeiten einer Zelle ist eine anspruchsvolle Aufgabe, welche durch die Aufzählung aller „Elementary Conversion Modes" (ECMs) bewerkstelligt werden kann. Diese Berechnung erfordert hohen rechnerischen Aufwand, was die Analyse für größere Modelle zeitlich unmöglich macht. Daher wurde im Zuge dieser Master-Arbeit ein neues Programm namens ECMproject entwickelt, welches den mplrs Algorithmus mit der neu entwickelten „mplrs project" Funktion anwendet. In Bezug auf die Gesamtzeit skaliert ECMproject mit zunehmender Modellgröße besser als ecmtool, das bisher schnellste Programm. ECMproject stellt daher eine gute Alternative für größere und komplexere Modelle dar.

Schlagworte (mind. 3, max. 6):

Elementary Conversion Modes; Elementary Mode Analyse; mplrs project; ECMproject; ecmtool; genomweite metabolische Modelle

Abstract:

Uncovering the complete metabolic capabilities of a cell is a challenging task, which can be accomplished by enumerating all Elementary Conversion Modes (ECMs) of the cell's metabolic model. Enumeration of ECMs demands high computational efforts making it difficult to analyse larger models. We developed ECMproject, utilizing the mplrs algorithm with the 'mplrs project' function for more efficient ECM enumeration. In terms of total computation time, ECMproject performs better than ecmtool, hitherto recognized as the swiftest tool for ECM enumeration. Consequently, ECMproject presents itself as a viable option for the analysis of larger, more complex models.

Keywords (at least 3, max. 6):

Elementary Conversion Modes; Elementary Mode analysis; mplrs project; ECMproject; ecmtool; Genome-scale Metabolic Models (GSMM)

Acronyms

BFS Basic Feasible Solution

ECM Elementary Conversion Mode

EFM Elementary Flux Mode

EFV Elementary Flux Vector

FBA Flux Balance Analysis

FC Flux Cone

genFC generator representation of the flux cone

ineqFC inequality representation of the flux cone

JSON JavaScript object notation

lrs lexicographic reverse search

RSS Resident Set Size

SBML Systems Biology Markup Language

SBO Systems Biology Ontology

VE vertex enumeration

Contents

1 INTRODUCTION

Exploring an organism's metabolism can yield significant practical benefits. For instance, understanding the metabolic capabilities of a pathogen provides insights into the ecological niches where it can thrive. Traditionally, obtaining information about metabolic capabilities necessitated experimental cultivation of the organism in a laboratory setting. With the advent of cost-effective whole genome sequencing, we can now construct metabolic models from genomic data [Mendoza et al., 2019], obviating the need for labor-intensive experimental growth in many cases. This can be very advantageous, especially for non-culturable organisms. Metabolic models are already existing on different databases (e.g. BiGG [King et al., 2016]) for several organisms. Specific models are designed to emphasize particular aspects of an organism's metabolism, such as the core model for *E. coli*. In contrast, genome-scale metabolic models encompass the entirety of an organism's metabolic processes, often consisting of thousands of reactions and metabolites. Comprehensive analysis of such models can get challenging quite fast.

Several tools have been developed to assess metabolic capabilities of metabolic models and thereby closing the gap between genotype and phenotype of an organism. When we have specific questions in mind like maximizing biomass production or a specific product, Flux Balance Analysis (FBA) [Orth et al., 2010] can provide valuable insights [Raman and Chandra, 2009]. However, FBA is limited to offering only one optimal solution and requires a clear objective [Buchner and Zanghellini, 2021; Terzer and Stelling, 2008]. For a more "unbiased" assessment of an organism's metabolic capabilities, the enumeration of Elementary Flux Modes (EFMs) [Zanghellini et al., 2013] proves to be a better ap-

proach, since FBA only concentrates on one optimal solution while EFMs describe the whole space of all possible solutions. EFMs showcase all the unique elementary pathway possibilities within a metabolic model at steady state while regarding all irreversibility constrains [Buchner and Zanghellini, 2021]. A pathway is a set of linked reactions which lead from one point of the metabolism to another one. In case of EFMs, these points are external reactions. Each reaction in the model is represented in a vector of reaction rates, where each reaction rate corresponds to one reaction.

1.1 Elementary Flux Modes (EFMs)

To get concentration changes of metabolites over time, reaction rate of the corresponding reaction has to be multiplied with the stochiometric factor of the reaction. Given a vector **dc/dt** with all concentration changes over time of all metabolites of the whole model, the stochiometric matrix (**N**) has to be multiplied with a reaction rate vector which includes all reactions of the whole model (**v**) (Eq. 1). All this feasible reaction rate vectors (**v**) which fulfill the steady state are called flux modes. The steady state is a commonly made assumption, as metabolism typically operates on significantly faster time scales compared to corresponding regulatory events [Terzer and Stelling, 2008] and assures that no internal metabolite can be accumulated or depleted over time. Mathematically, each internal metabolite time derivative **dc/dt** needs to be zero to fulfill the steady state (Eq. 1). All stochiometric vectors are building the stochiometric matrix (**N**). Each column within this matrix corresponds to a stoichiometric vector of an individual reaction across all metabolites, and conversely, each row embodies a stoichiometric vector of a particular metabolite across all reactions.

$$\frac{\boldsymbol{dc}}{\boldsymbol{dt}} = N\boldsymbol{v} \,|\, c_i = 0 \; if \; i \in Int, v_j \geq 0 \; for \; all \; j \tag{1}$$

An EFM, like all other flux modes, is also represented as a vector of reaction rates but differs to normal flux modes in the following properties:

- EFMs are non-decomposable. No reaction of an EFM which is nonzero (therefore is active) can be deleted without violating the steady state assumption [Klamt et al., 2017; Zanghellini et al., 2013]. This property can help by detecting minimal cut sets.

- EFMs are conformal generators of the flux cone (FC). All flux modes can be described as a conformal sum of EFMs [Klamt et al., 2017] without cancellation of a reaction. All flux modes which are a sum of EFMs with cancellation are EFMs themselves.

This property gives a more complete definition of EFMs, since all extreme rays of a flux cone are EFMs but not necessarily all EFMs are extreme rays. Nevertheless, there exists a neat trick to get all EFMs of a cone by just enumerating the extreme rays. If all reactions are set to irreversibility (by splitting all reversible reactions in two irreversible ones), the whole cone resides in the positive orthant. No cancellation can appear, since all EFMs are in the positive orthant.

- EFMs are support-minimal, which means, that no steady-state subsets of an EFM are possible. If any feasible reaction rate vector is a subset of another reaction vector, the latter is not an EFM.

  Another more mathematical explanation of support-minimality needs the definition of flux vector support. A support of a flux vector (**supp(v)**) is the index set of the nonzero part of this vector. In other words, all indices of reactions which are active in this vector are its support. Given an EFM, its support (**supp(e)**) cannot be a superset of another feasible flux mode [Zanghellini et al., 2013]. All supports of EFMs are already minimal, there is no feasible support of other flux modes, which could be a subset of an EFM support (Eq. 2).

In other words, EFMs cannot be expressed as the sum of other flux vectors without any cancellations occurring [Klamt et al., 2017]. Consequently all possible flux vectors which are not elementary are weighted combinations of EFMs [Zanghellini et al., 2013].

$$supp(v) \not\subset supp(e) \tag{2}$$

1.2 Flux Cone (FC)

All Elementary Flux Modes (EFMs) can be located in a flux cone (FC), a solution space of a metabolic model, which is a subset of the nullspace defined by the steady-state constraints. Additionally, the flux cone is defined by irreversibility constrains [Klamt et al., 2017]. If additional inhomogeneous linear constrains such as upper and lower reaction boundaries are required, an expanded characterization of EFMs becomes necessary [Buchner and Zanghellini, 2021]. Klamt et al. (2017) and Buchner and Zanghellini (2021) defined EFMs as convex-conformally non-decomposable pathways. This definition also includes Elementary Flux Vectors (EFVs) and allows for inclusion of inhomogeneous flux bounds which transform a flux cone to a flux polyhedron. The flux polyhedron is a subset of the flux cone [Klamt et al., 2017], since additional inhomogeneous linear constrains further restrict the solution space [Buchner and Zanghellini, 2021].

The flux cone has dimensions corresponding to the number of reaction rates in the model subtracted by the rank of the stochiometric matrix [Klamt et al., 2017] and can be seen as the solution space of all possible flux modes of the metabolic model. Since EFMs are the extreme vectors of a flux cone, all possible flux modes of the cone are a weighted combination of EFMs [Klamt et al., 2017; Zanghellini et al., 2013]. Therefore, EFMs are called "elementary".

1.3 Elementary Conversion Modes (ECMs)

As the model size increases, the number of EFMs grows combinatorial, making them computationally challenging and the enumeration only feasible for small and medium sized models with limited connectivity [Buchner and Zanghellini, 2021; Terzer and Stelling, 2008]. Given that numerous EFMs exhibit identical substrate-to-product con-

versions, thereby signifying equivalent metabolic conversion capabilities, their exhaustive enumeration is not always necessary. Instead, for many applications, it is enough to concentrate our attention on all feasible overall conversions that a cell is capable of catalyzing [Clement et al., 2021]. This is where Elementary Conversion Modes (ECMs) come into play. ECMs present the overall conversion of an organism/metabolic model, focusing on the initial substrates and final products rather than the internal pathways themselves. In Figure 1, we can observe an example of an EFM and ECM enumeration within a small model. Each ECM can have multiple EFMs (e.g. blue and green EFM in Figure 1) but no EFM can lead to more than one ECM. EFMs with multiple substrate or product production never lead to more than one ECM, since, if no other smaller ECMs exist, the EFM leads to exactly one ECM and if smaller ECMs exist, the resulting ECM can be described by the combination of smaller ECMs and therefore is just a normal conversion mode (e.g. orange EFM in Figure 1). Since many different EFMs result in the same substrate-to-product-conversion, the number of enumerated ECMs increases much slower than the number of EFMs [Clement et al., 2021]. For example, the model of *E. coli* core metabolism which consists of 72 metabolites and 95 reactions has 100,274 EFMs but only 689 ECMs [Clement et al., 2021]. The dimensionality of ECMs is normally smaller compared to dimensionality of EFMs, due to the reduced focus on external reactions and external metabolites [Clement et al., 2021]. External reactions are reactions, which take up a substrate (external metabolite intake) or excrete products (external metabolite output). These reactions are crossing the model border and the external metabolites are not in steady state (while all internal metabolites are assumed to be in steady state) [Zanghellini et al., 2013]. If only overall conversion capabilities of a metabolic model are in question, enumeration of ECMs is a better approach.

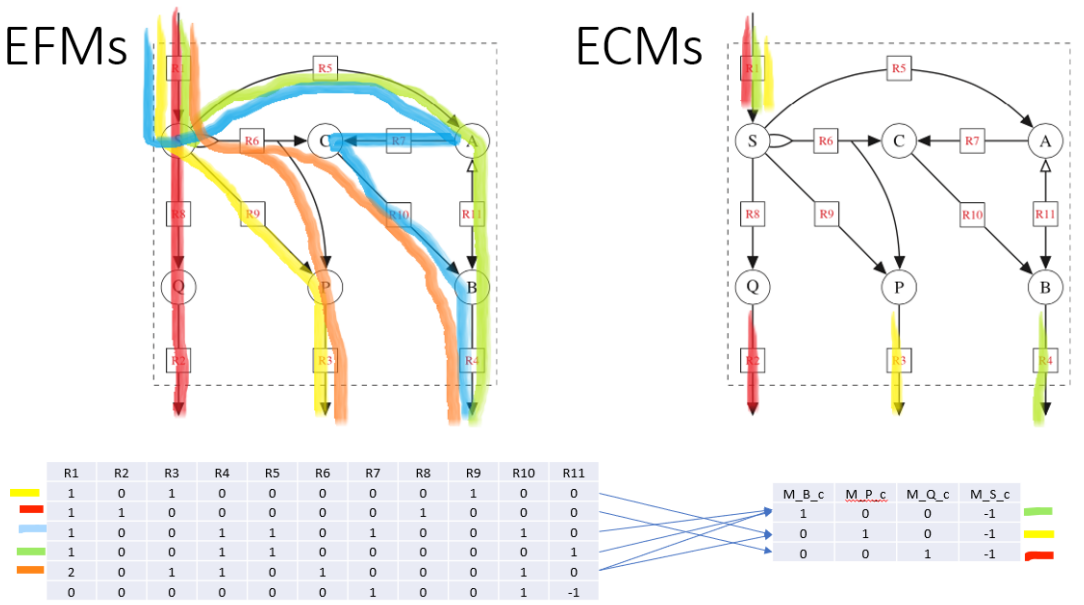

Figure 1: Comparison of EFMs and ECMs in a small toy model (*yfba_s1*). Five EFMs lead to just three ECMs. The tables are showing the corresponding EFMs and ECMs. Notably, two EFMs, represented by the colors green and blue, led to identical overall conversions, thereby both being represented by the green ECM. The orange EFM leads to two different ECMs, but, since the sum of these two ECMs is no ECM, effectively, the orange EFM leads to no ECM and just to a normal conversion mode.

An ECM is the product of a vector of reaction rates projected to external reactions and the stoichiometric matrix for external metabolites (Eq. 3), providing the rates of change for all external metabolite concentrations [Clement et al., 2021].

$$C = \{\mathbf{c} = N\boldsymbol{v} \,|\, c_i = 0 \; if \; i \in Int, v_j \geq 0 \; for \; all \; j\} \tag{3}$$

All ECMs together form a minimal set which generates all substrate-to-product conversions, analogous to all EFMs forming the minimal set which generates all steady-state flux modes [Clement et al., 2021]. Therefore, all ECMs can be found within a conversion

cone (C), where the conversion rate of external metabolites are dimensions. The conversion cone represents all possible substrate-to-product conversions the metabolic model can perform. Put simply, ECMs reveal the conclusive outcomes of EFMs concerning the external metabolites, i.e. the feasible stoichiometries between substrates and products [Clement et al., 2021]. Figure 2 illustrates an example of a conversion cone, providing a visual representation of this concept.

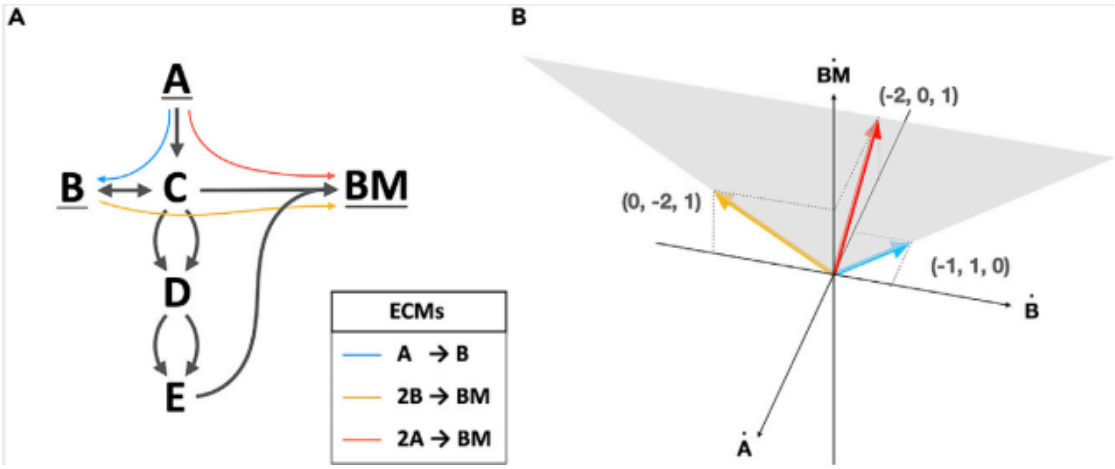

Figure 2: Illustration of ECMs of a model in a conversion cone. (**A**) A simple model which leads to three ECMs. (**B**) The resulting conversion cone of the model. The blue and the yellow ECMs are typical extreme rays, i.e., vectors on the edge of the cone. The hyperplane of metabolite B being zero cuts the cone, leading to two different orthants of the cone. Therefore, the red one lies within the cone and is a special case. The resulting conversion when adding the blue and the yellow ECM together leads to the same conversion as the red ECM does ($2A \rightarrow 2B + 2B \rightarrow BM = 2A \rightarrow BM$). But since it also leads to an elimination of the external metabolite B it is also an ECM and resides at the zero hyperplane of the external metabolite B ($B = 0$). (taken from Clement et al. (2021))

Consider the conversion cone, which represents the solution space of all possible conversion states. Each point within this cone can be viewed as a combination of different ECMs. Rather than showing all possible conversion combinations, which would be over-

whelming, we focus on presenting the extreme cases that can be achieved. ECMs can be thought of as the extreme cases/rays of the conversion cone. This concept holds true when the entire cone spans only one orthant of our coordinate system [Clement et al., 2021]. However, as depicted in Figure 2, not all ECMs lie at the edge of the cone if the cone spans more than one orthant. Thus, a broader and more precise definition is necessary. This led to an alternative definition of ECMs:

"*The set of elementary conversion modes (ECMs) is the minimal set of conversions (ecm_1, ..., ecm_K) such that each steady-state conversion can be written as a positive sum of ECMs, without the production of any external metabolite being canceled in that sum.*" [Clement et al., 2021]

If the combination of two ECMs cancels out the production of an external metabolite, that combination itself is an ECM. This can be observed in Figure 2, where the red ECM is a combination of the blue and yellow ECMs but cancels out metabolite B in the process. With this definition in mind, different tools have different solutions to enumerate all ECMs.

1.4 Lexicographic reverse search (lrs)

To solve the vertex enumeration problem and therefore enumerating all ECMs, two different methodologies have been identified: the Double Description method and the Lexicographic reverse search [Buchner and Zanghellini, 2021]. The Double Description method was first implemented and uses a so called "double pair" to convert a generator representation to an inequality representation [Clement et al., 2021].

Lexicographic reverse search is based on the idea of reversing the simplex method [Avis and Fukuda, 1992], which is a method to optimize a system of linear equations and inequalities. In its non-reverse form, the simplex method searches for an optimum based on the optimization of an objective via pivoting operations on a dictionary derived from the set of equalities and inequalities and therefore walks from one vertex of an object

to another adjacent vertex in each pivoting step until the vertex which maximizes the objective is found (Fig. 3). Vertex enumeration follows a reverse search pattern. First, an extreme vertex, which is assumed to be the optimal vertex for a chosen objective, is determined as the starting vertex [Avis and Jordan, 2018; Buchner and Zanghellini, 2021]. Conventionally, the origin of the object is taken as starting vertex [Avis and Fukuda, 1992]. Then the algorithm has to find a path from this vertex to all other vertices (Basic Feasible Solutions) of the object by moving along the edges of the object via reverse pivoting. Whenever a Basic Feasible Solution (BFS) is found the first time, a vertex/ray can be identified and recorded. All paths together form the reverse search tree with the starting vertex as root (Fig. 3) [Avis and Fukuda, 1992; Avis and Jordan, 2018]. A quite efficient algorithm for vertex enumeration using reverse pivoting is presented in Avis and Fukuda (1992). By parallelizing the search, multiple branches of the reverse search tree can be explored simultaneously. Once all possible subproblems are solved, and all BFSs are discovered, the vertex enumeration process is complete. Efficiency of the vertex enumeration strongly depends on the geometry of the enumerated object and therefore on the structure of the reverse search tree [Avis and Jordan, 2018].

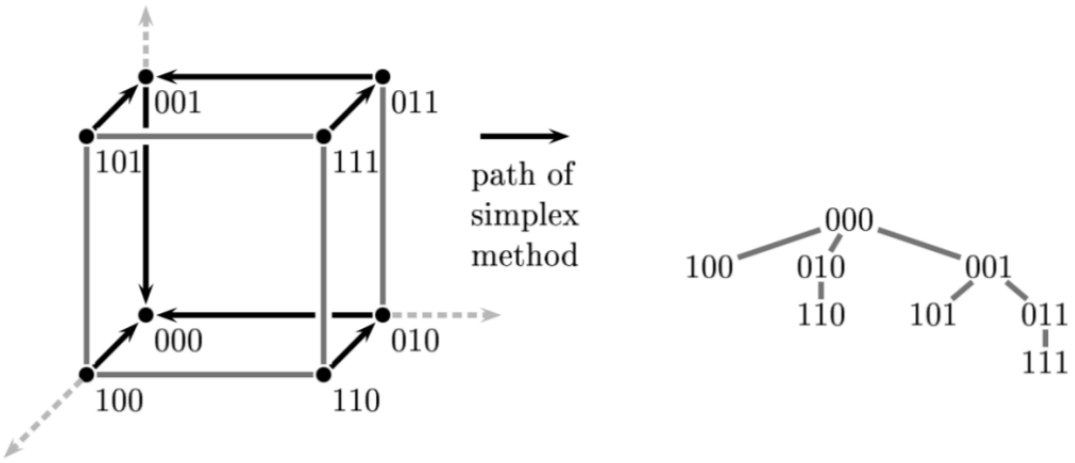

Figure 3: Simplex method and corresponding reverse search tree. Different paths of the simplex method to an optimal node in a cube correspond to the structure of a reverse search tree. (taken from Avis (1998))

1.5 mplrs

The state of art tool for performing lexicographic reverse search is mplrs, which has its origins in a tool for parallelized lexicographic reverse search (plrs) [Avis and Jordan, 2018]. The mplrs algorithm requires a minimum of three processes: one master, one worker, and one consumer, although multiple workers can also be utilized. Initially, since we have only one starting point, the master selects one worker responsible for dividing the entire problem into subproblems, which are then added to a list. In the context of vertex enumeration, each subproblem corresponds to a portion of the reverse search tree with nodes as vertices and edges as rays. After this initial setup phase, the master assigns subproblems from the list to each worker. Each worker attempts to solve its assigned subproblem within a specified number of iterations. If a BFS is obtained within the allotted iterations, the worker forwards the solution to the consumer, which then prints the result. If a worker fails to find a solution within the allotted number of iterations, it returns all the newly produced smaller subproblems back to the master, who adds them back to the list. This load-balancing mechanism contributes to the speed advantage achieved over plrs, resulting in faster processing times. Once all subproblems in the list have been solved, all BFSs are discovered. [Avis and Jordan, 2018]

1.6 ecmtool

Ecmtool, a tool for enumerating ECMs already written by Clement et al. (2021), utilizes the Double Description Method [Motzkin et al., 1953]. Ecmtool offers two distinct methods for ECM enumeration: an indirect method and a direct method [Clement et al., 2021]. Both methods are elaborated on in more detail in Supplementary material of Clement et al. (2021). The indirect method uses Double Description conversion twice: once at the beginning and once at the end. Between these two Double Description Steps, steady state constrains get implemented which leads to a large intermediate result. By splitting all external metabolites, ecmtool enumerates all ECMs of a Conversion Cone by calculating the extreme rays of this cone [Clement et al., 2021]. The direct method does

not use Double Description and adds steady state constrains directly iterative to the generator representation of a general conversion cone converting this cone to a specific conversion cone we are searching for [Clement et al., 2021]. The direct method was invented by Clement et al. (2021) themselves and therefore is a very unconventional method. While it conserves memory space by avoiding the creation of intermediate results, it exhibits relatively slower computational speed when compared to the indirect method [Clement et al., 2021].

Since even time performance of the indirect method of ecmtool got slow and memory intensive with growing model size quite fast, ecmtool has undergone further enhancements with the introduction of the '*mplrs*' method as a second option of the indirect computation to speed up the computation time and reduce Resident Set Size (RSS). Instead of Double Description, lexicographic reverse search is used to convert generator representations into inequality representations or vice versa (shown as VE steps in Fig. 4). Both of these representations are elaborated upon in the Material and Methods section. These developments have made the ECM enumeration process more efficient and effective and applicable for larger models. [Buchner et al., 2023]

1.7 Problem definition - ECMproject

Unfortunately, even ecmtool in '*mplrs*' mode is not able to enumerate all ECMs of the minimal cell JCVI-syn3A (304 metabolites, 338 reactions, 69 exchange reactions) in reasonable time on a complex medium with 19 (ingestible) excess nutrients (model name '*mmsyn_sm19*') [Buchner et al., 2023]. In order to address this time performance limitations, a new function called '*mplrs project*' in the lrslib [Avis, 2022] was explored. Subsequently, a new tool called ECMproject was developed from us as part of this thesis, specifically designed to work with '*mplrs project*'.

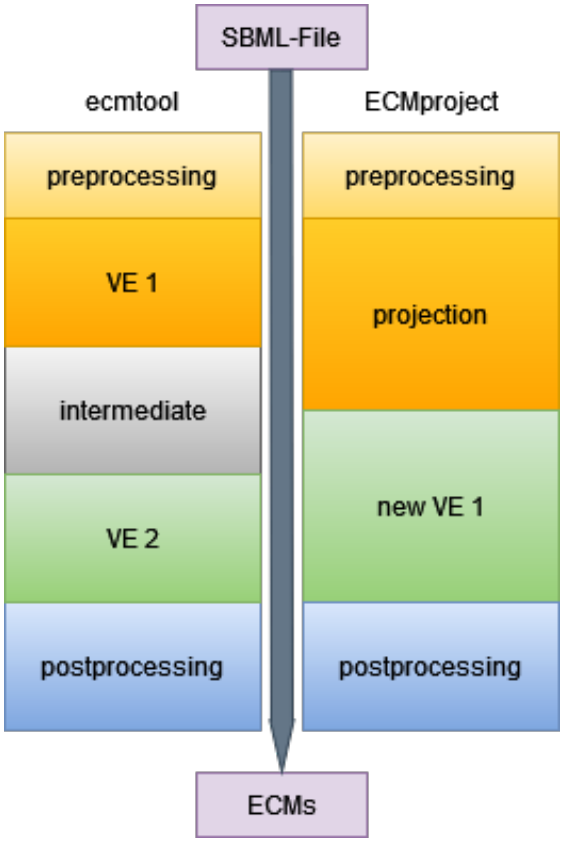

Figure 4: Comparison of the workflow of ecmtool in 'mplrs' mode and ECMproject. Ecmtool has an additional intermediate and an additional vertex enumeration (VE) step. Each VE step corresponds to the utilization of the 'mplrs' procedure, to solve the vertex enumeration problem. Second vertex enumeration (VE2) of ecmtool can be compared with the first one of ECMproject. The projection step of ECMproject gets performed by '*mplrs project*'.

The workflow disparities between ecmtool in '*mplrs*' mode and ECMproject are depicted in Figure 4. The main difference lies in the initiation of the conversion cone and in the integration of the steady-state constrains. ECMproject creates an inequality representation of a flux cone including the steady state conditions right at the beginning and transitions from the flux cone to the conversion cone in the postprocessing step. In contrast, ecmtool works with a general conversion cone right from the beginning of the analysis and applies the steady state in the intermediate step of the indirect approach.

The following questions are answered in the present work:

- Does ECMproject provide the same results as ecmtool? I.e., first, the new tool is validated against the standard in the field.
- Is ECMproject faster than ecmtool in '*mplrs*' mode, and if so, what is the extent of the speed improvement?
- How does the memory consumption (RSS) of ECMproject compare to ecmtool in '*mplrs*' mode?

2 MATERIALS AND METHODS

For the enumeration of ECMs, two tools, namely ecmtool and ECMproject, are investigated. Ecmtool, an established software, represents the current state-of-the-art for ECM enumeration, initially developed by Clement et al. (2021) and subsequently enhanced by Buchner et al. (2023). ECMproject was developed as a part of this thesis and can be downloaded from https://github.com/c-mayer/ECMproject. It contains many steps which will get explained in greater detail in this chapter.

2.1 Hardware and Software Components

The tool ECMproject underwent rigorous testing on the Jucuda Server, a part of the network cluster of the University of Vienna. To ensure repeatability of results, specific specifications of the server are listed in Table 1 and Table 2. Notably, we used two different Python conda environments, each equipped with essential packages, as outlined in Table 3.

To facilitate compatibility between ECMproject and ecmtool, we created one conda environment for ecmtool called '*ecmtool_env*' and one for ECMproject called '*ECMproject*'. For readers convenience, we have provided .yml files for both environments in the '*environment_files*' directory of https://github.com/c-mayer/ECMproject/tree/main/installation, simplifying the installation process.

Table 1: Hardware of the Jucuda server.

AMD EPYC 7542 32-Core Processor (x86_64), 128 CPUs
2 TB RAM DDR4
NVIDIA Tesla V100S-32GB

Table 2: 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: Conda environments.

packages	version	
	'ECMproject'	*'ecmtool_env'*
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

2.2 Preprocessing

2.2.1 Reading in input data

ECMproject relies on SBML-Files as its input data. These files are written and structured in Systems Biology Markup Language (SBML) [Hucka et al., 2019] which has a typical xml-scheme with a clear hierarchy. Most important for parsing is the subordination 'listOfReactions', which contains all reactions of the model. Assuming they fulfill the general conventions, the parsing process should proceed without any complications.

To accomplish this, we utilize the Python package called '*cobra*', which effectively transforms the SBML-File into Python objects. The model itself is one object, as well as each reaction. Certain attributes like id, name, products, reactants and so on are annotated to these objects. It is important to note, that the tagging of external reactions is not performed by the '*cobra*' package itself. We have implemented a custom function that takes care of tagging reactions by adding them an attribute with exchange True or False. This function applies specific criteria to identify external reactions accurately:

- The presence of the SBO-Term:0000627 in the reaction, which is a code for exchange reaction.
- The reaction ID containing either R_EX or EX.
- The reaction being one-sided without functioning as a sink or demand reaction.

At least one of these conditions must be met for the function to correctly identify and tag the intended external reaction. By incorporating this functionality, ECMproject ensures that all external reactions are appropriately recognized and handled. This allows a comprehensive analysis of the system.

2.2.2 Splitting reversible external reactions

To obtain all Elementary Conversion Modes (ECMs), it is crucial that the entire geometric object resides within a single orthant (see Section 1.3). Achieving this requires splitting all reversible external reactions into two separate irreversible external reactions. This splitting process involves transforming a reversible reaction object into two mirrored irreversible reaction objects. By performing this split, we ensure that each reaction only occurs in one direction, effectively confining the geometric object representing the system to a single orthant. This adjustment facilitates the accurate determination and representation of all ECMs as rays within the V-representation framework.

2.2.3 Writing H-Representation

Since SBML format is not suitable for mplrs calculations, all necessary information which defines our solution space are written into a matrix. The H-Representation, as a mathematical framework, is a set of equations derived from the given SBML-File. These equations define a solution space geometrically represented by a polyhedron which can be described by the following inequalities:

$$\mathbf{Ax} \geq \mathbf{b} \tag{4}$$

$$-\mathbf{b} + \mathbf{Ax} \geq \mathbf{0} \tag{5}$$

The matrix **A** can encompass a diverse range of configurations, for example, each row representing an individual reaction and each column signifying a distinct metabolite. Vector **x** conceptually can be the collection of reaction rates corresponding to the reactions within **A**. Additionally, vector **b** serves as a vector of constrain variables which establish the constrain boundaries (inhomogeneous if $\neq 0$; homogeneous if 0). In its general form, our framework is not confined to steady-state conditions and thus is written in form of inequalities.

The H-Representation consists of three components: the **b**-column, the stoichiometric matrix **N**, and the irreversibility matrix $\mathbf{I}_{Irr}$ (depicted in Fig. 5). The irreversibility matrix has dimensions of m x n (m is the number of irreversible reactions, n is the total number of reactions). Since the stochiometric matrix (l x n where l is the number of metabolites, n = total number of reactions) has the same number of columns n as the irreversibility matrix, they can be combined into one matrix represented as **A** (e x n where $e = m + l$) in the general Equation (4). The variable **x** is a flux vector with dimensions n x 1 (n = number of reactions) and represents one possible flux. Vector **b** can be understood as a vector of constrain variables and has dimensions of e x 1 (e is the number of equations) if no additional constrains beside steady state (rows of **N** or

l) and irreversibility (rows of $\mathbf{I}_{Irr}$ or m) are set.

Since we assume steady state, values in vector $\mathbf{b}$ corresponding to equations of the stochiometric matrix ($\mathbf{N}$) must have a zero value and a linearity mark for the lrs algorithm (see Fig. 5). Setting $\mathbf{b} = \mathbf{0}$ leads to a flux cone. Equation (4) can be written as Equation (6). Similar assumptions are made for the irreversibility constrains. A reaction is irreversible, if the reaction rate can have only positive direction. For an irreversible reaction this means $\mathbf{Ax} \geq \mathbf{0}$ which sets the values in the $\mathbf{b}$ vector corresponding to equations of the irreversibility matrix ($\mathbf{I}_{Irr}$) to zero, where Irr is an index set indicating the indices of the irreversible reactions.

$$\begin{aligned} \mathbf{Nx} &= \mathbf{0} \\ \mathbf{x}_i &\geq \mathbf{0}, \quad i \in Irr \end{aligned} \tag{6}$$

By transforming Equation (4) to a canonical form such that the right side is zero (Eq. 5), we obtain a form similar to each row of the H-Representation illustrated in Figure 5.

Once the H-Representation is established, we apply the '*mplrs redund*' procedure to remove redundant equations if they exist. By eliminating redundant rows, the projection step becomes more efficient. It is worth noting that since the '*mplrs*' project algorithm only accepts integers or fractions, all floating-point values need to be converted before being incorporated into the H-Representation. Converting floats to fractions involves rounding fractions with very large numbers of numerator and denominator to fractions with smaller numbers, thereby improving the time efficiency of the projection process. This rounding function needs a limit to know, at which size of the fraction rounding is preferable. If the fraction of the value to be rounded has a denominator exceeding a certain value, the fraction gets rounded. This value is set to 1e06 by default. With the '*–approximation*' option, the limit can be changed manually. For example, if a value like 0.001 gets converted to a fraction, the output would be 1/1000 and the denominator consequently 1000. If we set the limit lower than 1000, the fraction gets rounded. Although this approach theoretically introduces some level of inaccuracy in the results,

in practice, these inaccuracies are negligible when the rounding parameter is set to a high value (e.g., 1e06 or higher). However, if the parameter is set very high, it can significantly increase projection times.

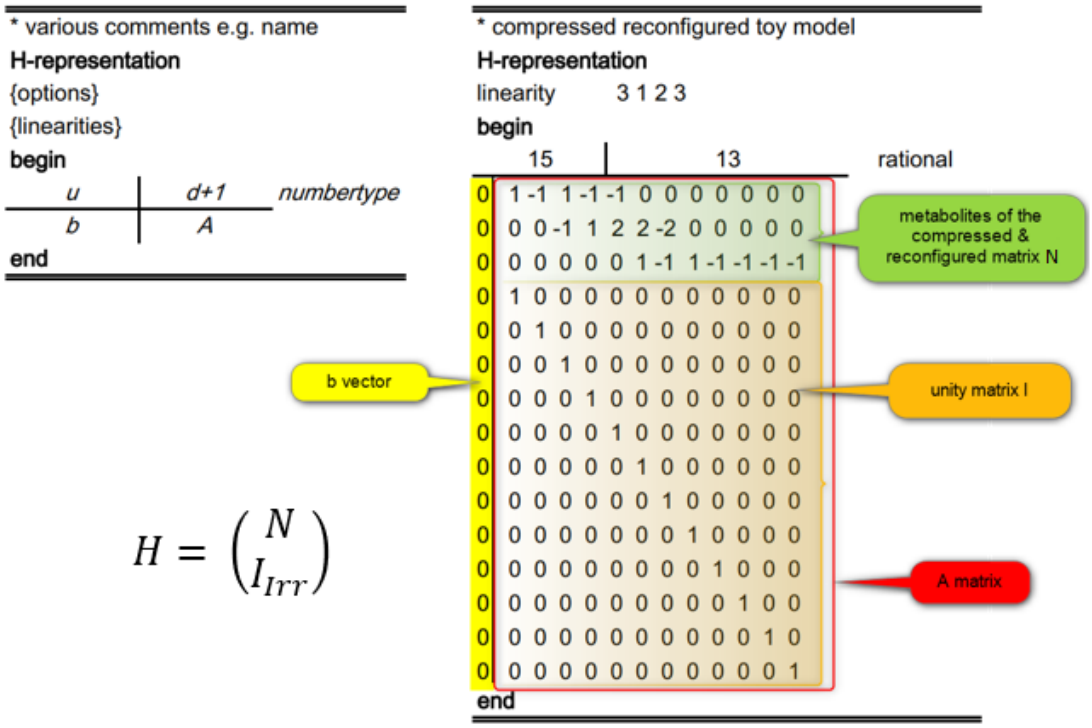

Figure 5: Example of an H-representation. The green part of the matrix is the stochiometric matrix (**N**). The orange part is the Irreversibility matrix ($\mathbf{I}_{Irr}$). The first column (yellow) is the **b**-vector. Each other column of the matrix is one variable of the equation and the entries in the column are the resulting coefficients for this variable in a certain equation. The linearity shows the character of the equation. In this case, the equations of rows one to three are solved with an equal sign (=). All other equations are solved with an equal or greater than sign (≥).

2.3 Projection

To perform the projection of the H-Representation (inequality representation), we utilize '*mplrs project*', a new function of the mplrs tool which reduces dimensions of a polyhedron or cone iteratively with Fourier Elimination [Avis, 2022]. In each iteration step, the inequality representation (ineq) respectively H-representation gets reduced by one internal reaction by projecting the corresponding column of the internal reaction onto the other columns. During the projection process, the dimensionality of the flux cone (FC) gets iteratively reduced to include only the fluxes associated with external reactions (ineq$_{ex}$FC) (Eq. 7). This reduction is achieved while preserving all the constraints imposed by the internal reactions.

$$ineqFC \rightarrow ineq_{ex}FC \tag{7}$$

There are two different projection modes available, which just differ in the way of execution. With choosing the option '*–mfel*', a "fel" algorithm of Avis (2022) gets executed to perform the looping of '*mplrs project*'. In standard application, this looping is performed by the main script itself. Beside Section 3.3.1, all results only got created with standard mode.

2.4 Vertex enumeration

To convert the solution space obtained from the projected H-Representation into a geometric object represented by vertices and rays, we employ a technique called vertex enumeration. This problem is solved using the *mplrs* tool on the reduced H-Representation.

The resulting V-Representation consists of the entire minimal generator set of the flux cone (genFC) containing all extreme rays (pre-ECMs), since the cone only resides in one orthant. All lines in the V-representation starting with '0' are rays, all lines starting with '1' are vertices [Avis, 2022]. Since we don't have any additional reaction constrains

beside the steady state and the irreversibility, we get rays and only one vertex at the origin of the cone (depicted in Figure 6). This transformation from an H-Representation to a V-Representation allows us to obtain a comprehensive set of rays and our basis vertex, accurately representing the solution space of the cone (Eq. 8).

$$ineq_{ex}FC \rightarrow genFC \tag{8}$$

```
*mplrs:lrslib_v.7.2_2022.3.6(hybrid arithmetic)3 processes
*Input taken from .//tmp/yfba_s1_h.projected
*Starting depth of 2 maxcobases=50 maxdepth=0 lmin=3 lmax=3 scale=100
*Phase 1 time: 0 seconds.
V-representation
begin
***** 5 rational
 1  0  0  0  0
 0  1  0  0  1
 0  1  1  0  0
 0  1  0  1  0
end
*Total number of jobs: 0, L became empty 0 times, tree depth 0, deepest vertex depth 0
*Finished with 1 64-bit, 0 128-bit, 0 GMP workers
*Totals: vertices=1 rays=3 bases=1 integer-vertices=1 vertices+rays=4
*Elapsed time: 0 seconds.
```

Figure 6: Example of an V-representation of the small fictional toy model '*yfba_s1*'. Each row of the matrix starting with '1' corresponds to one node. Each row starting with '0' corresponds to one ray. Three rays are leading to three pre-ECMs. The node is the origin of this concrete resulting cone object with each external metabolite being zero.

2.5 Postprocessing

In order to enhance the efficiency of the postprocessing stage, a multiprocessing implementation has been developed. When the multiprocessing option (*–parallel*) is enabled, the postprocessing step becomes significantly faster by using the '*multiprocessing*' python package to parallelize postprocessing (see Section 3.3.2). Early stages which used the

'*Pool()*' function for managing parallel processes had very high RAM demands. In the final version, this problem got solved manually without this function. Nevertheless, there is an option called '*–pool*', which performs parallelization of the '*multiprocessing*' package with the '*Pool()*' function if desired. Resident set size (RSS) comparisons between ECMproject with and without the '*–pool*' option are shown in Section 3.3.2.

2.5.1 Reading V-Representation and merging splitted reactions

During the processing of the V-Representation, it is handled in chunks, which allows for less RAM requirements and enables parallel-postprocessing, where each core converts one chunk at a time. The chunk size can be adjusted manually via the '*–chunksize*' option. Each row in a chunk can be interpreted as a ray/vertex (pre-ECM) and each column is one reaction. Since all reversible external reactions were split at the beginning, corresponding forward- and backward-reaction have to be merged again, to get ECMs with the real net conversion of the reversible external reaction. This can be accomplished by adding the value of the forward- and the value of the backward-reaction together. To merge all pre-ECMs of a chunk at the same time, columns of the forward- and backward-reaction get merged via vector addition.

2.5.2 Calculation of ECMs and normalizing

To convert the generator of the flux cone (genFC) into a generator of the conversion cone (genC) and thereby get all ECMs, the converted pre-ECMs from Section 2.5.1 get multiplied with the stochiometric matrix. However, before this operation can be performed, the stoichiometric matrix needs to be modified. First, we slice the stoichiometric matrix to include only external reactions. This step ensures that we focus solely on the external reactions involved in the ECM calculation. Additionally, the stoichiometric matrix is transposed to align the columns with metabolites and the rows with reactions. The resulting modified stoichiometric matrix and the converted rays are then multiplied together, as expressed in Equation (9), to compute the ECMs for the model. After com-

putation of a chunk of ECMs, each ECM gets normalized by default on the maximum value.

$$\begin{gathered} \text{genC} = \text{genFC } \boldsymbol{N}^T \\ \text{(ECM x metabolites)} = \text{(ray x reactions) x (reactions x metabolites)} \end{gathered} \tag{9}$$

2.5.3 Writing ECMs into result file

In large models, the high number of ECMs can result in large file sizes. To mitigate this effect, a rounding operation can be applied to the conversion rate values of an ECM, reducing their precision to a specified number of decimal places. By default, four decimal places are used, as it aligns with the typical precision found in the stoichiometry of reactions in input SBML-Files. However, the number of decimal places can be chosen manually via the '*–decimals*' option.

Additionally, to further optimize file size, the '*–gzipped*' option can be activated. This option enables the output file to be compressed in a binary format, reducing its overall size. By utilizing compression, the storage requirements for the ECM result file can be significantly reduced, which is particularly beneficial for models with a large number of ECMs. Unfortunately, using the '*–gzipped*' option can result in higher RSS utilization and slower postprocessing times in single- and parallel-mode. It is essential to strike a balance between speed and resource utilization, taking into account the size of the model and the available system resources, to determine whether the '*–gzipped*' option should be enabled or not.

Furthermore, there are additional options available to customize the output format. Users have the flexibility to set the delimiter to a format of their preference. By default, ECMs are written in CSV format with comma delimiters. This can be adjusted as per individual requirements via the '*–separator*' option.

These options provide flexibility in managing file sizes and output formats, allowing users to tailor the ECM results to their specific needs while optimizing storage and facilitating data analysis.

2.6 Validation

The comparison process aims to validate the accuracy and consistency of the ECM results obtained from ECMproject, providing means to verify its performance against the established ecmtool. By assessing the level of agreement between the two sets of results, researchers can gain confidence in the reliability and robustness of ECMproject for their specific applications (see 3.2 Validation).

To compare the ECM results obtained from ECMproject $\mathbf{v}$ with those from ecmtool $\mathbf{w}$, the model '*e_coli_core*', which describes the core metabolism of *E. coli*, was utilized. The comparison process was facilitated using the script '*validation.py*'. This script allows a comprehensive evaluation of the ECM results between the two tools.

The accuracy of the comparison can be adjusted using the available options. By default, the comparison script has a tolerance span ε of 1e-05. This tolerance level determines the acceptable difference between the ECM results from ECMproject and ecmtool for them to be considered as matching (Eq. 10). Adjusting this tolerance allows for flexibility in evaluating the similarity and agreement between the results.

$$||\mathbf{w} - \mathbf{v}||_2 \leq \varepsilon \tag{10}$$

2.7 Performance measurement

By capturing both the wall clock time and resident set size (RSS) usage, we can evaluate the efficiency and resource requirements of ECMproject and ecmtool, which enables us to make informed decisions regarding its usage and potential optimizations.

The evaluation of wall clock time and RSS usage is an important aspect of performance measurement in ECMproject and ecmtool. The wall clock time is measured using the Python package '*time*', which allows for tracking the execution time of the script. By utilizing the '*–time*' option of ECMproject, the total time taken by the script, as well

as the time taken by individual parts of the process, can be recorded and saved into a JSON file. To measure total time and execution time of different parts of ecmtool, time stamps were integrated. These modified ecmtool files save tracked times in the same way as ECMproject does.

Additionally, the RSS usage, which reflects the amount of memory consumed by the script, is measured using '*gnutime*'. This measurement provides insights into the memory requirements of ECMproject and ecmtool and aids in optimizing memory usage.

To conduct multiple repetitions of the benchmarking process and consolidate the results into a single JSON file, the '*benchmark_test.py*' script is used. This script enables the measurement of wall clock time and RSS across multiple runs and facilitates the analysis of the average performance.

3 RESULTS AND DISCUSSION

All results of ECMproject are created with parallelization of the postprocessing step by adding the '*–parallel*' option if not mentioned otherwise. Beside the '*e_coli_core*' model, we enumerated all ECMs in a genome-scale metabolic model (GSMM) of the synthetic minimal cell JCVI-syn3A created by Breuer et al. (2019). It contains 155 genes, 304 metabolites and 338 reactions of which 69 are external [Buchner et al., 2023]. This model, depending on the number of excess nutrients of the minimal medium, and hence number of external and internal reactions leads to the complexity levels ranging from 0 to 22. The models '*mmsyn_sm00*' up to '*mmsyn_sm22*' which are summarized in Table 4, were created to describe this complexity levels. All of these complexity models were provided by Buchner et al. (2023) and can be found on https://github.com/c-mayer/ECMproject.

3.1 Number of enumerated ECMs

Table 4 shows a breakdown of the most important components of each model and the resulting ECMs. The number of ECMs explodes with the increase of complexity. Each increase of the complexity level by one in this study means, that the synthetic minimal cell JCVI-syn3A got added two or three reactions, one of them being external and one or two metabolites (Table 4). The models of JCVI-syn3A ('*mmsyn_smxx*') increase not much by size but largely on complexity. With each increase by one in the model name, one external reaction is added leading to higher combinatorial possibilities of conversions. More biologically explained: Adding one component to the minimal medium of the synthetic minimal cell JCVI-syn3A leads to more metabolic possibilities.

Table 4: Size of the model and number of enumerated ECMs. With increasing number of external reactions, the number of ECMs grows very fast.

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

3.2 Validation

Table 5: Coverage of comparison of results between ECMproject and ecmtool on the '*e_coli_core*' model. A validation value of 100% indicates that all ECMs are identical within a certain tolerance which is shown in the first column. The column labeled 'decimal results' indicates the number of decimal digits rounded during the final step (see Section 2.5.3). The column 'rounding fractions' specifies the maximum denominator value chosen for rounding fractions. (see Section 2.2.3).

Tolerance	Decimals result	Rounding fractions	Validation [%]
1e-03	4	1e06	99.9
1e-04	4	1e06	12.5
1e-05	4	1e06	3.9
1e-03	5	1e06	100.0
1e-04	5	1e06	99.9
1e-05	5	1e06	88.2
1e-05	6	1e06	99.9
1e-06	6	1e06	12.9
1e-06	7	1e06	99.9
1e-07	7	1e06	11.3
1e-06	7	1e02	89.1
1e-06	7	1e03	99.9
1e-06	7	1e04	99.9
1e-06	7	1e15	99.9
1e-07	7	1e02	4.5
1e-07	7	1e03	8.3
1e-07	7	1e04	11.3
1e-07	7	1e15	11.3

The script '*validation.py*' was created to validate the results obtained from the '*e_coli_core*' model by comparing them with the outcomes generated by ecmtool (refer to Section 2.6). Table 5 presents the validation results, indicating that ECMproject can be considered successful with a coverage of approximately 99.9% when the validation tolerance (column "tolerance" in Table 5) is set one decimal lower than the decimals to be rounded (column "decimals result" in Table 5). The lower accuracy observed at stricter tolerances can be attributed to rounding errors in the final results when fewer digits are chosen (column "decimals result" in Table 5). Consequently, if greater 'precision' is desired, increasing the number of decimals rounded is recommended, as demonstrated in Table 5 with all the 99.9% results. One might expect, that rounded fractions as input for the H-Representation could introduce significant discrepancies. However, last lines in Table 5 demonstrate that this is not the case. Rounding errors only become apparent when the maximum denominator is set to 1e02 or 1e03, small values for rounding operations.

3.3 Optimizations of ECMproject

ECMproject underwent certain optimizations in its development. Most of these optimizations concerned the postprocessing step, but also projection underwent several iterations of optimization.

3.3.1 Projection standard vs mfel

As mentioned in Section 2.3, two different projection modes are available. Both got tested on the models '*mmsyn_sm05*' and '*mmsyn_sm10*' to proof which should be used further on. Figure 7 shows a distinct higher projection time on all process numbers except three, leading to a higher total running time if the "fel" algorithm of Avis (2022) is used. Interestingly, increasing the number of processes also increased projection time difference (Fig. 7). This effect starts for model '*mmsyn_sm05*' already on smaller process numbers than for model '*mmsyn_sm10*' which indicates that the chosen number of cores was too high at a certain point on each model and multiprocessing communication overhead raised. Since projection without the "fel" algorithm is faster when at least five processes are used, the standard projection mode with the projection loop directly implemented in the main script (starting ECMproject without '*–mfel*' option) got used for all further applications.

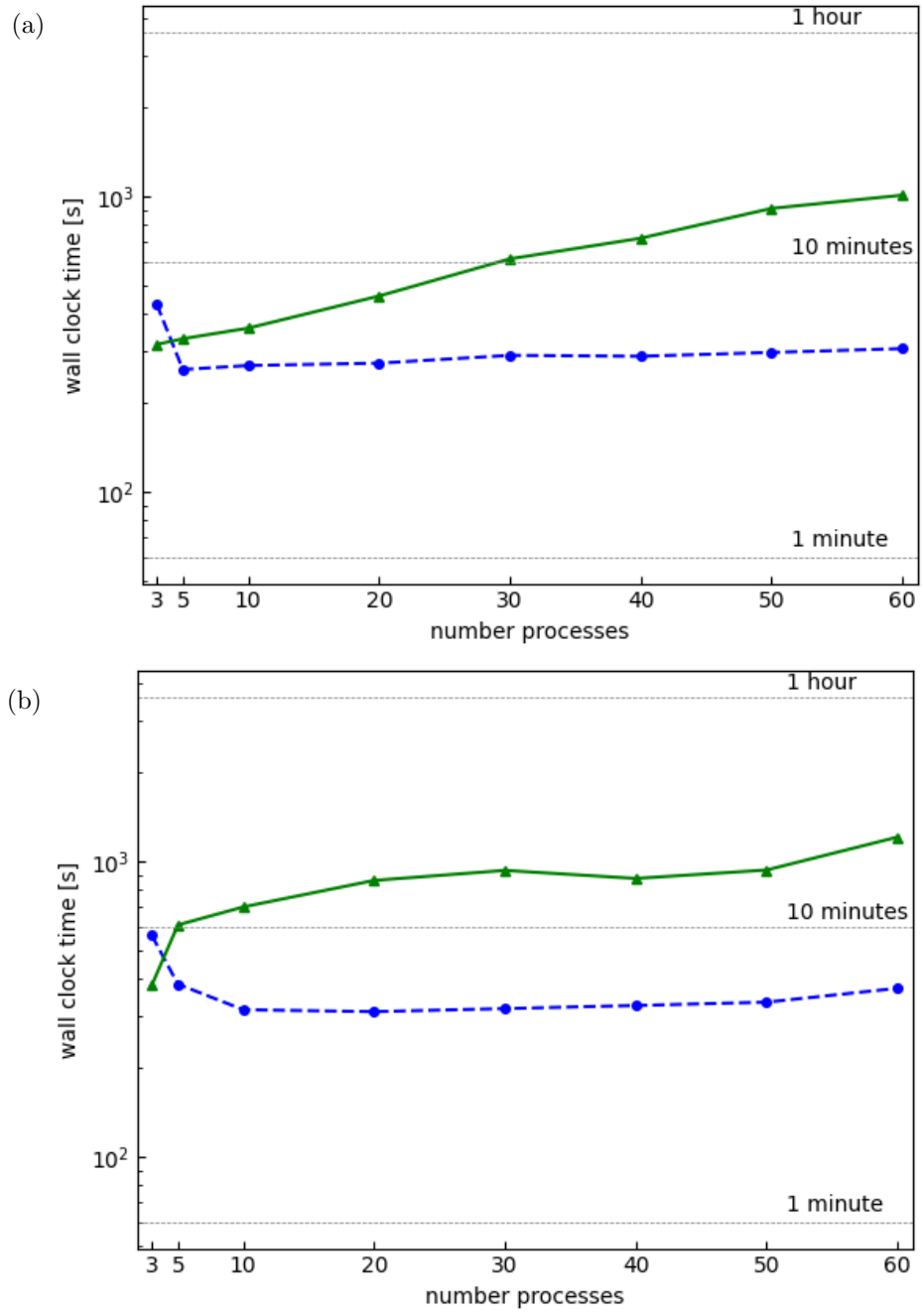

Figure 7: Average wall clock time of the projection step of ECMproject with the "fel" algorithm (green solid line) and without the "fel" algorithm (blue dashed line) as a function of used processes on the model **(a)** '*mmsyn_sm05*' and **(b)** '*mmsyn_sm10*'. Projection with the "fel" algorithm ('*–mfel*' option) takes more time than without. Three repeats where performed.

3.3.2 Parallelization of postprocessing

As already mentioned in Section 2.5, postprocessing of ECMproject can be performed with and without parallelization. A third variant is to perform parallelization of postprocessing with the '*Pool()*' function by using the '*–pool*' option (see Section 2.5). Time and memory performance of all three variants are shown in Figure 8, where 20 and 60 processes/cores were used for parallelization.

Figure 8a shows nearly equal time requirements for both parallel variants. On small models, which are fast computed, the '*–pool*' option is the faster parallelization variant since it has no waiting statement in the code. Nevertheless, the '*–pool*' option has ample RAM requirements in form of maximal RSS on greater model complexity (Fig. 8b) and therefore is not recommended for larger models. In default parallelization mode, the complexity of a model has no influence on maximal RSS requirements, making it usable even at servers with low RAM capacities.

Comparing both parallel versions with postprocessing without parallelization (single process), we can see a huge increase in wall clock time of the postprocessing step (Fig. 8a). For more complex models and therefore bigger V-representation files, running the postprocessing step in normal default parallel variant ('*-p*') is always recommended. Without parallelization, memory requirements are slightly higher as with default parallelization (Fig. 8b). On the minimal cell JCVI-syn3A with a complexity level of 22 (*mmsyn_sm22* corresponds to 316 reactions) default parallelization reaches a limit at 0.7 GB while without parallelization around 1 GB is reached. These RAM requirements might raise on more complex models. Nevertheless they are raising quite slow in comparison to parallelization with the '*–pool*' option enabled, which already reaches 93 GB on a complexity level of 18.

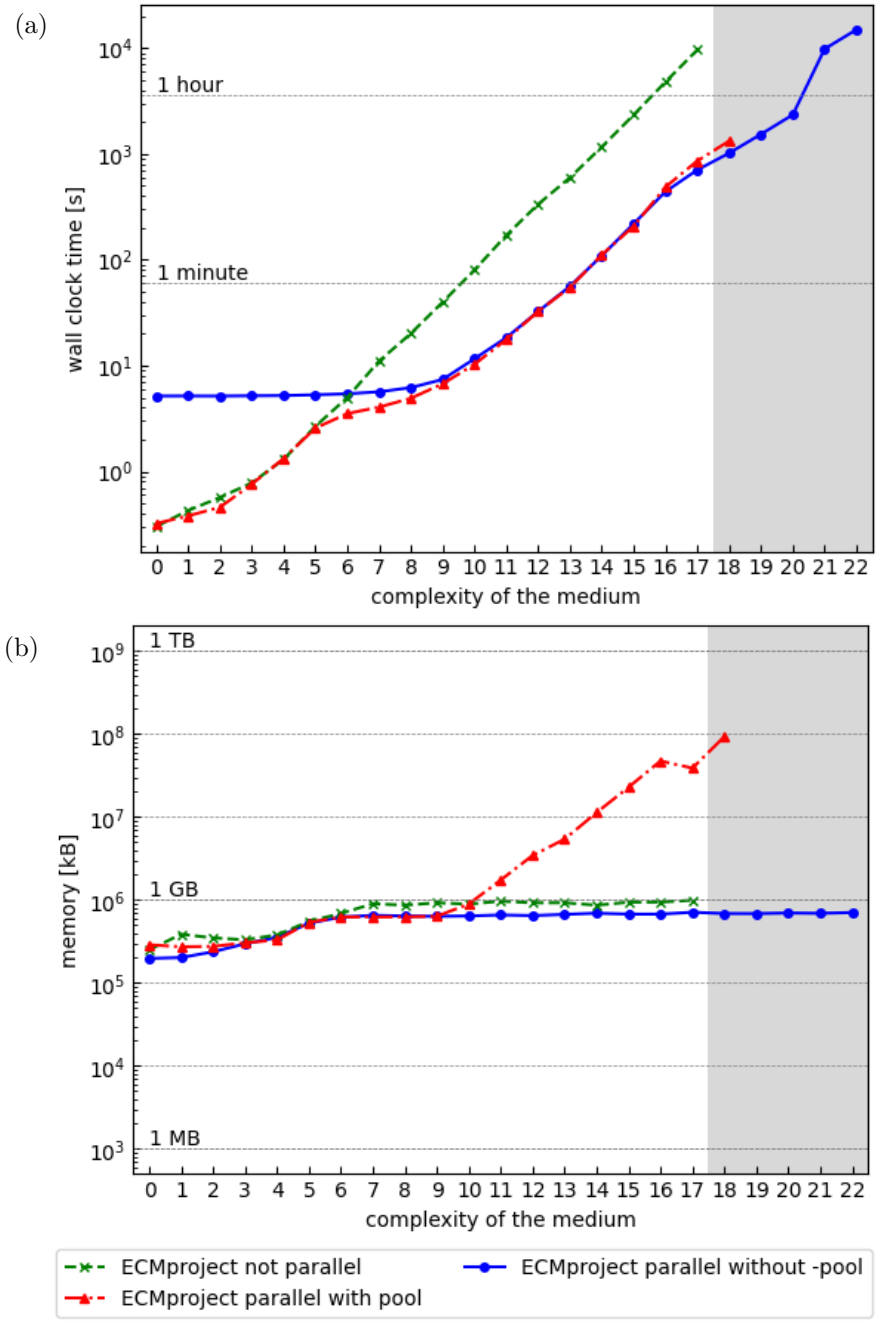

Figure 8: **(a)** Average postprocessing time and **(b)** max RSS of ECMproject in parallel mode with (red dash-dotted line) and without (blue solid line) '*–pool*' option and not in parallel mode (green dashed line) as a function of the complexity of the model. Complexity is defined as the number of excess nutrients in the minimal medium of JCVI-syn3A (see Table 4 for more details). 20 and 60 (shaded area) cores/processes were used. For complexity levels 0-20, three repeats were performed, while for 21 and 22 just one was performed.

3.3.3 Influence of used number of processes

One can choose the number of processes running ECMproject on several CPUs with the '*–n_processes*' option. In Figure 9, we can see the influence of chosen number of processes on total wall clock time of different models of complexity of the minimal cell JCVI-syn3A. For models with up to 66 external metabolites ('*mmsyn_sm10*'), using more than 30 processes does not bring any improvement of the computational time. Nevertheless, high numbers of processes show an effect on more complex models where more computational power is needed. The increase of processes from 10 to 60 decreases the wall clock time on a complexity level of 15 (see Table 4 for more details) by a factor of 5. Therefore, on more complex models, a high number of processes is recommended.

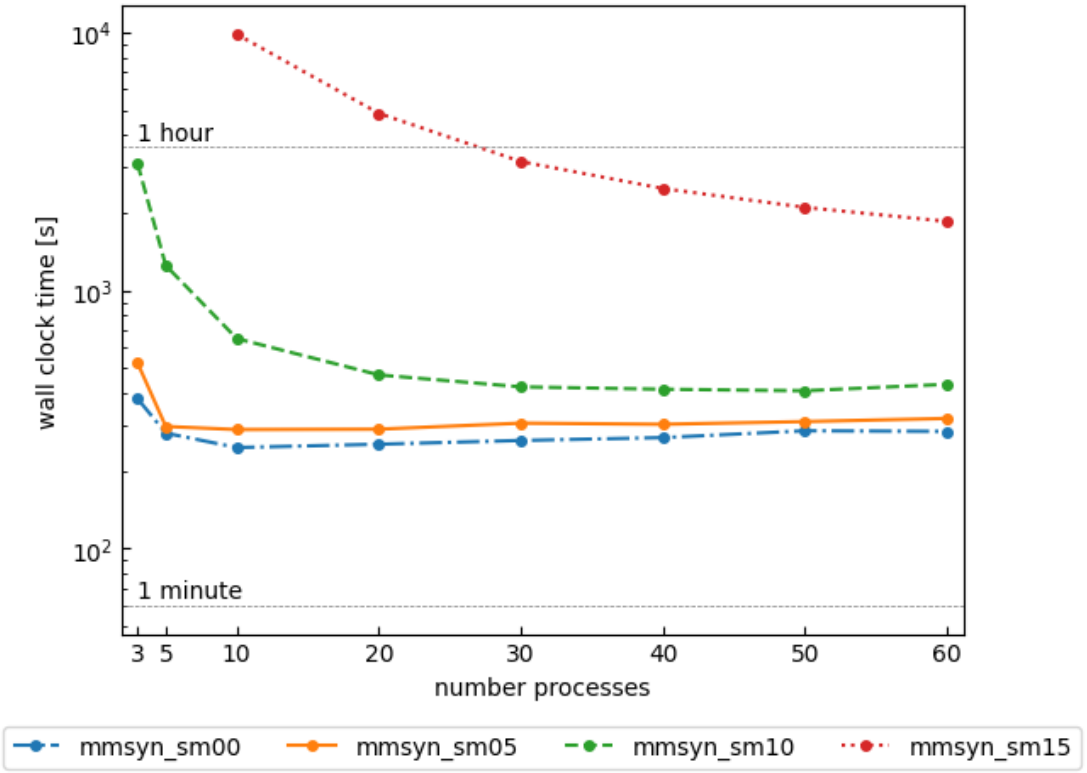

Figure 9: Comparison of different models/levels of complexity of the minimal cell JCVI-syn3A (see Table 4 for more details) on the average total wall clock time influenced by the number of processes on ECMproject. Average wall clock time decreases with increasing number of processes, especially on more complex models. Three repeats where performed.

Postprocessing and mplrs are parallelized, which lead to faster times of vertex enumeration and postprocessing at high numbers of processes (Fig. 10). Wall clock time of projection is not influenced by the number of processes. This implies that '*mplrs project*' is not really parallelized. Since each iteration has to generate one file which the next iteration has to access, parallelization seems not possible in terms of splitting iterations in subproblems to parallelize them like vertex enumeration did (see Section 1.4).

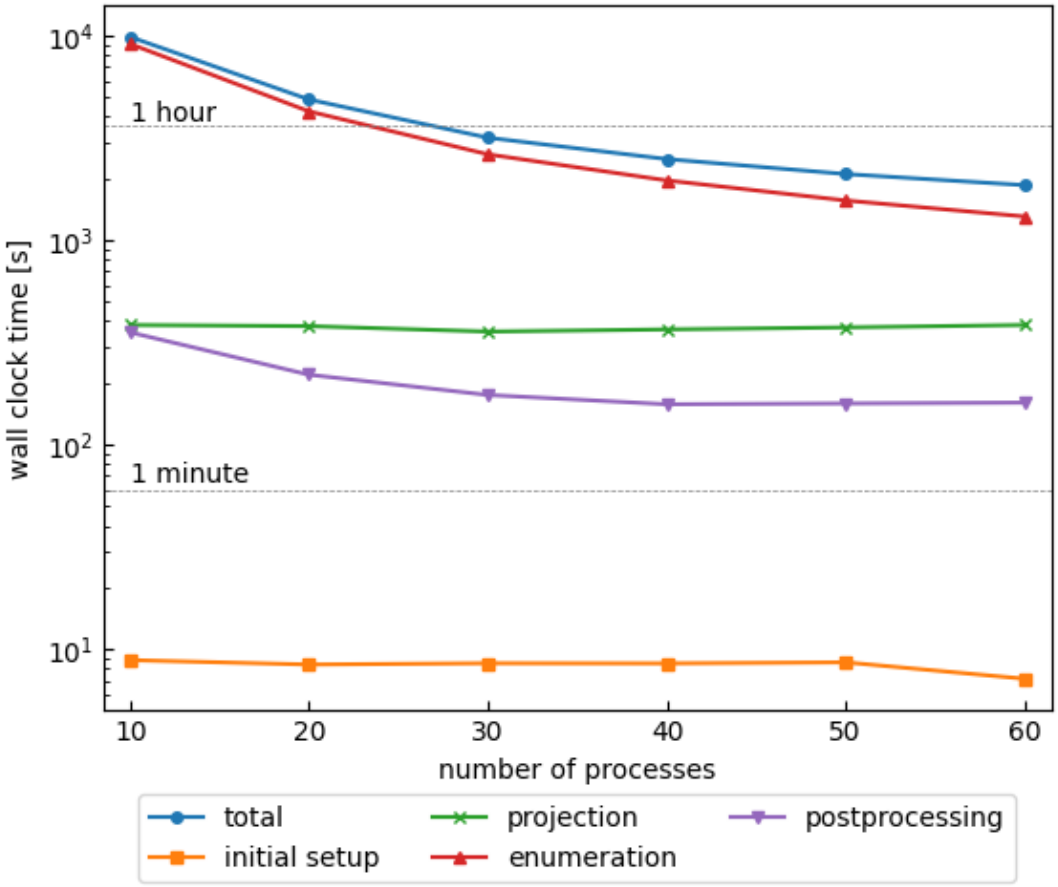

Figure 10: Comparison of the computational time for different steps of ECMproject on the synthetic minimal cell JCVI-syn3A with the complexity of the model *mmsyn_sm15*. Projection and preprocessing (initial setup) are not influenced by the number of processes. Enumeration and postprocessing are influenced. Especially the reduction of wall clock time in the enumeration step reduces total time. Three repeats where performed.

In terms of memory, the chosen number of processes had no clear effect on maximal RSS (Fig. 11). Maximal RSS only increased to a certain point with growing model complexity, which also can be seen in Figure 12b.

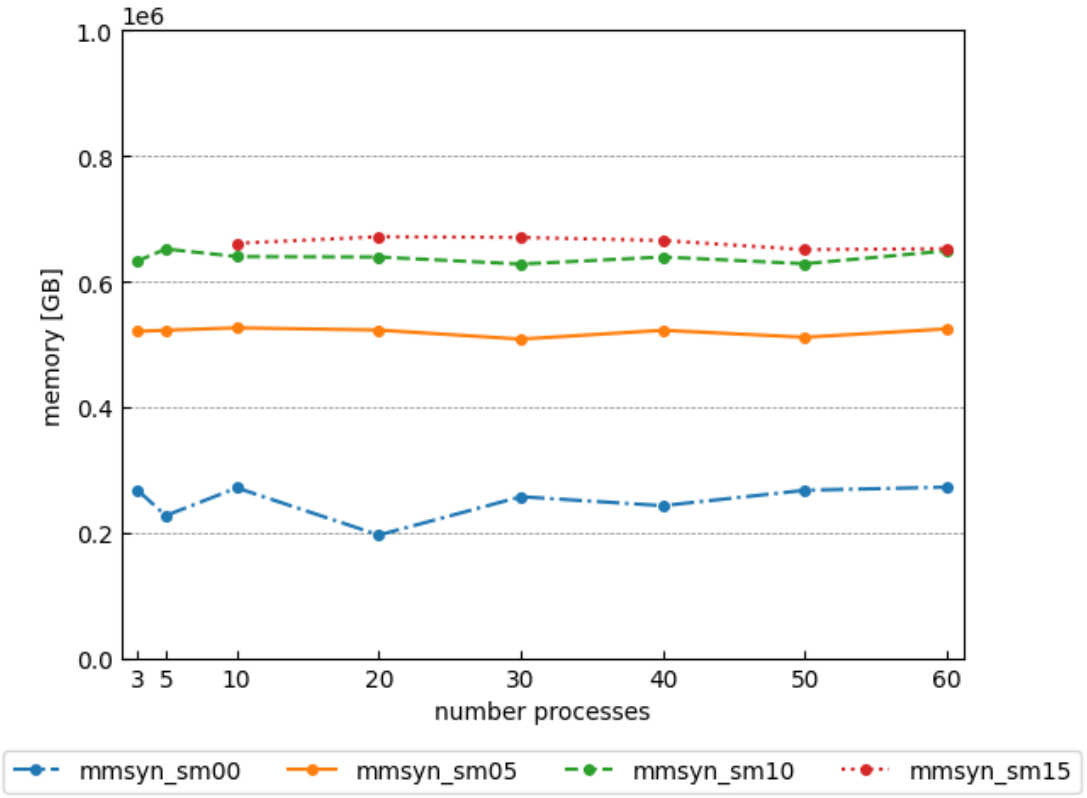

Figure 11: Comparison of the memory consumption for different models/levels of complexity on the synthetic minimal cell JCVI-syn3A. Maximal RSS is shown as a function of the number of used processes on ECMproject. Maximal RSS is not influenced by the number of processes. Three repeats where performed.

3.3.4 Zipped against unzipped mode

Since ECM results of more complex models can get very big in terms of file size, we implemented an option which compresses the file size (*–gzipped* option). Compression of data always needs to be done from the same process in multiprocessing (writing xxx.gz file with more than one process corrupts the file). Therefore, compressed writing of the

result file in multiprocessing mode is done by just one process, which slows down the postprocessing step. For a complexity level of 15 ('*mmsyn_sm15*), time of postprocessing rises from 219 to 3920 seconds (see Table 11). The zipped result file has a size of 1.1 GB while the unzipped result file has a size of 35.7 GB (see Table 11). High decrease of storage demands comes with high increase of postprocessing time.

3.4 Performance comparisons of ECMproject to ecmtool

3.4.1 Total time and max RSS

Total wall clock time of ECMproject and ecmtool increases with complexity level of the model (Fig. 12a), since an increased number of external reactions leads to more external metabolites, therefore to more ECMs and therefore to higher computational time. The exact number of enumerated ECMs for each complexity level can be seen in Table 4.

Ecmtool is faster on models with low complexity levels, therefore with low computational requirements (Fig. 12a). Since the time difference is always below five minutes, this speed advantage only is needed if many small models have to be calculated in high throughput. On a complexity level of 9 ('*mmsyn_sm09*' corresponds to 65 extern reactions), this speed advantage is already gone (Fig. 12a). With increasing model complexity, ECMproject gets faster and faster compared to ecmtool being 6 times faster on a complexity level of 14 and already 8 times faster on a complexity level of 17. From complexity level 18 upwards, all computations were performed with 60 instead of 20 processes, which explains the jump of the blue solid line in Figure 12a. With this process increase, ECMproject got even faster in comparison to ecmtool (Fig. 12a) being 13 times faster on a complexity level of 18. These factors are vast time savings considered the high total wall clock times of both tools on complex models. ECMproject needs just half of a day to compute a complexity level of 19 while ecmtool needs around ten days for the same computation. For maximal RSS, both tools are very efficient by just needing RAM below 1 GB (Fig. 12b). ECMproject has a stable maximal RSS usage while the usage of ecmtool is lower but varies more between different complexity levels.

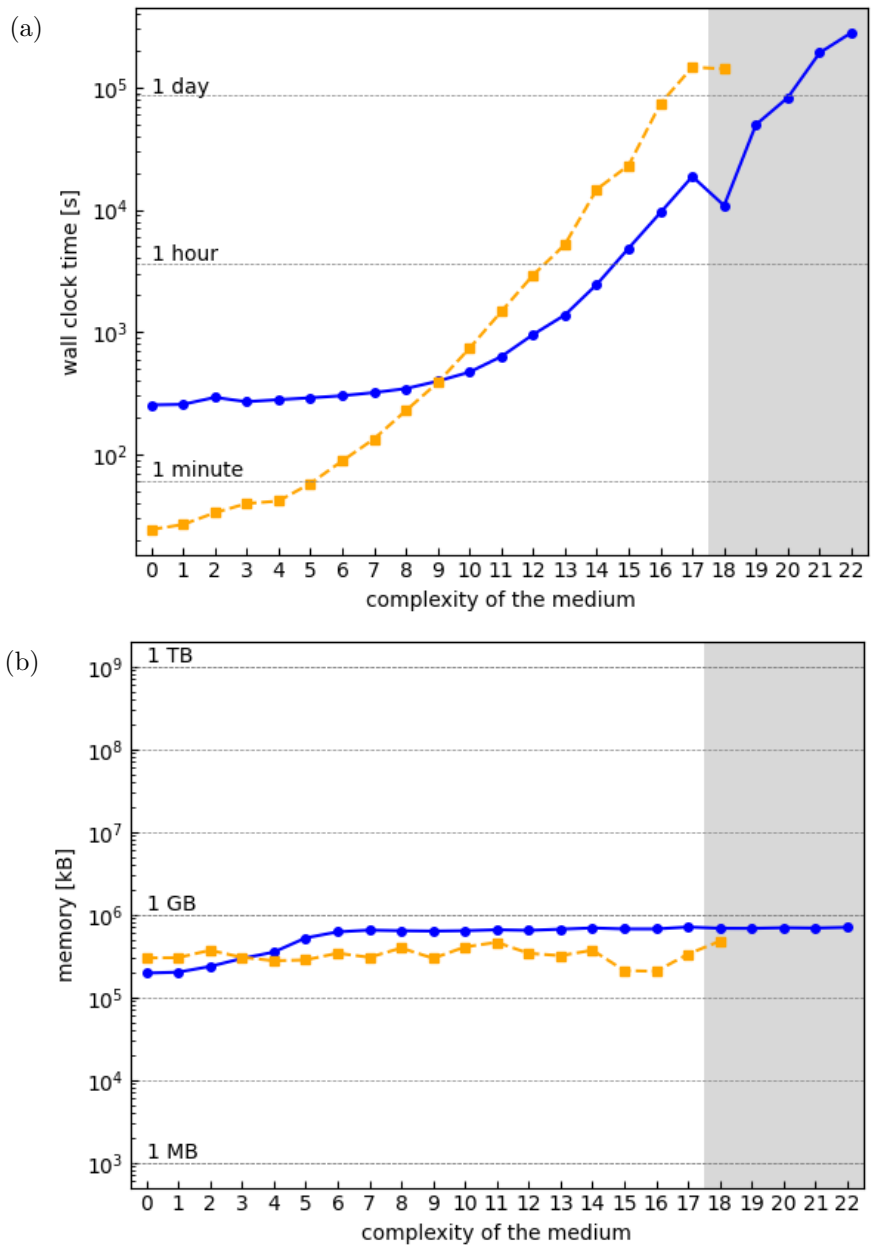

Figure 12: **(a)** Total time and **(b)** maximal resident set size (RSS) of ECMproject (blue solid line) and ecmtool (orange dashed line) as a function of the complexity of the model. Complexity is defined as the number of excess nutrients in the minimal medium of JCVI-syn3A (see Table 4 for more details). 20 and 60 (shaded area) threads/processes were used. For complexity levels 0-20, three repeats were performed, while for 21 and 22 just one was performed.

3.4.2 Performance of certain steps

Each tool got timestamps to be able to measure certain steps. These steps got differentiated like in scheme of Figure 4 in Section 1.7. In Figure 13, projection of ECMproject gets compared with first vertex enumeration (VE1) and intermediate processing of ecmtool (Fig. 13a). Vertex enumeration (VE) of ECMproject gets compared with the second vertex enumeration (VE2) of ecmtool (Fig. 13b), since both these steps convert an H-representation with implemented steady state conditions to a V-representation. The created V-representation gets converted into ECMs in the postprocessing steps of both tools.

As shown in Figure 13a, the projection of ECMproject needs constant more time than the first vertex enumeration and intermediate processing of ecmtool together. It also does not increase by much with increasing complexity of the model.

The vertex enumeration of ECMproject on a low complexity level of 5 (see Table 4) is 3 times faster than the second vertex enumeration of ecmtool (Fig. 13b). On a complexity level of 15, VE of ECMproject is 3.8 times faster than VE2 of ecmtool. With 60 processes on a complexity level of 18, VE of ECMproject is 9.6 times faster than VE2 of ecmtool. These differences are quite substantial since VE of ECMproject and VE2 of ecmtool are the most time-consuming steps on smaller but quite complex models. Actually, most of the time savings from ECMproject relative to ecmtool are the result of this circumstance if Figure 13b gets compared with Figure 12a.

Wall clock times of the postprocessing steps are shown in Figure 14. On low complexity levels, ecmtool is faster than ECMproject, leading (beside the faster VE1 and intermediate processing) to a faster total time of ecmtool compared to ECMproject (Fig. 12a). The switch lies between complexity levels 4 and 5. From this point on, ECMproject has the faster postprocessing time (Fig. 14) up to a factor of 50 on a complexity level of 18.

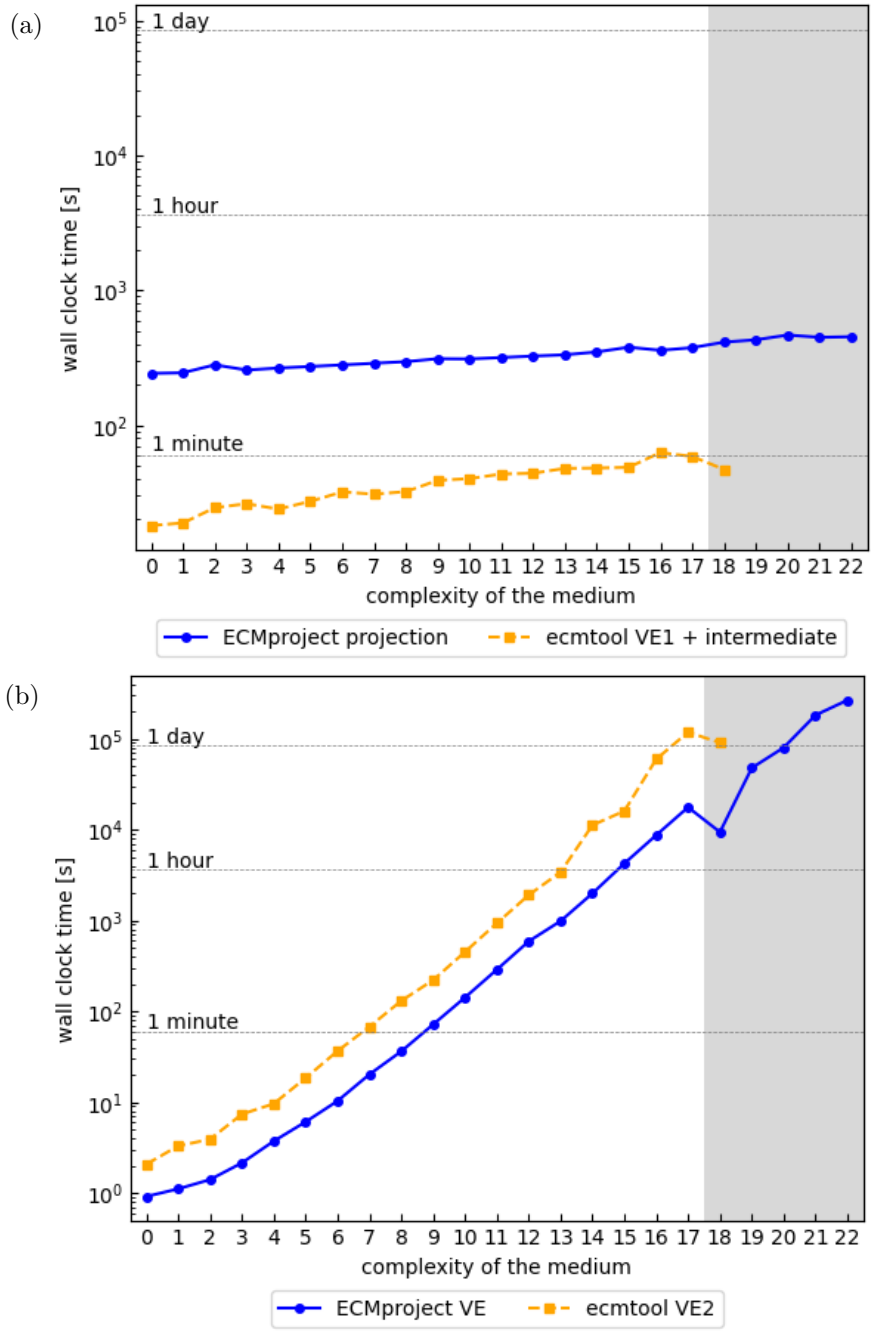

Figure 13: **(a)** Projection of ECMproject vs first vertex enumeration (VE1) and intermediate processing of ecmtool. **(b)** Vertex enumeration (VE) of ECMproject vs second vertex enumeration of ecmtool. Both subplots are functions of clock time against complexity of the model. Complexity is defined as the number of excess nutrients in the minimal medium of JCVI-syn3A (see Table 4 for more details). 20 and 60 (shaded area) threads/processes were used. For complexity levels 0-20, three repeats were performed, while for 21 and 22 just one was performed.

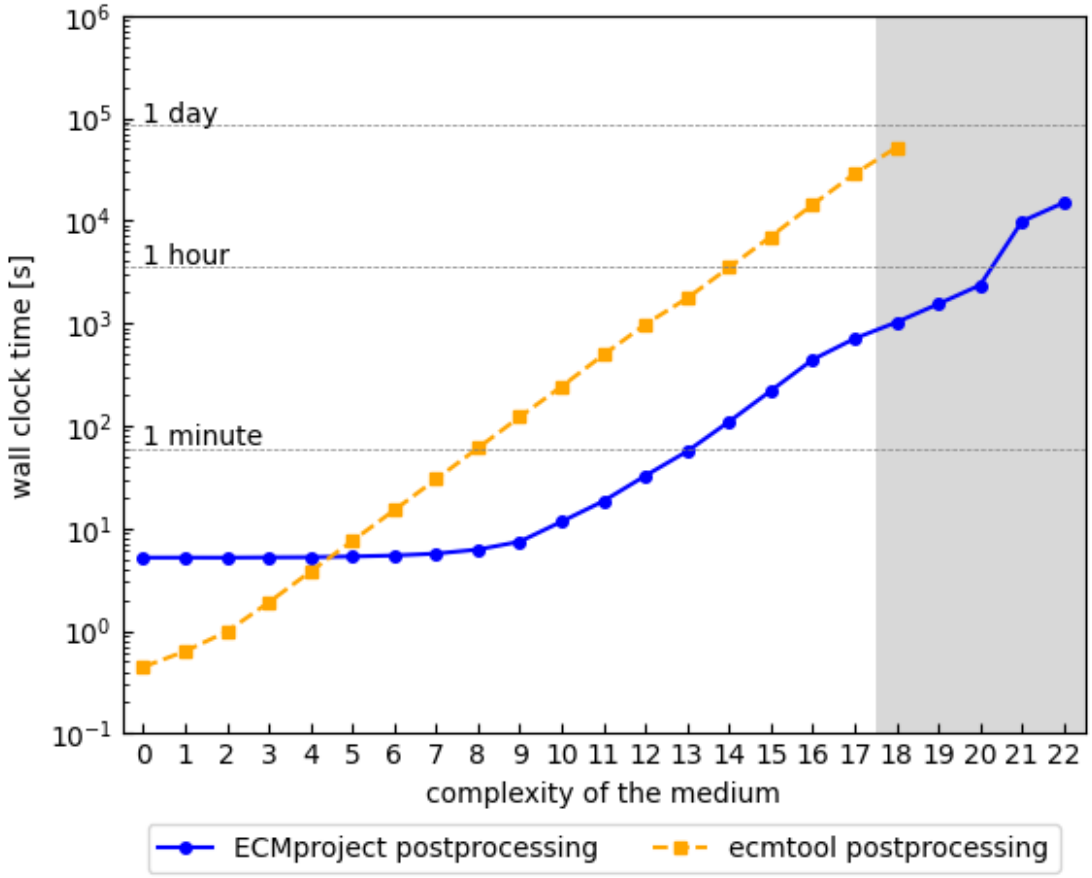

Figure 14: Postprocessing of ECMproject (blue solid line) and ecmtool (orange dashed line) as a function of the complexity of the model. Complexity is defined as the number of excess nutrients in the minimal medium of JCVI-syn3A (see Table 4 for more details). 20 and 60 (shaded area) cores/processes were used. For complexity levels 0-20, three repeats were performed, while for 21 and 22 just one was performed.

3.4.3 Influence of the number of processes on wall clock time of both tools

Total wall clock time between ECMproject and ecmtool got compared on different numbers of processes for parallelization on the synthetic minimal cell JCVI-syn3A with a complexity level of 15. With increasing number of processes, ecmtool decreases faster in

wall clock time than ECMproject does (Fig. 15). Both tools reach a valley at a certain number of processes. With higher complexity of the model, higher number of processes lead to a decrease of wall clock time before reaching a valley (shown in Figure 9 for ECMproject).

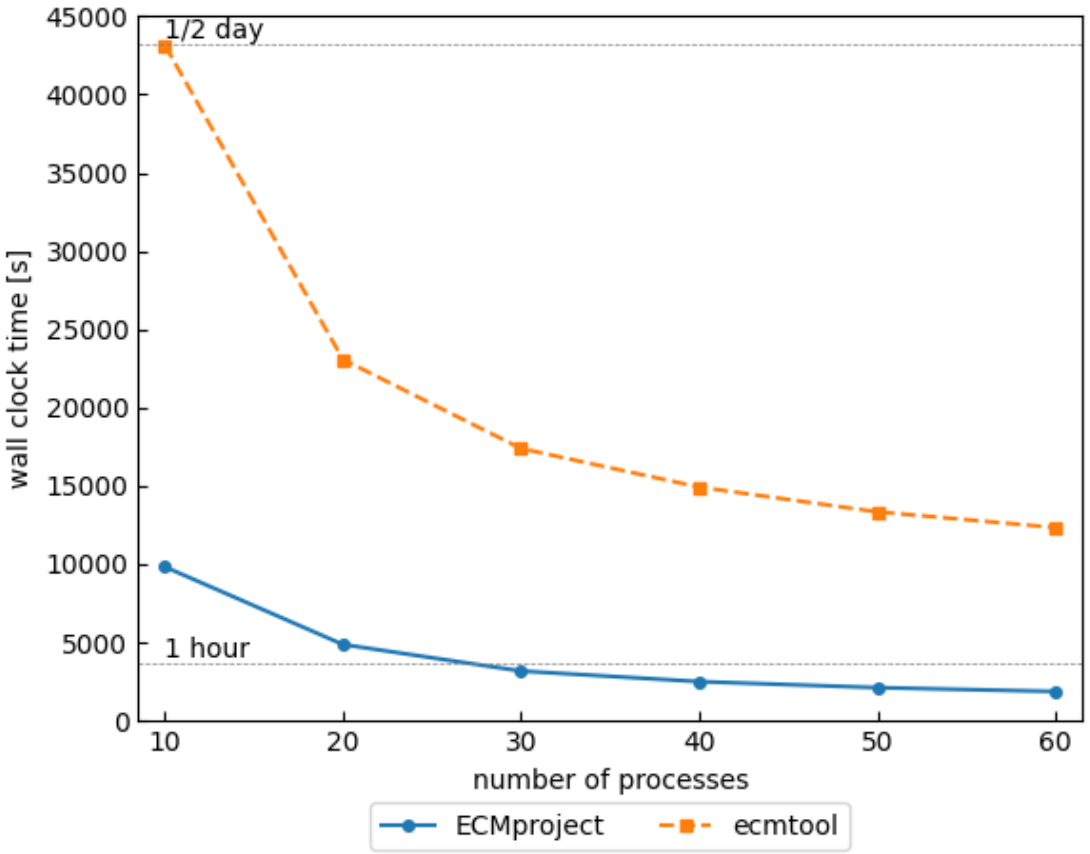

Figure 15: Comparison of average total wall clock time between ECMproject (blue solid line) and ecmtool (orange dashed line) on a minimal medium of JCVI-syn3A with the complexity of the model *mmsyn_sm15*. Total time of ecmtool is stronger influenced on the number of processes than total time of ECMproject. Three repeats where performed.

4 CONCLUSION

A new tool called ECMproject was developed in this Master Thesis which uses the mplrs algorithm with the newly developed '*mplrs project*' function to enumerate ECMs more efficiently and which enables the full enumeration of models currently too extensive to analyse. ECMproject got validated and compared with ecmtool, the fastest tool for complete ECM enumeration of entire metabolic models.

The validation process involved a meticulous examination of each ECM generated by ecmtool, compared with the corresponding ECM produced by ECMproject, particularly focusing on the core model of *E. Coli*. On all models of complexity of the minimal cell JCVI-syn3A, which corresponds to a range of 56 to 78 extern metabolites, the number of ECMs enumerated was the same for ECMproject and ecmtool.

Different options of ECMproject were tested on the core model of *E. coli* and the models of the minimal cell JCVI-syn3A. For users concerned about the size of result files, the '–gzipped' option provides a 32-fold reduction of storage requirements albeit postprocessing time has an 18-fold increase. It is important to note, that the "fel" algorithm of Avis (2022) ('*–mfel*' option) did not demonstrate superior performance compared to the default projection algorithm calling mplrs directly in the script and is therefore not recommended. The same applies to the parallelization algorithm with the '*Pool()*' function ('*–pool*' option) in the postprocessing step. The default parallelization of the postprocessing step ('*-p*') is the recommended choice, when the hardware can support the required number of cores. Using a greater number of cores for multiprocessing significantly expedites the enumeration (conversion) and postprocessing steps, particularly for more extensive models. All ECMs of the minimal cell JCVI-syn3A with a complexity

level of 15 ('*mmsyn_sm15*' with 296 reactions) get enumerated five times faster, when the number of processes (used cores) gets increased from 10 to 60.

A comparison of total wall clock time and RSS between ECMproject and ecmtool was undertaken. Both tools exhibited stable RSS usage, indicating consistent RAM requirements across the whole range of models of complexity of the minimal cell JCVI-syn3A and therefore indicating low RAM requirements even for big models. The primary limiting factor remains the total wall clock time. As the number of exchange reactions increases, the count of ECMs grows exponentially, making the vertex enumeration step progressively more time-consuming. An increase in the number of internal reactions prolongs the projection step in ECMproject.

On small scale models (up to 95 internal and external reactions, e.g. *E. Coli* core model), ecmtool exhibits a slight advantage over ECMproject in terms of speed (11 s computation time instead of 100 s for *E. coli* core model). This time advantage only gets substantial, if many small models are analysed. However, as model complexity and size increases, ECMproject demonstrates superior scalability and enables enumeration of models which ecmtool is not feasible of in a reasonable time scale. For example, total wall clock time of ECM enumeration of the minimal cell JCVI-syn3A with 305 reactions ('*mmsyn_sm18*') gets reduced from 39 h to 3 h if ECMproject is used instead of ecmtool (both on 60 processes).

The improved scalability of ECMproject primarily stems from the more efficient scaling of its enumeration step compared to the second vertex enumeration step in ecmtool. Additionally, ECMproject's postprocessing step exhibits better scalability in contrast to ecmtool's postprocessing procedure.

Nonetheless, even with ECMproject's enhanced scalability, computational demands increase exponentially with model size. To address the analysis of even more extensive models describing for instance the whole metabolism of a single-cell organism such as yeast or *E.coli*, further enhancements to ecmtool, ECMproject, or the introduction of novel computational concepts are essential.

Bibliography

Avis, D. (1998). Computational experience with the reverse search vertex enumeration algorithm. *Optimization Methods & Software*, *10*, 107–124. https://api.semanticscholar.org/CorpusID:8347988

Avis, D. (2022). Users guide for lrs - version 7.2 [Accessed: 2023-10-04]. http://cgm.cs.mcgill.ca/~avis/C/lrslib/USERGUIDE72.html#fourier

Avis, D., & Fukuda, K. (1992). A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computational Geometry*, *8*(3), 295–313. https://doi.org/10.1007/BF02293050

Avis, D., & Jordan, C. (2018). mplrs: A scalable parallel vertex/facet enumeration code. *Mathematical Programming Computation*, *10*(2), 267–302. https://doi.org/10.1007/s12532-017-0129-y

Breuer, M., Earnest, T. M., Merryman, C., Wise, K. S., Sun, L., Lynott, M. R., Hutchison, C. A., Smith, H. O., Lapek, J. D., Gonzalez, D. J., de Crécy-Lagard, V., Haas, D., Hanson, A. D., Labhsetwar, P., Glass, J. I., & Luthey-Schulten, Z. (2019). Essential metabolism for a minimal cell (Z. Nikoloski & N. Barkai, Eds.). *eLife*, *8*, e36842. https://doi.org/10.7554/eLife.36842

Buchner, B. A., Clement, T. J., de Groot, D. H., & Zanghellini, J. (2023). ecmtool: fast and memory-efficient enumeration of elementary conversion modes. *Bioinformatics*, *39*(3), btad095. https://doi.org/10.1093/bioinformatics/btad095

Buchner, B. A., & Zanghellini, J. (2021). EFMlrs: a Python package for elementary fux mode enumeration via lexicographic reverse search. *BMC Bioinformatics*, *22*(1), 547. https://doi.org/10.1186/s12859-021-04417-9

Clement, T. J., Baalhuis, E. B., Teusink, B., Bruggeman, F. J., Planque´, R., & de Groot, D. H. (2021). Unlocking Elementary Conversion Modes: ecmtool Unveils All Capabilities of Metabolic Networks. *Patterns*, *2*(1). https://doi.org/10.1016/j.patter.2020.100177

Hucka, M., Bergmann, F. T., Chaouiya, C., Dräger, A., Hoops, S., Keating, S. M., König, M., Novère, N. L., Myers, C. J., Olivier, B. G., Sahle, S., Schaff, J. C., Sheriff, R., Smith, L. P., Waltemath, D., Wilkinson, D. J., & Zhang, F. (2019). The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 2 Core Release 2. *Journal of Integrative Bioinformatics*, *16*(2). https://doi.org/10.1515/jib-2019-0021

King, Z. A., Lu, J., Dräger, A., Miller, P., Federowicz, S., Lerman, J. A., Ebrahim, A., Palsson, B. O., & Lewis, N. E. (2016). BiGG Models: A platform for integrating, standardizing, and sharing genome-scale models. *Nucleic Acids Research*, *44*(D1), D515–D522. https://doi.org/10.1093/nar/gkv1049

Klamt, S., Regensburger, G., Gerstl, M. P., Jungreuthmayer, C., Schuster, S., Mahadevan, R., Zanghellini, J., & Müller, S. (2017). From elementary flux modes to elementary flux vectors: Metabolic pathway analysis with arbitrary linear flux constraints. *PLOS Computational Biology*, *13*(4), e1005409. https://doi.org/10.1371/journal.pcbi.1005409

Mendoza, S. N., Olivier, B. G., Molenaar, D., & Teusink, B. (2019). A systematic assessment of current genome-scale metabolic reconstruction tools. *Genome Biology*, *20*(1), 158. https://doi.org/10.1186/s13059-019-1769-1

Motzkin, T. S., Raiffa, H., Thompson, G. L., & Thrall, R. M. (1953). *The double description method: Contributions to the theory of games (am-28), volume ii*. Princeton University Press. https://doi.org/10.1515/9781400881970-004

Orth, J. D., Thiele, I., & Palsson, B. Ø. (2010). What is flux balance analysis? *Nature Biotechnology*, *28*(3), 245–248. https://doi.org/doi.org/10.1038%2Fnbt.1614

Raman, K., & Chandra, N. (2009). Flux balance analysis of biological systems: applications and challenges. *Briefings in Bioinformatics*, *10*(4), 435–449. https://doi.org/doi.org/10.1093/bib/bbp011

Terzer, M., & Stelling, J. (2008). Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, *24*(19), 2229–2235. https://doi.org/10.1093/bioinformatics/btn401

Zanghellini, J., Ruckerbauer, D. E., Hanscho, M., & Jungreuthmayer, C. (2013). Elementary flux modes in a nutshell: Properties, calculation and applications. *Biotechnology Journal*, *8*(9), 1009–16. https://doi.org/10.1002/biot.201200269

List of Figures

List of Tables

Appendix

Data and Code Availability

The source code for ecmtool is freely available on GitHub at https://github.com/c-mayer/ECMproject. A manual can be found by using the help page of ECMproject. Additionally all options of the program are listed in this Appendix. All of the created time and memory data can be found at https://github.com/c-mayer/ECMproject in the format of JSON and CSV. Additionally, all mean values are shown in tables below in this Appendix.

Extra Figures

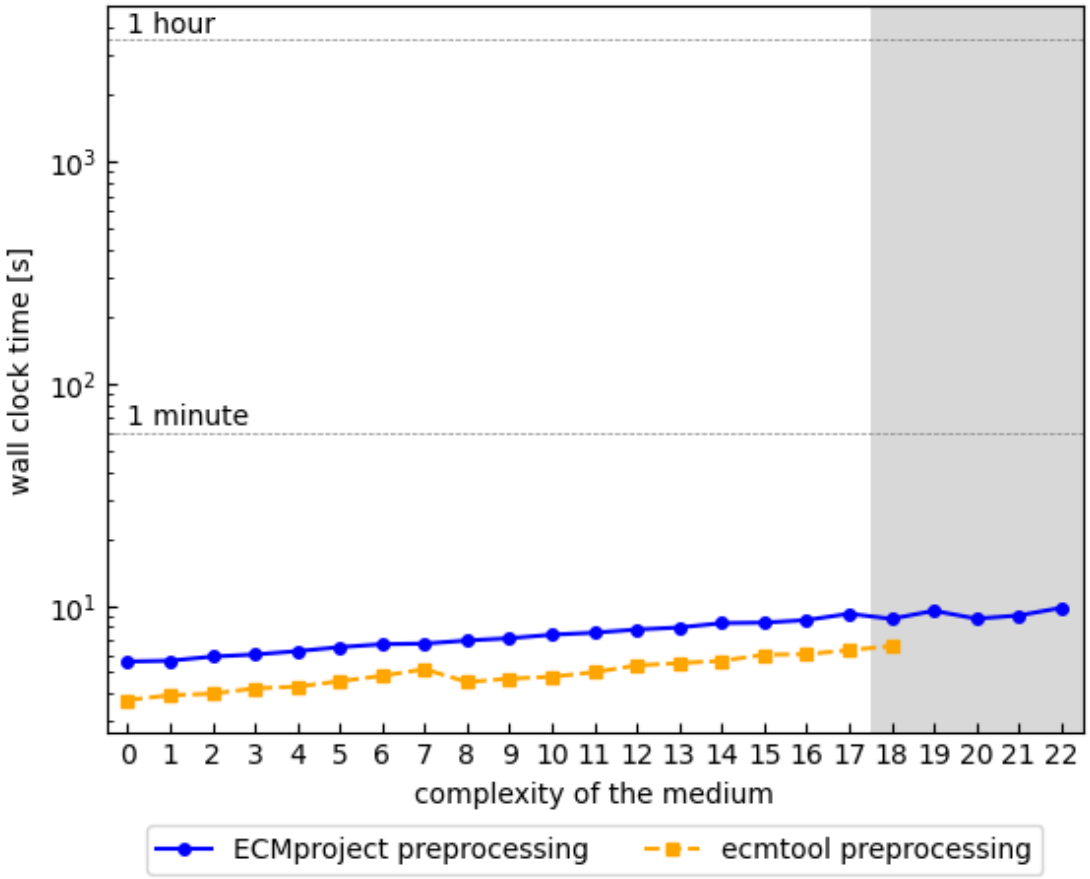

Figure 16: Preprocessing of ECMproject and ecmtool as a function of the complexity of the model. Complexity is defined as the number of excess nutrients in the minimal medium of JCVI-syn3A (see Table 4). 20 and 60 (shaded area) threads/processes were used. For complexity levels 0-20, three repeats were performed, while for 21 and 22 just one was performed.

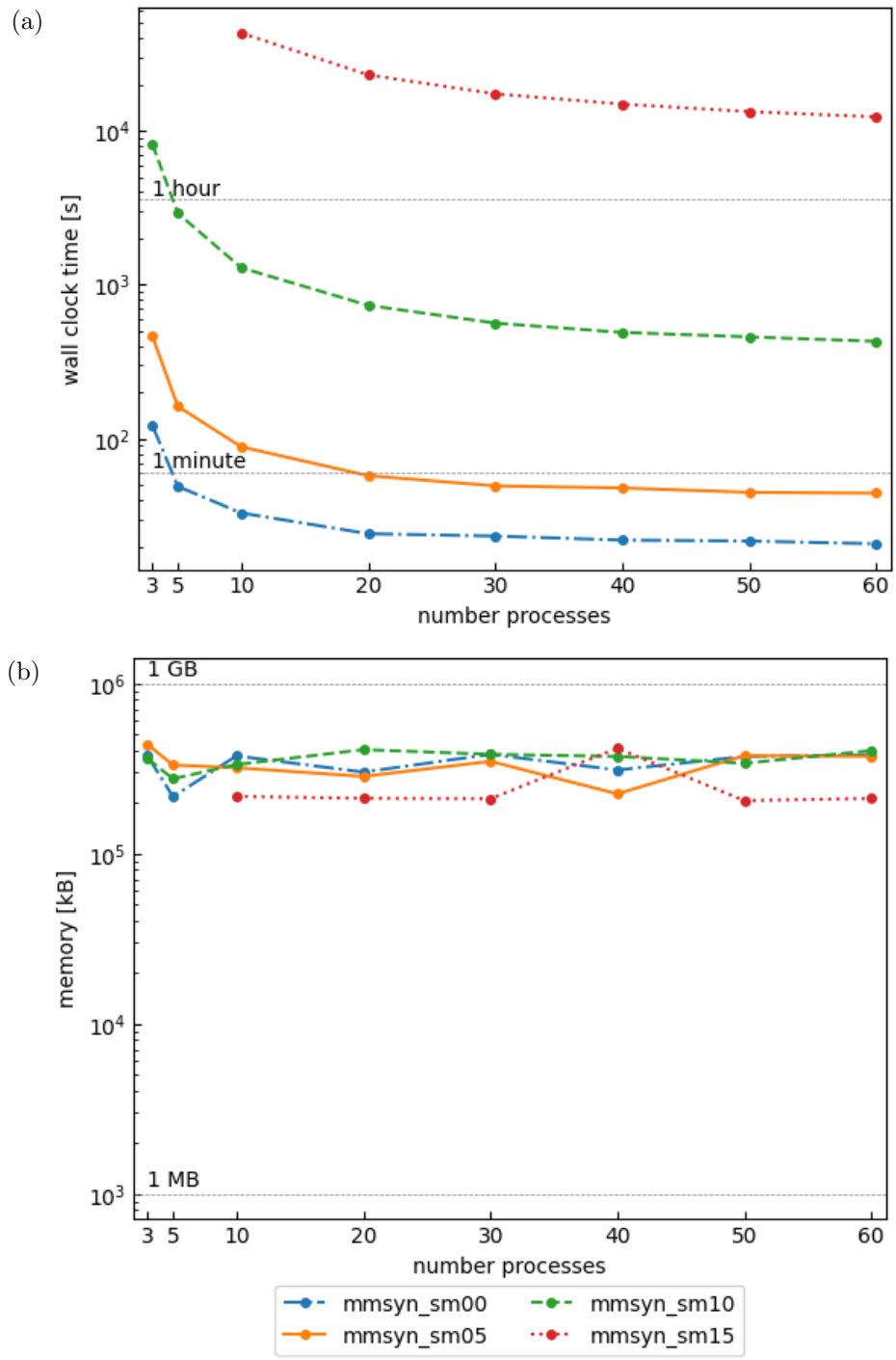

Figure 17: Comparison of different models of complexity of the minimal cell JCVI-syn3A on **(a)** total wall clock time and **(b)** maximal resident set size (RSS) influenced by the number of processes on ecmtool. On mmsyn_sm15, only one repeat was performed. On all other models, three repeats were performed.

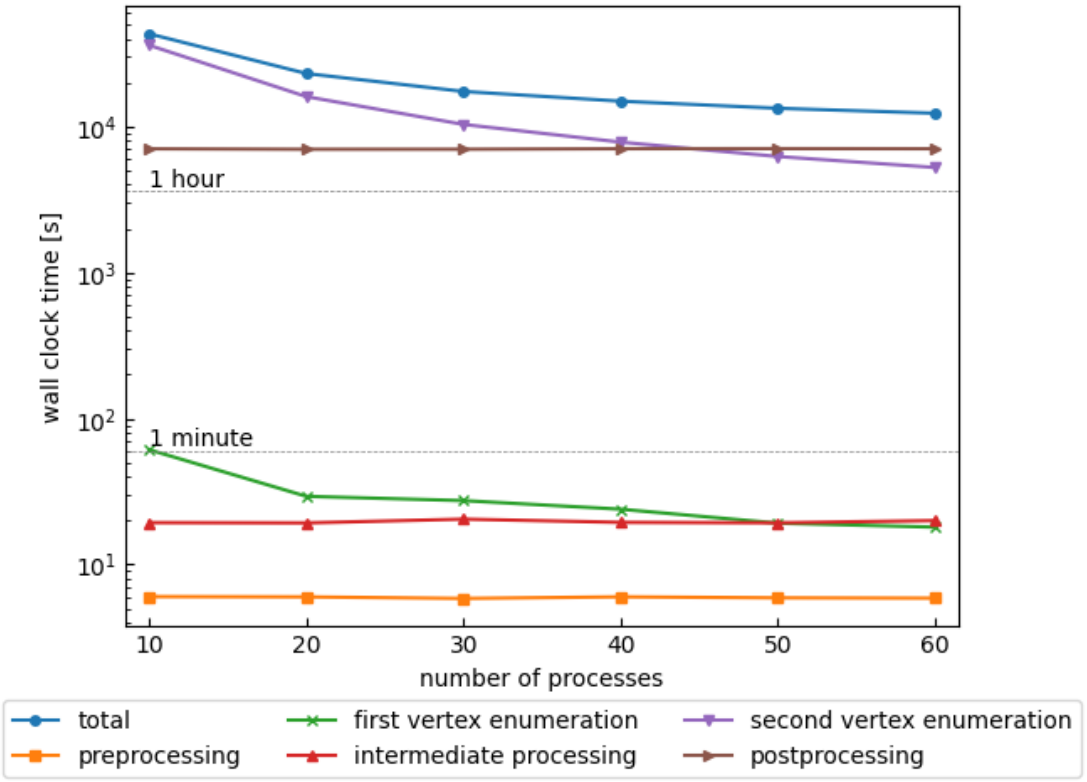

Figure 18: Comparison of different steps of ecmtool on the synthetic minimal cell JCVI-syn3A with the complexity of the model mmsyn sm15. Three repeats where performed.

Data Tables

All data tables are showing mean values. If all values are wanted lookup https://github.com/c-mayer/ECMproject/tree/main/performance.

Table 6: Settings & Results of ECMproject in default parallel postprocessing mode.

models	cores	total	initial setup	pro-jection	enumer-ation	postpro-cessing	ECM count	max_rss [GB]
e_coli_core	3	87.4	1.7	81.0	4.6	0.1	689	0.329
	5	101.3	1.5	97.5	2.2	0.1	689	0.329
	10	117.2	1.5	113.6	1.9	0.2	689	0.332
	20	100.1	1.5	96.8	1.6	0.2	689	0.328
	30	98.6	1.6	95.0	1.8	0.2	689	0.330
	40	124.9	1.6	121.6	1.4	0.3	689	0.330
	60	102.4	1.6	98.9	1.5	0.3	689	0.330
mmsyn_sm00	3	381.5	17.8	354.8	3.8	5.1	4132	0.268
	5	279.9	8.4	265.0	1.4	5.1	4132	0.227
	10	246.1	5.6	234.4	1.0	5.1	4132	0.272
	20	253.5	5.6	241.9	0.9	5.2	4132	0.196
	30	262.2	5.7	250.4	0.9	5.2	4132	0.257
	40	268.9	5.7	257.1	0.9	5.2	4132	0.243
	50	285.8	4.6	275.1	0.9	5.2	4132	0.268
	60	284.3	4.7	273.4	0.9	5.2	4132	0.273
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	3	525.3	23.4	430.8	66.0	5.2	77546	0.521
	5	297.0	10.1	258.9	22.7	5.3	77546	0.522

S & R of ECMproject in default parallel postprocessing mode. (continued)

models	cores	total	initial setup	pro-jection	enumer-ation	postpro-cessing	ECM count	max_rss [GB]
	10	289.0	6.3	267.1	10.2	5.3	77546	0.526
	20	289.9	6.5	272.1	6.1	5.3	77546	0.523
	30	305.9	6.5	289.2	5.0	5.3	77546	0.508
	40	302.9	6.5	287.1	3.9	5.3	77546	0.522
	50	310.3	5.3	295.8	3.8	5.3	77546	0.511
	60	319.1	5.6	304.7	3.4	5.4	77546	0.525
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	3	3113.4	30.3	561.4	2487.3	34.3	2481162	0.633
	5	1248.1	13.1	383.3	830.0	21.7	2481162	0.652
	10	650.4	7.6	314.0	313.5	15.3	2481162	0.640
	20	470.4	7.4	309.3	142.1	11.6	2481162	0.639
	30	422.8	7.4	316.8	88.8	9.9	2481162	0.628
	40	414.0	7.4	324.5	72.0	10.0	2481162	0.639
	50	408.4	7.6	333.1	57.8	9.9	2481162	0.628
	60	432.7	6.3	371.1	44.8	10.5	2481162	0.649
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	10	9816.3	8.8	383.1	9071.9	352.4	71024650	0.661
	20	4850.4	8.4	378.2	4245.0	218.8	71024650	0.671
	30	3155.2	8.5	355.7	2616.8	174.2	71024650	0.670

S & R of ECMproject in default parallel postprocessing mode. (continued)

models	cores	total	initial setup	pro-jection	enumer-ation	postpro-cessing	ECM count	max_rss [GB]
	40	2480.4	8.5	364.4	1950.6	156.9	71024650	0.665
	50	2095.9	8.6	372.7	1556.2	158.4	71024650	0.651
	60	1849.8	7.1	383.0	1299.8	159.8	71024650	0.652
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 7: Settings & Results of ECMproject in single postprocessing mode.

models	cores	total	initial setup	pro-jection	enumer-ation	postpro-cessing	ECM count	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

S & R of ECMproject in single postprocessing mode. (continued)

models	cores	total	initial setup	pro-jection	enumer-ation	postpro-cessing	ECM count	max_rss [GB]
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 8: Settings & Results of ECMproject in pool parallel postprocessing mode.

models	cores	total	initial setup	pro-jection	enumer-ation	postpro-cessing	ECM count	max_rss [GB]
mmsyn_sm00	3	365.7	17.8	343.9	3.7	0.3	4132	0.228
	5	235.5	8.4	225.4	1.4	0.3	4132	0.300
	10	241.3	5.5	234.5	1.0	0.3	4132	0.300
	20	258.9	5.6	252.0	0.9	0.3	4132	0.286
	30	255.4	5.7	248.5	0.9	0.4	4132	0.265
	40	263.7	5.7	256.7	0.9	0.4	4132	0.244
	50	269.7	4.6	263.9	0.9	0.4	4132	0.264
	60	278.4	4.7	272.3	0.9	0.4	4132	0.233

S & R of ECMproject in pool parallel postprocessing mode. (continued)

models	cores	total	initial setup	pro-jection	enumer-ation	postpro-cessing	ECM count	max_rss [GB]
mmsyn_sm01	20	250.4	5.6	243.3	1.1	0.4	6486	0.271
mmsyn_sm02	20	257.5	5.9	249.7	1.4	0.5	9702	0.275
mmsyn_sm03	20	264.4	6.0	255.3	2.2	0.8	19394	0.303
mmsyn_sm04	20	274.6	6.2	263.4	3.6	1.3	38778	0.328
mmsyn_sm05	3	508.2	23.4	413.6	65.9	5.2	77546	0.511
	5	295.4	10.1	258.0	22.0	5.3	77546	0.509
	10	288.4	6.3	267.1	9.7	5.3	77546	0.526
	20	319.1	34.6	273.1	6.1	5.3	77546	0.534
	30	296.6	6.5	279.9	4.9	5.3	77546	0.514
	40	303.5	6.5	287.7	3.9	5.3	77546	0.526
	50	310.6	5.3	296.1	3.8	5.3	77546	0.524
	60	317.6	5.6	303.8	2.8	5.4	77546	0.509
mmsyn_sm06	20	299.9	6.7	279.1	10.7	3.5	155082	0.620
mmsyn_sm07	20	317.4	6.8	287.6	19.1	4.1	310154	0.620
mmsyn_sm08	20	342.7	7.0	294.9	35.9	4.9	620298	0.622
mmsyn_sm09	20	385.5	7.2	301.4	70.2	6.7	1240586	0.633
mmsyn_sm10	3	3114.3	30.3	560.9	2491.6	31.6	2481162	1.443
	5	1236.7	13.1	372.9	830.8	19.9	2481162	1.328
	10	653.6	7.6	313.5	313.0	19.5	2481162	1.057
	20	508.3	7.3	348.0	142.6	10.3	2481162	0.890
	30	425.7	7.4	317.2	91.6	9.5	2481162	0.711
	40	412.0	7.4	325.1	69.7	9.8	2481162	0.733
	50	436.7	7.5	364.4	55.0	9.7	2481162	0.701
	60	403.7	6.3	342.8	44.8	9.8	2481162	0.732
mmsyn_sm11	20	629.9	7.6	316.1	288.6	17.7	4962314	1.731

S & R of ECMproject in pool parallel postprocessing mode. (continued)

models	cores	total	initial setup	projection	enumeration	postprocessing	ECM count	max_rss [GB]
mmsyn_sm12	20	952.8	7.8	324.5	588.0	32.5	9924618	3.432
mmsyn_sm13	20	1408.1	8.0	361.4	983.9	54.8	17756170	5.381
mmsyn_sm14	20	2448.3	8.4	339.4	1989.3	111.3	35512330	11.259
mmsyn_sm15	10	9752.8	8.8	344.2	9076.3	323.5	71024650	32.738
	20	4807.3	8.4	347.3	4243.9	207.8	71024650	22.758
	30	3197.0	8.5	384.6	2621.0	183.0	71024650	14.148
	40	2499.6	8.5	364.7	1947.2	179.2	71024650	11.084
	50	2128.2	8.6	384.6	1558.4	176.7	71024650	12.654
	60	1869.1	7.1	382.6	1299.0	180.4	71024650	12.263
mmsyn_sm16	20	9370.9	8.6	358.3	8673.8	330.2	142049290	47.149
mmsyn_sm17	20	18897.9	9.1	395.5	17634.1	859.2	284098570	38.628
mmsyn_sm18	60	11157.4	8.7	440.7	9383.2	1324.8	514523146	92.736

Table 9: Settings & Results of ECMproject with mfel projection.

models	cores	total	initial setup	projection	enumeration	postprocessing	ECM count	max_rss [GB]
mmsyn_sm05	3	407.1	23.5	314.9	66.2	2.5	77546	0.508
	5	364.4	10.1	329.2	22.5	2.5	77546	0.527
	10	375.9	6.3	357.6	9.4	2.5	77546	0.506
	20	474.8	6.5	459.4	6.4	2.5	77546	0.506
	30	627.7	6.5	613.7	5.0	2.6	77546	0.505
	40	733.9	6.5	720.6	4.2	2.6	77546	0.521

S & R of ECMproject with mfel projection. (continued)

models	cores	total	initial setup	projection	enumeration	postprocessing	ECM count	max_rss [GB]
	50	920.4	5.3	909.3	3.0	2.7	77546	0.503
	60	1019.1	5.6	1008.1	2.8	2.6	77546	0.519
mmsyn_sm10	3	2931.4	30.3	379.7	2487.6	33.8	2481162	0.643
	5	1613.5	15.8	608.1	966.6	22.9	2481162	0.635
	10	1111.0	10.1	699.9	384.9	16.1	2481162	0.642
	20	1077.2	8.5	860.5	195.5	12.6	2481162	0.627
	30	1082.9	8.4	930.5	134.2	9.8	2481162	0.624
	40	961.3	7.9	874.0	69.8	9.6	2481162	0.635
	50	1007.1	7.6	933.2	56.2	10.1	2481162	0.653
	60	1267.9	6.3	1206.3	45.3	9.9	2481162	0.626

Table 10: Settings & Results of ecmtool.

models	cores	total	preprocessing	first vertex enumeration	intermediate processing	second vertex enumeration	postprocessing	max rss [GB]
e_coli_core	3	15.5	2.9	1.6	4.8	6.2	0.0	0.370
	5	11.8	2.8	1.2	4.9	2.8	0.0	0.369
	10	11.2	2.8	1.2	5.0	2.2	0.0	0.366
	20	10.6	2.8	1.2	4.9	1.6	0.0	0.365
	30	10.8	2.8	1.3	4.9	1.7	0.0	0.366
	40	10.6	2.9	1.3	4.9	1.5	0.0	0.368
	60	10.6	2.8	1.4	4.9	1.5	0.0	0.368

S & R of ecmtool. (continued)

models	cores	total	preprocessing	first vertex enumeration	intermediate processing	second vertex enumeration	postprocessing	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_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	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_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

S & R of ecmtool. (continued)

models	cores	total	preprocessing	first vertex enumeration	intermediate processing	second vertex enumeration	postprocessing	max rss [GB]
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_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	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
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 11: Zipped vs unzipped mode of ECMproject. Zipped mode gets activated by using the –gzipped option.

models	cores	Filesize [GB]		postprocessing [s]	
		unzipped	zipped	unzipped	zipped
e_coli_core	20	96.8e-06	15.9e-06	0.2	31.2
mmsyn_sm05	20	34.7e-03	1.2e-03	5.3	35.4
mmsyn_sm10	20	1.2	37.1e-03	11.6	163.0
mmsyn_sm15	20	35.7	1.1	218.8	3919.6

ECMproject options

ECMproject provides the user with a wide range of different options. The help page with all options can be invoked by calling ECMproject with the '*–help*' option. Additionally, a list with all options and required arguments is given below:

options:

-h, –help

show this help message and exit

-n INT, –n_processes INT

Give number of processes mplrs will use. (default: 3)

-ch INT, –chunksize INT

Give the size of chunks by number of lines per chunk from V-representation for postprocessing via multiprocessing. Lower chunksizes (e.g. 50000) are preferable on lower sized models. (default: 100000)

-p, –parallel

If flag is given, postprocessing will be performed parallel with multiprocessing. This

has high RAM usage on big models but postprocessing step is much faster. (default: False)

-gz, –gzipped

If flag is given, result file will be .gz file. In parallel mode, creation of a .gz file will take significantly more time. (default: False)

-t, –time

If flag is given as an option, script creates an additional json file in outpath with time measurements of different working steps. Note, that total time doesnt measure time for import statements. (default: False)

–pool

If flag is given, postprocessing will be performed with mp.Pool instead of mp.Process. (default: False)

-op, –only_projection

If flag is given, only H-projection will be created in tmp directory. Stops after projection step. (default: False)

-fv, –fluxvertices

If flag is given, only V-representation will be created in tmp directory. Stops after vertex enumeration step. (default: False)

-po FILE, –only_postprocessing FILE

Give V-representation as input. Only postprocessing and certain steps of preprocessing which are necessary will be performed. Usefull, if comprehensive V-representation of a model already exists. (default: None)

-mp FILE, –mplrs FILE

Path to mplrs file. (default: /opt/lrslib/v072/mplrs)

-o PATH, –outpath PATH

Directory, where results shall be saved. (default: ./)

-rn STR, –result_name STR

File name of result. If this option is not given, name of result file will be outpath + model_name. (default: None)

-sep STR, –separator STR

Option, which separator output will have. If no –result_name is given, result file will end with .csv for [,;] and .txt for everything else. (default: ,)

-d INT, –decimals INT

Give the number of decimals, the resulting ECMs will have. (default: 4)

-ex R1 R2 R3 ... [R1 R2 R3 ... ...], –extern R1 R2 R3 ... [R1 R2 R3 ... ...]

Give reaction ids as input (e.g. -ex R1 R2). Just the given reactions are marked as extern and will show up in the ECMs. Use only for real extern reactions, not internal ones! Beware, if the chosen reactions are not all extern reactions of the model, results are unbalanced. (default: None)

-tmp PATH, –tmppath PATH

Directory, where tmp files get stored. (default: ./)

-dv, –developer

Intermediate files in tmp directory get not deleted. H- and V-representation are maintained. (default: False)

-v, –verbose

If flag is given, mplrs will show the whole output. (default: False)

-ap STR, –approximation STR

Approximation for float numbers converted to rationals for .ine files. The less strict the border, the higher rationals can get. Keep attention, large numbers can lead to intense performance issues. If approximation is set to None (default), border is 1e6. Floats will get "rounded" to fractions with max this number as denominator. (e.g.: 1e3 means denominator can be max 1000 → 1/1000: 0.001) (default: None)

-mf, –mfel

Mplrs project will be performed with mfel file if flag is given as an option. (default: False)

-ro INT, –rows INT

The number of rows per job for the mplrs algorithm. (default: 20)

-lr INT, –lastrows INT

The number of rows for the last lastp jobs for the mplrs algorithm. (default: 20)

-lp INT, –lastp INT

Give the percentage of processes, which get used for lastrows of mplrs algorithm. (default: 10)

required arguments:

-f PATH_TO_FILE, –file PATH_TO_FILE

Enter input sbml-File. (default: None)

-m STR, –model_name STR

Enter name of the model. (not filename) If no –result_name is given, model name is used to name result files. (default: None)