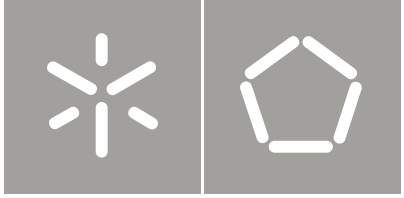
Universidade do Minho
Escola de Engenharia

Filipe Alexandre Wang Liu

Computational tools for pathway optimization towards metabolic engineering applications

Outubro de 2013

Computational tools for pathway optimization towards metabolic engineering applications
Filipe Alexandre Wang Liu
UMinho | 2013



Universidade do Minho
Escola de Engenharia

Filipe Alexandre Wang Liu

Computational tools for pathway optimization towards metabolic engineering applications

Tese de Mestrado
Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação do
Professor Doutor Miguel Rocha
Professora Doutora Isabel Rocha

Outubro de 2013

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. PhD. Miguel Francisco de Almeida Pereira da Rocha for the opportunity to work in this topic and for the continuous support that he provided me during the fulfillment of this work.

I would like to thank Prof. PhD. Isabel Rocha from the department of biological engineering, who agreed to co-supervise my dissertation and who gave a special attention to the scientific topics more related to systems biology in particular to metabolic engineering.

A special thanks to the Bioinformatics and System Biology Interdisciplinary Initiative group, were all the work was done, for their weekly group meetings that enriched my knowledge about topics concerning to systems biology, and for all the great moments of fun. I would also to like to emphasize a special thank to Paulo Vilaça from the group, for all the support he provided with his expertise in both computer sciences and systems biology.

I would also like to thank SilicoLife, a company that had a special interest for my work, providing me a greater motivation for this research topic.

A thank to all my friends, as without them, life would certainly be a much boring place. To Bruno Medeiros for the coffee moments, were we discuss science, politics and the meaning of life.

To my parents for their continuous effort to support my education.

Abstract

Metabolic Engineering targets the microorganism's cellular metabolism to design new strains with an industrial purpose. Applications of these metabolic manipulations in Biotechnological derive from the need of enhanced production of valuable compounds. The development of *in silico* metabolic models proposes a quantifiable approach for the manipulation these microorganisms. In this context, constraint based modelling is one of the major approaches to predict cellular behaviour. It allows to prune the feasible space of possibilities describing possible phenotype outcomes in terms of metabolic fluxes. Under these conditions, cellular metabolism can be represented as an algebraic system constrained by the laws of mass balance and thermodynamics.

These systems are prone to be represented as networks, taking advantage of different graph-based paradigms, including bipartite graphs, hypergraphs and process graphs. This thesis explores these representations and underlying algorithms for metabolic network topological analysis. The main aim will be to identify potential pathways towards the optimized biochemical production of selected compounds. Related to this task, algorithms will also be designed aiming to complement networks of specific organisms, taking as input larger metabolic databases, inserting new reactions making them able to produce a new compound of interest.

To address these problems, and also related tasks of data pre-processing and evaluation of the solutions, a complete computational framework was developed. It integrates a number of previously proposed algorithms from distinct authors, together with a number of improvements that were necessary to cope with large-scale metabolic networks. These are the result of problems identified in the previous algorithms regarding their scalability.

A case study in synthetic metabolic engineering was selected from the literature to validate the algorithms and test the capabilities of the implemented framework. It allowed to compare the performance of the implemented algorithms and validate the proposed improvements.

Keywords: Metabolic Networks; Flux Analysis; Synthetic Biology; Pathway Optimization; Network Topological Analysis; Subgraph Extraction;

Resumo

A Engenharia Metabólica visa a alteração do metabolismo celular dos micro-organismos com vista ao desenho de novas estirpes com fins industriais. As aplicações destas modificações genéticas na Biotecnologia derivam da necessidade de produzir de forma otimizada compostos de alto valor. O desenvolvimento de modelos computacionais propõe uma abordagem quantitativa para a manipulação destes organismos. Neste contexto, a modelação baseada em restrições constitui uma das abordagens mais usadas para a previsão do comportamento celular. Esta permite reduzir o espaço de soluções viáveis descrevendo o fenótipo celular a partir dos fluxos metabólicos. Nestas condições, o metabolismo celular pode ser representado como um sistema algébrico restringido pelas leis da conservação de massa e termodinâmica.

Estes sistemas podem ser representados como redes, tirando partido de diferentes paradigmas baseados em grafos, incluindo os grafos bipartidos, os hipergrafos e os grafos de processos. Esta tese explora estas representações e os respetivos algoritmos para a análise topológica de redes metabólicas. O objetivo principal será o de identificar potenciais vias metabólicas para a optimização da produção de compostos selecionados. Relacionado com esta tarefa, serão desenhados algoritmos com o objetivo de complementar redes de organismos específicos, tomando como entradas bases de dados metabólicas de maior dimensão, inserindo novas reações de forma a torná-los capazes da produção de novos compostos de interesse.

Para abordar estes problemas, bem como tarefas relacionadas ao nível do pré-processamento e avaliação das soluções, foi desenvolvida uma plataforma computacional completa. Esta integra um conjunto de algoritmos previamente propostos por diversos autores, em conjunto com melhorias significativas que foram necessárias para que estes pudessem lidar com redes metabólicas de grande escala. Estas melhorias resultam da identificação de problemas nos algoritmos no que diz respeito à sua escalabilidade.

Um caso de estudo na Engenharia Metabólica sintética foi selecionado da literatura para validar os algoritmos e testar as capacidades da plataforma implementada. Este permitiu comparar o desempenho dos algoritmos implementados e validar as melhorias propostas.

Palavras-chave: Redes Metabólicas; Análise de Fluxo; Biologia Sintética; Optimização de vias metabólicas; Análise topológica de redes; Extração de sub-grafos.

Contents

List of Figures

List of Tables

List of Acronyms

ACC	**A**nalysis**C**ontrol**C**enter
AOP	**A**spect-**O**riented **P**rogramming
CBA	**C**onstraints **B**ased **A**nalysis
DCC	**D**ata**C**ontrol**C**enter
EC	**E**nzyme **C**ommission number
EFM	**E**lementary **F**lux **M**ode
EP	**E**xtreme **P**athway
FA	**F**ind**A**ll
FBA	**F**lux **B**alance **A**nalysis
FP	**F**ind**P**ath
GSM	**G**enome-**S**cale **M**odel
JVM	**J**ava **V**irtual **M**achine
KEGG	**K**yoto **E**ncyclopedia of **G**enes and **G**enomes
LP	**L**inear **P**rogramming
ME	**M**etabolic **E**ngineering
MN	**M**etabolic **N**etwork
MP	**M**etabolic **P**athway
MSG	**M**aximal **S**olution **G**eneration
SBML	**S**ystems **B**iology **M**arkup **L**anguage
SSG	**S**olution **S**tructure **G**eneration

Chapter 1

Introduction

In the past decade, Biotechnology observed an increasing gain of popularity. One of its key advantages is the capability of *de novo* synthesis of secondary metabolites by genetic modification of microbe strains to produce high yield products [38, 47]. These microbes are often referred as cell factories, strategically selected microbial strains that are re-engineered to produce a target compound, offering a full catalogue of potential sustainable solutions for many modern problems. The use of cells as a factory for the production of compounds is exploited by many industries, such as pharmaceuticals, food companies, renewable energy sources, polymers and chemicals [13, 21].

The field of Metabolic Engineering (ME) studies the optimization of living organisms metabolic processes, by genetically modifying them to achieve a certain goal, such as increasing the production of a desired compound or allowing the production of a compound the wild type would not be able to produce. As such, it plays an important role in the Biotechnology industry [12].

Technological advances in high throughput screening and Bioinformatics tools allowed an exponential increase of biological information, that is likely to keep increasing for the next years. To accommodate all this information, several large *omics* databases are available to store and catalog all biological data from genomes to reactions. Such information became the core of biological research [29], demanding more sophisticated computational tools to analyze genome scale data sets, while manual processing became no longer viable.

In a traditional ME approach, industrial bioprocesses develop new strains by multiple rounds of random mutagenesis. This approach usually unrolls negative side effects, such as unwanted changes that may occur in the cell metabolism. Most of these side effects are usually hard to track and diagnose.

With access to experimental data, several genome scale models of a variety of organisms were re-constructed, enabling *in silico* whole cell simulation of phenotypic responses [15, 20]. Systems Biology developed a rational design process allowing to quantify and predict cellular responses to environmental and genetic modifications [48]. Under a systemic scope, the phenotypic interactions are expressed as a whole system instead of individual targets, by combining information from genomics, transcriptomics, proteomics, metabolomics and fluxomics into a multi-layer system.

Metabolic Engineering is powered by computational tools and models to design new strains. These systems quantify cell phenotype expression using metabolic models, allowing to strategically develop mutant strains capable of the production of high yield bio-compounds. The process is characterized by a cycle of three main steps: design, construction and analysis, involving both systems biology and synthetic biology [37].

The modelling of cells has been traditionally achieved through the use of dynamic models. However, these require information that is hard to gather, such as kinetic data, limiting the applicability of these models to small-scale systems. Due to the complexity in the parametrization of dynamic models, stoichiometric models are by far simpler to build. Stoichiometric models are based on the fundamental laws of mass balance, which requires only the stoichiometry of reactions, considering systems in steady state [9]. These stoichiometric models allowed the study of the metabolism of different organisms through several methods for phenotype simulation and structural analysis.

1.1 Motivation

The re-engineering of cells for *de novo* synthesis of secondary metabolites involves many different steps, from data collection and curation, to optimal strain selection, pathway identification and analysis of the best solutions.

It is common to use computer software to aid ME processes. In the past years, a vast catalogue of Bioinformatics software was developed to fit many topics in this field [10], ranging from network reconstruction and representation problems to data visualization and metabolic network analysis. Despite of this effort, most topics still present a big challenge to software development, since the reconstruction, analysis and optimization of large scale metabolic networks still face many challenges.

Advanced computational tools are currently able to identify optimal pathways through stoichiometric network analysis by either computing algebraically steady states of the stoichiometric network or from graph topological analysis. The computation of steady states relies in the analysis of the feasible solutions cone which represents all possible steady state flux distributions. Extreme Pathways (EP) [4] and Elementary Flux Modes (EFM) [55] both compute flux vectors through convex analysis [17]. However, these methods scale poorly and are inefficient for large scale networks. Several attempts have been made to adapt both EP and EFM to large scale data sets, either by taking advantage of modern multi-core processors [58] or using stochastic heuristics [44]. However, due to the exponential growth of combinatorial possibilities, for large scale systems, convex analysis still suffers from many pitfalls from memory problems to computational time.

An alternative approach is searching pathways with graph topology algorithms, based on graph search algorithms with extra rules that meet the criteria of a metabolic context. Graph traversals are fast and many algorithms were adapted to ME problems. However, graph searching for optimal solutions suffers equally from complexity issues, although for a single solution they are much faster compared to convex analysis methods [6].

Metabolic Engineering is powered by *in silico* analysis and, therefore, there is a demand for specialized integrated development environments, that still offers a challenge for software engineers. Most tools are developed as standalone programs or web services, making difficult the integration and analysis of results and most time is spent on writing computer scripts to parse the input and output of tools, which is not productive and it is not practical. Therefore, commercial applications such as MatLab, designed for numerical computing, are by far the most popular in the area [27] and there are only a few specialized open-source platforms available. OptFlux [52] is one of those platforms for *in silico* ME, integrating many techniques to tackle problems in this field, and designed to be modular. It allows the

development of plug-ins to integrate within the platform, thus allowing its easy extension.

1.2 Objectives

Given the context described above, the main aim of this work will be the development of computational tools for the optimization of pathways over a metabolic network, given specific design objectives under the realm of Metabolic Engineering applications. More specifically, the work will address the following scientific/ technological goals:

1. To build metabolic networks using graph-based representations, integrating distinct data sources including metabolic databases (e.g. KEGG or MetaCyc) or metabolic models, allowing flexible user defined filters to be applied and contemplating information related to the metabolic capabilities of each organism of interest.
2. To design and implement optimization algorithms that allow searching over these metabolic networks for the best routes from sets of source metabolites to target metabolites, given the specificities of the underlying graph representation and being able to optimize these paths according to different criteria.
3. To design and implement methods to evaluate the generated solutions, for the problems in 2.
4. To integrate the algorithms implemented into a computational framework, which is able to set up the entire process of computing synthetic ME pathways.
5. To validate the proposed algorithms with a selected case study from literature.

1.3 Structure of the Thesis

The document is divided in six chapters. In this first chapter, we provided a brief introduction of the motivation and the main aims of the work.

The second chapter, *Metabolic Engineering*, introduces several important aspects related to computational tools for metabolic modelling and optimization methods, as well as the introduction of the synthetic pathway extraction problems and the state of art of the available methods and algorithms for solving of it.

The third chapter, *Computing Synthetic Pathways*, presents a unified formal definition of several selected pathway extraction algorithms and the underlying graph representations. A detailed analysis of each of these algorithms is made to underline their weakness and strengths.

The fourth chapter, *The Biosynth Framework*, describes the framework design and implementation details, developed in this work to address the synthesis problem.

The fifth chapter, *Validation of the Framework*, applies the tools to a case study from the literature, to benchmark the developed framework by comparing the obtained results with the available solutions.

Finally the last chapter, *Conclusions*, presents the main conclusions of the work, also proposing future research topics.

Chapter 2

Metabolic Engineering

Metabolic engineering (ME) proposes a rational strategy to analyze and optimize cellular metabolic systems, recurring to mathematical models to quantify changes in the system. A metabolic system is a set of interconnected complex circuits of reactions and metabolites known as metabolic networks (MN).

Manipulation of such networks is usually a complex task, and quite impossible by naïve selection of biological entities. Indeed, it is nearly impossible to find the best combination of genes to express towards a desired phenotype outcome without resorting to computational tools. The analysis of MNs mainly relies on *in silico* modelling and computer algorithms to analyse and quantitatively simulate such networks.

2.1 Metabolic Modelling

Every cell conducts metabolism through a series of interconnected pathways, where a metabolic pathway (MP) can be defined as a coherent set of reactions that together conduct a primary metabolic function. A pathway typically converts a primary substrate (or several) to a target product through a combination of reactions. Another important aspect of a MP is that it should be feasible and observable, otherwise there would be little interest in defining it, as it would be inapplicable in real situations [57].

A metabolic model consists of a network of chemical reactions that allows to predict the behaviour (or limitations) of cellular micro-organisms. These models

can be targeted to a specific pathway, such as the central carbon metabolism, or integrate multiple subsystems to assemble genome scale models (GSM). The GSM allows interaction between subsystems, which increases the capability of phenotype prediction in different scenarios. These models are validated with experimental data to check whether the models correctly predict the desired outcomes. The reconstruction of GSMs is usually a cyclic process involving multiple rounds of validation and model tuning.

The basic elements required to describe a metabolic system are:

- A set of compartments (where reactions take place);
- Metabolites in the system. Since a metabolite can be present in different compartments and they are non-interchangeable, a single metabolite (e.g., water) can have multiple species (e.g., water_a, water_b, water_c) spread in distinct compartments;
- Chemical reactions that are able to transform metabolites inside a compartment or transfer metabolites between different compartments;
- Other features that are not relevant for this work: including genes and gene-reaction rules.

The analysis of GSMs is typically done with computer tools and algorithms. The Systems Biology Markup Language (SBML) is a free, open, XML based format for encoding metabolic models [28], that is one of most popular computer readable formats.

2.2 Metabolic Databases

Most *in silico* tools rely on the information available in biological databases. This is no exception for ME, as the reconstruction of GSMs involves the use of a lot of information from different public databases. There are several bioinformatics resources available in the web for metabolic pathways. The Kyoto Encyclopedia of Genes and Genomes (KEGG) [31, 32] is one of the main resources, integrating all genomic, chemical and protein information. For metabolic networks, the

set of entities of interest is the chemical universe mainly composed of metabolites (compounds) and chemical reactions. KEGG features four database (i.e., Compounds, Glycans, Reactions and Enzymes) to assemble biochemical information (Fig. 2.1(a)). The compound database hosts chemical compounds, while the Glycan database contains only carbohydrates structures, making the metabolite databases. The reaction database assembles chemical reactions, where each reaction contains a set of products and reactants from either of the metabolite types (i.e., compound or glycan). Finally, the enzyme database links reactions to genes, which serves as a gateway to the external resources.

KEGG offers a simple but accurate representation of cellular metabolic pathways. However, other databases such as the BioCyc [34] database consortium, contain several metabolic pathway databases with a much more complex entity relationship. While, in KEGG, only four entities assemble the universe of chemical reactions, the BioCyc schema represents the chemical universe of the metabolite components separating into basic compounds, RNA and proteins. Then, the association of reactions to enzymes is more complex (Fig. 2.1(b)).

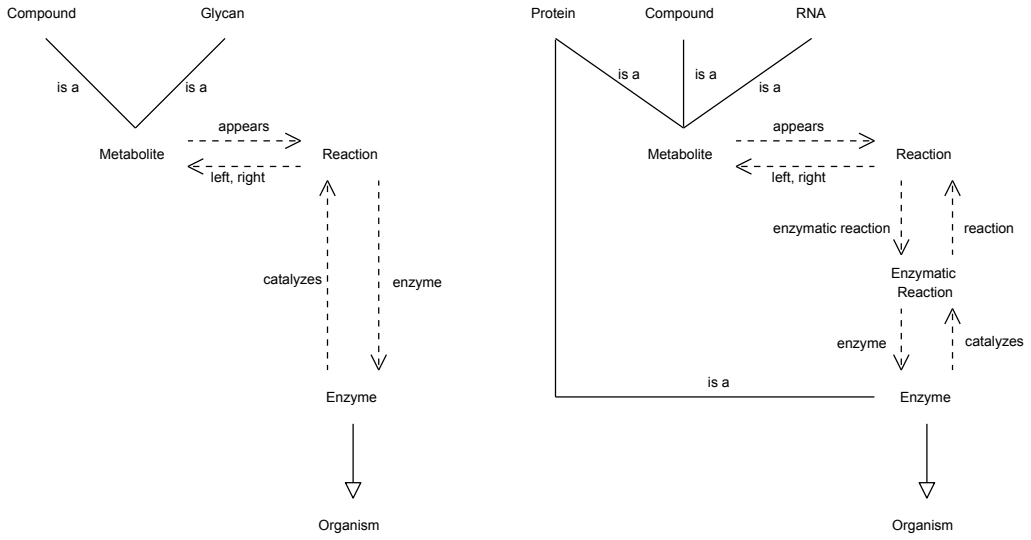


FIGURE 2.1: Partial schema of the chemical universe of KEGG and BioCyc.

The BioCyc database consortium contains several databases, each specific for a certain organism. The databases are separated in three tiers. The first tier consists of databases that have gone though multiple persons-years of heavy manual curation, while in the second tier records are also under curation but with less

effort. The last tier contains automatically generated databases from the BioCyc Pathway Tools [35, 36], with no curation effort. In the BioCyc consortium, the MetaCyc database (first tier) [7] is the only generic knowledge base containing information on multiple organisms.

The KEGG and BioCyc are the two major knowledge bases for metabolic information, nevertheless each of them presents a noticeable growth rate of information each year. There are several more metabolic databases such as Model SEED [2] and BiGG [54], but these are relatively small compared to the previous mentioned databases [33]. To complement these pathway databases, there are some more specific databases. For instance, BRENDA focuses on enzymatic reactions having more than 79 000 individual reactions from 10 500 organisms. The ExplorEnz [45] and ENZYME [3] are specific for enzyme and enzyme-catalyzed reactions description classified by the Enzyme Commission number (EC) a numerical classification scheme to classify enzymes by function.

The heterogeneity of the data information sources offers a rich set of metabolic information. However, without a standard representation, this provides a huge amount of redundant data. Most of these databases have cross references to each other for part of their records, but still there are many cases of inconsistency between these.

2.3 Constraint Based Analysis

Constraint Based Analysis (CBA) is one of most popular approaches adopted for *in silico* analysis of MNs. This approach differs from other traditional alternatives since it has not been limited by the availability of kinetic information, thus allowing the analysis of genome scale models capturing whole cell information. In CBA, the system is subject to several constraints, such as the laws of mass balance and thermodynamics. The system is constrained to the stoichiometric information, the reactions reversibility and the maximum flux allowed in each reaction [16]. It is, thus, possible to determine the feasible space, which allows to understand the capabilities of the system under these circumstances.

Flux Balance Analysis (FBA) uses an optimization approach to, over this feasible space, calculate the optimal flux distribution of a steady state network [42], where the net product of internal metabolites is equal to zero. The FBA method

optimally assign fluxes to reactions in a MN based on an objective function using a linear programming (LP) formulation. A limitation of FBA is that it only returns one optimal solution, as for a given MN, in some scenarios, there may exist multiple optimal flux distributions.

There are several other CBA methods, including variants of FBA, that will not be mentioned as these are not relevant to this work. Those are applied, for instance, when simulating the effects of gene knockouts or flux variations on a GSM [60].

2.4 The Synthetic Metabolic Problem

A synthetic (or retrosynthetic) ME problem can be summarized as the following: given a well defined GSM, the goal is to find a set of reactions that attached to the model would augment its capability to produce new compounds of interest, which are non native to the organism.

This can be viewed as a reverse optimization process. While, in ME, for the optimization of a MN using stoichiometric models, the objective function and all participating reactions are known and well defined, in a synthetic problem for *de novo* synthesis of a target compound, the problem is to find a set of reactions that are compatible with the host taken from a larger knowledge base of chemical reactions.

The synthetic problem starts with the selection of a target compound and a host organism, represented by a metabolic network, which is set as the chassis. Pathway extraction algorithms identify candidate reactions from a known domain that, if introduced to the chassis, allow to augment the capability of the organism to produce the target compound. A less traditional problem would be to optimize the selection of the chassis itself. In this scenario, the chassis is unknown, therefore the pathway extraction algorithms have to find the best pathway out of the entire domain.

The identification of optimal stoichiometrically balanced pathways is often problematic and offers many challenges. The number of combinatorial possibilities exponentially increases with the size of the search domain. Another important factor is the definition of what is an optimal MP. Depending on the criteria and

complexity, one could argue that an optimal MP would be the solution that returns the highest production yield with the minimum amount of reactions, but equally valid criteria may be defined.

There are several methods for pathway extraction from MN, that can be subdivided into two main categories: steady state analysis or graph topological analysis.

Extreme pathways (EP) and elementary flux modes (EFM) compute non decomposable minimal pathways from metabolic networks. A non decomposable minimal pathway is a minimal set of reactions that satisfies the steady state condition and that is non reducible, i.e., the removal of any reaction from this set invalidates the steady state condition.

EPs and EFMs can be used to analyze network robustness [64]. Since the amount of EFM and EP usually increases exponentially with the size of the network, it is impossible to compute them in most of the GSMs due to computational intractability. Because these methods are unable to analyse large GSMs, the computation of database size networks such as the KEGG or BioCyc is definitely out of their reach.

An alternative approach is to rely on topological analysis of metabolic networks, which offers a faster approach to identify potential MPs. There are several implementations based on a variety of graph structures and algorithms (Table 2.1). Some of those are more generic, to be able to search any network such as protein-protein interaction networks, others are designed specifically for metabolic networks.

TABLE 2.1: Available software tools for pathway extraction based on graphs

Software	Data Structure	Algorithm(s)
Pathway Hunter Tool [51]	Graphs	k-shortest path
Metabolic Path Finding [11]	Graphs	k-shortest path
Find Path [6]	Hypergraphs	pathway enumeration
Network Analysis Tool Set [19]	Bipartite Graph	k-shortest path kWalks
Reaction Pathway Identification [41]	Process Graph	Maximal Structure Generation Solution Structure Generation

In the next section, several metabolic network models and analysis methods are described.

2.4.1 Graph Based Approach

A graph is a common mathematical model to define relationships between entities. In the metabolic network context, there are two participating entities in the network: metabolites and reactions. Since graphs have only a single type of vertex, there are several alternative types of graph models.

r_0: ATP + L-Glutamate $\xrightarrow{2.7.2.11}$ ADP + L-Glutamyl 5-phosphate
r_1: Acetyl-CoA + L-Glutamate $\xrightarrow{2.3.1.1}$ CoA + N-Acetyl-L-glutamate
r_2: ATP + N-Acetyl-L-glutamate $\xrightarrow{2.7.2.8}$ ADP + N-Acetyl-L-glutamate 5-phosphate

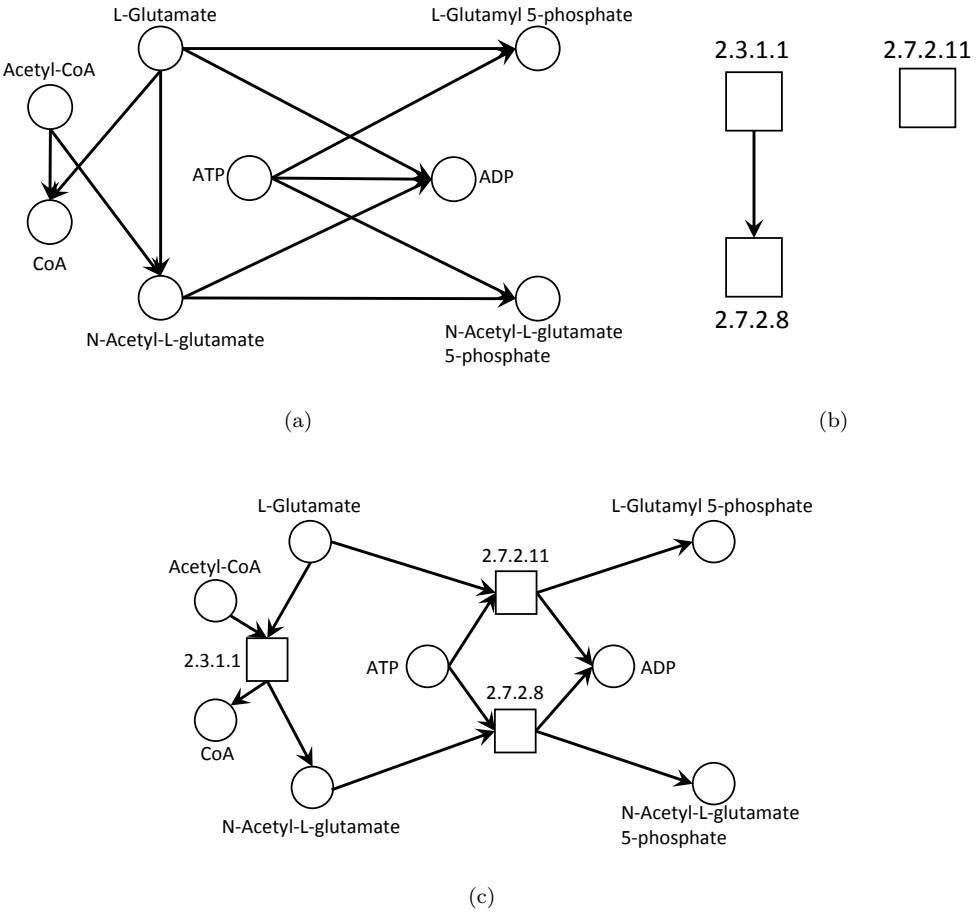


FIGURE 2.2: Three possible graph representations of reactions r_0, r_1 and r_2; a) compound graph; b) reaction graph; c) compound-reaction graph;

The compound graphs (Fig. 2.2(a)) allow the analysis of several topological properties, such as connectivity, length, or cluster density, among others [43]. In a compound graph, a link between compounds a and b means there is a reaction that requires a in order to get b, but since a reaction may require more than one substrate, a single reaction can spawn many edges between substrates and products. The dual of this model are reaction graphs (Fig. 2.2(b)), where an edge between two vertices (that are reactions), captures the property of the dependency. As an example, in Figure 2.2(b), the reaction 2.7.2.8 is dependent on the products of the reaction 2.3.1.1, and therefore, they are related. The combination of both compounds and reactions into a single graph model generates a bipartite graph (Fig. 2.2(c)), having two disjoint sets of vertices of compounds and reactions, where an edge between a compound and a reaction represents substrates, and the opposite represents products.

Graph topological analysis uses well known graph based traversal algorithms, most of them derived from a Breadth-first or Depth-first search approach. The main advantage of graph methods is the computation time of a single solution, which in most cases is achieved in polynomial time, and the availability of many well defined algorithms for graph analysis.

A common strategy to identify a possible optimal route between a substrate s and a product t is to use shortest path algorithms. However, since there is a possibility to have multiple variants between two vertices, it is common to use algorithms that enumerate k multiple alternative paths, such as the k-shortest path algorithm [19, 51].

Traditional graph algorithms suffer from two major pitfalls. It is possible for some pathways not to be linear, i.e., they do not correspond to a linear path between s and t. A path in a graph model is always a sequence of edges between s and t. This usually does not express exactly a pathway, but a portion of what could be the pathway, as most pathways fork into multiple routes because of multiple dependencies. Therefore, computing exact pathways from paths in graphs is a complex task, which requires merging multiple solutions.

Another problem is due to the fact that MNs are usually small world networks, where most nodes can be reached by a small number of hops from any other node [61]. This is due mainly to cofactor compounds (e.g., water, ATP or ADP), which

usually serve as central hubs connecting to most of reactions. This will, eventually, mislead the path finding algorithms to generate non relevant paths [11, 53].

Several alternatives are proposed to fix this issue. By weighting the graph edges by their degree, in shortest path algorithms, this allows to penalize hub compounds. However, this is still inaccurate as the degree is very dependent on the sample of the network. Also, several non-cofactor compounds have a relative high degree (e.g., pyruvate). Other authors propose more sophisticated strategies by using the annotated data, such as the KEGG reaction pair annotation [39], where each compound to compound pair has an special annotation that describes whether they are the main link or a cofactor link. This allows a more advanced filtering of the cofactors [18, 65]. Nonetheless, false positives and negatives still occur due to flaws in the data and by improper or incomplete characterization.

2.4.2 Set Systems

The ambiguity of graph models and the incapability of those to define multi dependency relationships demanded a different model to represent metabolic networks. Both hypergraphs and process graphs have been put forward for that purpose. They are very closely related as they are both set systems, so unlike graphs, where each edge connects single entities, in set systems, an edge establishes a connection between two sets.

A directed hypergraph is a graph where each edge may contain multiple source vertices (head) to multiple destination vertices (tail) (Fig. 2.3(a)). This edge is denoted as hyperedge for undirected edges and hyperarc for directed edges. This type of graph was introduced to model database schemas, being first described as functional dependency graphs because of the importance of their closure properties [1]. Later, it gained application in other fields, being one of them chemical reaction modelling [59].

A process graph is very similar to an hypergraph and a bipartite graph. This model was designed to represent chemical synthesis problems, as common graphs were unable to define properly these systems [23]. They are similar to bipartite graphs because the number of edges is equal to a compounds-reaction graph (Fig. 2.3(b)). However, the relationship between these edges is ambiguous as a single edge between a compound and a reaction has little meaning. The process graph

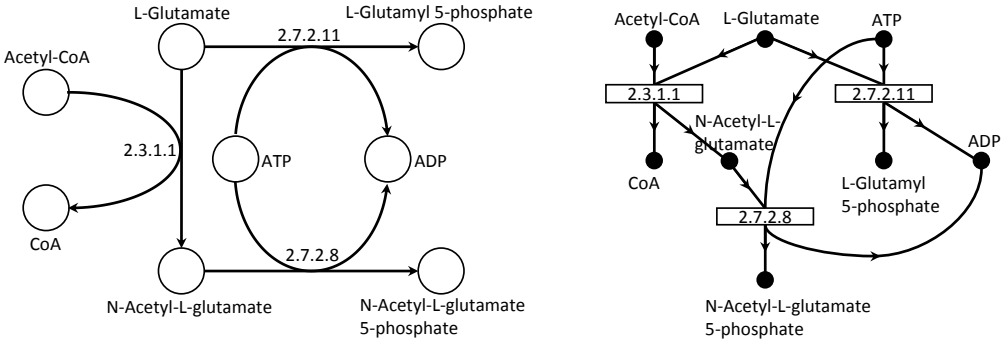


FIGURE 2.3: Two set systems. a) directed hypergraph; b) process graph;

uses operational units to define relationships between compounds, in a way that is similar to an hyperarc since each operational unit connects two disjoint sets of vertices. For the sake of simplicity, operational units will be called hyperarcs as they share the same properties.

In set systems, there is no exact notion of what exactly defines a path (or hyperpath) between two vertices. This definition varies depending on the problem context. For the metabolic context, a hyperpath $S-T$ between two sets of vertices S and T can be defined as: from an initial set of vertices denoted as substrates (S) to another set of vertices denoted as products (T), a hyperpath is a set of hyperarcs that can be satisfied by the initial set S, and satisfy the set T. In a dependency problem, a hyperarc needs to be satisfied by a set of vertices, and satisfies another set of vertices. The head of the hyperarc are the vertices that need to be satisfied, while the tail contains the dependencies that a hyperarc satisfies, when included in a system.

Mathematically, the objective is to compute, from a set of reactions denoted as the reaction universe, a subset that satisfies a certain set of constraints:

- The metabolites that are unsatisfied must belong to the set of substrates vertices S;
- All the products in T must be present in the subset;
- Every reaction in the subset must be connected to at least one product in T.

This problem can be viewed as a subgraph extraction problem, in which given a large set system which contains all possible reactions, the goal is to find combinations that satisfy $S - T$. The big advantage, when compared to traditional graph methods, is that it is now possible to obtain combinations of reactions that actually assemble the exact shape of the pathways. Because the paths are multi-dimensional, such that they do not always assemble a linear sequence of several edges, it is not possible to determine what is an optimal hyperpath between S and T. Thus, the algorithms relies on enumerating all possible hyperpaths from S to T.

There are several algorithms related to these models. The first category includes algorithms to prune the network [24]. Since this is a combinatorial problem, one question to address is to determine which building blocks (i.e., all the reactions that are available to assemble the pathways) are necessary. In a set of reactions, there may be several elements that can be discarded since they do not participate in any pathway due to compatibilities (e.g., a reaction that is unsatisfiable). Removing these reactions would reduce the size of the elements of the set, which in turn would reduce the complexity of the problem.

The other category are the algorithms to generate the hyperpaths that satisfy $S-T$ from a given set of reactions. The FindPath algorithm [6] allows to enumerate all minimal solutions from an hypergraph, while the Solution Structure Generation algorithm [25] computes all possible combinations of pathways from a process graph.

In the next chapter, a formal definition is introduced to describe and analyze the synthesis problem and the algorithms mentioned above. The set systems offer a potential solution to the synthetic ME problem. Therefore, the main focus of this work is to apply these algorithms to extract pathways from database networks (i.e, KEGG and BioCyc). However, they have their own drawbacks, which are analyzed in deeper detail in the following chapter.

Chapter 3

Computing Synthetic Pathways

In the previous chapter, several state of the art graph representations and algorithms to compute pathways from metabolic networks were introduced. Extreme Pathways (EP) and Elementary Flux Modes (EFMs) are both precise methods to generate pathways, but they are limited by the size of the networks, since up to date it is still impossible to apply these methods over Genome-Scale Models (GSMs). Therefore, the application of these methods to database size networks, which are many times larger than GSMs, is definitely inappropriate.

The graph topological analysis methods are more convenient to extract pathways from database size networks, as there are many cases of success applications of graph path searching algorithms to infer pathways. A major problem of these methods is the accuracy of the results. A solution is to apply set systems to model chemical networks, since these show more similarity to the structure of the network.

In this chapter, the synthetic Metabolic Engineering (ME) problem, the sets systems and related algorithms are introduced in a formal notation to precisely analyze several aspects related to their behavior, including aspects such as the complexity and precision of the algorithms, as well as other limitations.

3.1 Formal Definition and Data Modeling

3.1.1 Basic Definitions

To describe the algorithms and their properties, a notation is specified to represent the entities, special sets and functions that participate in a synthesis problem. Most of the notation follows the definitions and theorems defined by the process synthesis algorithms [23–25]:

Zero	$\emptyset$	Empty set
Calligraphy typing	$\mathcal{M}, \mathcal{R}, \mathcal{D}, \mathcal{Z}, \mathcal{S}, \mathcal{H}, \mathcal{P}$	Special Sets
Capital letters	M, R, S	Sets
Lowercase letters	m, r, e, o	Single entities

If not stated otherwise, all sets defined in this chapter are assumed as unordered sets, such that two sets containing $\{1, 2\}$ and $\{2, 1\}$, are considered as equal. Ordered sets will be explicitly presented, being defined by $\langle\rangle$ brackets. As an example, a tuple defined by $\langle a, b\rangle$ is an ordered pair of two elements, such that $\langle a, b\rangle$ is not equal to $\langle b, a\rangle$. Similarly, a triple is written as $\langle a, b, c\rangle$.

The basic set operations are presented in this section with minimal detail, as these have the same definition as in most set problems. These operations are the following:

- Element relation: a is said to be an element of a set M, written $a \in M$, if M contains a (for the opposite relation the symbol $\notin$ refers to "is not an element of");
- Inclusion relation: a set A is said to be a subset of B, written $A \subseteq B$, if B contains A (for the opposite relation the symbol $\nsubseteq$ refers to "is not a subset of");
- Proper Subset: a set A is said to be a proper subset of B, written $A \subsetneq B$, if B contains A, but A does not contain B (i.e., B contains A but the sets are not equal);
- Union operation: the union of two sets A and B is defined by $\{x : x \in A \text{ or } x \in B\}$, written as $A \cup B$;

- Intersection operation: the intersection of two sets A and B is defined by $\{x : x \in A \text{ and } x \in B\}$, written as $A \cap B$;
- Set Difference operation: the set difference of sets A and B is defined by $\{x : x \in A \text{ and } x \notin B\}$, written as $A \backslash B$;
- Size of a set: the size of a set A is defined by the function $|A|$, that counts the number of elements in A;
- Power Set: the power set of a set A is defined by the function $\wp(A) = \{X : X \subseteq A\}$, which is the set of all subsets of A;
- Cartesian product: the Cartesian product of two sets A and B, is denoted by $A \times B$, being defined as: $\{\langle a, b\rangle : a \in A \text{ and } b \in B\}$

For ordered sets, the $\pi_i(X)$ functions return the i-th element of an ordered set. As an example, $\pi_1(\langle a, b\rangle) = a$ while $\pi_3(\langle a, b, c\rangle) = c$.

Several useful definitions are given next, being used in further sections.

Definition 1. **(Findable)**

An element a is said to be findable in A, written as $a \dot{\in} A$, if and only if, the following occurs:

1. A is a set, and $a \in A$ verifies;
2. A is an ordered set of n elements, and at least one of $a \dot{\in} \pi_i(A)$, where $1 \leq i \leq n$, is true.

□

Definition 2. **(Included)**

A set A is said to be included in B, written as $A \dot{\subseteq} B$, if and only if, for each element $a \in A$, then $a \dot{\in} B$ must be true. □

The definitions 1 and 2 allows to later simplify complex operations of element and set inclusion, that otherwise would require a lot of recursive π_i operations to find elements in tuples that may be nested inside of set of tuples. Other symbols are introduced later in their corresponding sections.

In the next sections, two levels of entities and models will be presented: the biological level, which includes entities that are the main building blocks of the metabolic system, followed by their mathematical models, which are created from their biological counterparts, being the ones used by the algorithms that are analyzed in this chapter.

3.1.2 Biological Entities

Here, biological entities are defined as those representing biochemical components within metabolic networks, namely reactions and metabolites. Each of the entities contains a minimal set of characteristics used to capture the properties required to compute the metabolic synthetic problem.

A metabolite is a single entity that, in general, is defined by the symbol m (although later the symbols s and t will also be use to define metabolites to easily distinguish supply and target metabolites). The universal set of metabolites $\mathcal{M}$ is a finite set that contains all distinct metabolites of a system. The following example is a valid universal set:

$$\mathcal{M}_{e.g.} = \{m_0, m_1, m_2, m_3, m_{\text{Pyruvate}}, m_{\text{2-Acetolactate}}, m_{VT}, m_{O_2}, m_{\text{CO}_2}, m_{H_2O}\}$$

There are many attributes that could be assigned to a metabolite entity, like the chemical formula, but for the purpose of the pathway extraction algorithms, at this stage, these can be discarded.

A reaction entity r defines a relationship between metabolites, as follows:

Definition 3. **(Reaction)**

A reaction r is an ordered pair of two sets of metabolite-stoichiometry pairs α and β. Since a metabolite-stoichiometry pair is a tuple $\langle m, n\rangle \in \mathcal{M} \times \mathbb{R}^+$, a reaction r can be defined as:

$$r = \langle \alpha, \beta \rangle : \alpha, \beta \subseteq \{\langle m, n\rangle : m \in \mathcal{M}, n \in \mathbb{R}^+\}$$

□

The symbol r without superscript notation usually represents a reversible reaction, where r can either assume $\langle \alpha, \beta \rangle$ or $\langle \beta, \alpha \rangle$, which is equivalent to the $\overleftrightarrow{r}$ notation. If the arrow on top has a single direction this indicates that r is irreversible. The direction of the arrow corresponds to the orientation of the reaction (e.g., if $r = \langle \alpha, \beta \rangle$ then $\overrightarrow{r} = \langle \alpha, \beta \rangle$ and $\overleftarrow{r} = \langle \beta, \alpha \rangle$). Similarly to $\mathcal{M}$, the universal set of reactions is written by the symbol $\mathcal{R}$, which is a finite set that contains all reaction entities in a system. The following reaction:

$$m_{\text{2-Acetolactate}} + m_{\text{CO}_2} \overset{r_0}{\rightleftarrows} 2\, m_{\text{Pyruvate}}$$

would be represented by:

$$r_0 = \langle \{ \langle m_{\text{2-Acetolactate}}, 1 \rangle, \langle m_{\text{CO}_2}, 1 \rangle \}, \{ \langle m_{\text{Pyruvate}}, 2 \rangle \} \rangle$$

and would be an element of the following hypothetical universal set of reactions:

$$\mathcal{R}_{e.g.} = \{ r_0, \overleftarrow{r_1}, r_2, r_3 \}$$

A reaction r_i is identified by the subscript i, and any universal set may only contain one version of r_i (i.e., $\mathcal{R}'_{e.g.} = \{ r_0, \overleftarrow{r_0}, r_1 \}$ would not be a valid universal set).

The functions Ψ^- and Ψ^+ map reactions to metabolites. These metabolites are denoted as reactants (or substrates) and products of the reactions.

Definition 4. **(Reactants)**

The mapping of a single reaction to the set of its reactants is defined by the function Ψ'^-, while the mapping of the reactants of a reaction set is defined by the function Ψ^-. There are given by the following expressions:

$$\Psi'^- \colon \mathcal{R} \to \wp(\mathcal{M}) \qquad \Psi^- \colon \wp(\mathcal{R}) \to \wp(\mathcal{M})$$
$$\Psi'^-(r) = \bigcup_{p \,\in\, \pi_1(r)} \pi_1(p) \qquad \Psi^-(R) = \bigcup_{r \,\in\, R} \Psi'^-(r)$$

□

Definition 5. **(Products)**

On the other hand, the mapping of a single reaction to the set of its products is defined by Ψ'^{+}, while the correspondent mapping of the products of a reaction set is defined by the function Ψ^{+}, as follows:

$$\Psi'^{+}\colon \mathcal{R} \to \wp(\mathcal{M}) \qquad \Psi^{+}\colon \wp(\mathcal{R}) \to \wp(\mathcal{M})$$
$$\Psi'^{+}(r) = \bigcup_{p \,\in\, \pi_2(r)} \pi_1(p) \qquad \Psi^{+}(R) = \bigcup_{r \,\in\, R} \Psi'^{+}(r)$$

□

The function Ψ^{-} returns the set of all metabolites that participate as a reactant (or substrates) in the set of reactions R, while the function Ψ^{+} returns the set of all metabolites that are products of R. The function Ψ with no superscript returns the entire set of metabolites that participate in a set of reactions, which is equivalent to $\Psi(R) = \Psi^{-}(R) \cup \Psi^{+}(R)$.

The reverse functions are defined as consumers and producers of a metabolite set M, mapping metabolites to reactions.

Definition 6. **(Producers)**

The set of producers of a metabolite set is defined by the function φ^{-}.

$$\varphi^{-}\colon \wp(\mathcal{M}) \times \wp(\mathcal{R}) \to \wp(\mathcal{R})$$
$$\varphi^{-}(M, R) = \bigcup_{m \,\in\, M} \{r\colon r \in R \mid m \,\dot{\in}\, \pi_2(r)\}$$

□

Definition 7. **(Consumers)**

The set of consumers of a metabolite set is defined by the function φ^+:

$$\varphi^+ \colon \wp(\mathcal{M}) \times \wp(\mathcal{R}) \to \wp(\mathcal{R})$$
$$\varphi^+(M, R) = \bigcup_{m \in M} \{r \colon r \in R \mid m \,\dot{\in}\, \pi_1(r)\}$$

□

Because metabolites do not contain information about reactions, the φ functions require a set of reactions as a parameter. The φ is usually applied on the universal set of reactions, to identify all reactions related to a set of metabolites. In some cases, to simplify the notation, $\varphi(M, \mathcal{R})$ will be considered equivalent to $\varphi(M)$, assuming the omission of the reaction set parameter to correspond to the universal set of reactions.

Similar to the Ψ function, the φ without superscript is a function that returns all reactions in R, such that the metabolites in M participate either as a substrate or a product. This is equivalent to $\varphi(\langle M, R\rangle) = \varphi^-(\langle M, R\rangle) \cup \varphi^+(\langle M, R\rangle)$. These functions will play an important role in the algorithms to be describe later.

The metabolic domain $\mathcal{D}$ is a tuple, which contains all entities for a metabolic synthetic problem and will make the dataset supporting all algorithms. The domain associates an universal set of metabolites with a universal set of reactions, that together define all the chemical universe for the synthetic problem.

Definition 8. **(Metabolic Domain)**

A metabolic domain $\mathcal{D}$ is a tuple containing two universal sets, each corresponding to the universal set of each of the two entities that play a role in the metabolic synthetic problem: metabolites and reactions, defined as follows:

$$\mathcal{D} = \langle \mathcal{M}, \mathcal{R}\rangle, \text{ where } \Psi(\mathcal{R}) \subseteq \mathcal{M}$$

□

Definition 9. **(Synthetic Problem)**

Let $\mathcal{D} = \langle \mathcal{M}, \mathcal{R} \rangle$, be an arbitrary domain, then the triplet $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$, where $S, T \subseteq \mathcal{M}$ and $S \cap T = \emptyset$, is the definition of a synthetic metabolic problem. □

A synthetic problem describes the constraints of the metabolic pathway extraction algorithms. These correspond to the set T, containing the target metabolites that must be reached in the solutions, while the set S contains the initial substrates where the solutions may origin from. For a problem $\mathcal{Z}$ to make sense, there must exist at least a single initial compound and product, i.e., $|S| \geq 1$ and $|T| \geq 1$.

3.1.3 Set Systems

To address a synthetic problem, biological entities are mapped to a mathematical representation. A metabolic system can be represented using a variety of mathematical models, where some are less ambiguous than others at the expense of complexity. As we saw in the last chapter, the best representation for these systems are hypergraphs (Def. 10).

Definition 10. **(Hypergraph and Hyperarc [6])**

A directed hypergraph is a pair $\mathcal{H} = \langle V, E \rangle$, where $V = \{v_0, v_1, \ldots, v_n\}$ is the set of vertices and $E = \{e_0, e_1, \ldots, e_m\}$ is the set of hyperarcs. A hyperarc (which is a directed hyperedge) e_i is an ordered pair $e_i = \langle X_i, Y_i \rangle$ of disjoint subsets of V, i.e., $X_i \subseteq V, Y_i \subseteq V, X_i \cap Y_i = \emptyset, i = 0, \ldots, m$. □

In a directed hypergraph, the vertices represent metabolites, while each hyperarc corresponds to a reaction. The mapping of metabolites to vertices is direct , while in mapping reactions to hyperarcs, the stoichiometry is discarded. Although this could be included by adding an extra set to the edges that holds the stoichiometry value of each metabolite in each reaction, since topological analysis algorithms do not account for the stoichiometry value of the reactions, these can be discarded.

Definition 11. **(Vertex Degree)**

Given an arbitrary hypergraph $\mathcal{H} = \langle V, E \rangle$, the in-degree of a vertex v is the number of incoming edges at v. The out-degree is the number of outgoing edges of v. The degree is the total number of edges connected to v, i.e., the sum of the in and out degrees. □

Metabolites with zero in-degree or out-degree are denoted as dead-end nodes, being disconnected from the remaining nodes of the network. A vertex v_i with zero in-degree is a substrate-only metabolite, as there are no producers v_i, while a vertex with zero out-degree is a product-only metabolite.

Similar to the hypergraph, an alternative representation are the process graphs (p-graphs). The p-graphs were designed to solve the ambiguity of the directed graphs and signal-flow graphs to model chemical synthesis problems [23]. The p-graphs, although very similar to hypergraphs, have a distinct nomenclature for its components: instead of vertices, in p-graphs these are named materials (or species), and the operational units in p-graphs correspond to hyperarcs in a directed hypergraph. A p-graph is defined as follows:

Definition 12. **(Process graph and Operational Unit [23])**

A process graph is a pair $\mathcal{P} = \langle M, O \rangle$, where $M = \{m_0, m_1, \ldots, m_i\}$ is the set of materials (or species) and $O = \{o_0, o_1, \ldots, o_j\}$ is the set of operational units. An operational unit o is a tuple such that $o \subseteq \wp(M) \times \wp(M)$. Moreover, the vertices of the process graph are the elements of:

$$\mathcal{V} = M \cup O$$

and the arcs of the process graph are the elements of:

$$\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$$

where $\mathcal{A}_1$ contains the arcs that point towards the operating units:

$$\mathcal{A}_1 = \{\langle x, y \rangle \mid y = \langle \alpha, \beta \rangle \in o \; and \; x \in \alpha\}$$

and $\mathcal{A}_2$ keeps the arcs that point outwards of the operating units:

$$\mathcal{A}_2 = \{\langle y, x \rangle \mid y = \langle \alpha, \beta \rangle \in o \; and \; x \in \beta\}$$

□

The process graph is much similar to a common graph, containing vertices and arcs (or edges) that connect two vertices. Furthermore, by the definition of the p-graph (Def. 12), it is easy to identify that every p-graph must be bipartite as the arcs are either $\langle o, m \rangle$ or $\langle m, o \rangle$. Therefore, the set of vertices M and O are disjoint. Actually the p-graph is an adaptation of a common graph, where arcs are no longer identified by a connection of two vertices, but rather by a set of arcs defined by an operational unit (which is very similar to a hyperarc).

In a metabolic context, a hypergraph is similar to a process graph, but they diverge in the edge count as an edge in a process graph is a link between a material node and an operational unit. So, an operational unit has many edges, while in a hypergraph an edge corresponds directly to a reaction.

3.1.4 Conversion of biological domains to set systems

In most scenarios, the biological domain must be translated to a mathematical model prior to the execution of an algorithm. Given an arbitrary metabolic domain $\mathcal{D} = \langle \mathcal{M}, \mathcal{R} \rangle$, a function is required to map each element of $\mathcal{D}$ into the elements of a mathematical model (e.g., hypergraphs and process graphs).

For the presented models, the mapping is very simple simple, as both share a similar structure with the metabolic domain. The mapping of metabolites to vertices and materials is trivial since there is a one to one (bijective) transformation.

The mapping of reactions to hyperarcs and operational units is defined as following:

$$f_{\mathcal{R} \mapsto E \text{ or } O}(r) = \begin{cases} \{\langle X, Y \rangle, \langle Y, X \rangle\} & \text{if } \overleftrightarrow{r} \\ \{\langle X, Y \rangle\} & \text{if } \overrightarrow{r} \\ \{\langle Y, X \rangle\} & \text{if } \overleftarrow{r} \end{cases} \text{, where } X = \Psi^-(\{r\}) \text{ and } Y = \Psi^+(\{r\})$$

Reversible reactions are mapped into two edges in both directions. As an example, let $\mathcal{D} = (\{m_0, m_1, m_2, m_3\}, \{\overrightarrow{r_0}, \overleftarrow{r_1}, \overleftrightarrow{r_2}\})$ be a metabolic domain, this would be equivalent to the following directed hypergraph $\mathcal{H}$ and process graph $\mathcal{P}$:

$$\mathcal{H} = (\{v_0, v_1, v_2, v_3\}, \{\overrightarrow{e_0}, \overrightarrow{e_1}, \overleftarrow{e_2}, \overrightarrow{e_2}\})$$

$$\mathcal{P} = (\{m_0, m_1, m_2, m_3\}, \{\overrightarrow{o_0}, \overrightarrow{o_1}, \overleftarrow{o_2}, \overrightarrow{o_2}\})$$

For any irreversible reaction, a left to right operational unit and hyperarc is created, where the right to left reactions have their pairs swapped.

Since the directed hypergraphs and process graphs will be generated from a metabolic domain, by writing $\mathcal{Z} = \langle \mathcal{H}, S, T \rangle$ it will be equivalent to $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$,

being $\mathcal{H}$ the hypergraph created from $\mathcal{D}$ as stated above. The same applies for process graphs.

3.2 The Subgraph Extraction Problem

3.2.1 Solution Structures

In the previous section, all components of the synthetic ME problem were defined. In this section, the definition of a solution is introduced.

Topological algorithms compute structures, being the solutions characterized by a set of reactions (that can be represented by hyperarcs and operational units) that verify a set of conditions. These conditions are what defines a valid solution for a given synthetic problem.

Given an arbitrary synthetic problem $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$, defined over a metabolic domain $\mathcal{D} = \langle \mathcal{M}, \mathcal{R} \rangle$, a subset F from $\mathcal{R}$, is a solution to the synthetic problem if it satisfies $\Theta(F, \mathcal{Z})$, defined as follows.

Definition 13. **(Solution Structure)**

Let $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$ be an arbitrary synthetic problem, with $\mathcal{D} = \langle \mathcal{M}, \mathcal{R} \rangle$ the correspondent metabolic domain. The proposition $\Theta(F, \mathcal{Z})$, where $F \subseteq \mathcal{R}$, is true, if and only if:

1. for every $t \in T$, there is at least one reaction $r \in R$, that produces t;
2. for every $m \in \Psi^{-}(R)$, that is not in S, a reaction $r \in R$ exists, that produces m;
3. for every $r \in R$, at least one consumer of a product of r exists, unless r produces a metabolite that is an element of T.

□

Every solution structure is related to a synthetic problem, and therefore, is associated with a metabolic domain. To simplify, a set of reactions F and a synthetic problem $\mathcal{Z}$, that verifies the proposition $\Theta(F, \mathcal{Z})$ (or $\Theta_{\mathcal{Z}}(R)$), can be represented

by $\sigma(\mathcal{Z})$, meaning that σ is a solution structure of a synthetic problem $\mathcal{Z}$. Therefore, σ is a set of reactions and $\Theta(\sigma, \mathcal{Z})$ is verifiable. Because every solution is related to a particular synthetic problem $\mathcal{Z}$, for simplification in later sections, the $\mathcal{Z}$ is omitted.

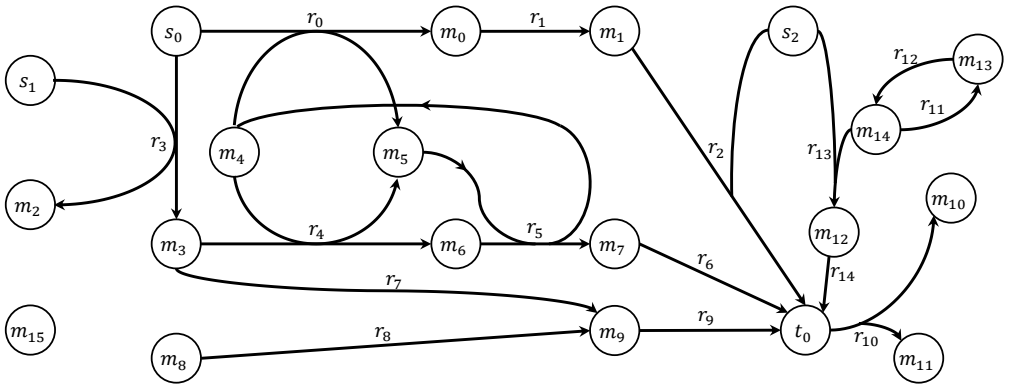


FIGURE 3.1: Example directed hypergraph: $V = \{t_0, s_0, s_1, s_2, m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}, m_{11}, m_{12}, m_{13}, m_{14}, m_{15}\}$; $E = \{r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}\}$

Let $\mathcal{Z}_{example} = \langle \mathcal{H}_{example}, \{s_0, s_1, s_2\}, \{t_0\} \rangle$ be a synthesis problem, and $\mathcal{H}_{example} = \langle V, E \rangle$ a directed hypergraph (Fig. 3.1). The condition 1 of Θ requires that if F is a solution of $\mathcal{Z}_{example}$, then $T \subseteq \Psi^+(F)$ (Ψ^+ reads as "products of"). In this particular example, the condition would be $t_0 \in \Psi^+(F)$. The second condition requires that every metabolite, that is a reactant in F, must be satisfied except for the ones in S (there is no need to satisfy the substrates of the problem). Together, these are enough to build fully satisfied solutions. The last condition is used to eliminate meaningless solutions.

Let $F_0 = \{r_3, r_4, r_5, r_6, r_7\}$ be a subset of $\pi_2(\mathcal{H}_{example})$, then $\Theta(F_0, \mathcal{Z}_{example})$ is not a valid solution structure, although F_0 is fully satisfied. Indeed, the hyperarc r_7 contains no element in T and no products of r_7 have a consumer in F_0, not obeying the third condition. This implies that this hyperarc can be discarded, although R_0 satisfies conditions 1 and 2 of Θ.

A solution structure space $\mathcal{S}$ is the set of all solution structures of a synthetic problem, being defined more formally as follows.

Definition 14. **(Solution Space)**

Let $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$ be an arbitrary synthetic problem. Then, the solution space of $\mathcal{Z}$, is defined by the set $\mathcal{S}$, as follows:

$$\mathcal{S}(\mathcal{Z}) = \{R : R \in \wp(\mathcal{R}) \mid \Theta(R, \mathcal{Z})\}$$

□

The solution space is a finite set of all subsets of the universal set of reactions in the metabolic domain, such that the property of solution structure is verified in each of these sets. This set is dependent of the synthetic problem, therefore it is relative to the domain and the specific constraints of the synthesis problem.

A solution structure is normally closed under union, i.e., for given two distinct σ_a and σ_b, then $\sigma_a \cup \sigma_b$ is also a solution structure, since T is included in both and every $M \backslash S$ metabolites in both sets are satisfied.

Theorem 1. **(Solution Structure Closure under Union [23])**

The union of two solution structures remains a solution structure, that is, if

$$\sigma_1 \in \mathcal{S}(\mathcal{Z}) \text{ and } \sigma_2 \in \mathcal{S}(\mathcal{Z})$$

then

$$\sigma_1 \cup \sigma_2 \in \mathcal{S}(\mathcal{Z})$$

□

The above theorem is not true if the reversible reactions are splitted into two irreversible reactions (i.e., $\overleftrightarrow{r} \mapsto (\overleftarrow{r}, \overrightarrow{r})$). In these scenarios, exclusivity should be guaranteed between the two directions, i.e., only one of the directions can be used in a solution. For the purpose of analysis of the algorithms, this property will be ignored and both of the versions are assumed as distinct.

This property allows to characterize the following types of solution structures:

Definition 15. **(Minimal Structure)**

Let σ be a valid solution structure, then σ is minimal if and only if there is no valid solution structure σ_x, such that $\sigma_x \subsetneq \sigma$ □

Definition 16. **(Combined Structure)**

Let σ be a valid solution structure, then σ is a combination of solution structures if and only if there are two valid solution structures σ_x and σ_y, that $\sigma_x \cup \sigma_y$ is equal to σ and $\sigma_x \subsetneq \sigma_y \wedge \sigma_y \subsetneq \sigma_x$ □

The minimal solution structures are the most relevant, as these are able to generate other solution structures through combinations and extensions.

The next section is dedicated to the maximal structure, that is written by the symbol μ, which is also a set of reactions that is a solution structure with a special relevance for the computation of other solutions.

3.2.2 The Maximal Solution Structure

An important aspect of subgraph computation is to determine several properties of the domain. It is relevant to identify if the problem is feasible, that is, if there is at least one valid solution structure. Because the σ is closed under union, the union of all $\sigma \in \mathcal{S}_{\mathcal{Z}}$ is a super structure that covers all valid solution structures. This super structure is defined as the maximal structure μ.

Definition 17. **(Maximal Structure [24])**

Let $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$ be an arbitrary synthesis problem. The maximal structure of $\mathcal{Z}$ is defined by:

$$\mu(\mathcal{Z}) = \bigcup_{\sigma \in \mathcal{S}(\mathcal{Z})} \sigma$$

□

Given an arbitrary synthesis problem $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$, if a maximal structure exists, then the problem is feasible (Def. 17). The maximal structure also allows to prune the entities of a domain. Let $\mathcal{D} = \langle \mathcal{M}, \mathcal{R} \rangle$ be an arbitrary domain, a network mapped from this domain can have: *a*) unreachable metabolites (i.e., metabolites that are associated with no reactions); *b*) two or more disjoint partitions of the network; *c*) reactions that are unsatisfiable. All of these elements can be discarded from the metabolic domain, as they only increase complexity for solution generating algorithms.

There are several algorithms to compute the maximal structure. However, the exact set is hardly achieved. The algorithms later introduced can only obtain an approximation to the maximal structure. An approximation of a set can be defined as follows:

Definition 18. **(Lower approximation [49])**

Let A be an exact well defined set and X be an arbitrary set.

If an element x, such that $x \in X$ and $x \in A$, this is written as $x \smile_X A$ ("x surely belongs to X in A"). This is defined as a strong membership. If for every $x \in X$, $x \smile_X A$, then X is a lower approximation of A, defined as:

$$\underline{Apr}_X(A) = \{x : x \smile_X A\}$$

□

Definition 19. **(Upper approximation [49])**

Let A be an exact well defined set and Y be an arbitrary set.

If an element y, such that $y \in Y$, but is not clear whether $y \in A$ or $y \notin A$, this is written as $x \frown_Y A$ ("y possibly belongs to Y in A"). This is defined as a weak membership. If for every $y \in Y$, $y \frown_Y A$, then Y is a lower approximation of A, defined as:

$$\overline{Apr}_Y(A) = \{y : y \frown_Y A\}$$

□

Let $\mathcal{D} = \langle \mathcal{M}, \mathcal{R} \rangle$ be an arbitrary metabolic domain, and $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$ a related synthetic problem. If, given an arbitrary set of reactions $F \subseteq \mathcal{R}$, where $F \subseteq \mu(\mathcal{Z})$, then F is an lower approximation of the maximal structure since the elements of R surely belongs to $\mu(\mathcal{Z})$. If, given an arbitrary set of reactions $F' \subseteq \mathcal{R}$, where $\mu(\mathcal{Z}) \subseteq F'$, then F' is an upper approximation of the maximal structure since elements of R' possibly belong to $\mu(\mathcal{Z})$.

For simplification, the lower approximation of the maximal structure is written as $\underline{\mu}$, such that $\underline{\mu} = \underline{Apr}_R(\mu)$, while upper approximation is written as $\overline{\mu} = \overline{Apr}_R(\mu)$.

3.2.3 Algorithms to Compute the Maximal Structure

The exact computation of the maximal structure μ of a synthesis problem $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$ is quite hard to achieve. Therefore, the algorithms shown in this section compute approximations of μ.

The maximal structure generation (MSG) algorithm was designed for the purpose of computing the exact maximal structure [24], being given as Algorithm 1[1]. The MSG computes μ in two steps: the reduction phase followed by the composition one.

The reduction step performs the following tasks:

- Elimination of every metabolite that operates as a single island (i.e., every metabolite that is not associated with any reaction).
- Elimination of every reaction that contains metabolites without producers that do not belong to the substrate set S. The elimination of any reaction might generate new reactions that are unsatisfiable, therefore, this task is iterative, being achieved by the loop of the reduction section (Algorithm 1).

At the end of the reduction step, it is expected that the domain $\mathcal{D}$ contains only reachable elements. Not all these elements belong to μ. For instance, if the network is composed by two disjoint sets of reactions R_1 and R_2, and T is a set with a single element, nothing implies that both R_1 and R_2 are satisfiable by S. However, since R_1 and R_2 are disjoint, T can only be present either on R_1 or R_2.

The composition step picks the domain computed by the reduction step and, by starting from the set T, it moves up to S by successively adding reactions until reaching S. If T is not present in the domain, then no reaction is added, so $\mu = \emptyset$ (i.e., the problem is infeasible).

The MSG algorithm fails only on one scenario, where there is a fully closed loop (futile cycle). As an example, consider the following reactions:

$$m_0 + m_1 \xrightarrow{r_0} m_2 + m_3$$

[1] Although the MSG algorithm has been designed to compute the exact maximal structure, we will show later that this is not the case.

Algorithm 1 Maximal Structure Generation

procedure MSG(M, O, R, P) ▷ M, set of materials, O set of operating units, R set of raw materials (equivalent to substrates S), P set of products (equivalent to target compounds T)
  $O \leftarrow O \backslash \varphi^{-}(R)$
  $M \leftarrow \Psi(O)$
  $r \leftarrow \Psi^{-}(O) \backslash (\Psi^{+}(O) \cup R)$
  while $r \neq \emptyset$ **do** ▷ Reduction loop
    let x **be an element of** r
    $M \leftarrow M \backslash x$
    $o \leftarrow \varphi^{+}(x)$
    $O \leftarrow O \backslash o$
    $r \leftarrow (r \cup (\Psi^{+}(o) \backslash \Psi^{+}(O))) \backslash x$
  end while
  if $P \cap M \neq \emptyset$ **then**
    STOP ▷ There is no maximal structure
  end if
  $p \leftarrow P$
  $m \leftarrow \emptyset$
  $o \leftarrow \emptyset$)
  while $p \neq \emptyset$ **do** ▷ Composition loop
    let x **be an element of** p
    $m \leftarrow m \cup x$
    $o_x \leftarrow \varphi^{-}(x)$
    $o \leftarrow o \cup o_x$
    $p \leftarrow (p \cup \Psi^{-}(o_x)) \backslash (R \cup m)$
  end while
  $m \leftarrow \Psi(o)$
  return m
end procedure

$$m_2 + m_3 \xrightarrow{r_1} m_4 + m_5$$

$$m_4 + m_5 \xrightarrow{r_2} m_0 + m_1$$

Reactions r_0, r_1, r_2 perform a loop, which is an elementary mode as it operates without any substrates, having net flux of zero. Now assuming the reaction $m_2 + m_4 \xrightarrow{r_A} m_6$ is added to the set, the algorithm would consider r_A to be reachable, as every substrate of r_A has a producer and their producers are satisfiable. From this example, it is clear that every element included that is dependent of the loop r_0, r_1, r_2 will eventually lead to infeasible solutions, as they are false positives.

This scenario is much more complex when dealing with reversible reactions. A reversible reaction is duplicated in the network, such that given a reaction $A \overset{r_Y}{\rightleftarrows} B$, this would generate two edges corresponding to $A \xrightarrow{\overrightarrow{e_Y}} B$ and $B \xrightarrow{\overleftarrow{e_{Y'}}} A$. The edges $\overrightarrow{e_Y}$

and $\overleftarrow{e_{Y'}}$ self satisfy for any A and B. Therefore, all reversible reactions connected to T are included, no matter if they are satisfiable or not. This shows that $MSG(\mathcal{Z})$ computes $\overline{\mu}(\mathcal{Z})$, that is an upper approximation, as only cases of false positives were pinpointed.

The FindAll (FA) algorithm [6] (Alg. 2) is a subroutine of the FindPath algorithm (a subgraph extraction algorithm that is described in a later section). The purpose of FA is to provide a function that identifies if the problem is feasible and sort the reactions in the domain in s specific order. It is shown as Algorithm 2.

Algorithm 2 Find All

```
1: procedure FindAll(H, S)                    ▷ H hypergraph, S source metabolites
2:     for each r ∈ H do
3:         m[r] ← Ψ⁻(r)
4:     end for
5:     V ← S
6:     D ← S
7:     F ← ∅
8:     while V ≠ ∅ do
9:         let x be an element of V
10:        V ← V\x
11:        D ← S ∪ x
12:        for each r ∈ H ∧ x ∈ m[r] do
13:            m[r] ← m[r]\x
14:            if m[r] = ∅ then
15:                F ← {F, r}
16:                for each j ∈ Ψ⁺(r) ∧ x ∉ D do
17:                    V ← V ∪ j
18:                end for
19:            end if
20:        end for
21:    end while
22:    return F
23: end procedure
```

Let R be an ordered set of reactions and S a set of substrates, the set R is traversable if, for every element $r_i \in R$, then r_i must be satisfiable by its previous elements plus the substrate set, i.e., by $\langle S, \{r_0, \ldots, r_{i-1}\}, \rangle$.

The $m[r]$ (Alg. 2, line 3), maintains all dependencies of the hyperarcs (which correspond to reactions). The sets of metabolites D and V are initialized as S, then the algorithm successively removes elements in V from $m[r]$. If a given $m[r]$ has no dependencies, then the tail elements of the hyperarc r are added to V, such that these contribute to new substrates that might satisfy other reactions.

The main problem of this algorithm is the existence of circular dependencies. Given an arbitrary domain $\mathcal{D}$, a metabolite m is a circular dependency if the following condition is verified: every producer of m_x requires as substrates m_y and every producer of m_y requires as substrates m_x.

Assuming that both $\{m_x, m_y\}$ are not included in S, then to acquire m_x a producer of m_x is required to be included. However, every producer of m_x requires the substrate m_y, thus m_x can only be included if m_y is achieved, since m_y is never reached because it requires m_x. Then, reactions containing $\{m_x, m_y\}$ are excluded.

A cyclic dependency does not necessarily generate infeasible solutions. A common example includes the following reactions patterns:

$$A + m_{ATP} \overset{r_0}{\rightleftarrows} m_{ADP} + B$$

$$C + m_{ATP} \overset{r_1}{\rightleftarrows} m_{ADP} + D$$

$$E + m_{ATP} \overset{r_2}{\rightleftarrows} m_{ADP} + F$$

These reactions are feasible if at least a pair of reactions from $\{r_0, r_1, r_2\}$ is added. Therefore, this algorithm may generate false negatives, which means that the Find-All algorithm computes a lower approximation of μ. This later has consequences in solution generating algorithms, since a lower approximation of μ implies loss of solution structures.

3.2.4 Domain Partition

It is common that metabolic databases assemble massive networks of reactions and metabolites. Depending on the constraints of the problem, the computation of pathways from these networks usually takes an unpractical amount of time.

A partition strategy is proposed to break the networks into multiple sets. The goal is to apply a meaningful approach to reduce the domain, such that the solutions of the subsets share a common property.

Given an arbitrary domain $\mathcal{D}$, the space of the reactions is split into several layers, defined as a radius, such that $\mathcal{D}^i = \langle \mathcal{M}^i, \mathcal{R}^i \rangle,\ i \in \mathbb{N}_0^+$, where i is the *i-th* radius of the domain. The decision to assign each reaction and metabolite to a specific

radius is based on a starting set of metabolites C. These metabolites are defined as center metabolites, that belong to $\mathcal{D}^0 = \langle \mathcal{M}^0, \emptyset \rangle$, where $\mathcal{M}^0 = C$.

The elements of $\langle \mathcal{M}^1, \mathcal{R}^1 \rangle$ are adjacent entities of $\mathcal{D}^0$, where $\mathcal{R}^1$ are producers of metabolites in $\mathcal{M}^0$ and $\mathcal{M}^1$ are reactants of reactions in $\mathcal{R}^1$. These form the first layer $\mathcal{D}^1$ of the domain centered in C.

The *i-th* radius can be defined by:

$$\mathcal{D}^i(C, \mathcal{R}) = \begin{cases} \langle C, \emptyset \rangle & \text{if } i = 0 \\ \langle \Psi^-(\mathcal{R}^i) \cup \mathcal{M}^{i-1}, \mathcal{R}^i \rangle & \text{if } i > 0 \end{cases}$$

$$\text{where } \mathcal{M}^{i-1} = \pi_1(\mathcal{D}^{i-1}(C, \mathcal{R})), \mathcal{R}^i = \varphi^-(\mathcal{M}^{i-1}, \mathcal{R})$$

This implies that $\mathcal{D}^i \subseteq \mathcal{D}^{i+1}, n \geq 1$. The elements in $\mathcal{D}^i \cap \mathcal{D}^{i+1} = (\mathcal{M}^i \cap \mathcal{M}^{i+1}, \mathcal{R}^i \cap \mathcal{R}^{i+1})$, will be the elements specific of $\mathcal{D}^{i+1}$ (i.e., the newly added elements in radius $i + 1$).

Let $\mathcal{Z} = \langle \mathcal{D}, S, T \rangle$ be an arbitrary synthetic problem. By assigning C in the previous definition to T (the set of target metabolites for the problem, the *i-th* radius of $\mathcal{Z}$ is represented by $\mathcal{Z}^i$, such that $\mathcal{Z}^i \subseteq \mathcal{Z}$. Because the *i-th* radius elements are adjacent elements to the previous radius $(i - 1)$, this implies that $\mathcal{Z}^i = \langle \mathcal{D}^i, S, T \rangle$ contains all solution structures of size i.

3.3 Computing Solution Structures

Now that the properties of the subgraphs are defined, in this section the algorithms to compute pathways (subgraphs) are analyzed. The goal is to identify the capabilities of each algorithm and the scope of the solution structures that each of these are able to compute. These algorithms operate over set systems, by performing a topological analysis over the networks to extract solution structures.

For most of the algorithms, no complexity analysis is performed. This is due to the fact that most algorithms are not trivial to analyze, and the average scenario is highly dependent on the topology of the network.

3.3.1 Minimize Algorithm

The Minimize algorithm [6] is a subroutine of the FindPath algorithm (explained in a later section) that extracts a single minimal solution from a directed hypergraph.

Given an arbitrary hypergraph $\mathcal{H}$, a set of substrates S and a set of target products T, a hyperarc e of $\mathcal{H}$ is critical if by removing e from $\mathcal{H}$, then there is no maximal structure and, therefore, the problem is infeasible.

Definition 20. **(Critical Edge)**

Let $\mathcal{Z} = \langle \mathcal{H}, S, T \rangle$ be an arbitrary synthetic problem, and $S \subseteq \mathcal{S}(\mathcal{Z})$ a set of solution structures, a hyperarc $e \in \mathcal{H}$ is critical in S if and only if $e \in \bigcap_{\sigma \in S} \sigma$.

□

A minimal solution is equivalent to a set of hyperarcs, where every hyperarc is critical. The Minimize algorithm (Alg. 3) tests each hyperarc for the criticality property, and for every hyperarc that fails the test it is removed from $\mathcal{H}$. This algorithm is given as Algorithm 3.

In the definition of the algorithm (Def. 3) it receives an extra parameter R_f, a set of reactions that cannot be removed from $\mathcal{H}$ (Alg. 3, line 8). This parameter is only used in the FindPath algorithm to obtain alternative solutions, which are not necessary to compute σ_{min}.

The Minimize algorithm obtains a minimal solution, but not a specific minimal solution. A hypergraph $\mathcal{H}$ may contain multiple minimal solutions. The σ_{min} returned from the Minimize algorithm is dependent on the ordering of the hyperarcs in $\mathcal{H}$. Let $\mathcal{H}$ be a hypergraph defined by the set of edges in Figure 3.2, where $\mathcal{S}(\mathcal{Z}) = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ are all minimal structures for a hypothetical synthesis problem (the set of substrates S and product T are irrelevant for this example as the minimal structures are given), then $\mathcal{H} = \mu = \bigcup_{i=0}^{4} \sigma_i$.

The hyperarcs are sorted by index. If the algorithm (Alg. 3, line 7) picks each hyperarc by this order, then the output would be σ_4. If the algorithm picks the edges by the reverse order (or the edges are sorted in the reverse order), then the output would be σ_0.

Algorithm 3 Minimize

```
1: procedure MINIMIZE(H, R_f, S, T)   ▷ H hypergraph, R_f reactions to not test, S source set, T target set
2:     F ← FindAll(H, S)   ▷ 2-4 Test if μ ≠ ∅
3:     H' ← H
4:     if T ∩ Ψ+(F) = ∅ then
5:         H' ← ∅   ▷ μ = ∅ return ∅
6:     else   ▷ μ ≠ ∅ proceed to minimization
7:         for each r ∈ H do   ▷ For each reaction not in R_f test if μ ≠ ∅ for H\r
8:             if r ∉ R_f then
9:                 F ← FindAll(H\r, S)
10:                if T ∩ Ψ+(F) ≠ ∅ then
11:                    H' ← H'\r   ▷ Remove reaction from hypergraph
12:                end if
13:            end if
14:        end for
15:    end if
16:    return H'   ▷ Return either ∅ or a minimal solution structure of H
17: end procedure
```

$$E_{\mathcal{H}} = \{r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}\}$$
$$\sigma_0 = \{r_0, r_2, r_5, r_6\}$$
$$\sigma_1 = \{r_0, r_3, r_4, r_8\}$$
$$\sigma_2 = \{r_1, r_7, r_{12}, r_{13}\}$$
$$\sigma_3 = \{r_1, r_9, r_{10}, r_{11}\}$$
$$\sigma_4 = \{r_1, r_{14}, r_{15}\}$$

FIGURE 3.2: Hypothetical synthetic problem $\mathcal{Z} = \langle \mathcal{H}, S, T \rangle$, with $\mathcal{S}_{min}(\mathcal{Z}) = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$

3.3.2 FindPath Algorithm

The FindPath (FP) algorithm [6] combines the FA and Minimize algorithms to enumerate all minimal pathways.

Let $\mathcal{H}$ be an arbitrary hypergraph $\mathcal{H} = \langle V, E \rangle$, and $\mathcal{Z} = \langle \mathcal{H}, S, T \rangle$ a synthesis problem, where T is a singleton set with the metabolite t_0. If $\mu \neq \emptyset$, then $Minimize(\mathcal{H}, \emptyset, S, \{t_0\}) = \sigma_0 \neq \emptyset$, ($\sigma_0$ is the first solution structure obtained by the Minimize algorithm). Let F be an ordered list of hyperarcs in σ_0, where $e_i \in F$ is the last element, the one closest to the product (i.e, $F[i]$ produces t_0, where i is the index of the last element of F), then the alternative solutions are either the following:

1. the solution structures that do not contain e_i;

2. the solution structures that contain e_i, but do not contain e_{i-1}.

As previously demonstrated, the FA computes a lower approximation of μ, such that there are false negatives. This implies that some hyperarcs that are excluded might belong to solutions, that are excluded from the results.

Algorithm 4 Find Path

1: **procedure** FINDPATH($\mathcal{H}, R_f, S, T$) ▷ $\mathcal{H}$ hypergraph, S source metabolites, T target metabolites, R_f for branching solutions (initially as $\emptyset$)
2: $F \leftarrow FindAll(\mathcal{H}, S)$
3: $\mathcal{H}' \leftarrow \emptyset$
4: $\mathcal{H}' \leftarrow \mathcal{H}' \cup F \cup R_f$
5: $\mathcal{H}_\sigma \leftarrow Minimize(\mathcal{H}', R_f, S, T)$ ▷ $\mathcal{H}_\sigma$ the first minimal solution
6: $En \leftarrow \emptyset$
7: **if** $\mathcal{H}_\sigma \neq \emptyset$ **then**
8: $En \leftarrow \mathcal{H}_\sigma$
9: $F \leftarrow FindAll(\mathcal{H}_\sigma, S)$
10: **for** $k \in \{|F|..1\}$ **do** ▷ for each element in F (i.e., hyperarcs of $\mathcal{H}_\sigma$) branch alternative solutions
11: $r = F_k$
12: **if** $r \notin R_f$ **then**
13: $En \leftarrow \{En,$ FindPath$(\mathcal{H} \backslash r, R_f, S, T)\}$
14: $R_f \leftarrow R_f \cup r$
15: **end if**
16: **end for**
17: **end if**
18: **return** En
19: **end procedure**

The algorithm (given as Algorithm 4) starts by computing a single solution. Then, to branch to other solutions, the FA computes subproblems with the domain (hypergraph) smaller for each subproblem, using either alternative 1 (Alg. 4, line 13) or alternative 2 (the remaining of the loop). Each cycle of the loop generates more branches of type 1.

3.3.3 Solution Structure Generation Algorithm

The Solution Structure Generation (SSG) [25] algorithm enumerates all possible combinations of reactions. To test every combination of reactions, this would require a huge amount of wasteful computation, as there are much more infeasible combinations, than the actual solution structures. The total amount of possible

combinations can be expressed by 2^n, where n is the number of reactions. This is equivalent to the power set of the reactions set (i.e., $\wp(\mathcal{R})$), since $\mathcal{S} \subseteq \wp(\mathcal{R})$.

Let $\mathcal{P}_1 = \langle \mathcal{M}_1, \mathcal{O}_1 \rangle$ be a process graph (Fig. 3.3), where $\mathcal{M}_1 = \{s_0, s_1, m_0, m_1, t_0\}$, and $\mathcal{O}_1 = \{r_0, r_1, r_2\}$. Here,

$$\wp(\mathcal{O}_1) = \{\{\emptyset\}, \{r_0\}, \{r_1\}, \{r_2\}, \{r_0, r_1\}, \{r_0, r_2\}, \{r_1, r_2\}, \{r_0, r_1, r_2\}\}$$

Let $\mathcal{Z}_1 = \langle \mathcal{P}_1, S_1, T_1 \rangle$ be a synthesis problem, such that $S_1 = \{s_0, s_1\}$ and $T_1 = \{t_0\}$; $\mathcal{S}(\mathcal{Z}_1) \subseteq \wp(\mathcal{O}_1)$ by the definition of the solution space (Def. 14); where $\wp(\mathcal{O}_1)$ contains the solutions of the problem $\mathcal{Z}_1$, being $\mathcal{S}(\mathcal{Z}_1) = \{\sigma_0, \sigma_1, \sigma_2\}$, , where to $\sigma_0 = \{r_0, r_1\}, \sigma_1 = \{r_0, r_2\}, \sigma_2 = \{r_0, r_1, r_2\}$.

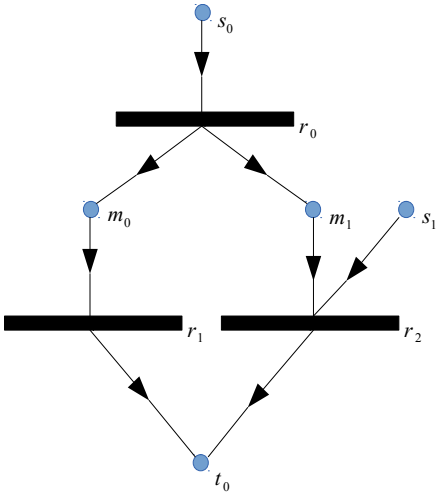


FIGURE 3.3: Example P-graph.

A requirement of a solution structure is that every set of reactions that satisfies Θ must contain at least one producer of each element in T_1, so at least one element of $\varphi^+(T_1) = \{r_1, r_2\}$ must be included for every element of $\mathcal{S}(\mathcal{Z}_1)$. This $\varphi^+(T_1)$ is defined as a maximal decision mapping of T_1, written as $\Delta(T_1) = \varphi^+(T_1, \mathcal{O}_1)$ (or $\varphi^+(T_1)$ as defined earlier by omitting the reaction parameter, the universal set of reactions is assumed).

Definition 21. **(Decision Mapping)**

A solution structure can be represented by a map $\delta[m]$, denoted as decision mapping of a metabolite m, that maps the corresponding metabolite to its set of producers (reactions). The maximal decision mapping $\Delta(m)$ is the set of all possible reactions (which corresponds to the producers of m), that can be mapped to the metabolite m. □

The problem can be decomposed into subproblems. Consider the subproblem, where the producers of $T_1 = \Delta(T_1)$ are excluded from $\mathcal{P}_1$. Let $\wp_i$ be an element of $\wp(\Delta(T_1))\backslash\emptyset$, where $\wp_0, \wp_1, \wp_2$ are equal to $\{r_1\}, \{r_2\}, \{r_1, r_2\}$, respectively. The solution structures of $\mathcal{Z}_1$ is the union of each subproblems $(\wp_i \cup \mathcal{S}(\mathcal{Z}_{\wp_i}))$ and $\mathcal{Z}_{\wp_i} = \langle \mathcal{P}_1\backslash\Delta(T_1), S_1, \Psi^+(\wp_i)\rangle$.

As an example, if $i = 0$, then $\wp_0 = \{r_1\}$, where $\mathcal{Z}_{r_1} = \langle \mathcal{P}_1\backslash\{r_1, r_2\}, S_1, \{m_0\}\rangle$, then $\mathcal{S}(\mathcal{Z}_{r_1}) = \{\{r_0\}\}$ and $r_1 \cup \mathcal{S}(\mathcal{Z}_{r_1}) = \{\{r_1, r_0\}\} = \{\sigma_0\}$.

Any subproblem is a synthesis problem, therefore each subproblem can further be decomposed into smaller subproblems. This recursive strategy of problem decomposition is the strategy applied in the SSG algorithm (Alg. 5). Using this strategy, it is possible to compute every combination of solution structures, without resorting to exhaustive enumeration.

Algorithm 5 Solution Structure Generation

```
1: procedure SSG(T, M, δ[M])
2:     if T = ∅ then
3:         return δ[M]                          ▷ δ[M] is a solution structure
4:     end if
5:     let x ∈ P
6:     C ← ℘(Δ(x))\{∅}                          ▷ Generate all combinations of Δ(x)
7:     for c ∈ C do                             ▷ For each combination test if is valid
8:         if ∀y ∈ m, c ∩ δ̄(y) = ∅ ∧ (Δ(x)\c) ∩ δ(y) = ∅ then
9:             δ[m ∪ {x}] ← δ[m] ∪ {(x, c)}
10:            SSG((p ∪ φ⁻(c))\(R ∪ m ∪ {x}), m ∪ {x}, δ[m ∪ {x}])
11:        end if
12:    end for
13:    return
14: end procedure
```

3.4 Improvements to the Algorithms

In this section, several modifications are proposed for both of the solution extraction algorithms, for the purpose of computing solutions in large database metabolic networks, such as the KEGG and BioCyc.

3.4.1 Minimize: Binary Search Heuristic

The Minimize algorithm (Alg. 3) reduces the network to a minimal set of reactions. A limitation is the quadratic complexity, since as the size of the hypergraph increases there is a significant increase in computational time.

An alternative heuristic is here proposed to scale it for large data sets, such as the KEGG Ligand chemical universe.

Let $\mathcal{Z} = \langle \mathcal{H}, S, T \rangle$ be an arbitrary synthetic problem, with $\mu \neq \emptyset$, then there is at least one solution for the synthesis problem $\mathcal{Z}$. This strategy assumes that for the specific problem $\mathcal{Z}$, the elements of $\mathcal{S}_{min}(\mathcal{Z})$ are much smaller than μ, i.e, the problem contains a huge amount of small solutions, that together assemble a big maximal structure. This ratio is expressed by the ω symbol.

Definition 22. **(Solution Structure Ratio)**

The solution structure ratio ω will be a real number, in the range $0 < \omega \leq 1$, such that, $\omega = \frac{|\sigma(\mathcal{Z})|}{|\mu(\mathcal{Z})|}$

□

Let σ_{min} be an arbitrary solution structure. If $\omega = 1$, then μ definitely has only one solution structure, such that $\sigma = \mu$; otherwise, the number of reactions in μ is greater than the number of reactions in σ_{min} (i.e., $\omega < 1$). The sequential test of each reaction is most suited for synthesis problems where $\omega > 0.5$, since there are more reactions to keep that to remove. However, in real data sets, where synthesis problems apply to an organism chassis, the substrate set S is usually extensive, and this implies less substrates must be satisfied, thus it is more probable to have smaller solution structures.

As an hypothesis, we will assume a scenario where the domain is much larger than the size of the solutions, meaning that the domain is composed by many small solutions. Therefore, each of these solutions has $\omega < 0.5$.

In these cases, the number of reactions to remove is higher than those to keep. A consequence of this property is that the objective is shifted to identify which reactions to keep instead of those to remove, as most reactions can be discarded.

The proposed heuristic is to use a binary search strategy similar to the bisection optimization method, such that, instead of testing each reaction one by one, a

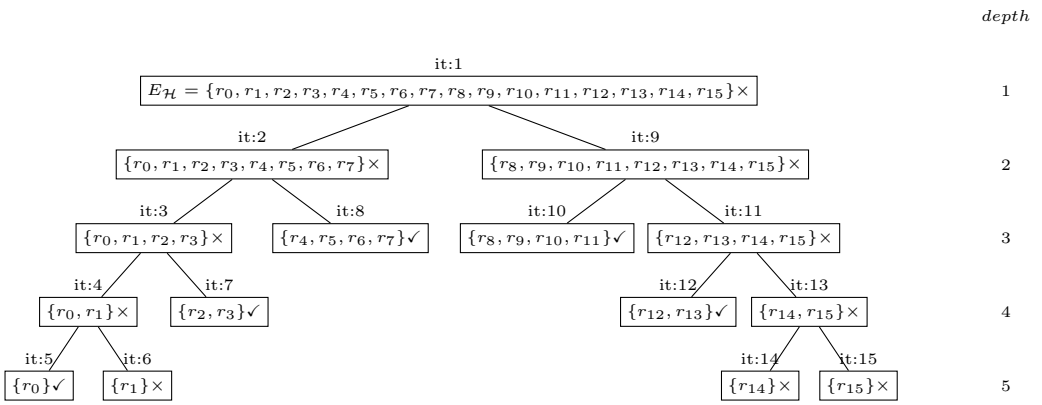


FIGURE 3.4: Binary Search recursion tree. × - critical set. ✓ - discarded set. The leafs identified as critical corresponds to $\sigma_4 = \{r_1, r_{14}, r_{15}\}$

binary search is performed to identify critical reactions to keep and discard the rest of the reactions in the sets.

As an example, in the hypothetical synthesis problem $\mathcal{Z}$ in the previous section (Fig. 3.2), the minimal solution structure has $(\omega_0, \omega_1, \omega_2, \omega_3, \omega_4) = (0.25, 0.25, 0.25, 0.25, 0.1875)$. The sequential search version of Minimize required a total of 16 FA calls (which is the number of edges in $E_{\mathcal{H}}$) to compute the first minimal solution structure (σ_4).

The binary search heuristic attempts to identify which portion of the set is non critical. The algorithm starts by selecting the entire set of reactions $E_{\mathcal{H}}$, then it recursively attempts to remove the set by splitting it into two equally sized subsets. The critical reactions eventually become singleton sets with only one element. If this element is not discardable, then it is a critical reaction.

Figure 3.4 shows the generated recursion tree using the binary search heuristic, for the hypothetical synthesis problem $\mathcal{Z}$ (Fig. 3.2). From the given example, there are no critical reactions in $E_{\mathcal{H}}$, such that, $\sigma_0 \cap \sigma_1 \cap \sigma_2 \cap \sigma_3 \cap \sigma_4 = \emptyset$, which implies that any reaction can be removed. However, if r_0 is excluded from $\mathcal{H}$ then the remaining set of solution structures is $\{\sigma_2, \sigma_3, \sigma_4\}$, where $\sigma_2 \cap \sigma_3 \cap \sigma_4 = \{r_1\}$. The hyperarcs r_0 and r_1 can not be removed simultaneously. This heuristic is similar to the previous attempts to remove the leftmost reaction. The algorithm descends to depth 5 and removes r_0, then is forced to keep r_1. The output solution structure is σ_4 because it has the rightmost reactions, where σ_2 and σ_3 are excluded in iterations 8 and 10.

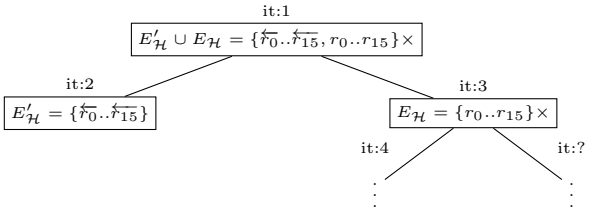


FIGURE 3.5: Binary Search recursion tree for the extended problem with the reversible reactions. $\times$ - critical set. ✓ - discarded set. The leafs identified as critical corresponds to $\sigma_4 = \{r_1, r_{14}, r_{15}\}$

In this example, the presented heuristic did not profit much, the sequential would require 16 iterations (i.e., one for each reaction), while the new heuristic observed only a gain of 1 iteration.

Now assuming the size of the problem is doubled, when, for each reaction the inverse direction is included (e.g., the reactions are now reversible). If arbitrary reactions are added to the network, the number of solutions remains always equal or greater than the original problem. This is because new reactions do not invalidate the original set of reactions, unless the problem constraints change.

Let $E'_{\mathcal{H}}$ be a set of reverse reactions of $E_{\mathcal{H}}$, then by adding $E'_{\mathcal{H}}$ to the head of $E_{\mathcal{H}}$, we have, $E'_{\mathcal{H}} \cup E_{\mathcal{H}} = \{\overleftarrow{r_0}..\overleftarrow{r_{15}}, \overrightarrow{r_0}..\overrightarrow{r_{15}}\}$. The original solutions still hold for all $\sigma_0, \ldots, \sigma_4$ plus new possible combinations of solution structures using the newly added reverse reactions (belonging to the set Σ'). Independently of the content of Σ', the solution structure σ_4 still holds for the rightmost solution, this is because the set $\mathcal{H}'$ is added to the head of the original set. The new domain $\mathcal{H}' \cup \mathcal{H}$ would require 32 iterations in the sequential version, however only a increase of two more iterations is required to obtain σ_4 (Fig. 3.5) by using a binary heuristic. It is also possible to observe that there is a decrease of ω by half for all $\sigma_0, \ldots, \sigma_4$ as the size of μ doubled.

This heuristic is most profitable when ω is very small. Analogous to the original approach (Alg. 3), the ordering of the reactions in the set does impact the output solution. Assuming that a new ordering is applied, where the reactions are ordered as follows: $\mathcal{H} = \{r_0, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_1, r_{14}, r_{15}\}$, this would decrease the amount of iterations to 10 (Table 3.1), a decrease of 1/3 of the required iterations.

To compute $\mathcal{S}_{min}$, this is still in charge of the FindPath algorithm, as branching technique is still required to obtain alternative solution structures.

TABLE 3.1: Recursion table for the example described in Figure 3.2. The reactions are sorted as follows: $E_{\mathcal{H}} = \{r_0, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_1, r_{14}, r_{15}\}$

depth	*iteration*	set to test	remove
2	1	$\{r_0, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$	✓
2	2	$\{r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_1, r_{14}, r_{15}\}$	×
3	3	$\{r_9, r_{10}, r_{11}, r_{12}\}$	✓
3	4	$\{r_{13}, r_1, r_{14}, r_{15}\}$	×
4	5	$\{r_{13}, r_1\}$	×
5	6	$\{r_{13}\}$	✓
5	7	$\{r_1\}$	×
4	8	$\{r_{14}, r_{15}\}$	×
5	9	$\{r_{14}\}$	×
5	10	$\{r_{15}\}$	×

The disadvantage of this heuristic is when $\omega \approx 1$, this will force to branch most of the leafs with singular sets. Assuming the example above $\mathcal{H} = \mu = \sigma_0$, the recursion tree would actually breach to every node with the singular set of $r_i \in \sigma_0$. This happens because all reactions belongs the only existing minimal solution. This is the worst case scenario for this heuristic, where the number of FA calls is equal to $2n - 1$, where n is the number of elements in the root node.

3.4.2 Solution Structure Generation: Power Set

A limitation of the SSG algorithm is the requirement to compute a power set. The number of the elements of the power set is exponential (2^n), implying that for a reasonable small set, i.e., 10 elements would generate 1024 possible combinations to be tested. In a large metabolic network, several metabolites can easily exceed a set of 10 or more producers, generating a colossal amount of combinations to test and consuming a huge amount of memory.

In this section, an alternative strategy is proposed in order to force the SSG routine to compute only σ_{min}. Consider the example synthesis problem and the p-graph described in the previous section (Fig. 3.3). A solution structure, if it exists, must be built by either one of the elements of the $\wp(\Delta(t_0))$.

The power set of n elements can be subdivided into n subsets, where the *n-th* set corresponds to a subset that contains every combination of n elements. These subsets are defined as $\wp_n(X)$, $0 \leq n \leq |X|$.

Assume that a solution exists for a combination $c \in \wp_i(X)$, then every combination of higher degree $\wp_{i+1}(X)$, that contains c, can be excluded, as these do not generate the minimal solution.

In the previous example, it was shown that a solution structure must be either composed by any of the combinations of $\{r_1, r_2\}$, such that $\wp_1(\{r_1, r_2\}) = \{\{r_1\}, \{r_2\}\}$ and $\wp_2(\{r_1, r_2\}) = \{\{r_1, r_2\}\}$. If σ exists built on $\{r_1\}$, then every $\wp_{i>1}(\{r_1, r_2\})$ containing $\{r_1\}$ is not minimal, and this renders $\wp_2(\{r_1, r_2\}) = \emptyset$.

Assuming σ_{r_1} exists built from r_1, this implies that a subgraph $\mathcal{P} \subseteq \mathcal{P}' \backslash \{r_1, r_2\}$ exists, that attached to r_1 is a solution structure. Then, if a solution structure $\sigma_{r_1 \cup O}$ exists, built from $r_1 \cup O$, which is any combination containing r_1. The r_1 of $\sigma_{r_1 \cup O}$ can be satisfied by $\mathcal{P}$ and the remaining O must be either satisfied by $\mathcal{P}$ or $\mathcal{P} \cup \mathcal{P}', \mathcal{P}' \subseteq \mathcal{P}' \backslash \{r_1, r_2\}$. By adding $\mathcal{P}$, this implies that $\sigma_{r_1} \subset \sigma_{r_1 \cup O}$, therefore $\sigma_{r_1 \cup O}$ is not a minimal solution structure.

This approach restricts the SSG algorithm to generate only minimal solutions, which in return greatly increases the size limit of the p-graph.

3.5 Microorganism Selection

A different but related problem is to find the microorganism that is most fit for the production of a certain compound (or set of compounds). Although, in this case, the objective has shifted, the problem remains similar: to compute pathways that link substrates to the target set of compounds.

The major difference is that, at the end, the result is an organism instead of a pathway. This requires to establish a link between the reactions in the metabolic network and the microorganisms that are able to achieve those.

The reactions can be associated with genes from the different microorganisms. One way to achieve this link is to associate reactions to enzymes, that are identified by Enzyme Commission numbers (EC), and use the EC to find related genes in the microorganisms. This allows to find which organisms contain a given reaction in the in metabolic system. However, this is highly dependent on the annotation of the genes and the information contained in the databases.

In our algorithms, the identification of the most suitable organisms can be performed after the computation of the solutions. The solution structures are solved as a synthesis problem as before. Afterwards, the solutions are characterized for the microorganism that best fits. A best fitting microorganism is the one that covers the maximum number of reactions (eventually covering all reactions), meaning that the generated pathway is most probably included in the microorganism metabolic system.

A problem in mapping reactions to organisms is the quality of the annotation. With the colossal amount of taxonomy, it is impossible to assign every organism to each of the reactions or enzymes. Indeed, most biochemical databases contain only a limited amount of microorganisms, having many reactions unassigned.

To deal with unassigned reactions, they are either excluded or considered that they can occur in any organism. By excluding these, this may ultimately lead to no solutions if they are critical reactions. However, by considering any organism, it is also possible to bias the selection, which eventually requires user analysis of the results.

Chapter 4

The Biosynth Framework

The selection of pathway optimization algorithms is just one of the steps of the problem characterized in the previous chapter. There are many other crucial aspects, such as raw data processing, management and quantification of results. In a biological context, it is common for raw data to contain redundancies and inconsistencies. These should be identified prior to the execution of the algorithms, otherwise inconsistent solutions may occur.

Another problem to address is the diverse type of input structures that are required by the algorithms. Distinct algorithms operate on distinct data structures depending on their implementation. Therefore, it is not possible to define a generic data structure and the raw chemical data must be translated to a specific data model prior to the computation. Finally, it is necessary to implement a strategy to quantify the quality of the solutions obtained from the algorithms.

All these issues are addressed by the framework for synthetic pathway optimization developed in this work. The architecture of this framework, as well as some implementation details, are given in this chapter.

4.1 The Framework Architecture

A three stage pipeline is proposed (Fig. 4.1) to address the synthesis problems. An initial stage for pre-processing the raw data is followed by the algorithmic or computational stage and, lastly, by a post-processing analysis stage.

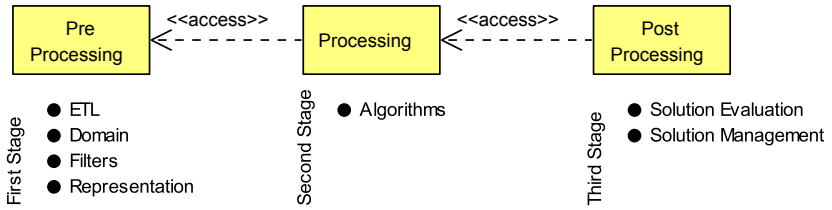


FIGURE 4.1: Work flow of the overall architecture

The architecture of these stages follows a loosely coupled design to create a light and modular system. This will allow the easy introduction of new algorithms and features for each of these stages. The specifications of each stage are discussed in the following subsections.

4.1.1 Pre-Processing

The pre-processing stage is the initial step of the pipeline. Here, data is collected from a diverse set of data interfaces to load and interpret raw data. The lack of standards for biochemical data requires flexibility in the system to adapt to multiple sources. Besides data collection and interpretation, local storage mechanisms are also required to store and catalog collected information for future sessions.

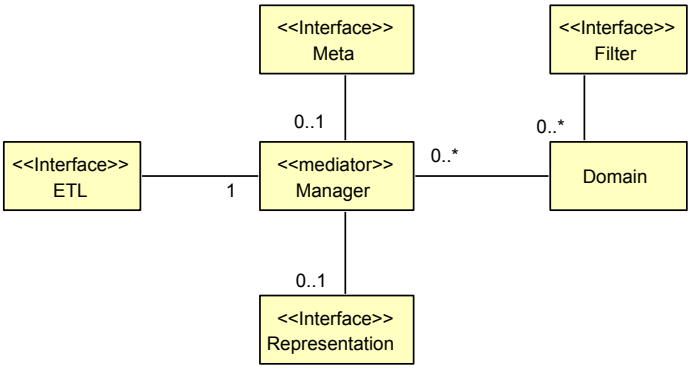


FIGURE 4.2: Pre-processing layer implementation architecture

These type of operations are commonly denoted as extraction, transformation and loading (ETL) processes, commonly applied in data warehousing, where distinct sources of non integrated data are put together into a central repository for analysis. The rest of the application can make decisions based on the loaded data.

The central manager (Fig. 4.2) serves the purpose of a pseudo data warehouse, that is responsible to link the data to the computational end point. The central

entity collects raw data from distinct sources, by recurring to a source interface. This interface performs all ETL operations to process the raw data, and stages the operational data into the central manager. Finally, the data in the staging area are packed into a metabolic domain $\mathcal{D}$ (as defined in the previous chapter). Here, several filters can be applied to remove unwanted data. These could be use to remove redundant or malformed data, but filters can also be applied to keep only data obeying a certain constraint. This domain is later transformed into a computational representation for processing (as explained in detail previously).

To qualify data, a metadata generator interface is attached to characterize each of the entities. This interface generates tags (e.g., "balanced reaction", "generic metabolite") that identify multiple properties of the entities. These tags allow a qualitative analysis of the problem domain.

Finally, a representation interface is attached to execute the required transformations of the raw data to the desired data structure (i.e., graphs, hypergraphs, p-graphs, stoichiometric matrices). The representation interface is the function that maps the data staged in the central manager to a specific data model. This provides a link between the biological information and the mathematical models. The product of this stage is a data model, which serves as an input structure for the next stage.

4.1.2 Processing

The processing layer is responsible for all the algorithmic computation. In this layer, a specified algorithm takes the respective inputs (e.g., problem domain) and computes the solution structures. The purpose of this layer is to integrate the solution generating algorithms (explained in the previous chapter) into the framework (Fig. 4.3). Because of the diverse amount of data structures and the heterogeneity of the algorithms, a common end interface must be assigned to interpret their results to a structure, over which the platform can later perform operations.

In this layer, it is necessary to support the following features:

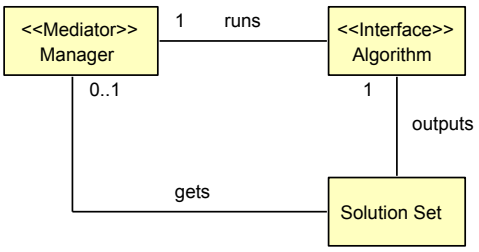


FIGURE 4.3: Processing layer implementation architecture

1. A common interface that translates the output of the algorithms to a common solution set structure (see definition in the previous chapter). This is mandatory, otherwise the solutions cannot be interpreted and post-processed.

2. A control system to limit the execution of the algorithms to a given time frame. Because most algorithms might take unpractical amounts of time to conclude, a timeout system is desired to address this problem.

In addition, the algorithm configurations are specified here. The configurations vary from algorithm to algorithm. However, there are a set of base constraints that must be defined for all of the algorithms, such as the substrate and product sets that define the synthesis problem.

4.1.3 Post-Processing

Finally, after the computation of the solutions, an evaluation strategy is required, that allows to identify the best solutions in a certain user defined sense. Depending on the goal of the user, the notion of solution quality might suffer some variations, because a single solution may contain many features (e.g., size, number of metabolites, fluxes), and it is up to the end user to define which attribute is most relevant. This demands flexibility in the analysis of the solution, so that an user must be able to define how to evaluate the solutions.

An evaluator is an entity that computes one or more features of a solution. This design allows several evaluators to be combined to generate a report (Fig. 4.4). For the sake of simplicity, the evaluators are independent of each other. Similarly with the previous stages, the central manager is responsible to establish the link between all the entities that play a significant role in this layer.

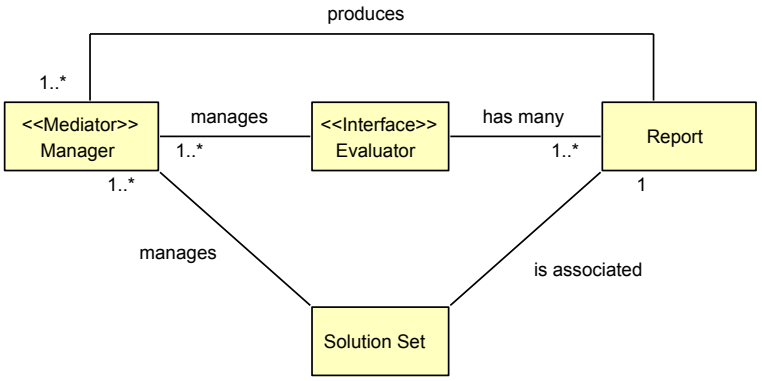


FIGURE 4.4: Post-processing layer implementation architecture

4.2 Implementation Details

With the definitions set and the architecture defined, a framework was implemented to support the solving of the synthesis problems. The Java programming language was chosen to implement the framework, making use of a number of object oriented design patterns. The usage of Java also allows an easy interaction with the components of OptFlux [52], a metabolic engineering platform, that later is required to perform model simulation.

In the previous chapter, several methods were introduced to solve the synthesis problems. While some methods use similar strategies and data models, there are multiple approaches to solve the synthesis problems (e.g., convex analysis vs graphs or set systems). This demands the framework to be flexible, operating on an abstract level.

Fortunately, these abstractions are highly explored by software engineering design methods. As an example, the inversion of control is a technique that allows to decouple the implementation from the framework, that was applied in this work.

The framework is subdivided into five main Java libraries (Fig. 4.5). This allows to increase modularity and reusability between each library. On the top, the Biosynth-Components library defines the basic data structures of the framework, an essential dependency of the entire framework. Below, three libraries define the subsystems of each stage of the pipeline described in the previous section. Finally, the core library connects all components and subsystems.

Some details of the implementation of each of these libraries are discussed in the next sections. To describe the framework implementation, the Unified Modeling

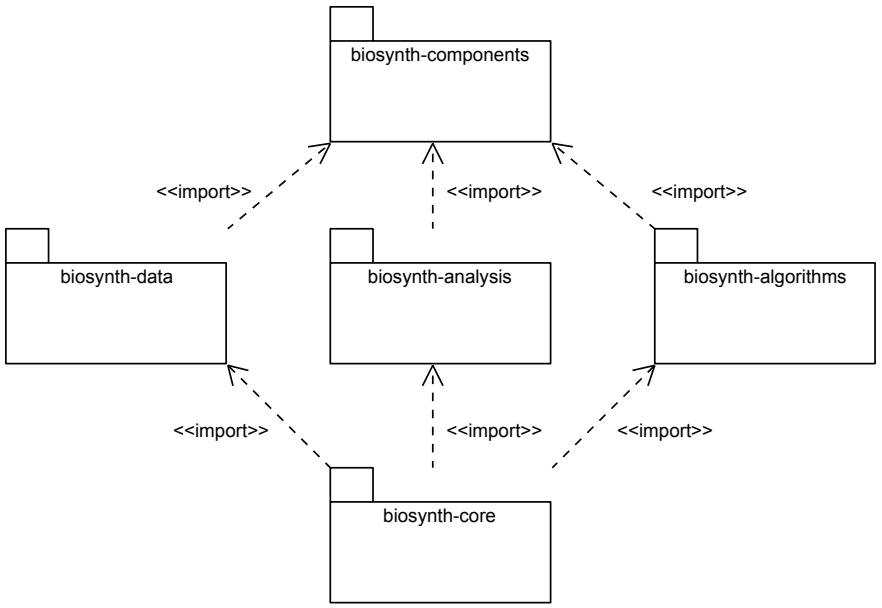


FIGURE 4.5: BioSynth framework library diagram

Language (UML) is used to describe visually the relationship of the components of the framework. The UML diagrams have only the purpose to describe the relationship, while the functions and the roles of each component is described in their corresponding section.

There are two distinct components in the framework. The first group includes the ones that are tightly coupled to assemble the framework's core. Changing any of components can have an high impact on the overall framework execution. The other group includes components that are attached to the framework. These are loosely coupled components and their change or removal has little impact in the function of the framework itself.

To distinguish these components, the solid borders represent classes that make the core of the framework, where the dashed borders stand for implementations that are injected into the framework.

4.2.1 Biosynth-Components

The components library implements all the basic data structures of the framework. These are subdivided into biological data structures and mathematical models. The biological structures are components that belong to the framework, hereby these can be defined as framework entities. A framework entity implies that their

definition is relevant to the integrity of the framework. Also, the framework itself is dependent on the implementation of these entities. While the mathematical representations are not in this group, these have the purpose to be used by solution generating algorithms.

All the biological entities are inherited from the `AbstractGenericEntity`, which defines: *a*) a unique key identifier; *b*) a descriptive name; *c*) the source of origin (e.g., a database as KEGG, MetaCyc or BiGG; a file name); *d*) a description of the entity. The unique key identifier is the only feature relevant for the rest of the framework, as each entity is identified by this property. However, the others can still be used for filtering purposes.

Since the specific implementation of the mathematical structures (i.e., p-graphs, hypergraphs) is transparent to the framework, they are independent of each other and independent of the biological entities. However, an interface is required to map the transformation of the framework entities into these models. This is done by the `IMetabolicRepresentation` interface, which requires to define the mapping function of the framework entities to the equivalent mathematical structure (e.g., `GenericReaction` to `HyperArc`).

To keep the original representations independent of the framework demands, metabolic versions of the mathematical models are defined inheriting the original representation (Fig. 4.6). This makes the original implementation unaware of the framework demands and offers a non invasive implementation. All the components that are not dependent of the framework follow the same philosophy, where the insertion of every component into the framework avoids any modification of the original components. In this scenario, the original implementations of `ProcessGraph` and `HyperGraph` are inherited by the framework versions (`MetabolicProcessGraph` and `MetabolicHyperGraph`), and only these implement the mapping interfaces that link the mathematical models to the framework metabolic models.

4.2.2 Generic Data Models

In this section, the generic data structures are described. Their definition is essential for the data interchange between the subsystems of the framework. For a start, the framework contains only three entities that define the chemical universe,

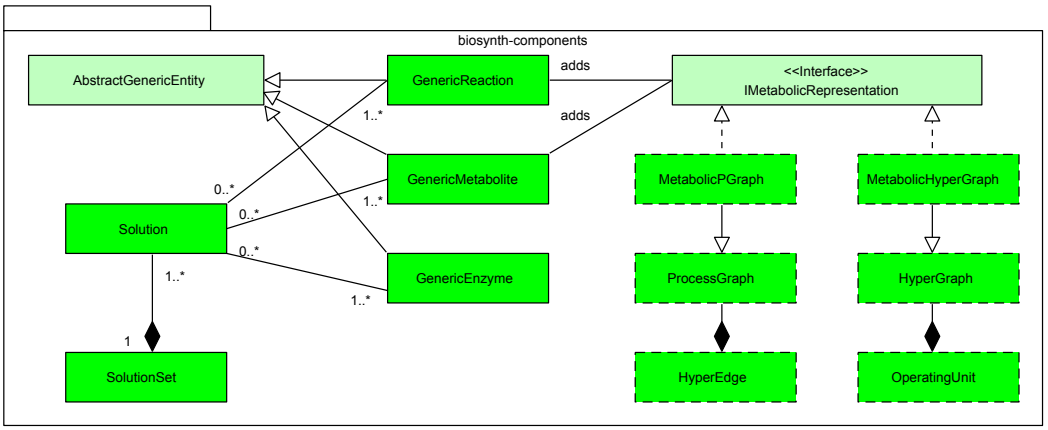


FIGURE 4.6: Data structures of the library biosynth-components

which are enough to define metabolic networks. These are the metabolites and reactions, defined in the previous chapter, and the enzymes for the microorganisms characterization problem.

There is a minimum set of features that these entities must include. The base features are inherited from the class `AbstractGenericEntity`. Each entity (i.e., metabolite, reaction, enzyme) has a distinct structure, that describes their own features.

The determination of specific features is not straight forward, specially when these vary a lot between different metabolic databases. However, the basic requirements of each individual entity type is defined in chapter 3. The complementary are obtained from the attributes of the entity models in the *KEGG Ligand* database.

The attributes are separated into two categories, that are either quantitative (e.g., reversibility of a reaction, substrates, products) or qualitative (e.g., formula of a metabolite, name). The quantitative data are relevant for the outcome of the processing algorithms, as the qualitative are only needed for the end user analysis.

The `GenericMetabolite` is a basic entity representing a metabolite. The attributes of the metabolites are the following:

- *Formula*: Describes the composition of the metabolite. This is usually a chemical formula, except for some exceptions such as the *KEGG* glycans.
- *Type*: The type of the entity (e.g., Compound, RNA, Protein, Glycan).

Like described in the previous chapter, the metabolite contains no quantitative attributes, which makes each of them optional with the exception for the unique identifier.

The `GenericReaction` is the entity that represents reactions, being the main player in the pathway extraction algorithms. Its attributes are the following:

- *Equation*: The equation that describes the reaction.
- *Reactants*: List of metabolite/stoichiometry pairs belonging to the reactants set.
- *Products*: List of metabolite/stoichiometry pairs belonging to the products set.
- *Reversibility*: Orientation of the equation.
- *Enzymes*: List of related enzymes that catalyze this reaction.

The reactants and products are followed by a stoichiometric constant (≥ 0), and the reversibility is an integer, being zero for reversible reactions; positive for irreversible reactions with orientation left to right (the orientation applies to the equation) and negative for reactions occurring from right to left. This allows to characterize the definition of a reaction from the previous chapter (Def. 3), where reactants and products corresponds to the stoichiometry pairs α, β, respectively, and the reversibility identifies whether a reaction r is $\overleftarrow{r}$, $\overrightarrow{r}$, or $\overleftrightarrow{r}$.

The final entity is the `GenericEnzime` representing enzymes, which relates reactions to genes that are associated to an organism:

- *Reactions*: List of reactions associated with the enzyme.
- *Genes*: List of gene/organism pair.

The enzyme entity is only used for evaluation purposes in the last stage of the pipeline. At this stage no attempt was made to integrate enzymes into the networks, as it would also require to modify or implement new algorithms that would use that information, a task out of the scope of this work due to temporal constraints.

4.2.3 Biosynth-Data

The data library implements all the input/output interfaces that interpret raw data to create the generic entities of the framework. This library plays an important role in the first stage of the pipeline.

An hierarchy of interfaces was designed to distinguish the features that each data source supported (Fig. 4.7). In the top of the hierarchy, there is the `ISource` interface that implements the basic functions that every data source requires.

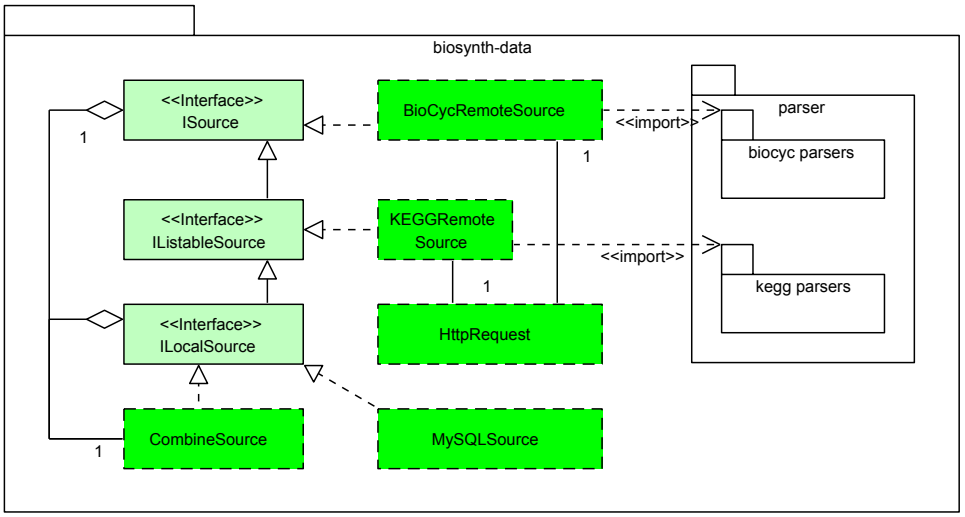


FIGURE 4.7: Components of the data library

The base interface contains the functions to fetch and transform a specific record from the raw source into a framework biological entity. The `IListableSource` interface is a more sophisticated source that implements functions to query all the entities. Finally, the `ILocalSource` interface implements what can be called a complete source, that allows fetching, querying, inserting and deleting entities.

The `KEGGRemoteSource` and `BioCycRemoteSource` are both classes implementing `ISource`, allowing to gather data from the KEGG Ligand and BioCyc databases. Both databases feature a RESTful application programming interface, that allows easy interaction with client systems, using the HTTP protocol. The KEGG interface also supports several listing operations, and therefore the `KEGGRemoteSource` can implement the `IListableSource` interface. The BioCyc database features a web query interface that interprets BioVelo [40] a specific querying language that allows users without programming expertise to obtain precise results. However, the interface shown slow performance and at this stage the `BioCycRemoteSource` does not feature querying.

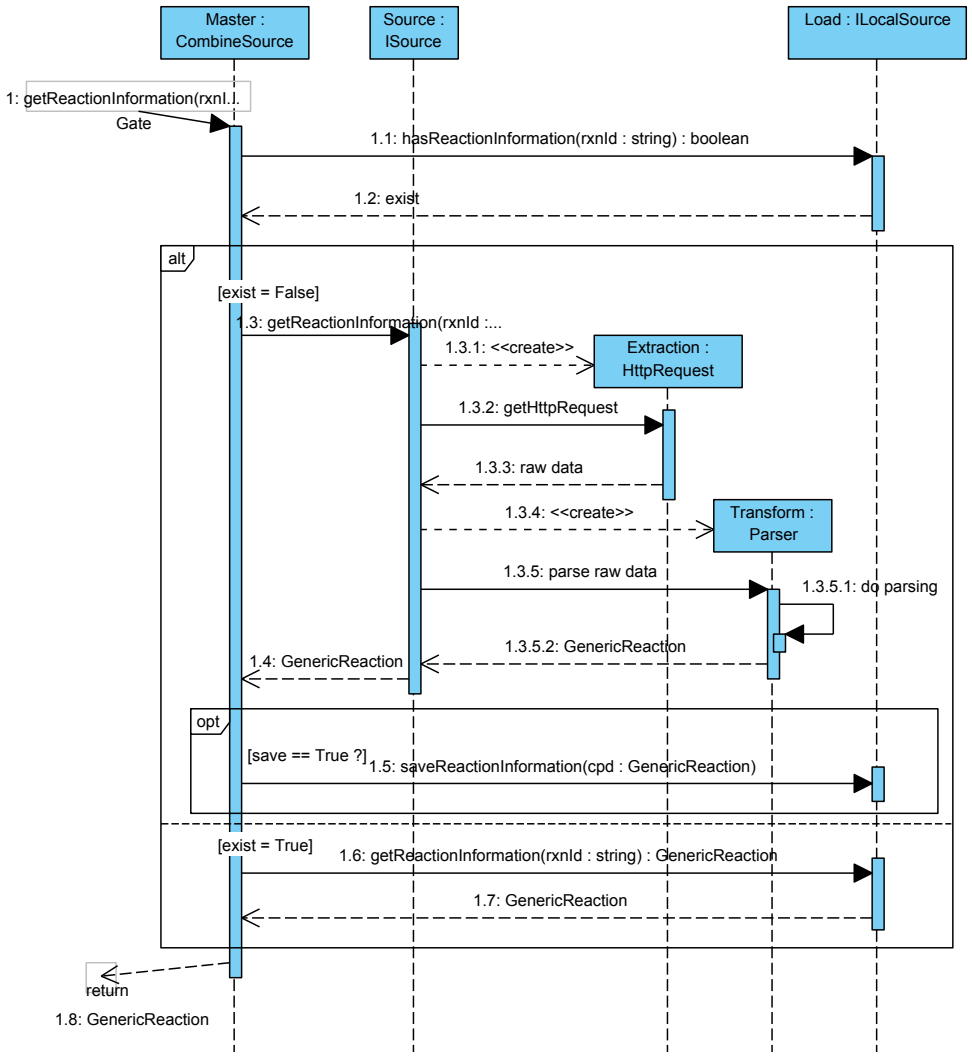


FIGURE 4.8: Sequence diagram of the `CombineSource` class to extract a single reaction

The transformation of the raw data is performed by several parsers. Each of the databases outputs raw information in their own format. This lack of a standard file format to describe biochemical reactions implies that for a specific source a set of parsers must be implemented. As an example, the `KEGGRemoteSource` requires several text file parsers, while the BioCyc offers more computer readable XML files.

The `MySQLSource` implements the `ILocalSource`. It allows the framework to connect to a MySQL database to store the retrieved information. This serve as a cache system to allow reuse of the retrieved data for future sessions.

The `CombineSource` connects an `ILocalSource` to an `ISource`, implementing the ETL process to transfer data from remote databases to an local medium (Fig.

4.8). This cheap implementation allows interaction between these two entities, but no integration is made between sources, and distinct identifiers are treated as distinct entities. This makes different web sources unable to share information, since they all have their own identifiers.

4.2.4 Biosynth-Algorithms

The algorithms library contains the implementation of the algorithms and the platform to connect the algorithms to the rest of the framework. All the algorithms described in the previous chapter are implemented in the framework. Each of them operates independently using their corresponding data structures (i.e., MSG/SSG over p-graphs; FA/Minimize/FP over hypergraphs). They are implemented following the algorithms described in the previous chapter. Since most of the algorithms apply set functions (i.e., union, intersection, difference), the Java Collections Framework is used to implement the data structure. This eventually contributes to a negative performance impact, but for a working prototype this library allows cheap and correct implementation of the required operations.

To integrate an algorithm to the rest of the framework some mandatory features must be satisfied. Many of the algorithms display a complex and messy implementation. It is avoided, whenever, possible to modify the original implementation of the solution generating algorithms. A non invasive strategy is adopted to augment the required features, such that the original implementation is left unmodified.

The `IMetabolicAlgorithm` interface implements the required functionalities that any algorithm must satisfy to operate correctly in the framework. These are: *a*) a solve handler that runs the algorithm, which is inherited by the `Runnable` interface; *b*) an extraction method to push the solutions from the algorithm, that also translates the algorithm output to a `SolutionSet` data structure. This is required to define a uniform structure for a solution, so that subsystems can interact with each other. Similarly to the data representations, the original algorithms are extended or wrapped into a new class that implements `IMetabolicAlgorithm` interface.

To reduce the execution time of the algorithms to a limited amount of time, there are several requirements to be satisfied. The `IInterruptableAlgorithm` extends the `IMetabolicAlgorithm` (Fig. 4.9), so that interruption handlers can be

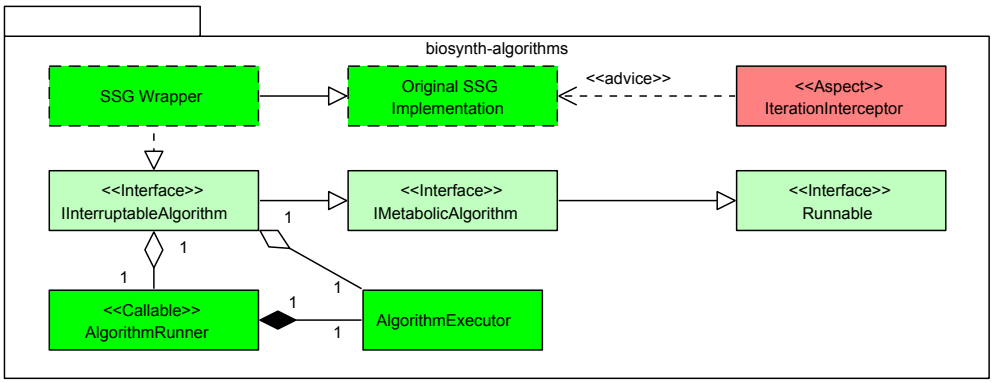


FIGURE 4.9: Components of the algorithmic library.

enforced. A *kill* method is required to make the algorithm to stop the execution. However, in a threaded environment, it is not possible to forcefully terminate threads. Therefore, the threads must gracefully terminate by themselves or kill the entire process needs to be killed (this would terminate all the threads including the program itself). This implies that, at some points of the algorithm, this must verify either to stop (i.e., if the kill handler was announced) or to continue execution. To avoid insertion of mechanisms into the algorithm kernel, a non-invasive strategy is applied based on the object-oriented and aspect-oriented paradigms (AOP).

An AOP programming approach is applied to intercept the core iteration of the algorithm. The AOP paradigm allows to design oblivious systems that minimize the degree to which programmers have to change their behavior [22]. The AOP allows augmenting the program features with a non invasive approach. Some applications of AOP are used in logging and monitoring [8] or in more advanced applications, such as alteration of the running environment to a grid system [56], as these are usually parallel features to the main logic of the programs.

An `IterationInterceptor` aspect intercepts the main iteration of the algorithm. By verifying if the *kill* handler was announced, it can skip the iteration completely or resume the computation. By skipping the computational iterations the algorithm eventually terminates. However, there is a demand that the core computation of each algorithm is wrapped into a separate method. Also, if the name of this method varies between algorithms, then individual aspects must be implemented to intercept each of these.

The `AlgorithmExecutor` uses the executor services from the Java Concurrent Library to run an `IInterruptableAlgorithm`. This creates an auxiliary `AlgorithmRunner` which implements the `Callable` interface. The executor service launches

a `Callable` object within a specified time frame. Upon depletion of the supplied time frame, a timeout exception is thrown. This makes the `AlgorithmExecutor` perform the *kill* handler to terminate the `IInterruptableAlgorithm` followed by the extraction of the solutions that were generated in the mean time.

4.2.5 Biosynth-Analysis

The analysis library operates in the third layer of the architecture, implementing several evaluators to characterize the solution structures.

An evaluator is defined by the `ISolutionEvaluator` that implements: *a*) the evaluation handler, which computes a set of features given a solution structure; *b*) auxiliary methods to get the information of the features that an evaluator computes; these auxiliary methods allow the framework to understand which and how many features an evaluator computes, since distinct evaluators have different amounts of features and some are also dependent on the parameters given to the evaluator.

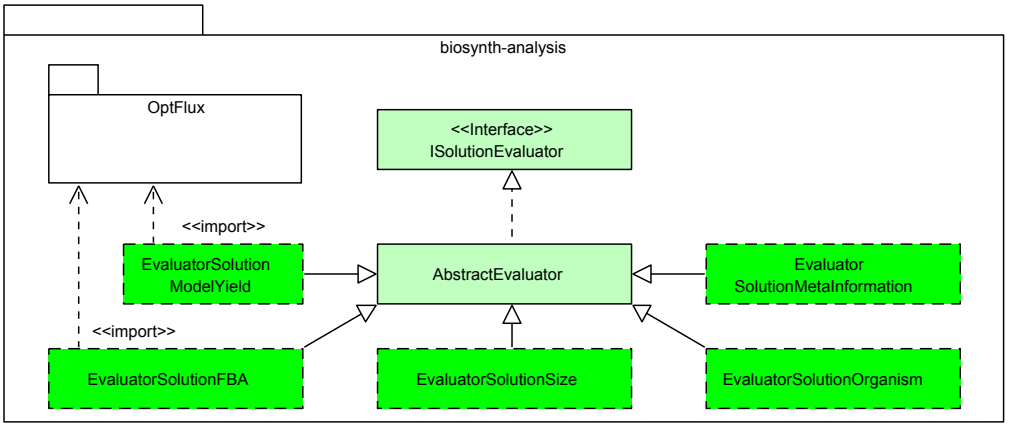


FIGURE 4.10: Components of the evaluator library

The `AbstractEvaluator` defines several elements common to all evaluators, and more specific to the framework, such as a description and a unique identifier to distinguish distinct evaluators.

The framework currently implements five evaluators to compute several characteristics of the solutions. The `EvaluatorSolutionSize` is a simple evaluator that outputs the number of reactions in the solution. The `EvaluatorSolutionMeta-Information` computes the meta features of the entire solution (e.g., for a given solution, if the reactions are 100% balanced).

Some other more complex evaluators are described in the next sections.

4.2.5.1 Solution Feasibility Analysis

As described in the previous chapter, a solution structure does not guarantee that it is feasible, meaning that there may not exist a feasible flux distribution in the metabolic network that produces the target set of products despite the fact of having all metabolite nodes satisfied. This happens because subgraph extraction algorithms do not take account the stoichiometry of the products and substrates, eventually leading to infeasible solutions. A common example are feedback loops.

To identify these solutions, Flux Balance Analysis (FBA) can be applied by maximizing the product flux (see chapter 2 for details), allowing to obtain the maximum flux value considering only the isolated pathway (solution). If a zero flux is obtained, then the solution is infeasible, meaning that no feasible flux distribution exists to reach the target product.

The `EvaluatorSolutionFBA` computes the optimal flux vector to maximize a single product in the solution structure by solving a linear programing problem. The objective function is the maximization of the product, which is supplied as parameter. This also allows to characterize the quality of the solution, the total yield of the product compound, the number of required substrates and the number of byproducts. The formulation and computation of the FBA linear program are done by the OptFlux library for metabolic simulation, which implements several interfaces to external linear programing solvers (i.e., GNU Linear Programming Kit, Coin-OR CLP, IBM ILOG CPLEX).

4.2.5.2 Metabolic Model Integration

The `EvaluatorModelIntegration` integrates the solution structures into a metabolic model and computes the theoretical yield of a target product. This evaluator requires several pre-conditions.

The model must be stored in the SBML format, which is an XML format to define metabolic models. The OptFlux library is used to read these models, which in turn uses the JSBML, a Java library for working with SBML files [14].

The metabolic model of interest must contain identifiers that match our domain entities identifiers. Otherwise, linkage of the solution to the model is not possible. However, if the identifiers of the model do not match, it is possible to supply a conversion map, that for each metabolite in the model assigns one in the domain.

The metabolites in the model are distributed in compartments. Since the solution structure does not have this representation, to be integrated into the model, a suitable compartment must be first identified. Every required substrate of the solution must be present in a single compartment, because distinct compartments do not share metabolites. A first step is to find a positive compartment match. After a positive match is found, the metabolite identifiers in the solution (if necessary) are striped to the corresponding species names in the model, providing a linkage to the newly inserted reactions.

Finally, a simulation is performed using the entire model to maximize the product of interest, by FBA. This procedure is similar to the previous evaluator, but now using the pathway interpreted in a full model. The result is the theoretical yield of the product in the supplied metabolic model.

4.2.5.3 Organism Fitting

The `EvaluatorSolutionOrganism` operates on a higher layer of the hierarchy of the chemical universe. It intends to identify which organism covers most of the reactions, given a solution. The `EvaluatorSolutionOrganism` searches for the *n-th* (supplied as parameter) best fitting organism which covers most of the reactions of a given solution.

An organism that covers more reactions can be viewed as the most suitable for synthesis of the target compound, since it is most likely that the solution size might decrease if the reactions are already contained in the organism metabolism.

4.2.6 Biosynth-Core

The core library connects the components of each subsystem and links each of the subsystems that compose the pipeline for the synthesis problem. The assembled pipeline consists of the three layers of operations that were introduced in the architecture (Fig. 4.1).

For each subsystem, a mediator class is implemented, that combines the components of each individual subsystem. The merger of these components allows to expose new features that require interaction and communication. The mediators implemented are the `DataControlCenter` and the `AnalysisControlCenter`, that correspond to the pre-processing and post-processing layers, respectively. The processing layer is discarded for now, as there is no need for additional interoperability between components, although the `AlgorithmExecutor` could be viewed as a mediator for the processing stage.

The `DataControlCenter` (DCC) is the central manager of the pre-processing stage. It connects the data source to the metabolic domain and, subsequently, to the metabolic representation. It performs all the operations of the pre-processing state, which are: *a*) to serve as a middleware for the metabolic entities (i.e., Metabolite; Reaction; Enzyme); *b*) to stage the entities obtained from the `ISource`; *c*) to define the `MetabolicDomain`; *d*) to perform the transformation of the `MetabolicDomain` to the mathematical models. The main purpose of the DCC is to organize and transform the biological entities.

The DCC also applies a gathering engine that describes a strategy to obtain data from the `ISource` interface. Two engines were implemented to fetch data from the sources: a näive method that obtains the entire dataset and a heuristic search method that gathers all connected elements from an initial target set. The näive method (`DefaultGatherer`) requires a `IListableSource`, since it is only possible to gather the entire dataset if all the identifiers are known. The `NodeSearchGatherer` applies a discovery based strategy, that is based on a breadth-first search, which attempts to find all elements that can be related to the initial set of metabolites. This is most profitable since it only fetches elements related to the synthetic problem and allows to discover the entities of basic sources (i.e., `ISources`'s that only support single fetch operations).

The `AnalysisControlCenter` connects the evaluators to the solution set. The link between these is performed by the `ScoreBoard` component. The `ScoreBoard` assembles a matrix where each row corresponds the a solution and each column to the values/properties returned by the set of evaluators applied.

As an example, if a solution set contains 16 solution structures to be evaluated by two evaluators *EVAL1* and *EVAL2*, that compute 3 and 5 features respectively,

the `ScoreBoard` will assemble a matrix of 16 rows and 8 columns, and each cell corresponds to the result of each feature.

4.2.7 User Interface

No effort was made to implement a graphical user interface, as the main purpose was to develop a framework that could be easily integrated and extended in any system.

To expose each of the features to a more user friendly environment, for each of the subsystems, a facade is implemented. A facade is a design pattern similar to the mediator, but it does not implement any new functionally having the only purpose to help expose and simplify the features of the entities that are wrapped into.

For each of three subsystems: an environment was developed to be used in a shell environment: the `DataEnv`, the `SolverEnv` and the `EvalEnv`. Each of those wraps the pre-processing, processing and post-processing layers, respectively. Each contains a function that loads its own default configuration.

This allows easy usage of the features implemented in each of the three subsystems, so that an unexperienced user can use most of the features with a minimum effort to understand the interaction between the components.

The `DataEnv` supports operations to control the data source and to manage the metabolic domains and as a parameter it requires that the user to supply a source interface. All the operations of the `DataEnv` are towards the construction and the definition of a metabolic domain, but here the interactions of each component are all wrapped into a single function (Table 4.1).

The `SolverEnv` automatically sets up the algorithm based on properties defined by an helper object `AlgorithmConfiguration` or by specifying an algorithm using the enumerator `AlgorithmType`, that in this case applies the default parameters for the selected algorithm. The auxiliary object `AlgorithmConfiguration` contains all the parameters for the implemented algorithms (statically coded) that allow an user to easily identify what parameters are available to setup. Additionally, the `SolverEnv` requires as parameter a `DataEnv` to push and transform the default metabolic domain into the representation that matches the algorithm.

TABLE 4.1: A brief list of supported operations of the `DataEnv`

Operation (Parameters)	Description
`flushStagingArea`	Clears the staging area
`gatherAll`	Collects the entire remote dataset
`gatherFrom` (String...vertices)	Collects the remote dataset using the search heuristic
`createDomain` (String)	Creates a new domain
`createSubdomain` (Integer)	Creates a new domain (subdomain)
`showMetaInfo`	Shows the Meta Statistics of the domain
`changeDomain` (String)	Changes the default domain
`filterBy...` (parameters vary by filter)	Applies a filter to the default domain
`removeDomain` (String)	Removes a domain
`showDomain` (String)	Shows the elements of the domain

The `EvalEnv` provides assistance to manage and run the evaluators and prepare the results. Most of the operations are for managing the evaluators (Table 4.2), executing evaluators and to view the results. These must be exported to an external CSV file.

TABLE 4.2: A brief list of supported operations of the `EvalEnv`

Operation (Parameters)	Description
`addSolutionSet` (SolutionSet)	Adds a new solution set
`changeSolutionSet` (String)	Changes the default solution set
`removesolutionSet` (String)	Removes a solution set
`addEvaluator...` (parameters vary by evaluator)	Adds a new evaluator to the solution set
`clearEvaluators`	Removes all evaluators assigned
`evaluate`	Execute the evaluators over the default solution set
`exportCSV` (Path)	Export the evaluation results into CSV file

The Java programing language is not designed for scripting purposes, but there are several runtime interpreters built on the Java Virtual Machine (JVM) mainly to port interpreted languages such python (Jython) and ruby (Jruby) to the JVM. These provide a shell that offers scripting opportunities for Java programs. In the appendix B, an example usage shows how to perform the entire pipeline using the Jruby interpreter.

Chapter 5

Validation of the Framework

To validate the implemented framework, a real world example was selected. The *de novo* synthesis of vanillin is a common case study applied in synthetic metabolic engineering. Synthetic pathways of vanillin production were extracted from the literature, being referred as reference pathways in the remaining. These are compared with the solution structures obtained using from the algorithms in the implemented framework. The details and discussion of the results is subdivided into three sections: pre-processing, processing and post-processing, which follow the framework's pipeline layers.

5.1 The Case Study

Natural vanilla is an important flavoring agent with a wide application range. It is used in many fields, like the food and pharmaceutical industries. Its main constituent is vanillin (4-hydroxy-3-methoxybenzaldehyde), a plant secondary metabolite found in the bean pod of the tropical Vanilla orchid. The production of vanillin exceeds 16000 tons per year, being less than 1% percent extracted from vanilla seeds, as the majority is obtained from synthetic processes. This is due to the cost of vanillin obtained from seed pods, which is highly expensive exceeding 1000$ per kilograms, while synthetic vanillin cost less than 15$ [26, 62].

A common process towards the production of synthetic vanillin is the application of metabolic engineering, approaches by designing cell factories for vanillin synthesis. Several organism hosts where successfully redesigned for the synthesis

of the vanillin compound, including *Escherichia coli* [50], *Pseudomonas* [30, 63], *Schizosaccharomyces pombe* and *Saccharomyces cerevisiae* [5, 26].

To obtain synthetic pathways for vanillin synthesis, several pre-requirements must be assured:

- The existence of genome scale metabolic model of the target host;
- A rich dataset of reactions and compounds;
- In some cases, a conversion table that maps species of the model to the dataset entities (reactions and metabolites). This is only necessary if the identifiers of the entities in the model do not match with the dataset.

In our case study, the *S. cerevisiae* (commonly known as baker's yeast) was selected as the organism host. The genome scale metabolic model used for this problem is the *iMM904* [46]. The reference pathways for synthetic production of vanillin are obtained from the synthesis of vanillin in fission yeast [5, 26], consisting in the addition of 3 reactions, being denoted as σ_{hansen} [26]. The other alternative comes from the synthesis of vanillin in *E. coli*, containing also 3 reactions and will be denoted as $\sigma_{pharkya}$ [50].

The datasets that are selected as metabolic domains for these studies are the chemical universe of KEGG Ligand and MetaCyc (which is multi-organism and manually curated) from the BioCyc database consortium.

5.2 Pre-Processing

The pre-processing components provide the tools and algorithms to aid in the construction of the domain and the data models. The initial step is to define the synthesis problem (recall def. 9 in chapter 3). The synthesis problem is defined by a triple, which contains the metabolic domain, defining all the metabolites and reactions, the set of substrates and the set of products. At this stage, the following tasks are performed:

- Construction of the metabolic dataset from KEGG and BioCyc databases;

- Mapping the reference pathways with the obtained datasets;
- Qualitative and quantitative analysis of the dataset contents;
- Definition of the synthesis problem.

5.2.1 The *iMM904* Genome Scale Model

The synthesis problem requires a set of initial substrates, that must be supplied by the user. As an alternative, assuming that the problem is organism related, which is what is intended in this case study, the initial set of substrates can be obtained from the metabolic model, by defining the set of substrates equal to the set of metabolites present in the metabolic model. This allows, in a later phase, to attach solutions into the model, also enabling the whole model simulation of the obtained results.

The *iMM904* GSM contains a total of 1392 species, that correspond to 700 metabolites in 8 compartments. The identifiers of the species diverge from the ones used in the datasets (i.e., KEGG and BioCyc). To map the identifiers, a table was constructed recurring to the BiGG database [54]. To obtain the BioCyc mapping, the corresponding KEGG cross-reference is gathered from the BioCyc databases. A total of 551 metabolites were mapped to the corresponding KEGG Ligand identifiers and 507 to BioCyc identifiers. All the unmapped metabolites are discarded from the substrate set.

5.2.2 Dataset Collection

The KEGG and BioCyc datasets are obtained using the source interfaces implemented in the framework. Although it would be preferable that the domains were joined together, since no data integration strategy was implemented at this stage, each of the domains will remain separate (i.e., there is no information exchange between the two datasets).

The KEGG domain consists of the entire KEGG Ligand database, while the BioCyc one contains only the MetaCyc entities, since this is the only database in the

BioCyc consortium that contains detailed information of the entities (e.g., cross-references, multi-organism). The KEGG and BioCyc domains will be denoted as $\mathcal{D}_{KEGG}$ and $\mathcal{D}_{BioCyc}$, respectively.

For comparison purposes, the two implemented strategies to obtain datasets from web databases were used. The first method is by querying all the records in a database, while the other is to use a search heuristic (breadth-first search) to query only elements related to an initial record (see section 4.2.6 for details). The domains obtained from these strategies are denoted as $\mathcal{D}^{A}_{KEGG/BioCyc}$ and $\mathcal{D}^{B}_{KEGG/BioCyc}$, for each of these methods, respectively. In the later strategy, the initial record selected to search neighbor elements will be the product compound of the case study (C00755 for KEGG and VANILLIN for BioCyc).

As expected, the number of entities in the $\mathcal{D}^{B}$ domains is far inferior to the ones in the $\mathcal{D}^{A}$ domains (Tab. 5.1). The number of reactions remains quite the same for both datasets, however there is a huge difference between the number of metabolites. This is due to the fact that a huge amount (over > 20000) metabolites in $\mathcal{D}_{KEGG}$ have no reaction associated (e.g., C00098 - Oligopeptide, C00034 - Manganese, G01194 - Dextran). Obtaining the entire set of metabolites from KEGG would be inefficient as the majority of the data is unusable.

TABLE 5.1: Datasets sizes for both $\mathcal{D}_{KEGG}$ and $\mathcal{D}_{BioCyc}$ metabolic domains using the two mechanism implemented for data collection.

	$\mathcal{D}^{\mathcal{A}}{}_{KEGG}(\mathcal{D}^{B}_{KEGG})$	$\mathcal{D}^{A}_{BioCyc}(\mathcal{D}^{B}_{BioCyc})$
Metabolites	28069 (7504)	10965 (9634)
Reactions	9397 (9361)	11570 (11464)
Enzymes	6043 (4390)	0 (0)

The major advantage of using the search heuristic is the reduced total time required to obtain the entire dataset with the useful information (related to the problem to solve) from a remote data source. To get the entire dataset from KEGG, it took at total of 860 minutes to obtain all the 43509 records, with an average of 1.09, 1.3 and 1.41 seconds for metabolites, reactions and enzymes, respectively. The real time to gather all of the entities was around 45 minutes running with 20 threads. This huge gain is due to the I/O stalls related to server response and transfer time.

Using the search heuristic greatly reduced the amount of total time to 362 minutes, that corresponds to 20 minutes of real time equally using 20 threads. The average

time for each record remain quite the same, with the exception for the reactions which had a noticeable reduction to 0.81 seconds per record (which might been related to the Internet connection and the sever response time). Nevertheless, the majority of the gain was due to the reduced amount of compounds.

Obtaining an entire dataset from a web database shown to be time expensive, and not all of the data may be relevant for the problem to solve. Using a search heuristic shown worthwhile, saving time and resources.

5.2.3 The Reference Pathways

The reference pathways σ_{hansen} and $\sigma_{pharkya}$ are manually mapped to KEGG and BioCyc identifiers (Tab. 5.2). The $\sigma_{pharkya}$ is successfully mapped in both datasets, while the σ_{hansen} one contains missing reactions in the KEGG dataset. Therefore:

$$\sigma_{pharkya}, \sigma_{hansen} \subseteq \mathcal{D}_{BioCyc}$$

and

$$\sigma_{pharkya} \nsubseteq \mathcal{D}_{KEGG}, \sigma_{hansen} \subseteq \mathcal{D}_{KEGG}$$

TABLE 5.2: Mapping of the reactions of the reference pathways with the identifiers of $\mathcal{D}_{KEGG}$ and $\mathcal{D}_{BioCyc}$. The (-) symbol refers to situations where no mapping was identified.

ID	E.C.	$\mathcal{D}_{KEGG}$	$\mathcal{D}_{BioCyc}$
3DSD	4.2.1.118	R01627	DHSHIKIMATE-DEHYDRO-RXN
ACAR	1.2.3.9	-	RXN-8091
hsOMT	2.1.1.-	-	RXN-8873
	1.2.1.46	R00604	FORMALDEHYDE-DEHYDROGENASE-RXN
	1.14.13.82	R05274	RXN-10891
	1.2.1.67	R05699	1.2.1.67-RXN

The reactions of reference pathway $\sigma_{pharkya}$ and σ_{hansen} are given by the following E.C. numbers {1.2.1.67, 1.14.13.82, 4.2.1.118} and {1.2.3.9, 2.1.1.-, 4.2.1.118}, respectively.

5.2.4 Data Set Analysis

A qualitative analysis is applied for both datasets. This allows to identify some properties of the reaction sets. Both KEGG and BioCyc show a high amount of balanced reactions. However, there is a slight difference between the amount of missing and generic reactions. A reaction is generic when its formula is not complete (e.g, KEGG - C00071, Aldehyde - CHOR), and consequently these reactions cannot be balanced. The reactions characterized as missing are those that contain metabolites without a formula. This is highly common in BioCyc, because the metabolites can be also proteins and RNA compounds (Fig. 2.1(b)). The reactions that are characterized as infeasible are the reactions that contain the same metabolite as reactant and product.

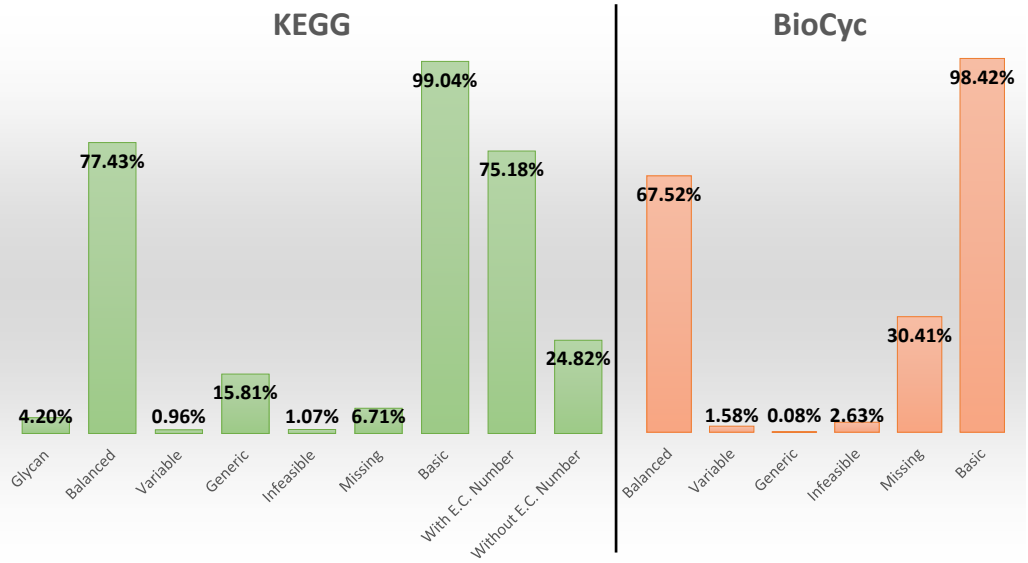


FIGURE 5.1: Properties of the reactions for both KEGG and BioCyc datasets

No E.C. number analysis is given for $\mathcal{D}_{BioCyc}$, since BioCyc reactions are not characterized directly with E.C. numbers. Since balanced reactions are the majority, the unbalanced ones are discarded from the metabolic domain. This guarantees that the stoichiometry of all generated solutions is balanced.

5.3 Processing

The implemented algorithms (SSG and FP) were executed to generate solution structures for both datasets. The domains applied to generate solutions are those

obtained from the heuristic search method ($\mathcal{D}^B$) and filtered to only keep balanced reactions. Because of the complexity of the algorithms, it is not possible to obtain the complete set of solutions for the whole domains. Therefore, the partitioning strategy is used for this problem, to generate multiple levels of subdomains for both $\mathcal{D}_{KEGG}$ and $\mathcal{D}_{BioCyc}$.

The following tasks are performed at this stage:

- Characterize and partition the domains into subdomains;
- Apply the pathway extraction algorithms to compute the solution sets on each of the subdomains;
- Compare the proposed Minimize heuristic to the original implementation.

5.3.1 Domain Analysis

The partitioning strategy generates subdomains from an initial set of target substrates. For this problem, the selected elements for the initial set contain only the target compound. This allows to obtain nearby elements of the target by distance radius, such that the solutions of size i are all present in the subdomain of radius i.

Both the domains $\mathcal{D}_{KEGG}$ and $\mathcal{D}_{BioCyc}$ are run up to twenty subdomains. These are defined as $\mathcal{D}^i$, such that $1 \leq i \leq 20, i \in \mathbb{N}^+$, where i corresponds to the i-th radius of the domain.

To test if each domain is feasible, i.e., if there is at least one feasible solution structure, a single run of the *Minimize* algorithm is executed. This allows to extract a single minimal solution. If this solution exists, the subdomain is feasible. A better approach would be to compute the exact maximal structure (which is the union of all solutions) of the subdomain, as this could later be used as input for the solution generating algorithms. However, but since in the previous chapter it was shown that obtaining the exact maximal structure is not trivial, and in many cases only approximations are obtained, the computation of maximal solution is left only for measurement purposes to analyze the performance of the algorithms on real datasets.

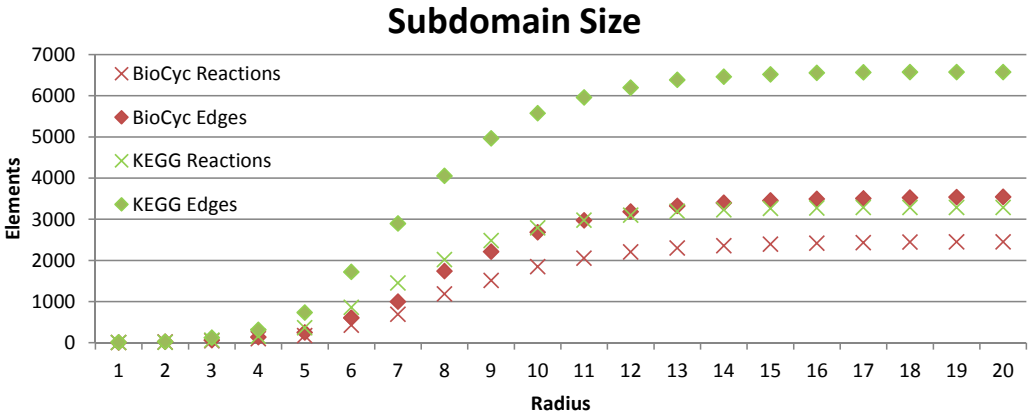


FIGURE 5.2: Number of elements of the domain (reactions) and the hypergraph (hyperedges) model for each radius size.

The major difference between KEGG and BioCyc is the reversibility, which impacts the growth speed of each subdomain and the number of edges in the graph representations (directed hypergraph and process graph) (Fig. 5.2). In $\mathcal{D}_{KEGG}$, all reactions are reversible. Therefore, to build a graph model representation, for each reaction r, it is necessary to add two versions of a hyperarc, such that $\overrightarrow{r}$ to corresponds to the "normal" direction and to $\overleftarrow{r}$ the opposite direction. This implies that a graph model has twice the amount of edges (Fig. 5.2) for each reaction included. Contrary to $\mathcal{D}_{KEGG}$, $\mathcal{D}_{BioCyc}$ contains irreversible reactions, which greatly reduces the size of the resulting graph model.

Both maximal structure computation algorithms are run against each subdomain (Figure 5.2). This allows to compare the behavior of both algorithms. Later, it is possible also to compare the exact maximal structure, that is assembled from the solution space of the minimal solutions.

Recall from the third chapter that the MSG and FA algorithms compute an upper and lower approximation, respectively. There is a huge gap between the two approximations, meaning that there is a high error rate from either of the algorithms. On the other hand, in $\mathcal{D}_{Biocyc}$, there is a noticeable difference. Nevertheless, the results show that the maximal structure obtained from MSG and FA are inconclusive.

The maximal structure algorithms were unable to prune the domain properly. Therefore, the solution generating algorithms are run against the original subdomains of D^i_{KEGG} and D^i_{Biocyc}, instead of over the pruned domain (maximal

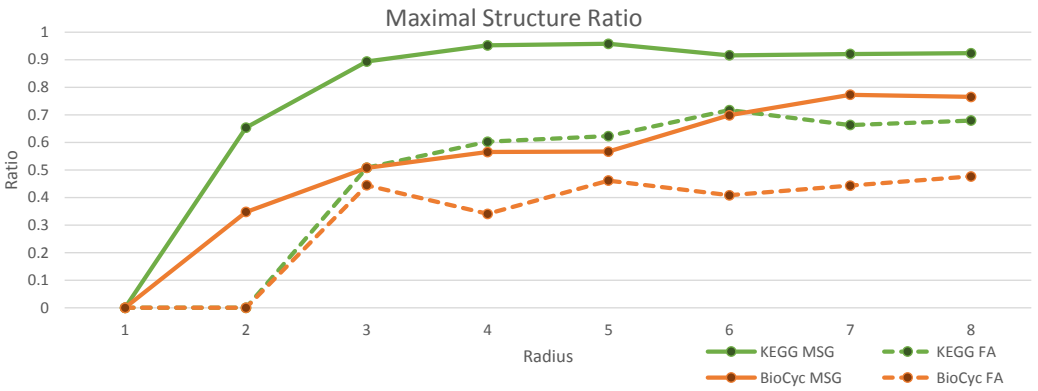


FIGURE 5.3: The percentage of the edges (hyperarcs or operating units) accepted by the MSG and FA algorithms in the maximal structure of the first eight subdomains; ratio = $\frac{\text{number of edges in the maximal structure}}{\text{number of edges in the domain}}$

structure). Later, the solution structures obtained can be assembled to compare with these maximal structures of both MSG and FA methods.

5.3.2 Pathway Extraction

For each subdomain generated, three configurations of the algorithms are tested. This allows to compare the solution set obtained from each algorithm and the performances of these. Each of the algorithms is limited to one hour of execution to generate solutions. For those that were not able to obtain solutions after this time frame, all solutions are discarded, so that no incomplete solution sets are included in this analysis. To generate solution structures, the following algorithms and configurations are tested:

- FP: Minimal solution structures, using the binary search heuristic proposed in this work;
- SSG: Minimal and extended solution structures. This uses the newly implemented strategy to compute partial power sets;
- SSG_{Δ}: Every solution structures. The original implementation of the SSG algorithm;

The results are given in Figure 5.4, where only the FP and SSG results are shown, since the SSG_{Δ} was only able to compute solutions for the first three subdomains.

There is a noticeable difference between the performance of the algorithms in both datasets. The FP algorithm outperformed the improved SSG algorithm in the KEGG subdomains. A major problem of the SSG was dealing with a fully reversible domain. Indeed, the reaction reversibility in a network greatly increases its connectivity. In the subdomains $\mathcal{D}^{i}_{KEGG}$, all reactions are reversible, and this generates many closed loops that promotes the computation of infeasible combinations, while also increasing the number of connections for each metabolite. Therefore, the size of $\Delta(m)$ (defined as the number of reactions available to satisfy an arbitrary metabolite m in the network) increases, also increasing the size of the power set (i.e., the number of combinations of producers of m).

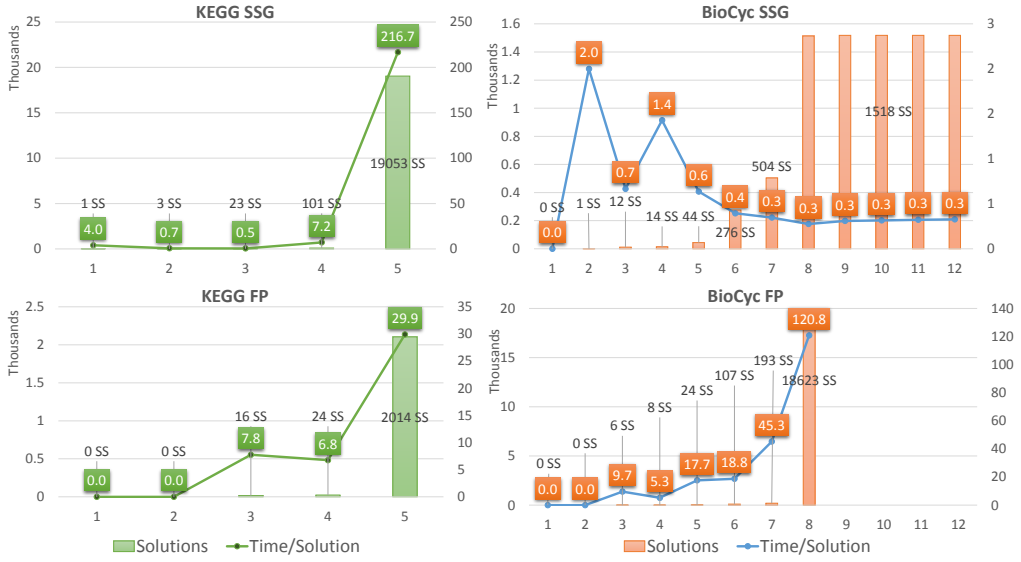


FIGURE 5.4: The solution count and the average time per solutions for the SSG and FP algorithms in both domains; Bar plot - the number of unique solution structures (in thousands) (left axis). Line plot - time per each solution structure in microseconds (right axis). SS - Solution Structure.

In general, the FP algorithm is more reliable than the SSG, being usually able to generate more efficiently the solutions, while the SSG guarantees the whole solution space of a domain (i.e., $\mathcal{S}_{min}$ since the new implementation only obtains minimal solutions). However, in the BioCyc dataset, the FP was unable to compute solutions beyond radius 8. The incapacity of the FA to detect the feedback loops, forces the Minimize algorithm to include an extra path that connects to the node in order to proceed.

Consider the example solutions structure $\sigma_{example}$ (Fig. 5.5), which is one of the many solutions that were generated by FP, for the eighth radius of the BioCyc

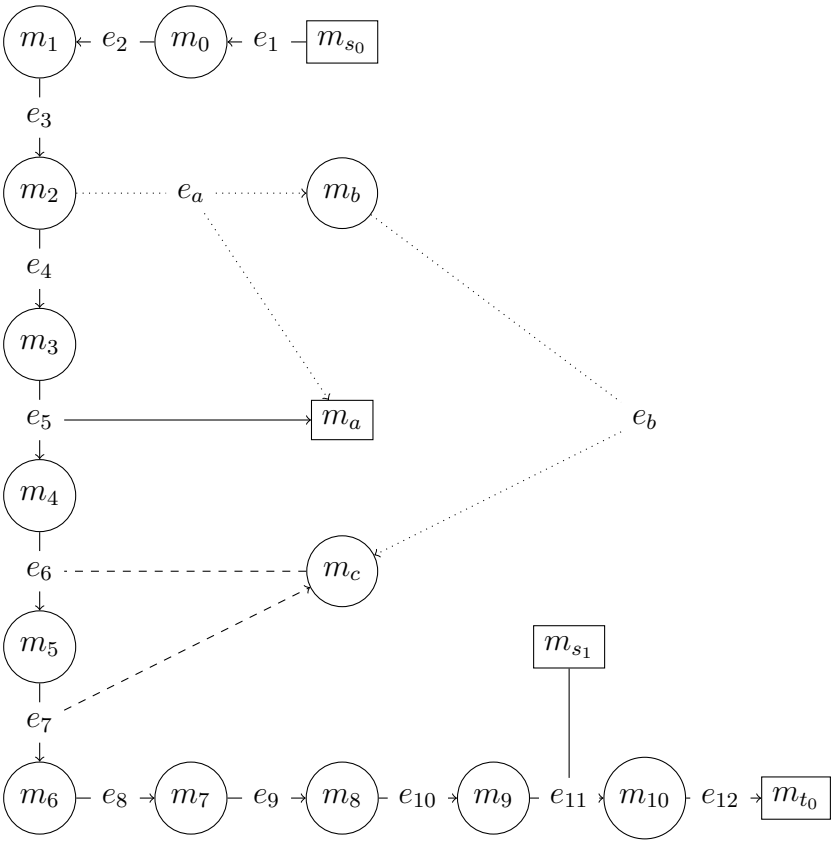


FIGURE 5.5: Solution structure $\sigma_{example}$ generated by the FP algorithm ($m_{s_0}, m_{s_1} \in S$ and $m_{t_0} \in T$). Rectangles are substrates and products (or byproducts). Circles represent intermediate metabolites. Each m_i is a hyperarc of the network (reactions). Solid line - main connection from S to T. Dashed line - feedback loop dependency. Dotted line - alternative connection from S' to $m_c, S' \subseteq S$.

dataset (a total of 18623), although the names were simplified to clarify the analysis (the original names can be found in the appendix A.1. Let $\sigma'_{example}$ be another solution structure similar to $\sigma_{example}$, but without the hyperarcs $\{e_a, e_b\}$ (the dotted hyperarcs in fig. 5.5). Then $\sigma'_{example}$ is also a solution structure and $V(\sigma'_{example}) \subsetneq V(\sigma_{example})$ and $E(\sigma'_{example}) \subsetneq E(\sigma_{example})$, therefore by def. 15, $\sigma_{example}$ is not a minimal solution. This compromises the role of the Minimize algorithm in generating minimal solutions, since it is not able to guarantee that every solution structure is minimal.

The FA algorithm in order to accept an edge e_i, this requires e_i be fully satisfied by the previous accepted edge, plus the starting substrates. In this particular example, the edge e_6 is dependent of vertexes m_4 and m_c, while m_4 is provided by the e_5 (which is previous to e_6), there is no edge providing the m_c vertex. This forces the algorithm to find an alternative route to obtain m_c (in this particular

case, the solution was to add e_a and e_b, that are not needed in the minimal solution).

The abnormal amount of solutions are due to the existence of redundant solutions. The feedback loop (e_6, e_7) requires a alternative path from S (initial substrates) to m_c. This can be viewed as a subproblem to satisfy $\sigma_{example}$. Every combination of this alternative path is a new solution. Therefore, for every instance of feedback loops, combinations of alternative solutions are generated, that eventually lead to the massive amount of solutions.

The alternative paths are redundant, so for any path P that satisfies m_c from S in $\sigma_{example}$, P can be discarded, as it is unnecessary. Later, the evaluator that computes the optimal flux distribution can be used to compute the flux distribution of these solutions, using a FBA simulation. This allows to identify the unique flux vectors of the set of solutions.

The SSG_Δ, which is the original implementation of SSG algorithm, only manages to compute only small subdomains, up to 3 and 4 for KEGG and BioCyc, respectively. The domain $\mathcal{D}^3_{KEGG}$ and $\mathcal{D}^2_{BioCyc}$ generated 211 and 141 solution structures for a small maximal structure of 10 and 13 reactions. Beyond these domain radius, the amount of solutions surpasses 4 gigabyte of memory (limit allowed for the Java virtual machine). Combining every solution is definitely not a viable choice as the upper bound amount of these solutions exponentially grows with the number of minimal solutions.

5.3.3 The Binary Search Heuristic

The new proposed heuristic for the Minimize algorithm is expected to have a noticeable impact when applied to large datasets. The original implementation of Minimize is compared against the newly proposed heuristic. It is expected that the new implementation outperforms the original version if the domain size increases.

The properties to compare are following: the execution time to find a single minimal solution and the ω ratio (see chapter 3, def. 22) of the obtained solution to the size of the number of edges of the hypergraph.

To compare the performance of both heuristics, the previous $\mathcal{D}^i_{KEGG}, i \geq 3$ subdomains are tested to reach a single minimal solution. A single run of Minimize

obtains the first solution of the domain. This is non deterministic, since the output varies depending on the order of the edges in the set.

In most scenarios, σ_0 contains only 3 reactions (Tab. 5.3) and ω is very small (i.e., < 0.1). Also, since $\mathcal{D}^i_{KEGG} \subseteq \mathcal{D}^{i+1}_{KEGG}$ and $\mathcal{S}^i_{KEGG} \subseteq \mathcal{S}^{i+1}_{KEGG}$, as the radius i increases, ω gets even smaller.

TABLE 5.3: The size of the first solution structure (σ_0) obtained from Minimize

i	$\lvert\sigma_0\rvert$	ω	Edges
3	3	0.023	128
4	3	0.009	336
5	8	0.001	812
6	3	0.003	2014
7	3	0.001	3532
8	8	0.001	5200
9	3	< 0.001	6462
10	3	< 0.001	7366

The σ_0 in the $\mathcal{D}^3_{KEGG}$ consists of the edges $\{e_{63}, e_{102}, e_{121}\}$, from a total of 128 edges. This implies that 125 edges are removed to obtain σ_0, in the sequential version of Minimize. A total of 128 FA evaluations are performed to test each of the edges, while the new heuristic manages to find $\{e_{63}, e_{102}, e_{121}\}$ in only 31 FA evaluations.

As the domain size increases, the sequential version takes a longer time to compute a single minimal solution (Fig. 5.6), while the binary search removes hyperedges by batch sets. Without this heuristic, the FP is rendered unable to generate solutions in large datasets.

5.4 Post processing

The post-processing is the last stage of the pipeline. Here, it is intended to provide aid to the user in identifying interesting solution structures obtained in the previous stages. This is achieved through the framework's evaluator system.

The following evaluation tasks are performed:

- Solution feasibility analysis and potential yield;

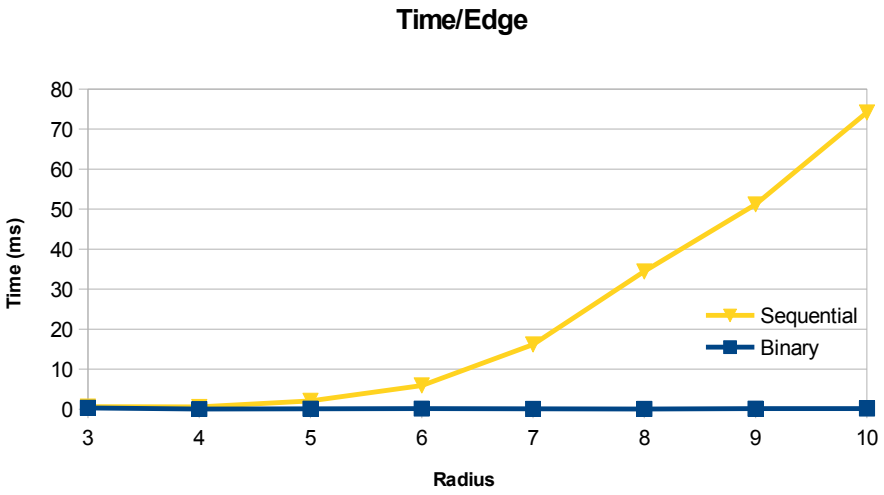


FIGURE 5.6: Computational time per each σ_i in $\mathcal{D}^i$ (i is the radius)

- Integration of the solutions with the *iMM904* GSM;
- Identification of potential organism chassis for the generated solutions.

Each of these tasks is described in the following subsections.

5.4.1 Solution Feasibility Analysis

In the third chapter, it was shown that in some scenarios the generated solution structures are infeasible, meaning that there is no possible flux distribution containing a positive flux value for all target compounds, even if all demands are satisfied. A common scenario are feedback loops that cut off the flux from a certain part of the solution.

The framework's `EvaluatorSolutionFBA` computes the optimal flux vector for a solution. This allows to identify the number of feasible solutions for each solution set. By computing the optimal flux distribution, it is also possible to identify unnecessary reactions in the solution. As an example, the solution $\sigma_{example}$ (Fig. 5.5) would generate zero flux for both hyperarcs e_a and e_b, because distributing flux through these arcs would only reduce the flux value of e_4 and increase output of m_c. Since the objective is to maximize m_{t_0}, decreasing e_4 would be non optimal.

The SSG, in general, was able to obtain more solutions than the FP, since the FA algorithm excludes some reactions due to false negatives. That eventually has consequences in the loss of solutions that contain these reactions.

All the 18623 solutions obtained in D^8_{BioCyc} by the FP algorithm are feasible. However, they do not have unique optimal flux distributions, as $\sigma_{example}$ can have multiple variations that generate the same optimal flux distribution. By filtering the solutions to only those who have unique flux distributions, the number of solutions decreases from 18623 to 863 (Fig. 5.7). On the other hand, the SSG main drawback is the computation of infeasible solution. By combining reactions, it generates solutions with cycles, that also promote the computation of infeasible solutions. Thus, unlike FP, the SSG contains a high ratio of infeasible solutions.

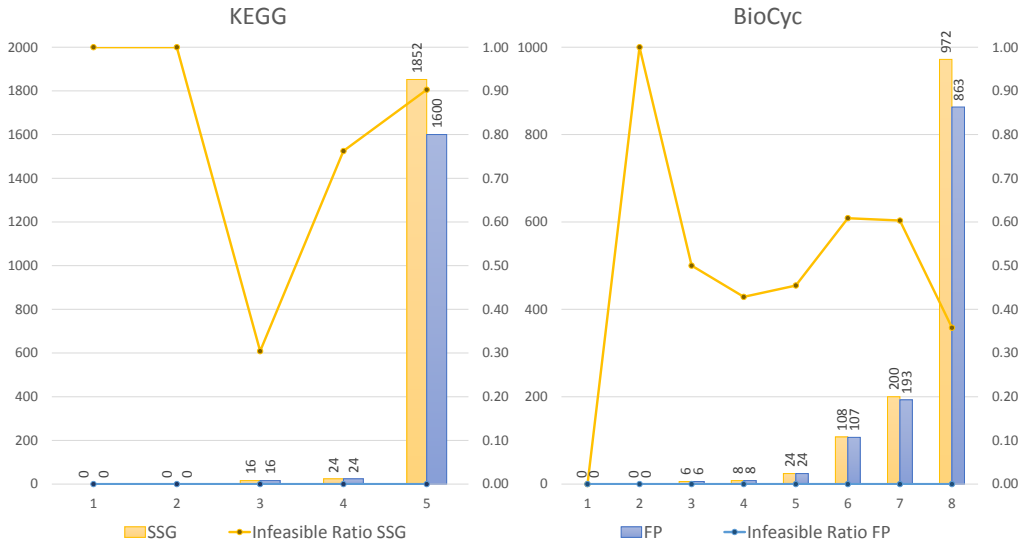


FIGURE 5.7: The number of unique feasible flux distributions and the infeasible ratio of the previous solution sets. Left axis shows the number of flux distributions. Right axis shows the ratio of unfeasible solution structures.

As a conclusion, this evaluation is strictly necessary for both algorithms, as there were many solutions generated by the SSG method that were identified as infeasible, while for the FP algorithm the redundancy is a major problem.

5.4.2 Model Integration

The `EvaluatorSolutionModelYield` allows to integrate each individual solution into a metabolic model and evaluate the optimal theoretical yield of a desired compound. The solution sets obtained from domains $\mathcal{D}^5_{KEGG}$ and $\mathcal{D}^8_{BioCyc}$, filtered

by unique feasible flux vectors, are integrated into the metabolic model of *S. cerevisiae* (iMM904), with the vanillin selected as the target compound.

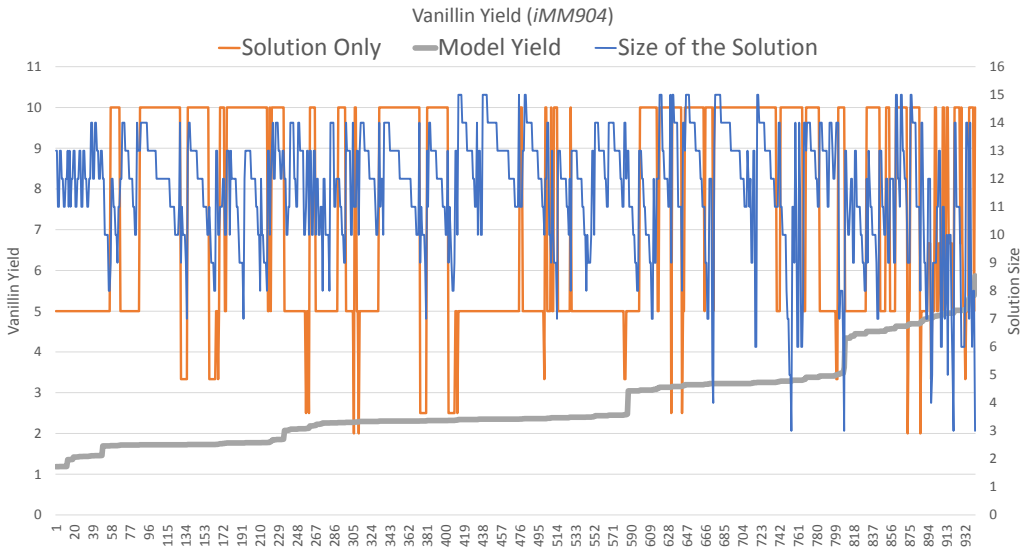


FIGURE 5.8: The unique feasible solutions of the D^8_{BioCyc} subdomain generated by the *SSG* algorithm. The solutions are sorted by increasing yield of vanillin.

The results obtained show that there is no correlation between the maximum flux value obtained from the previous evaluator (simulating the solutions isolated from a model) and the yield of the solution simulated when inserted into the model (Fig. 5.8). Also, the size does not matter much as for every range of vanillin yield there are solutions of a variety of sizes.

The best solution of the $\mathcal{D}^8_{BioCyc}$ obtains 5.86 units of vanillin per 10 units of glucose. This solution is actually equal to the reference pathway $\sigma_{pharkya}$, while the σ_{hansen} obtains only a yield of 3.61. However, the optimal solution of $\mathcal{D}^5_{KEGG}$ does not match any of the reference pathways. This solution gives a higher yield of 6.98 and contains only 3 reactions (R05273, R10136, R01627).

5.4.3 Organism Matching

Finally, the `EvaluatorSolutionOrganism` is the evaluator implemented in the framework, that allows to identify the k organisms that cover the most part of each generated pathway. This allows to predict which alternative host is most fitted to accept a certain solution. The test was run with the evaluator set to rank the 100 best organisms most fit for each solutions. Only the solution set from the

subdomain $\mathcal{D}^5_{KEGG}$ is evaluated, as the KEGG dataset is the only dataset where the solutions contains E.C. number mapping.

The total coverage is summed up for each organism and divided by the amount of solutions. This allows to identify the organism that is present in most of the pathways with highest coverage ratio.

TABLE 5.4: Top 10 organisms with highest coverage ratio.

Organism Id	Average Cover Ratio	Organism Name
NONE	0.6153	No Organism Found
RHA	0.1662	*Rhodococcus jostii RHA1*
VPE	0.1577	*Variovorax paradoxus EPS*
REU	0.1557	*Ralstonia eutropha JMP134*
VEI	0.1552	*Verminephrobacter eiseniae EF01-2*
VAP	0.1510	*Variovorax paradoxus S110*
BUR	0.1489	*Burkholderia sp. 383*
SJP	0.1472	*Sphingobium japonicum UT26S*
POL	0.1414	*Polaromonas sp. JS666*
PDX	0.1378	*Pseudonocardia dioxanivorans CB1190*

The most present organism in all generated pathways is the "No Organism Found" (Tab. 5.4), which shows the high number of reactions without any organisms attached. With a average coverage ratio of 61%, this shows that most reactions are uncharacterized, which makes the information inconclusive about the organism selection. This happens because root reactions (i.e., reactions that produce vanillin, e.g., R05273 and R05699) of the solutions for this case study do not have any organism assigned in the KEGG database, meaning that no solution can be fully characterized by a single organism.

As alternative would be recurring to enzyme to organism databases, such as the BRENDA[1] or ENZYME[2]. This could allow to obtain a better taxonomy of organisms. However, this would require a more sophisticated strategy to integrate multiple distinct data sources, out of the scope of this work at this stage.

[1] http://www.brenda-enzymes.info/
[2] http://enzyme.expasy.org/

Chapter 6

Conclusions

The main goal of this thesis was to implement optimization methods to assemble synthetic pathways, that are capable to augment existing metabolic models, aiming at the production of new compound of interest, within the realm of Metabolic Engineering. The assembling of synthetic pathways can be interpreted as a subnetwork extraction problem, where given a reactions network, the objective is to extract a subset satisfying a set of criteria. For this purpose, topological analysis algorithms were analysed and implemented, applying searching or combinatorial techniques.

The combinatorial explosion of the number of solutions is the main bottleneck of the computation of large networks. The selected case study allowed to conclude that most problems are caused by infeasibility and redundancy, as these solutions contribute for inefficacy in the computation of larger domains. Indeed, with the increase in the size of the domain, the new elements naturally contribute to new solutions, but also to the increase of the unwanted solutions.

To complement these algorithms, several tools for data processing were developed, that allow to construct datasets from data available in web-based metabolic databases, such as the KEGG and BioCyc. These tools are crucial to solve these problems, as most bioinformatics algorithms are highly dependent on the massive amount of biological data available in the world wide web. Also, the developed evaluation tools allow to support the user in the analysis of the obtained solutions, since it is common that a large amount of solutions are generated in large networks.

All the implemented algorithms and tools are assembled into a single framework implemented in the Java programming language. A real case of synthetic biology was selected to test and validate the framework capabilities.

6.1 Contributions

In this work, we have shown that the previous existing algorithms (i.e., Solution Structure Generation and FindPath) are not adequate to handle large networks, such as the chemical universe of KEGG and BioCyc. Indeed, it is inconceivable to extract solutions over these networks with those algorithms. The modifications applied to both allowed to increase the size limitation of these algorithms. This allowed to extract minimal solutions from the networks, although extracting every minimal solution from large networks is still impossible, due to time constraints.

The strategy applied in the case study shown a viable approach to obtain portions of solutions in an organized approach (i.e., by increasing size). This is applicable in general on any network size.

And, finally, this work contributed with the implementation of a complete framework, to support all the processes related the synthetic metabolic problem, as well as its pre-processing and post-processing. A major advantage of this framework is its simplicity and flexibility, allowing the easy introduction of new components. By adding a new component, this immediately is connected to all other components previously available, be it either a new data source, a new algorithm or a new evaluation method. This is most suitable for solving such problems, as there is no standard database, no perfect algorithm, nor the best evaluation strategy.

6.2 Limitations

It was shown that computing solutions for these problems is very difficult, as there are many scenarios that trick the algorithms to induce the computation of infeasible or redundant solutions. These errors were exposed in both datasets, but it is difficult to show that other similar cases do not exist.

The non-invasive philosophy of the framework allows inclusion of new components with none or at least minimal changes to the original components. This indeed

greatly reduces the maintenance cost of these. However, there is a significant performance cost specially in the computation step. Most of the algorithms were implemented as a prototype for a working example, meaning their performance was never a concern.

The methods applied in this work, for synthesis problems, do not take account into several other properties that might be relevant. In a real situation it is not that simple to optimize microorganisms to produce non native compounds. Several other relevant properties such as compound toxicity is also of high importance.

6.3 Future Work

There are many topics of this thesis that can still be further enhanced. From the algorithms perspective, it was shown that the computation of the maximal structure was not an easy task. This is essential to prune large networks in order to understand several properties, such as, the elements that contribute to solutions, the feasibility of the problem and most important to reduce the size of network by removing reactions that to not contribute to solutions. Indeed, these reactions only increase the complexity of the algorithms.

A strategy to identify redundant and infeasible network branches or patterns would highly increase the capability of the computation of solutions in bigger networks, as these are the main bottlenecks of the algorithms. Another approach is to apply distributed computing strategies on the solution generating algorithms. This would allow to further increase the computational capability in solving larger and more complex domains.

A key missing element in the pre-processing phase are the data integration techniques, since there is no universal data set for biological chemical reactions. Although the *KEGG* and *BioCyc* offer a rich dataset, the integration of information of these and other databases would definitely allow obtaining a richer set of solutions.

Appendix A

Results

TABLE A.1: Compounds corresponding to the verteces in the Figure 5.5

Node	BioCyc ID	Name
m_0	CPD-674	*trans*-cinnamate
m_1	CINNAMOYL-COA	(*E*)-cinnamoyl-CoA
m_2	CPD-513	3-hydroxy-3-phenylpropionyl-CoA
m_3	CPD-514	3-oxo-3-phenylpropionyl-CoA
m_4	BENZOYLCOA	benzoyl-CoA
m_5	CPD-6443	benzylbenzoate
m_6	BENZOATE	benzoate
m_7	CPD-290	1,2-*cis*-dihydroxybenzoate
m_8	CATECHOL	catechol
m_9	3-4-DIHYDROXYBENZOATE	protocatechuate
m_{10}	VANILLATE	vanillate
m_a	ACETYL-COA	acetyl-CoA
m_b	BENZALDEHYDE	benzaldehyde
m_c	BENZYL-ALCOHOL	benzyl alcohol
m_{s_0}	PHE	L-phenylalanine
m_{s_1}	FORMALDEHYDE	formaldehyde
m_{t_0}	VANILLIN	vanillin

Appendix B

Example Usage

This appendix gathers the set of commands executed in a command line interface to run a simple example. The following command sequence is performed using the Jruby console (version 1.7.4). The tests were performed using the Windows version of Jruby.

To load the framework, a ruby script is run, which simply imports all the required Java libraries.

```
>load 'C:\biosynth_init.rb'
```

The next example shows how to initialize the `DataEnv` from a MySQL database as local storage and KEGG as dataset:

```
>dataEnv = DataEnv.new 'kegg_database', 'db_user', 'db_pass',
SourceDatabase::KEGG
#<Java::EduUminhoBiosynthEnv::DataEnv:0x24d1c210>
```

This is equivalent to setting the sources manually:

```
>mysqlSource = MySQLSource.new 'kegg_database', 'db_user', 'db_pass'
#<Java::EduUminhoBiosynthCoreDataIoLocal::MySQLSource:0x30aec067>
>keggSource = KEGGRemoteSource.new
#<Java::EduUminhoBiosynthCoreDataIoRemote::KEGGRemoteSource:0x7c84e2e1>
```

```
>comboSource = CombineSource.new keggSource, mysqlSource
#<Java::EduUminhoBiosynthCoreDataIoLocal::CombineSource:0x58df96db>
>dataEnv = DataEnv.new comboSource
#<Java::EduUminhoBiosynthEnv::DataEnv:0x25f12d71>
```

The next step is to load the data:

```
>dataEnv.gatherAll
```

or by using the search heuristic (C00755 corresponds to the vanillin compound)

```
>dataEnv.gatherFrom 'C00755'
```

To build a domain simply invoke `createDomain` followed by a string to name it:

```
>dataEnv.createNewDomain 'keggDomain'
#<Java::EduUminhoBiosynthEnv::DataEnv:0x24d1c210>
>puts dataEnv.showDomain
keggDomain
----cpd filters---------
----rxn filters---------
----rpr filters---------
----statistics----------
cpd#: 28059
rxn#: 9387
nil
```

To restrict the domain to a certain criteria, filters can also be added:

```
>dataEnv.createNewDomain 'keggDomain_filtered_by_balanced_reactions'
#<Java::EduUminhoBiosynthEnv::DataEnv:0x24d1c210>
>dataEnv.filterByTag 'BALANCED'
>puts dataEnv.showDomain
keggDomain_filtered_by_balanced_reactions
----cpd filters---------
```

```
----rxn filters---------
Includes Only  Reactions: [R01634, R01635, R01636, R01630, R01631,
R01632, R ...
----rpr filters---------
----statistics----------
cpd#: 28059
rxn#: 7266
nil
```

All these actions are possible with any dataset. To use BioCyc simply use `SourceDatabase::BioCyc` instead. However, the BioCyc interface requires to select an internal database from all the available databases of the BioCyc consortium (if no database selected the MetaCyc is assumed as default):

```
>dataEnvBioCyc = DataEnv.new 'biocyc_database', 'db_user', 'db_pass',
SourceDatabase::BioCyc, 'YEAST'
#<Java::EduUminhoBiosynthEnv::DataEnv:0x399f5184>
```

The remaining operations are roughly the same. Now, moving back to the KEGG example, to run an algorithm, first we must either specify `AlgorithmType` (which set ups an algorithm with the default settings) or define the `AlgorithmConfiguration`. In the next example, it is shown how to configure a simple execution of the FindPath algorithm.

```
>findPathConfig = AlgorithmConfiguration.new AlgorithmType::FindPath
#<Java::EduUminhoBiosynthEnv::AlgorithmConfiguration:0x1ad4ba55>
>puts findPathConfig
==========CONFIG==========
RUNNER:
AlgorithmType: FindPath
TimeLimit: 600
TimeUnits: SECONDS
INPUT:
Substrates #0: []
Products #0: []
ALGORITHM FP CONFIG:
```

```
FastMinimize: true
==========================
nil
```

Then, we load the host genome-scale model as input substrates and select the target product. Also, for this example, we set the execution time limit to 5 minutes:

```
>findPathConfig.setInputAsModel 'C:\iMM904.xml'
#<Java::EduUminhoBiosynthEnv::AlgorithmConfiguration:0x1ad4ba55>
>findPathConfig.setTargetProduct 'C00755'
#<Java::EduUminhoBiosynthEnv::AlgorithmConfiguration:0x1ad4ba55>
>findPathConfig.setTimeLimit 5
#<Java::EduUminhoBiosynthEnv::AlgorithmConfiguration:0x1ad4ba55>
>findPathConfig.setTimeUnit TimeUnit::MINUTES
#<Java::EduUminhoBiosynthEnv::AlgorithmConfiguration:0x1ad4ba55>
>puts findPathConfig
==========CONFIG==========
RUNNER:
AlgorithmType: FindPath
TimeLimit: 5
TimeUnits: MINUTES
INPUT:
Substrates #551: [C00119, C00118, C14818, C00116, C03263, ...]
Products #1: [C00755]
ALGORITHM FP CONFIG:
FastMinimize: true
==========================
nil
```

To run the algorithm just pass the configuration to a `SolverEnv`. A `DataEnv` containing a metabolic domain is also required. The `SolverEnv` automatically maps the domain into the correct representation. Afterwards, just execute the run handle and this will output a `SolutionSet` object which contains all solution structures computed in the defined time frame.

```
>solverEnv = SolverEnv.new findPathConfig, dataEnv
```

```
#<Java::EduUminhoBiosynthEnv::SolverEnv:0x30000e9a>
>solverEnv.run
#<Java::EduUminhoBiosynthCore::SolutionSet:0x541b0d70>
>solutions = _
>puts solutions
#ID: c699edc2-8c68-469a-b0e9-7324a9d741b6
#DESCRIPTION:
#CREATED_AT: 2013-10-31 23:02:27.78
0: {C00230=[R01298], C00001=[R01298], C00755=[R05273], C0000
4=[R05273], C00006=[R01298], C00101=[R10136], C00080=[R05273]
, C06672=[R10136], C00007=[R05273]}
1: {C00230=[R01298], C00001=[R01298, R05699], C00755=[R05699
], C00003=[R05699], C00006=[R01298], C00101=[R10136], C06672=
[R10136]}
2: {C01494=[R03366], C00852=[R04342], C00013=[R02194], ...
```

The object *solutions* contains all the solutions that the FindPath algorithm was able to compute in the 5 minutes time frame. The next step is to evaluate these solutions, using the evaluation system. The `EvalEnv` requires a solution set as parameter. In this example, we use the previous computed set:

```
>evalEnv = EvalEnv.new solutions
#<Java::EduUminhoBiosynthEnv::EvalEnv:0x3e21819>
```

To add an evaluator, we use the `addEvaluator...` functions: in the next example, we added the Flux Balance Analysis evaluator:

```
>evalEnv.addEvaluatorFBA 'C00755'
#<Java::EduUminhoBiosynthEnv::EvalEnv:0x3e21819>
```

It is also possible to add more evaluators (even if they are of the same type). For this example we will also added the model integration evaluator that requires explicitly the target compound, the model file and two files (one for compounds and another for reactions) that maps identifiers between the model and the dataset:

```
>evalEnv.addEvaluatorModelIntegration 'C00755', 'C:\iMM904.xml',
'C:\compoundMap.txt', 'C:\reactionMap.txt'
#<Java::EduUminhoBiosynthEnv::EvalEnv:0x3e21819>
```

Then, the `evaluate` function can be used to compute the evaluation matrix:

```
>evalEnv.evaluate
#<Java::EduUminhoBiosynthEnv::EvalEnv:0x3e21819>
>evalEnv.exportToCSV 'C:\results.csv'
nil
```

The results are stored in a file (CSV format), which can be easily viewed with any text reader or spreadsheet processor. Additionally, the solution sets can be stored into file for future sessions. The `BioSynthUtilsIO` contains several helper functions for I/O operations: the following example shows how to save and read solution set files:

```
>BioSynthUtilsIO.saveSolutionSet solutions, 'C:\solutions.sls'
nil
>BioSynthUtilsIO.loadSolutionSet 'C:\solutions.sls'
#<Java::EduUminhoBiosynthCore::SolutionSet:0x602df8c7>
```

Bibliography

[1] G. Ausiello, A. D'Atri, and D. Saccà. Graph Algorithms for Functional Dependency Manipulation. *Journal of the ACM*, 30(4):752–766, Oct. 1983.

[2] R. K. Aziz, S. Devoid, T. Disz, R. A. Edwards, C. S. Henry, G. J. Olsen, R. Olson, R. Overbeek, B. Parrello, G. D. Pusch, R. L. Stevens, V. Vonstein, and F. Xia. SEED servers: High-Performance Access to the SEED Genomes, Annotations, and Metabolic Models. *PloS one*, 7(10):e48053, Jan. 2012.

[3] a. Bairoch. The ENZYME database in 2000. *Nucleic acids research*, 28(1):304–5, Jan. 2000.

[4] S. L. Bell and B. O. Palsson. Expa: a program for calculating extreme pathways in biochemical reaction networks. *Bioinformatics*, 21(8):1739–1740, Apr. 2005.

[5] A. R. Brochado, C. Matos, B. L. Mø ller, J. r. Hansen, U. H. Mortensen, and K. R. Patil. Improved vanillin production in baker's yeast through in silico design. *Microbial Cell Factories*, 9(1):84, Jan. 2010.

[6] P. Carbonell, D. Fichera, S. B. Pandit, and J.-L. Faulon. Enumerating metabolic pathways for the production of heterologous target chemicals in chassis organisms. *BMC systems biology*, 6(1):10, Jan. 2012.

[7] R. Caspi, H. Foerster, C. A. Fulcher, P. Kaipa, M. Krummenacker, M. Latendresse, S. Paley, S. Y. Rhee, A. G. Shearer, C. Tissier, T. C. Walk, P. Zhang, and P. D. Karp. The MetaCyc Database of metabolic pathways and enzymes and the BioCyc collection of Pathway/Genome Databases. *Nucleic Acids Research*, 36(Database issue):D623–31, Jan. 2008.

[8] M. A. Cibrán, B. Verheecke, W. Vanderperren, D. Suvée, and V. Jonckers. Aspect-oriented Programming for Dynamic Web Service Selection, Integration and Management. *World Wide Web*, 10(3):211–242, Mar. 2007.

[9] B. L. Clarke. Stoichiometric network analysis. *Cell biophysics*, 12:237–53, 1988.

[10] W. B. Copeland, B. A. Bartley, D. Chandran, M. Galdzicki, K. H. Kim, S. C. Sleight, C. D. Maranas, and H. M. Sauro. Computational tools for metabolic engineering. *Metabolic Engineering*, 14(3):270–280, May 2012.

[11] D. Croes, F. Couche, S. J. Wodak, and J. van Helden. Metabolic PathFinding: inferring relevant pathways in biochemical networks. *Nucleic Acids Research*, 33(Web Server issue):W326–30, July 2005.

[12] K. A. Curran and H. S. Alper. Expanding the chemical palate of cells by combining systems biology and metabolic engineering. *Metabolic engineering*, 14(4):289–97, July 2012.

[13] B. de Jong, V. Siewers, and J. Nielsen. Systems biology of yeast: enabling technology for development of cell factories for production of advanced biofuels. *Current Opinion in Biotechnology*, 23(4):624–30, Aug. 2012.

[14] A. Dräger, N. Rodriguez, M. Dumousseau, A. Dörr, C. Wrzodek, N. Le Novère, A. Zell, and M. Hucka. JSBML: a flexible Java library for working with SBML. *Bioinformatics (Oxford, England)*, 27(15):2167–8, Aug. 2011.

[15] M. Durot, P.-Y. Bourguignon, and V. Schachter. Genome-scale models of bacterial metabolism: reconstruction and applications. *FEMS microbiology reviews*, 33(1):164–90, Jan. 2009.

[16] I. Famili, J. Förster, J. Nielsen, and B. O. Palsson. Saccharomyces cerevisiae phenotypes can be predicted by using constraint-based analysis of a genome-scale reconstructed metabolic network. *Proceedings of the National Academy of Sciences of the United States of America*, 100(23):13134–9, Nov. 2003.

[17] I. Famili and B. O. Palsson. The convex basis of the Left Null Space of the Stoichiometric Matrix Leads to the definition of metabolically meaningful pools. *Biophysical Journal*, 85(1):16–26, July 2003.

[18] K. Faust, D. Croes, and J. van Helden. Metabolic pathfinding using RPAIR annotation. *Journal of Molecular Biology*, 388(2):390–414, May 2009.

[19] K. Faust, P. Dupont, J. Callut, and J. V. Helden. Pathway discovery in metabolic networks by subgraph extraction. *Bioinformatics*, 26(9):1211–1218, 2010.

[20] A. M. Feist, M. J. Herrgå rd, I. Thiele, J. L. Reed, and B. O. Palsson. Reconstruction of biochemical networks in microorganisms. *Nature Review Microbiology*, 7:129–143, 2009.

[21] N. Ferrer-miralles, J. Domingo-Espín, J. L. Corchero, E. Vázquez, and A. Villaverde. Microbial factories for recombinant pharmaceuticals. *Microbial Cell Factories*, 8:1–8, 2009.

[22] R. E. Filman and D. P. Friedman. Aspect-oriented programming is quantification and obliviousness. Technical report, 2000.

[23] F. Friedler, K. Tarján, Y. W. Huang, and L. T. Fan. Graph-theoretic approach to process synthesis: axioms and theorems. *Chemical Engineering Science*, 47(8):1973–1988, June 1992.

[24] F. Friedler, K. Tarjan, Y. W. Huang, and L. T. Fan. Graph-Theoretic Approach to Process Synthesis: Polynomial Algorithm for Maximal Structure Generation. *Computer chemical Engineering*, 17(9):929–942, 1993.

[25] F. Friedler, J. B. Varga, and L. T. Fan. Decision-Mapping: A tool for consistent and complete decisions in process synthesis. *Chemical Engineering Science*, 50(11):1755–1778, 1995.

[26] E. H. Hansen, B. L. Mø ller, G. R. Kock, C. M. Bünner, C. Kristensen, O. R. Jensen, F. T. Okkels, C. E. Olsen, M. S. Motawia, and J. r. Hansen. De Novo Biosynthesis of Vanillin in Fission Yeast (Schizosaccharomyces pombe) and Baker's Yeast (Saccharomyces cerevisiae). *Applied and Environmental Microbiology*, 75(9):2765–74, May 2009.

[27] K. Hübner, S. Sahle, and U. Kummer. Applications and trends in systems biology in biochemistry. *The FEBS journal*, 278(16):2767–857, Aug. 2011.

[28] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, a. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, a. a. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup

language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar. 2003.

[29] T. Ideker, T. Galitski, and L. Hood. A New Spproach to Decoding Life: Systems Biology. *Annual Review of Genomics and ...*, 2001.

[30] M. U. Kabir, S. M. Abdulkarim, R. Son, A. H. Azizah, and N. B. Saari. Metabolic engineering of functional phytochemicals. *International Food Research Journal*, 20(1):35–41, 2013.

[31] M. Kanehisa and S. Goto. KEGG: kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30, Jan. 2000.

[32] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori. The KEGG resource for deciphering the genome. *Nucleic acids research*, 32(Database issue):D277–80, Jan. 2004.

[33] P. D. Karp and R. Caspi. A survey of metabolic databases emphasizing the MetaCyc family. *Archives of Toxicology*, 85(9):1015–33, Sept. 2011.

[34] P. D. Karp, C. A. Ouzounis, C. Moore-Kochlacs, L. Goldovsky, P. Kaipa, D. Ahrén, S. Tsoka, N. Darzentas, V. Kunin, and N. López-Bigas. Expansion of the BioCyc collection of pathway/genome databases to 160 genomes. *Nucleic Acids Research*, 33(19):6083–6089, Jan. 2005.

[35] P. D. Karp, S. Paley, and P. Romero. The Pathway Tools software. *Bioinformatics (Oxford, England)*, 18 Suppl 1:S225–32, Jan. 2002.

[36] P. D. Karp, S. M. Paley, M. Krummenacker, M. Latendresse, J. M. Dale, T. J. Lee, P. Kaipa, F. Gilham, A. Spaulding, L. Popescu, T. Altman, I. Paulsen, I. M. Keseler, and R. Caspi. Pathway Tools version 13.0: integrated software for pathway/genome informatics and systems biology. *Briefings in bioinformatics*, 11(1):40–79, Jan. 2010.

[37] I.-K. Kim, A. Roldão, V. Siewers, and J. Nielsen. A systems-level approach for metabolic engineering of yeast cell factories. *FEMS yeast research*, 12(2):228–48, Mar. 2012.

[38] D. Klein-Marcuschamer, P. K. Ajikumar, and G. Stephanopoulos. Engineering microbial cell factories for biosynthesis of isoprenoid molecules: beyond lycopene. *Trends in biotechnology*, 25(9):417–24, Sept. 2007.

[39] M. Kotera, M. Hattori, M. Oh, and R. Yamamoto. RPAIR: a reactant-pair database representing chemical changes in enzymatic reactions. *Genome Informatics*, (1):3–4, 2004.

[40] M. Latendresse and P. D. Karp. An advanced web query interface for biological databases. *Database: The Journal of Biological Databases and Curation*, 2010:13, Jan. 2010.

[41] D.-y. Lee, L. T. Fan, S. Park, S. Yup, S. Shafie, B. Bertók, F. Friedler, and S. Y. Lee. Complementary identification of multiple flux distributions and multiple metabolic pathways. *Metabolic Engineering*, 7(3):182–200, May 2005.

[42] J. M. Lee, E. P. Gianchandani, and J. A. Papin. Flux balance analysis in the era of metabolomics. *Briefings in bioinformatics*, 7(2):140–150, June 2006.

[43] H. Ma and A.-P. Zeng. Reconstruction of metabolic networks from genome data and analysis of their global structure for various organisms. *Bioinformatics (Oxford, England)*, 19(2):270–7, Jan. 2003.

[44] D. Machado, Z. Soons, K. R. Patil, E. C. Ferreira, and I. Rocha. Random sampling of elementary flux modes in large-scale metabolic networks. *Bioinformatics (Oxford, England)*, 28(18):i515–i521, Sept. 2012.

[45] A. G. McDonald, S. Boyce, and K. F. Tipton. ExplorEnz: the primary source of the IUBMB enzyme list. *Nucleic acids research*, 37(Database issue):D593–7, Jan. 2009.

[46] M. L. Mo, B. O. Palsson, and M. J. Herrgå rd. Connecting extracellular metabolomic measurements to intracellular flux states in yeast. *BMC Systems Biology*, 3:37, Jan. 2009.

[47] J. H. Park and S. Y. Lee. Towards systems metabolic engineering of microorganisms for amino acid production. *Current Opinion in Biotechnology*, 19(5):454–60, Oct. 2008.

[48] J. H. Park, S. Y. Lee, T. Y. Kim, and H. U. Kim. Application of systems biology for bioprocess development. *Trends in biotechnology*, 26(8):404–12, Aug. 2008.

[49] Z. Pawlak. Rough Sets. *International Journal of Computer & Information Sciences*, 11(5):341–356, Oct. 1982.

[50] P. Pharkya, A. P. Burgard, and C. D. Maranas. OptStrain : A computational framework for redesign of microbial production systems. *Genome Research*, (814):2367–2376, 2004.

[51] S. a. Rahman, P. Advani, R. Schunk, R. Schrader, and D. Schomburg. Metabolic pathway analysis web service (Pathway Hunter Tool at CUBIC). *Bioinformatics (Oxford, England)*, 21(7):1189–93, Apr. 2005.

[52] I. Rocha, P. Maia, P. Evangelista, P. Vilaça, S. a. Soares, J. P. Pinto, J. Nielsen, K. R. Patil, E. C. Ferreira, and M. Rocha. OptFlux: an open-source software platform for in silico metabolic engineering. *BMC Systems Biology*, 2010.

[53] A. Samal, S. Singh, V. Giri, S. Krishna, N. Raghuram, and S. Jain. Low degree metabolites explain essential reactions and enhance modularity in biological networks. *BMC Bioinformatics*, 10:1–10, 2006.

[54] J. Schellenberger, J. O. Park, T. M. Conrad, and B. O. Palsson. BiGG: a Biochemical Genetic and Genomic knowledgebase of large scale metabolic reconstructions. *BMC bioinformatics*, 11:213, Jan. 2010.

[55] S. Schuster and C. Hlgetag. On Elementary Flux Modes in Biochemical Reaction Systems at Steady State. *Journal of Biological Systems*, 2(2):165–182, 1994.

[56] E. Sousa, R. C. Gonçalves, D. Neves, and J. a. L. Sobral. Non-Invasive Gridification through an Aspect-Oriented Approach. In *2nd Iberian Grid Infrastructure Conference*, 2008.

[57] G. Stephanopoulos. Metabolic fluxes and metabolic engineering. *Metabolic engineering*, 1(1):1–11, Jan. 1999.

[58] M. Terzer and J. Stelling. Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics (Oxford, England)*, 24(19):2229–35, Oct. 2008.

[59] M. Thakur and R. Tripathi. Linear connectivity problems in directed hypergraphs. *Theoretical Computer Science*, 410(27-29):2592–2618, June 2009.

[60] N. Tomar and R. K. De. Comparing methods for metabolic network analysis and an application to metabolic engineering. *Gene*, 521(1):1–14, May 2013.

[61] a. Wagner and D. a. Fell. The small world inside large metabolic networks. *Proceedings. Biological sciences / The Royal Society*, 268(1478):1803–10, Sept. 2001.

[62] N. J. Walton, M. J. Mayer, and A. Narbad. Vanillin. *Phytochemistry*, 63(5):505–515, July 2003.

[63] N. J. Walton, A. Narbad, C. Faulds, and G. Williamson. Novel approaches to the biosynthesis of vanillin. *Current opinion in biotechnology*, 11(5):490–6, Oct. 2000.

[64] T. Wilhelm, J. Behre, and S. Schuster. Analysis of structural robustness of metabolic networks. *Systems biology*, 1(1):114–20, June 2004.

[65] W. Zhou and L. Nakhleh. The strength of chemical linkage as a criterion for pruning metabolic graphs. *Bioinformatics (Oxford, England)*, 27(14):1957–63, July 2011.