Compilation de connaissances pour l'analyse des réseaux métaboliques

Maxime Mahout

université
PARIS-SACLAY

Rapport de stage Master 2 AMI2B

Compilation de connaissances pour l'analyse des réseaux métaboliques

Stagiaire : M. Maxime Mahout
Tuteur : Mme Sabine Peres
Parrain : M. Olivier Lespinet

Laboratoire de Recherche en Informatique, Orsay
Équipe BIOINFO

17 Février 2020 - 15 Août 2020
En collaboration avec Pr. Laurent Simon
Laboratoire Bordelais de Recherche en Informatique

Table des matières

Présentation

J'ai réalisé ce stage dans l'équipe BioInfo du Laboratoire de Recherche en Informatique, encadré par Mme Sabine Peres, et en collaboration avec Pr. Laurent Simon, responsable de l'équipe Modélisation et Vérification du Laboratoire Bordelais de Recherche en Informatique.

Dans le cadre de ce stage nous étudions l'intégration de contraintes de régulation au calcul des Modes Élémentaires de Flux (EFMs), une méthode d'analyse des réseaux métaboliques. Deux outils sont comparés : une méthode de programmation logique, Answer Set Programming (ASP) et une méthode de compilation de connaissances, les Diagrammes de Décision Binaires (BDDs).

Ce stage fait suite à mon stage précédent l'année dernière également encadré par Mme Sabine Peres, durant lequel j'ai développé une méthode de calcul des Modes Élémentaires de Flux à base de Answer Set Programming (ASP) et de Programmation Linéaire (LP) utilisant le solveur clingo[LP]. Il s'agit de l'outil que nous allons utiliser pour calculer les EFMs.

Pour évaluer l'efficacité d'une compilation préalable des contraintes de régulation dans un Diagramme de Décision Binaire, nous la comparerons avec le traitement direct des contraintes de régulation pendant le calcul des EFMs. Pour cela, nous interfaçons les BDDs au solveur clingo[LP] et nous exprimons les contraintes de régulation soit en ASP soit en BDD.

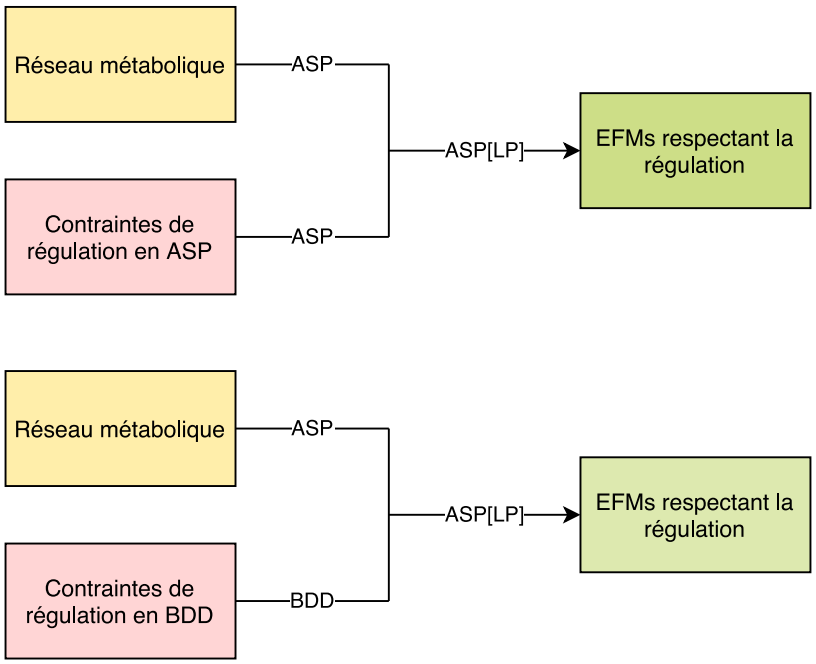

Figure 1: Schéma d'exécution des deux méthodes d'intégration des contraintes de régulation

1 Introduction

En biologie des systèmes, les réseaux métaboliques sont étudiés afin de comprendre les propriétés des systèmes vivants. Le métabolisme est l'ensemble des réactions biochimiques qui se déroulent au sein de la cellule.

Les réseaux métaboliques sont représentés par des hypergraphes orientés où les nœuds représentent les composants biochimiques (métabolites) et les hyper-arêtes représentent les réactions enzymatiques.

Un exemple de réseau métabolique est présenté ci-dessous en Figure 2. Il s'agit du métabolisme central de la bactérie Escherichia coli [1].

On peut y observer des voies métaboliques particulières, des chemins sur ce graphe, qui vont consommer un ensemble de métabolites en entrée et produire un autre ensemble en sortie.

Ces voies peuvent être décrites par des vecteurs de flux, c'est à dire des vecteurs qui donnent pour chaque réaction du réseau la vitesse estimée de métabolite consommée.

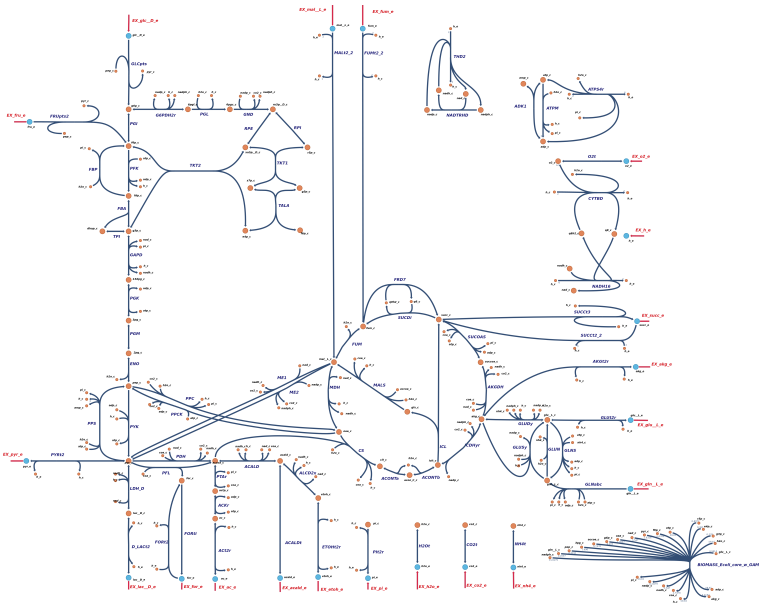

Figure 2: Réseau métabolique central d'Escherichia coli (Orth, Fleming, Palsson, 2010)

Pour analyser les réseaux, on peut étudier les flux métaboliques à l'état stationnaire, c'est-à-dire l'état où la variation de concentration en métabolites est nulle. On distingue alors sur l'hypergraphe les métabolites dits externes – les sources et destinations des flux – des autres métabolites dits internes.

Les flux à l'état stationnaire permettent de déterminer les propriétés du réseau (comme la robustesse, le couplage, les gènes essentiels, etc.) avant de réaliser une étude dynamique (nécessitant beaucoup de paramètres).

Le Flux Balance Analysis [2] est une des méthodes d'analyse des flux à l'état stationnaire. Il s'agit d'obtenir un flux à l'état stationnaire en utilisant la programmation linéaire, en maximisant ou minimisant une fonction objectif. En revanche, ces flux peuvent s'avérer différents de ceux qui sont observés *in vivo*, car les solutions trouvées ne sont pas uniques.

Une autre méthode d'analyse est d'énumérer les Modes Élémentaires de Flux [3], des solutions particulières qui permettent de caractériser tout l'espace de solution des flux à l'état stationnaire.

2 Modes Élémentaires de Flux

2.1 Formalisme mathématique

On considère une matrice stœchiométrique $S \in \mathbb{R}^{m \times r}$ de m métabolites internes en ligne et de r réactions en colonne. Pour chaque réaction, on reporte sur la matrice les quantités de chaque métabolite produites ou consommées, ainsi les réactifs auront un coefficient négatif dans la matrice.

Une distribution de flux est un vecteur $v \in \mathbb{R}^r$ donnant les taux de réactions pour chaque réaction. Son support est l'ensemble de réactions actives : $Supp(v) = \{i \mid v_i \neq 0\}$.

On distingue les réactions irréversibles, dont le flux est un réel positif ou nul, des réactions réversibles, dont le flux peut également être négatif.

L'évolution des concentrations $X(t)$ des métabolites en fonction du temps s'écrit par un système d'équations différentielles :

$$\frac{dX(t)}{dt} = S.v(x(t))$$

A l'état stationnaire, la variation de ces concentrations est nulle, on a $dX(t)/dt = 0$ et donc $S.v(x(t)) = 0$. L'ensemble des solutions des flux v est donc dans le noyau de S.

On note C l'ensemble des vecteurs du noyau de la matrice stœchiométrique qui respectent l'orientation des réactions irréversibles. Cet ensemble de solutions est un cône convexe polyhédrique.

$$C = \left\{ v \in \mathbb{R}^r \mid Sv = \vec{0} \text{ et } \forall i \text{ irréversible } v_i >= 0 \right\}$$

Les modes élémentaires de flux (Elementary Flux Mode ou EFM) sont alors les solutions de support minimal au sens de l'inclusion :

$$E = \{e \in C \mid \nexists e' \in C \; Supp(e') \subset Supp(e)\}$$

Dans le cas où toutes les réactions sont irréversibles, le cône C est pointé en zéro et les modes élémentaires E sont les arêtes du cône. Il est donc intéressant de décomposer les réactions réversibles en réactions irréversibles.

$$S = \begin{array}{c} \begin{matrix} \text{T1} & \text{T2} & \text{T3} & \text{T4} & \text{R1} \end{matrix} \\ \begin{pmatrix} 1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & -1 & 1 \end{pmatrix} \end{array}$$

$$E = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Figure 3: Modes élémentaires de flux d'un réseau métabolique simple

Un exemple de réseau métabolique et de modes élémentaires de flux est présenté en Figure 3 et Figure 4. Les réactions en rouge sur la Figure 4 sont celles avec un flux non nul et indiquent le support des modes élémentaires de flux.

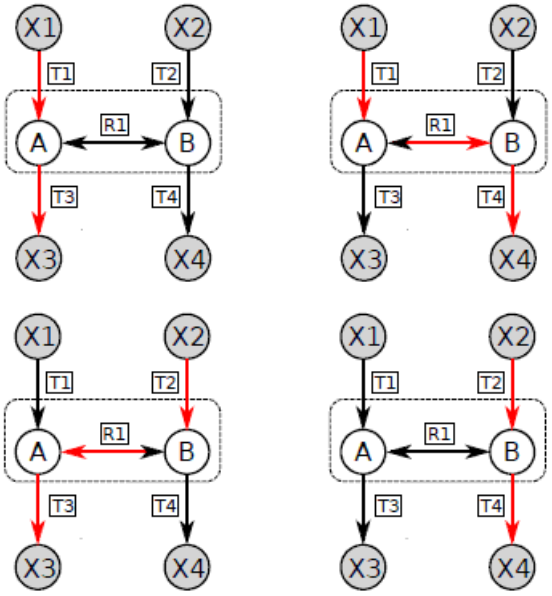

Figure 4: Modes élémentaires de flux d'un réseau métabolique simple

Puisque les EFMs sont des solutions de support minimal, il n'existe pas deux EFMs différents avec le même support. Par la suite, nous dénoterons donc les EFMs par leur support, de la manière suivante: $M1 = \{T1, T3\}$, $M2 = \{T1, R1, T4\}$, $M3 = \{T2, R1, T3\}$, $M4 = \{T2, T4\}$.

2.2 Méthodes de calcul

Traditionnellement, l'algorithme utilisé pour calculer les modes élémentaires de flux est la Double Description [4]. Il s'agit d'un algorithme efficace à partir de calcul matriciel. En revanche, plus les réseaux sont grands, plus il y a d'EFMs et moins il est envisageable de réaliser ces calculs : il y a une explosion combinatoire. Il est pour le moment impossible d'énumérer les EFMs sur des réseaux dits à l'échelle du génome qui peuvent avoir jusqu'à 10 000 réactions [5].

Un autre problème inhérent au calcul des EFMs consiste à trouver des EFMs d'intérêts. En effet, sur un réseau métabolique, parmi les nombreux EFMs trouvés, seule une petite partie est en réalité observée en laboratoire. La majorité des modes élémentaires trouvés ne sont plus plausibles lorsque l'on prend en compte l'ensemble des contraintes biologiques. Il est alors nécessaire d'intégrer des contraintes de sélection des EFMs directement pendant le calcul, car cela est beaucoup moins coûteux en temps et en espace mémoire que de calculer l'ensemble des EFMs puis les filtrer.

Une première implémentation du calcul des EFMs est MetaTool [6], sous MATLAB. L'implémentation la plus efficace à ce jour, utilisant la Double Description, est EfmTool [7], sous Java. EfmTool utilise toutes les techniques de compression et réduction de réseau connues ainsi que des structures de données plus performantes. Il est possible d'intégrer des contraintes au calcul des EFMs par la Double Description [8][9], cependant celles-ci doivent être monotones sur le support.

Dans le but d'énumérer directement les EFMs sous contraintes, de nouvelles méthodes ont vu le jour, notamment à base d'optimisation linéaire. Ces méthodes exploitent le fait que retrouver un premier EFM peut se faire en temps polynômial avec un programme linéaire. C'est énumérer l'ensemble des EFMs qui est un problème dans la classe de complexité #P-Difficile [10][11].

En 2009, de Figueireido et al [12] proposent d'énumérer les modes élémentaires les plus courts avec une méthode de programmation linéaire mixte MILP (Mixed Integer Linear Programming) nommée k-shortest EFMs, qui énumère les EFMs les plus courts jusqu'à l'itération k, où k est le nombre de réactions de l'EFM.

Cette méthode a été revisitée à plusieurs reprises [13][14], en particulier pour d'autres applications telles que les GFMs (Generating Flux Modes, sous-ensemble des EFMs) [15] et les MCS (Minimal Cut Sets) [16], une application majeure des modes élémentaires de flux qui permet de voir les réactions essentielles au réseau métabolique.

En 2014 et 2016, le problème de la satisfiabilité booléenne (SAT) et les Satisfiability Modulo Theories (SMT) sont envisagés pour le calcul des EFMs sous contraintes [17][18]. La résolution linéaire se fait cette fois-ci au niveau de la théorie Linear Real Arithmetic du solveur SMT.

Ces méthodes parviennent à énumérer des EFMs à la volée sur des réseaux de très grande taille, pour lesquels la méthode de la Double Description peut peiner à renvoyer un résultat.

Nous allons nous intéresser plus particulièrement à l'énumération des EFMs respectant des contraintes de régulation transcriptionnelles, un type de contraintes booléennes qui ont pu être prises en charge par l'outil *SMTTool* [18] et intégrées dans la Double Description par l'outil *regEFMTool* [8].

3 Réseaux de régulation

3.1 Présentation

Un réseau de régulation transcriptionnelle représente un ensemble d'interactions entre des gènes et des facteurs de transcriptions, généralement de type activation ou inhibition. Dans un modèle métabolique, chaque réaction métabolique est associée à des enzymes susceptibles de la catalyser. Nous allons nous intéresser à l'intégration de la régulation des gènes codant pour ces enzmyes.

Une application majeure du Flux Balance Analysis (FBA) est de pouvoir effectuer des prédictions d'ordre phénotypique [19]. Dans un modèle métabolique simple, seules les concentrations en métabolites externes et les bornes sur les flux peuvent varier. Si l'on souhaite avoir des prédictions proches de la réalité, il faut alors intégrer des contraintes biologiques [20].

Une première approche d'intégration de données transcriptionnelles de régulation aux modèles métaboliques a été réalisée par Covert, Schilling et Palsson en 2001 sur un réseau jouet [21]. La régulation transcriptionnelle est alors prise en charge dans la méthode de dynamic FBA, développée plus tôt [2]. La méthode est ensuite mise en application sur un réseau d'Escherichia Coli à petite échelle par Covert et Palsson en 2002 [22].

Chaque réaction peut être régulée ($\Rightarrow$) par des protéines de régulation ou des métabolites externes d'après une formule booléenne composée de NOT ($\neg$), de AND ($\wedge$) et de OR ($\vee$)[1].

$$
\begin{aligned}
FUMAR &\implies \neg\,(ArcA \vee FNR)\\
FUMBR &\implies FNR\\
ADHER &\implies \neg\,(O2xt \wedge FruR)
\end{aligned}
$$

Les auteurs réalisent ensuite en 2004 un réseau métabolique à l'échelle du génome d'Escherichia Coli intégrant des données de régulations issues directement de données transcriptomiques [23], avec lequel ils prédisent de nouveaux phénotypes.

En 2010, Orth et al. publient un guide pour la reconstruction de modèles biologiques incluant des règles de régulation transcriptionnelles, avec comme exemple présenté un modèle du métabolisme central d'Escherichia Coli [1].

Plutôt que de contenir une réaction pour chaque isoenzyme, ce qui était le cas du modèle de 2002, ce modèle définit une relation entre gènes et réactions, puis les contraintes transcriptionnelles sur les gènes. Le nombre de réactions est alors considérablement diminué.

Notons que pour les associations entre gènes et réactions, le NOT ($\neg$) n'est pas autorisé, seuls AND ($\wedge$) et OR ($\vee$) sont permis.

$$
\begin{aligned}
FRD7 &\implies (frdA \wedge frdB \wedge frdC \wedge frdD)\\
FUM &\implies (fumA \vee fumB \vee fumC)\\
adhE &\implies (\neg\, o2e) \vee (\neg\, FruR) \vee Fis
\end{aligned}
$$

Il existe aujourd'hui un certain nombre de modèles à l'échelle du génome contenant des règles de régulations transcriptionnelles, répertoriés sur le site BiGG [5] ou sur le site BioModels [24] en format SBML (Systems Biology Markup Language) [25]. Associer des gènes à chaque réaction est possible dans la spécification la plus récente du format SBML pour l'analyse de flux [26].

[1] Code couleur: Réactions, Métabolite de l'environnement, Régulateurs, Gènes
Le premier exemple est tiré de Covert, Palsson, 2002. Le second est tiré de Orth et al., 2010.

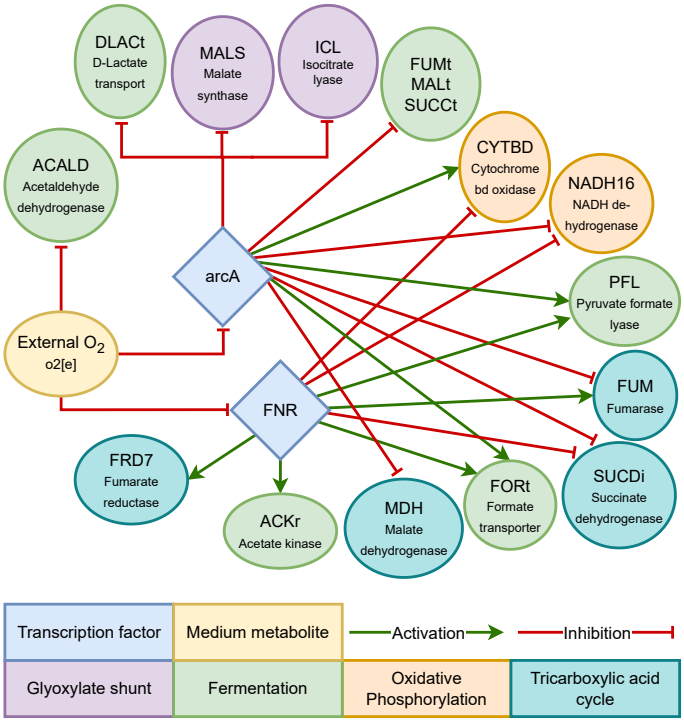

Figure 5: Réseau de régulation transcriptionnelle d'EcoliCore: Régulation liée à l'oxygène

Quant à ce qui concerne les EFMs, en 2003, Covert et Palsson ont calculé les *extreme pathways* respectant les règles de régulation sur leur réseau jouet présenté en 2001 [27]. Les *extreme pathways* sont un sous-ensemble des EFMs d'un réseau métabolique, mais dans le cas de ce modèle, tous les EFMs sont des *extreme pathways*. Leurs calculs prennent en compte les modifications que l'environnement apporte à la régulation.

Le logiciel *regEFMTool* de Jungreuthmayer et al. [8] a été testé quant à lui sur le réseau central d'Escherichia Coli de Orth et al., 2010. Il ne semble pas à notre connaissance prendre en compte les modifications apportées par l'environnement.

Pour notre étude des EFMs sous contraintes de régulation, nous allons donc nous pencher sur les deux réseaux précédents, que nous appellerons respectivement Csp2001 et EcoliCore. Les informations sur la taille des modèles sont présentées en Tableau 1. Le réseau EcoliCore est représenté en Figure 2 et son réseau de régulation en Figure 5.

Réseau métabolique	Csp2001	EcoliCore
Nombre de réactions	20	95 (dont 20 réactions d'échange)
Nombre de métabolites	11 internes, 8 externes	72 internes, 20 externes
Nombre de gènes	Pas de gènes	137 gènes, associés aux réactions
Nombre de règles de régulation	11 (7 réactions régulées, 4 régulateurs)	78 (56 gènes régulés, 17 régulateurs)
Nombre total d'EFMs	80 (59 respectent la régulation)	$226.3 \cdot 10^6$

Tableau 1: Tableau récapitualif des informations des deux réseaux métaboliques étudiés

Afin de pouvoir prendre en compte les contraintes apportées par l'environnement nous introduisons des règles de régulation supplémentaires liées aux réactions de transport (exemple: un transporteur d'oxygène ne peut être présent que si de l'oxygène externe est présent).

3.2 Formalisme

Une règle de régulation R_i sur une variable x_h du réseau de régulation est de la forme :

$$R_i : x_h \Longrightarrow \phi_i(x_{b_1}, \ldots, x_{b_m}) \quad (1)$$

$$R_i : x_h \Longleftrightarrow \phi_i(x_{b_1}, \ldots, x_{b_m}) \quad (2)$$

Où ϕ_i est une formule booléenne associée, et $x_{b_1}, \ldots, x_{b_m}$ d'autres variables booléennes du réseau.

Le réseau est ainsi formé des variables $x_1, \ldots, x_n$ et de la conjonction de règles $R_1 \wedge \cdots \wedge R_p$.

On dispose de quatre types de variables contrôlées par des règles de régulation sur le réseau : protéines de régulations, réactions actives, gènes, et métabolites présents dans l'environnement.

Ces quatre types de variables sont inspirés de ce que l'on retrouve dans la littérature décrivant des réseaux de régulation pour les modèles à base de contraintes.

3.3 Protéines de régulation

Il s'agit de produits intermédiaires qui sont activés par l'environnement, les réactions actives ou d'autres protéines de régulation.

Ils correspondent à des régulateurs connus du métabolisme, tels que les facteurs de transcription *ArcA* et *FNR* chez *Escherichia coli*, ou alors à des indicateurs fictifs indiquant un état du réseau métabolique (par exemple: "surplusFDP").

La valeur de ces variables booléennes est déterminée automatiquement par inférence et l'utilisateur n'a aucun moyen de la contrôler.

3.4 Métabolites présents dans l'environnement

La présence de métabolites externes dans l'environnement de départ (exemple: milieu de croissance d'une bactérie) va déterminer quelles protéines de régulation et quelles réactions vont être actives.

Les métabolites de l'environnement sont aussi impliqués dans des réactions de transport : enlever un métabolite externe de l'environnement va automatiquement désactiver son transporteur.

Les variables booléennes indiquent la présence ou l'absence de ces métabolites et sont contrôlées par l'utilisateur.

3.5 Réactions actives, gènes

Les réactions actives sont régulées par les protéines de régulation ou directement par les métabolites externes de l'environnement pour les cas trop complexes. D'un point de vue biologique, la régulation agit sur la transcription de l'enzyme catalysant la réaction.

A partir d'un mode de flux donné, on peut déterminer quelles réactions sont actives et quelles réactions sont inactives (support du mode de flux).

Pour chaque mode de flux, l'utilisateur peut donc mettre les variables booléennes à 1 pour les réactions qui sont actives et 0 pour les réactions inactives, et interroger le réseau de règles afin de savoir si le mode de flux est conforme à la régulation.

Dans les réseaux métaboliques à l'échelle du génome, les réactions sont associées aux gènes et les régulations sont effectuées au niveau des gènes. En pratique, la valeur des variables booléennes des gènes est déterminée automatiquement par l'inférence comme pour les protéines de régulation.

3.6 Exemple

L'exemple suivant présente les règles de régulation du réseau Csp2001. Il s'agit d'un réseau jouet modélisant les réactions standard étudiées sur un réseau métabolique.

Reactions

$$r2a \implies \neg rpb$$
$$r5a \implies \neg rpo2$$
$$r5b \implies rpo2$$
$$r7 \implies \neg rpb$$
$$r8a \implies \neg rph$$
$$rres \implies \neg rpo2$$
$$tc2 \implies \neg rpc1$$

Regulation proteins

$$rpo2 \iff \neg oxygen$$
$$rpc1 \iff carbon1$$
$$rph \iff th$$
$$rpb \iff r2b$$

Metabolite inputs

$$tc2 \implies carbon2$$
$$tc1 \implies carbon1$$
$$th \implies h$$
$$tf \implies f$$
$$to2 \implies oxygen$$

Reactions : $\{r1, r2a, r2b, r3, r4, r5a, r5b, r6, r7, r8a, r8b, rres, tc2, tc1, th, tf, to2, growth, td, te\}$
Regulation proteins : $\{rpo2, rpc1, rph, rpb\}$
Metabolite inputs : $\{carbon1, carbon2, h, f, oxygen\}$

Le mode élémentaire $\{r2b, r3, r4, r5b, r8b, rres, th, to2, growth\}$ n'est pas conforme à la régulation. En effet, on a: $rres \implies \neg rpo2$ et $r5b \implies rpo2$, une contradiction. De plus, si l'environnement ne contenait pas d'oxygène, alors la règle $to2 \implies oxygen$ ne pourrait plus être respectée.

4 Answer Set Programming

Lors d'un stage de deux mois l'année précédente, j'ai pu développer un outil de calcul des EFMs à partir du paradigme de programmation logique Answer Set Programming (ASP).

Le principe de calcul est dans la continuité des méthodes à base de SAT/SMT et de celles utilisant MILP. Les solveurs utilisés sont *clingo* [28] pour la programmation logique ASP, et *cplex* pour la programmation linéaire, à l'aide de l'interface appelée *clingo[LP]* [29].

4.1 Présentation

Answer Set Programming est un langage de programmation logique avec une syntaxe très riche. Le langage utilise le postulat de Monde Fermé (Closed World Assumption) pour son raisonnement, ainsi ce qui a été énoncé sera considéré comme vrai et ce qui n'a pas été énoncé sera automatiquement considéré comme faux. Les solutions d'un programme logique ASP sont appelées ensembles réponses (Answer Sets).

On distingue trois types de propositions logiques en ASP:

- Les faits: `atome(a).` *A est vrai*
- Les règles: `atome(a) :- atome(b0) ; ... ; atome(bn).` $B_0 \wedge \cdots \wedge B_n \implies A$
- Les contraintes d'intégrité: `:- atome(b0) ; ... ; atome(bn).` $B_0 \wedge \cdots \wedge B_n$ *est faux*

Il est possible de calculer des prédicats de logique de premier ordre: les noms de variables sont alors en majuscules et les noms d'instances en minuscule. Par exemple, si on a les faits `reaction(r1). reaction(r2). support(r2).` et la règle `inactive(R) :- reaction(R); not support(R).`, alors il y aura un unique ensemble réponse, contenant l'atome `inactive(r1)`.

Le langage de *clingo[LP]* ajoute à cela des atomes dits de théorie, commençant par une éperluette (`&`), destinés au solveur de programmation linéaire. Ainsi `&dom` définit le domaine des variables pour le solveur linéaire et `&sum` définit des contraintes linéaires.

Afin de calculer les EFMs d'un réseau métabolique, nous encodons le réseau métabolique et ses contraintes en ASP, puis nous lançons le solveur avec en entrée le réseau, les contraintes et le programme logique ASP[LP] présenté en sous-section 4.2.

Le programme logique est composé de plusieurs règles dont une définissant des variables linéaires positives ou nulles donnant le flux pour chaque réaction (i.e. un vecteur de flux v), une assurant que la direction thermodynamique des flux est respectée, une évitant la solution triviale et enfin la contrainte de stationnarité des flux ($Sv = 0$).

Pour chaque solution, les atomes `support(R)` présents dans l'ensemble réponse désignent les réactions du support : les réactions actives telles que leur flux est non nul.

À la différence des méthodes MILP, aucune minimisation n'est imposée: on ne souhaite pas les EFMs les plus courts. Le calcul des EFMs, solutions dont le support est minimal par inclusion, est fait à l'aide des heuristiques de *clingo* spécifiques aux domaines [30].

clingo[LP] énumère les EFMs en ajoutant un nogood sur le support à chaque fois qu'une solution est trouvée (ex: si la solution est $\{T1, R1, T4\}$, le solveur ajoute la clause booléenne négative suivante : $\neg(T1 \wedge R1 \wedge T4)$, c'est ce qu'on appelle un nogood).

4.2 Encodage du calcul des EFMs

```
% Borne supérieure pour les valeurs de flux
#const nb=5000.

% Toutes les réactions sont irréversibles: elles ont un flux positif ou nul
&dom{0..nb} = flux(R) :- reaction(R).

% Deux réactions irréversibles issues de la séparation d'une réaction
% réversible sont mutuellement exclusives
:- support(R1); support(R2); 1 {reversible(R1, R2); reversible(R2, R1)} 1;
   reaction(R1); reaction(R2).

% Au moins une réaction doit être utilisée
:- not support(R) : reaction(R).

% Pour chaque métabolite, la somme pondérée par la stoechiométrie
% des flux de chaque réaction le consommant ou le produisant est nulle
&sum{C*flux(R) : stoichiometry(M, R, C), reaction(R)} = 0 :- metabolite(M).

% Définition du support, atomes dont la présence indiquent un flux non nul
support(R) :- &sum{flux(R)} > 0; reaction(R).

% Heuristique sur les atomes support pour la minimisation
#heuristic support(R). [1, false]
```

4.3 Contraintes booléennes et linéaires

L'avantage de posséder un framework tel que *clingo[LP]* est de pouvoir intégrer très facilement des contraintes booléennes et linéaires. Les contraintes booléennes vont être prises en compte par le solveur de programmation logique *clingo* et les contraintes linéaires par le solveur de programmation linéaire *cplex*. Tous deux sont des leaders dans leurs catégories en terme de performance.

De plus, l'aspect interface et practicabilité est grandement pris en compte pour un biologiste. Nous pensons que cette méthode de raisonnement automatique est plus intelligible qu'une méthode computationnelle telle que *EFMTool* ou que le précédent outil *SMTTool* à base de clauses booléennes.

Les exemples suivants présentent deux types de contraintes qui peuvent être intégrées dans *clingo[LP]* pour obtenir des EFMs cohérents biologiquement: les contraintes de régulation (contraintes booléennes) et les contraintes thermodynamiques (contraintes linéaires).

```
% Le régulateur Reg active T1 : T1 => Reg
:- support(t1); not regul(reg).
% Le régulateur Reg inhibe T2 : T2 => ~ Reg
:- support(t2); regul(reg).
```

```
% Valeurs des constantes d'équilibre apparentes
keq(t2, 1). keq(t3, -1).
keq(r1, 1). keq(r1_rev, -1).
keq(t1, 1). keq(t4, 1).
% Règle thermodynamique
&sum{K*flux(R) : reaction(R), keq(R, K)} > 0.
```

4.4 Contraintes positives

Une contrainte est dite positive si elle est équivalente à une contrainte linéaire quelconque de la forme $b^T \nu \geq c$.

D'après cette définition, toutes les contraintes booléennes sur le support qui sont des clauses de littéraux positifs sont des contraintes positives.

Par exemple, demander à ce que les EFMs calculés contiennent la réaction de biomasse et la réaction produisant de l'ATP est une contrainte positive.

```
% Ensemble ces règles expriment la contrainte "Biomass AND ATP"
:- not support("Biomass"). % impossible d'avoir flux("Biomass") = 0
:- not support("ATP"). % etc.
```

De même pour les contraintes de ratios de flux, du type:

```
&sum{flux("R1"); -3*flux("R2")} > 0. % R1 - 3 R2 > 0 <=> R1 > 3 R2
```

Si l'on ajoute plus d'une contrainte positive lors de la recherche de solutions, on n'a aucune garantie que les résultats trouvés soient des modes élémentaires de flux : il peut s'agir de modes non élémentaires, des combinaisons de plusieurs EFMs.

En effet, les modes élémentaires sont les extrémités d'un cône formé par l'ensemble des contraintes imposées par l'hypothèse de stationnarité. Or, rajouter des contraintes linéaires peut donner un cône différent [13][18].

La recherche d'EFMs respectant des contraintes positives est donc plutôt réalisée lors du traitement post-processing, une fois que toutes les solutions sont calculées.

4.5 Application

Fort heureusement, les contraintes de régulations peuvent toutes être exprimées sous forme de contraintes qui ne sont pas positives.

Il suffit de toujours exprimer les règles de régulation sous la forme ($Reaction \implies Regulateur$) et jamais sous la forme ($Regulateur \implies Reaction$).

Par exemple, voici deux programmes ASP illustrant l'action des règles de régulation. Le sous-programme présenté ci-dessous correspond à plusieurs règles de régulation issues du réseau Csp2001 (voir sous-section 3.6).

```
% TO2 => Oxygen
:- support(to2); not env(oxygen).
% R5B => RPO2
:- support(r5b); regul(rpo2).
% RRes => ~RPO2
:- support(rres); not regul(rpo2).
% RPO2 est présent ssi oxygen est absent.
reg(rpo2) :- not env(oxygen).
```

Celui présenté ci-dessous comporte l'instance invoquée : on cherche à vérifier si en présence d'oxygène l'EFM suivant : $\{r2b, r3, r4, r5b, r8b, rres, th, to2, growth\}$ respecte la régulation.

```
% Exemple de solution calculée par les règles clingo[LP] pour Covert Palsson
support(r2b; r3; r4; r5b; r8b; rres; th; to2; growth). % p(a;b). <=> p(a). p(b).
% On souhaite les EFMs respectant la régulation en présence d'Oxygène
env(oxygen).
```

Exécuter ensemble ces sous-programmes ASP avec *clingo* ne nous renvoie aucune solution donc cet EFM du réseau Csp2001 n'est pas conforme à la régulation en aérobie.

4.6 Résultats

Le réseau exemple Csp2001 nous a permis de tester la méthode. Nous retrouvons les 80 EFMs du réseau dont 59 sont conformes aux règles de régulation. En particulier, nous avons vérifié que nous obtenons bien les mêmes EFMs que ceux listés dans l'article correspondant [27].

Comme présenté plus haut, il est possible à l'utilisateur de modifier l'environnement avec les atomes `env(E)`, et ainsi obtenir les différents EFMs correspondant à chacun des environnements possibles. Chaque exécution du solveur ASP[LP] sur ce réseau prend 5 secondes ou moins.

La méthode a de plus été testée sur le réseau EcoliCore. D'après l'article de regEFMTool, en ajoutant les règles de régulation, il leur est possible d'obtenir $2.1 \cdot 10^6$ EFMs sur les $226.3 \cdot 10^6$ EFMs après post-processing pour un temps total de 7h. En revanche, leur méthode ne semble pas prendre en compte les contraintes imposées par l'environnement.

En appliquant notre méthode sur ce réseau central d'Escherichia coli dans un environnement minimal composé de Glucose, Pi, H, H2O, NH4 et CO2, nous passons de $226.3 \cdot 10^6$ EFMs à seulement 5007 EFMs respectant la régulation. Pour être exact, 4027 EFMs lorsque l'O2 est dans l'environnement et sinon 980 EFMs ; l'anaérobie changeant drastiquement la régulation et donc les réactions actives.

L'exécution prend au total environ 25 minutes avec notre méthode plus lente (environ 1210s pour les 4027 EFMs en aérobie et 330s pour les 980 EFMs en anaérobie).

En post-processing, on récupére les 3608 EFMs produisant de la biomasse, puis on analyse ceux qui sont pareto-optimaux en terme de ratios dioxygène/biomasse et carbone/biomasse ; il y en a 16. Parmi les pareto-optimaux, on prend ceux qui sont sur l'enveloppe convexe des solutions; il y en a 6.

Cette sélection, inspirée des travaux de Carlson and Srienc, 2004. [31], est présentée en Figure 6. Les courbes sont réalisées avec *matplotlib* sous Python.

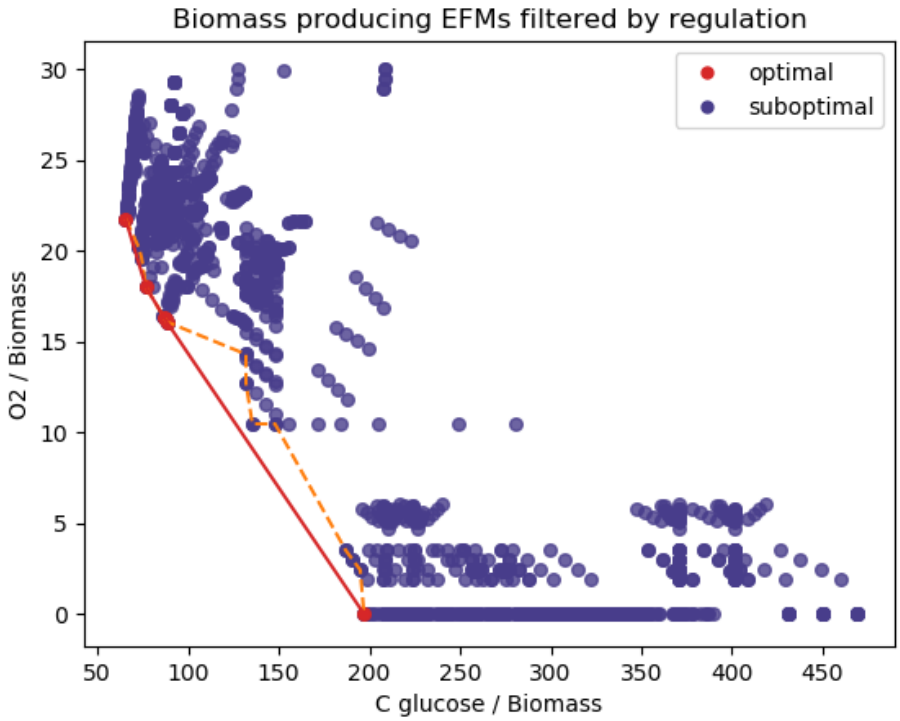

Figure 6: EFMs optimaux en rendements de production de biomasse respectant la régulation

4.7 Perspectives

Malgré l'utilisation de logique booléenne en plus d'un solveur linéaire, l'implémentation du calcul des EFMs en ASP[LP], comme celle en MILP, reste bien plus lente que les méthodes concurrentes à base de calcul matriciel.

Ainsi, même si contrairement à la Double Descritpion nous sommes capables d'énumérer à la volée un nombre prédéfini d'EFMs respectant les contraintes voulues, nous estimons que ce n'est pas suffisant.

Nous cherchons à pouvoir poser des questions sur les EFMs et avoir des réponses en temps polynômial, en particulier le nombre d'EFMs respectant une contrainte, par exemple.

A l'heure actuelle nous nous retrouvons obligés d'énumérer les solutions pour connaître ce nombre et attendre des temps d'exécution dépassant régulièrement 5 minutes sur des réseaux de grande taille.

Plutôt que d'utiliser la programmation logique, nous proposons de nous intéresser à des méthodes basées sur le principe de la compilation de connaissances.

L'idée serait de stocker toutes nos contraintes biologiques dans une structure compacte en mémoire et d'en déduire les EFMs en un temps polynômial.

5 Binary Decision Diagrams

Pour illustrer la compilation de connaissances, nous allons détailler une méthode pionnière, les Diagrammes de Décision Binaire ou BDD. Le principe de compilation de connaissances repose sur l'existence de langages de compilation cibles qui ont des propriétés particulières telles que la possibilité de compter et d'énumérer des solutions en temps polynômial [32].

Par exemple, un BDD est une représentation compacte des solutions d'une formule booléenne. Toute formule booléenne peut être représentée par un BDD. Il est alors possible de vérifier en temps linéaire sur le BDD si une affectation de valeurs aux variables booléennes est une solution de la formule booléenne. La complexité du problème est transférée à l'étape de compilation de la formule booléenne en BDD, qui peut prendre beaucoup de temps.

Soit $X = \{x_1, x_2, \ldots, x_n\}$ un ensemble de variables booléennes. Une interprétation $I \in \mathbb{B}^n$ de X est une affectation de chaque variable booléenne de X soit à la valeur Vrai ($\top$), soit à la valeur Faux ($\bot$).

Soit une formule booléenne $\phi : \mathbb{B}^n \mapsto \mathbb{B}$. ϕ associe une interprétation de X soit à Vrai, soit à Faux. On appelle solutions S de ϕ les interprétations de X qui sont vraies par la formule ϕ.

Un Diagramme de Décision Binaire pour une formule booléenne ϕ est un graphe orienté acyclique (DAG) tel que chaque interprétation possible I de X correspond à un chemin sur le graphe qui mène soit à la valeur Vrai, soit à la valeur Faux.

Il y a donc deux types de nœuds: les nœuds terminaux qui sont les valeurs Vrai et Faux et les nœuds non-terminaux qui correspondent aux variables booléennes.

Ainsi, chaque nœud possède un label : soit Vrai ou Faux pour les deux nœuds terminaux, soit une variable booléenne (on peut avoir plusieurs nœuds pour la même variable booléenne).

Un nœud non terminal possède toujours deux arcs sortants: l'arc `low` qui correspond au cas où la variable prend la valeur Faux, et l'arc `high` pour le cas où la variable prend la valeur Vrai.

On peut de plus définir un ordre sur les variables de X, par exemple généralement les variables sont numérotées de $1, 2, \ldots,$ à n.

Un Diagramme de Décision Binaire est dit Ordonné (OBDD) si à tout moment du graphe les variables respectent un ordre donné $\{x_1 < x_2 < \cdots < x_n\}$, c'est-à-dire que si $i < j$, alors un nœud étiqueté x_i ne peut être un descendant d'un nœud étiqueté x_j.

On décrit maintenant chaque nœud par un triplet $(var, low, high)$: var est le label du nœud, low est le label du nœud fils par l'arc sortant `low` et $high$ est le label du nœud fils par l'arc sortant `high`.

Un BDD est dit Réduit (ROBDD) [33] si:

- chaque nœud est dit unique : il n'existe pas deux nœuds distincts avec le même triplet $(var, low, high)$
- il n'existe pas de nœud $(var, low, high)$ tel que $low = high$ (on arrive à la même variable que var soit Vrai ou Faux, il s'agit d'un test redondant)

Exemple: soit ϕ_1 une formule booléenne sur $\{t1, t2, t3, t4\}$ telle que

$$\phi_1 = (t1 \Rightarrow \neg t2) \wedge (t3 \vee t4)$$

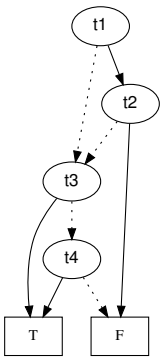

Figure 7: ROBDD pour la formule $\phi_1 = (t1 \Rightarrow \neg t2) \wedge (t3 \vee t4)$

Un exemple de ROBDD pour la formule ϕ_1 est représenté en Figure 7. Les arcs `low` sont représentés en pointillés tandis que les arcs `high` sont représentés en trait plein. Par exemple, pour l'interprétation $\{\bot, \bot, \top, \bot\}$, $t1$ est affecté à Faux donc on suit l'arc en pointillés, puis on affecte $t3$ à Vrai ce qui nous amène en suivant le trait plein au nœud Vrai. Il s'agit d'une solution. Notons que ici la valeur de $t2$ et $t4$ n'ont pas influé sur le résultat.

Pour l'interprétation $\{\bot, \top, \bot, \bot\}$, $t1$ est affecté à Faux, $t3$ est affecté à Faux, $t4$ est affecté à Faux donc on arrive au nœud Faux. Il ne s'agit pas d'une solution. Effectivement, on peut voir que cette solution ne respecte pas l'expression booléenne $(t3 \vee t4)$ dans la formule ϕ_1.

Communément, ce que l'on appelera BDD par la suite désignera en fait des diagrammes qui sont réduits et ordonnés (ROBDDs).

En suivant ce principe, nous allons donc chercher à compiler nos formules booléennes correspondant aux réseaux de régulation en BDDs.

Le calcul des modes de flux n'est pas pour l'instant considéré car les contraintes de stœchiométrie nécessaires sont numériques, or les BDDs ne nous permettent que de regarder les contraintes booléennes.

On cherche donc pour l'instant à tester si un ensemble de réactions actives, issu du calcul des modes de flux, respecte ou non les contraintes de nos réseaux de régulation.

En d'autres termes, nous allons compiler un BDD du réseau de régulation de telle sorte que si notre mode de flux est conforme à la régulation, son support correspondra à un chemin du BDD qui amène au nœud Vrai. Inversement, s'il n'est pas conforme, le chemin amènera au nœud Faux. La formule booléenne considérée sera la conjonction de toutes les règles du réseau.

5.1 Application

Il existe un certain nombre d'opérations possibles sur un BDD. Nous allons détailler celles qui vont nous intéresser: `COUNT` et `RESTRICT` [33].

- Supposons qu'on possède un BDD u pour une expression ϕ. L'opération `COUNT`(u) nous renvoie le nombre de solutions S de ϕ. Cette opération est réalisée en temps linéaire en la taille de u, autrement dit $O(|u|)$.
- Supposons qu'on possède un BDD u pour une expression ϕ, une variable $x \in X$ et une valeur de vérité $b \in \mathbb{B}$. L'opération `RESTRICT`$(u,\, x,\, b)$ nous renvoie alors un BDD $u[x = b]$ pour la sous-expression $\phi[x = b]$ tel que la variable x a été affectée à la valeur b. Cette opération est réalisée en temps linéaire en la taille de u, autrement dit $O(|u|)$.

Appelons `LET`(u, V, B) l'opération qui consiste à obtenir un BDD $u[V = B]$ tel que un ensemble de k variables $V \subset X$ ont été affectées à k valeurs de vérités $B \in \mathbb{B}^k$. Cette opération consiste à effectuer k fois l'opération `RESTRICT`. Elle est donc toujours en temps polynômial $O(|u| \cdot k)$.

Créons maintenant un BDD u_r corrrespondant au réseau de régulation d'un réseau métabolique. Les réactions sont des variables booléennes de notre réseau de régulation et donc de notre BDD.

Pour un ensemble K de k réactions données, il est alors possible de vérifier en temps polynômial s'il existe une solution dans laquelle toutes les réactions sont affectées à Vrai ($\top$). Il suffit de vérifier que l'opération `COUNT`(`LET`$(u_r, K, \top^k)$) renvoie un résultat strictement positif.

Pour chaque mode élémentaire trouvé, il est donc possible de vérifier sur le BDD en temps polynômial s'il respecte la régulation.

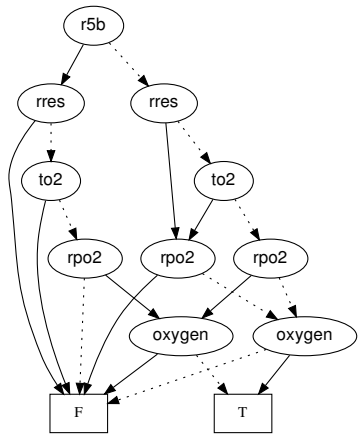

Figure 8: ROBDD compilant des contraintes de régulation de Csp2001

On peut observer sur la Figure 8 un BDD compilant les contraintes issues du Csp2001 présentées précédemment avec ASP (voir sous-section 4.5). Appelons ce BDD u_1.

Il est possible d'observer directement que les réactions $r5b$ et $rres$ ne peuvent pas apparaître ensemble dans un même EFM. En effet, en suivant les traits pleins depuis les nœuds $r5b$ et $rres$, on arrive au nœud Faux.

En revanche, si l'on se tient à ce BDD qui ne représente qu'une petite partie du réseau de régulation complet, tout EFM ne contenant aucune des 3 réactions $\{r5b, rres, to2\}$ est une solution, que ce soit en aérobie ou en anaérobie. Il existe alors 2 solutions possibles pour les variables restantes, 2 chemins alternatifs : $\{rpo2 : \top, oxygen : \bot\}$ et $\{rpo2 : \bot, oxygen : \top\}$.

En d'autres termes, le nombre de solutions renvoyé par $\texttt{COUNT}(\texttt{LET}(u_1, \{r5b, rres\}, \{\top, \top\}))$ est 0. Le nombre de solutions renvoyé par $\texttt{COUNT}(\texttt{LET}(u_1, \{r5b, rres, to2\}, \{\bot, \bot, \bot\}))$ est 2.

5.2 Implémentation

Pour tester les BDDs, nous utilisons la librairie *dd* en Python. Cette librairie interface la librairie standard *cudd* pour les BDD réalisée en langage C, en utilisant Cython. Le code Python exécuté appelle donc du code C, ce qui permet des optimisations majeures en temps de calcul.

```
from dd import BDD
bdd = BDD() # Nouveau Manager de BDD
bdd.declare(variables) # Déclaration des variables
function = bdd.add_expr(expression) # Déclaration de la formule booléenne
restricted_function = bdd.let({'R1': True}, function) # LET
nb_sols = restricted_function.count() # COUNT
```

Nous allons désormais détailler l'intégration du BDD dans le solveur ASP. L'idée consistera à faire un appel aux fonctions `LET` et `COUNT` sur notre BDD lorsque ASP décide quelles réactions vont être actives pour éliminer prématurement les ensembles de réactions incompatibles.

5.3 Intégration dans clingo

En Figure 9, nous proposons une illustration générale de l'intégration dans clingo des BDD avec le propagateur de théorie, prenant pour example la formule Booléenne de la Figure 7. Dans le contexte ci-dessous, chaque flèche standard est une affectation à vrai ou "propagate", et "backtrack" fait référence à un évènement "undo". Enfin, "nogood" fait référence à un ajout d'une clause négative sur les littéraux restants $t2$ et $r1$ ensemble, ajout qui, comme l'appel au BDD, est effectué à l'intérieur de l'implémentation de la fonction "propagate".

Afin de tester les avantages de la compilation de connaissances, nous avons décidé d'intégrer les BDDs dans notre implémentation en clingo[LP]. Pour cela, nous pouvons utiliser l'interface des *propagateurs de théorie* de clingo. Un propagateur de théorie est une structure externe à clingo capable d'inspecter chaque affectation et ajouter des nogoods pendant la résolution ASP.

L'interface de propagateur de théorie est disponible depuis la version 5 du solveur clingo [28], et peut être implémentée en Python. Il s'agit de la même interface qui est utilisée pour intégrer la programmation linéaire dans clingo[LP] (voir Figure 10).

Nous avons d'abord tenté d'utiliser deux propagateurs, un pour les programmes LP cplex et un pour les BDD, ce qui est possible avec clingo 5. En revanche, nous n'étions pas satisfaits du résultat, car le propagateur BDD semblait être appelé après le propagateur LP.

Nous avons donc finalement modifié le code de clingo[LP] afin d'y intégrer le propagateur pour le BDD directement avant tout appel au solveur linéaire, et ainsi couper les solutions incompatibles avec la régulation le plus tôt possible.

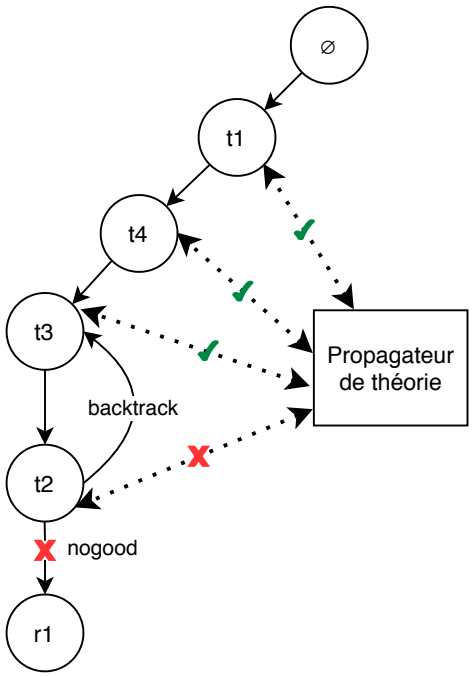

Figure 9: Illustration générale de l'intégration d'un BDD avec le propagateur de théorie, prenant la formule Booléenne $\phi_1 = (t1 \Rightarrow \neg t2) \wedge (t3 \vee t4)$. A chaque propagation de littéral, un appel est réalisé au BDD, utilisant les méthodes `LET` et `COUNT` afin de vérifier en temps polynômial si l'affectation respecte ou non la formule.

L'interface de propagateur définit des méthodes suivantes :

- "init" : l'initialisation du propagateur, on doit y définir les atomes à observer, on obtient alors les littéraux correspondants.
- "propagate" : cette méthode est appelée chaque fois qu'un littéral observé est affecté à Vrai, on a de plus contrôle sur la résolution et on peut ajouter des clauses ou des nogoods (clauses de littéraux négatifs) selon la valeur des littéraux observés.
- "undo" : cette méthode est appelée chaque fois que l'on "annule" un précédent "propagate", c'est à dire qu'on assigne à Faux un littéral qu'on avait précédemment affecté à Vrai. Cette méthode ne donne pas de contrôle de la résolution.
- "decide" : cette méthode est appelée lorsque le solveur doit décider quels littéraux assigner à Vrai, elle peut être utilisée pour implémenter des heuristiques.

Pour clarifier, si le littéral correspondant à l'atome observé est vrai, alors l'atome observé est présent dans la solution, et inversement.

Dans notre cas, les atomes à observer sont les atomes `support(R)`. Chaque fois qu'un littéral est affecté à Vrai ("propagate"), on ajoute la réaction `R` correspondante dans l'ensemble, et chaque fois que cette décision est annulée ("undo"), on le retire de l'ensemble.

A chaque "propagate", l'ensemble de réactions stocké correspond donc aux réactions présentes dans la solution actuellement calculée. On peut donc faire des requêtes à notre BDD pour vérifier si ces réactions peuvent apparaître ensemble dans une même solution. Le BDD nous répond en temps polynômial, et s'il n'y a pas de solution possible contenant ces réactions, on ajoute alors un nogood sur les littéraux.

Par exemple, avec notre implémentation de BDD en *dd* Python, nous faisons un simple appel à la méthode *count* afin de compter le nombre de chemins possibles en assignant toutes les réactions à Vrai avec *let*. S'il n'y a aucun chemin possible ($count = 0$), alors l'EFM correspondant n'est pas acceptable (voir sous-section 5.1): on ajoute un nogood.

On souhaite que le BDD nous aide au niveau des clauses booléennes disjonctives ou conjonctives de plus de deux litéraux. Les règles de régulation de nos réseaux de type "P $\implies$ Q" sont plutôt bien gérées par ASP. De ce fait, l'utilisation du BDD qui est une interface extérieure pourrait devenir une perte de temps. Lorsque nous avons suffisament de clauses, nous pensons que le BDD peut être plus rapide que ASP dans la gestion des contraintes.

5.4 Résultats

Nous avons compilé en BDDs nos deux réseaux de régulation étudiés, Csp2001 et EcoliCore, puis nous avons compilé deux BDDs des EFMs solutions sur le réseau EcoliCore.

Les caractéristiques de chaque BDD ainsi que leur temps de construction sont présentés en Tableau 2.

Réseau	Csp2001	EcoliCore	EcoliCore	
Type	Régulation		Solutions	
Environnement	Encodés dans le BDD		Minimal	Enrichi
Nombre total d'EFMs possibles	jusqu'à 80	$226.3 \cdot 10^6$	4027	40551
Nombre de nœuds	438	5576	6752	18521
Nombre de variables	29	252	154	154
Nombre de solutions	5250	$1.1 \cdot 10^{37}$	4027	40551
Temps de construction	699 µs	17.3 ms	2.17 s	32.7 s

Tableau 2: Caractéristiques de chaque BDD

Comme pour ASP, le réseau de régulation de Csp2001 nous a permis de tester la méthode. Nous retrouvons bien pour chaque environnement différent les mêmes résultats que ceux présentés dans l'article de Covert et Palsson.

Les environnements sont modifiés en faisant un appel à *let* pour affecter en conséquence les valeurs des variables correspondant aux métabolites externes. Les métabolites externes qui sont dans l'environnement sont affectés à Vrai, les autres métabolites externes à Faux.

On a donc un unique BDD compilant tous les environnements. Selon l'environnement on récupère le sous-BDD correspondant.

Pour le réseau EcoliCore, nous avons testé les deux environnements suivants avec le BDD de régulation : l'environnement minimal {Glucose, Pi, H, H2O, CO2, NH4, O2} et l'environnement enrichi {Glucose, Pi, H, H2O, CO2, NH4, O2, Lactose, Pyruvate}.

En plus des réseaux de régulations, nous avons compilé deux BDDs des solutions du réseau EcoliCore. En effet, les EFMs, ou du moins leurs supports, peuvent s'exprimer par une clause conjonctive de littéraux positifs.

Le premier BDD de solutions correspond aux EFMs de l'environnement minimal tandis que le second correspond aux EFMs de l'environnement enrichi. Si l'on donne ces solutions comme contraintes à clingo[LP], le solveur pourra alors nous recalculer les mêmes EFMs.

5.5 Comparaison

Nous comparons maintenant l'efficacité des contraintes compilées en BDD par rapport aux mêmes contraintes en ASP évaluées directement pendant la résolution. Pour cela, nous prenons les règles de calcul des EFMs en ASP[LP], le réseau EcoliCore et les contraintes de régulation traduites soit en ASP soit en BDD. Dans le cas des contraintes BDD, nous utilisons notre version modifiée de clingo[LP] (voir sous-section 5.3) qui effectue des appels aux BDD via la librairie *dd*. Des résultats moyennés sur 5 éxécutions pour chaque condition sont présentés en Tableau 3.

EcoliCore		Environnement minimal	Environnement enrichi
Contraintes BDD	Nombre d'EFMs	4027	40651
	Temps (s)	1211.48	8973.29
	LPs résolus	51414	252777.4
	Nogoods ajoutés	22	20
Contraintes ASP	Nombre d'EFMs	4027	40325.6
	Temps (s)	1218.81	8914.80
	LPs résolus	43804	251836.4

Tableau 3: Comparaison de la performance des méthodes avec les contraintes de régulation

On n'observe pas de différence significative pour le temps de calcul entre les contraintes ASP ou BDD, quelque soit l'environnement proposé. Le nombre de LPs résolus par clingo[LP] est cependant légèrement plus faible pour les contraintes ASP que pour les contraintes BDD.

Comme expliqué précédemment, cela est dû aux règles de régulation de type "P $\implies$ Q", qui sont très bien gérées à la fois par clingo et par le BDD. Notons que le nombre de nogoods ajoutés est très faible, ainsi avoir compilé le réseau en BDD n'apporte pas ici d'aide précieuse.

En revanche, sur nos BDDs des solutions (voir sous-section 5.4) – avec plus de nœuds et qui ont pris significativement plus de temps à construire – nous allons pouvoir observer des différences majeures. En effet, les solutions sont encodées sous forme de clauses conjonctives de nombreux littéraux positifs.

Nous présentons en Tableau 4 les résultats d'une seule exécution pour chaque condition, avec les solutions comme contraintes. On observe un nombre conséquent de nogoods ajoutés par le BDD pour chaque environnement.

De plus, face aux clauses conjonctives, les contraintes ASP font appel au solveur LP près de 10 fois plus que les contraintes BDD. Le temps de calcul est lui aussi significativement plus élevé pour les contraintes ASP.

EcoliCore		Environnement minimal	Environnement enrichi
Contraintes BDD	Nombre d'EFMs	4027	40088
	Temps (s)	611	4239
	LPs résolus	37416	233179
	Nogoods ajoutés	652	7752
Contraintes ASP	Nombre d'EFMs	4027	40551
	Temps (s)	1746	27599
	LPs résolus	300796	2239529

Tableau 4: Comparaison des méthodes avec les solutions en tant que contraintes

Bien que ce test nécessite en premier lieu la connaissance des solutions, il montre les avantages possibles apportés par la compilation des contraintes en BDD par rapport à une résolution entièrement en ASP.

En effet, ces résultats laissent à penser que nous aurions observé un gain de temps par rapport à l'utilisation de ASP si nous avions compilé en BDD un réseau de régulation transcriptionnelle avec plus de règles de types clauses conjonctives.

6 Discussion

Au cours de ce stage, nous avons pu développer l'intégration des réseaux de régulation au calcul des EFMs avec Answer Set Progamming et avec les Binary Decision Diagrams.

Nos travaux ont montré des résultats équivalents pour les deux méthodes sur les réseaux de régulation, aussi bien en calculant les contraintes dans une phase en ligne avec ASP que dans une phase hors ligne de compilation avec les BDDs. De plus, pour les contraintes de types clauses construites à partir des solutions, le BDD est significativement plus performant que ASP pour retrouver les EFMs correspondants.

Nous avons donc fait une première application de la compilation de connaissances au calcul des EFMs qui nous donne des résultats prometteurs.

Afin de continuer dans cette voie, nous souhaiterions appliquer les principes de la compilation de connaissances directement au calcul des valeurs de flux, le cœur du problème. En particulier, nous souhaitons compiler aussi des contraintes de types linéaires et numériques.

Cela nécessitera alors d'autres techniques de compilation de connaissances plus avancées que les BDDs qui ne sont prévus que pour les contraintes booléennes.

Dans un autre temps, nous souhaitons approfondir le problème des contraintes positives posé par le calcul des EFMs et essayer d'y trouver une solution afin d'intégrer un plus grand nombre de contraintes biologiques.

Quant aux réseaux de régulation transcriptionnelle, nous avons confirmé qu'ils permettent de filtrer un très grand nombre d'EFMs lors de leur énumération. En revanche, leur construction et leur vérification reste un enjeu biologique majeur et une tâche difficile [34].

Certaines contraintes de régulation booléennes s'avèrent en réalité trop strictes et éliminent certains EFMs qui seraient empruntés par la cellule dans des conditions expérimentales. En particulier, nous discutons en ce moment avec Pr. Ross P. Carlson d'une contrainte de régulation éliminant toute production du formate en aérobie. Cette contrainte est incohérente avec le constat que *Escherichia coli* produit du formate en présence de faibles quantités d'oxygène.

Pour répondre à cette problématique, un plus grand panel de données transcriptomiques, telles que des données numériques ou probabilistes, pourra être étudié par la suite.

Nous souhaitons également explorer d'autres modèles métaboliques, tel que celui de la bactérie *Pseudomonas aeruginosa*, dans le cadre d'une collaboration avec Pr. Ross P. Carlson.

7 Conclusion

En conclusion, ce stage m'a appris de nombreuses connaissances en biologie et en informatique. Ce sujet m'a apporté beaucoup d'intérêt et je suis particulièrement motivé pour l'approfondir autant dans l'aspect informatique et méthodes de compilation de connaissances que dans l'aspect biologie et traitement et modélisation de l'information biologique.

Je souhaite donc continuer à travailler sur ce sujet de bio-informatique et sur ces deux aspects dans le cadre d'un doctorat au Laboratoire de Recherche en Informatique.

Je tiens à remercier fortement Mme Sabine Peres pour m'avoir spécialement proposé ce sujet de stage ouvert vers un sujet doctoral ainsi que pour son accompagnement continuel tout au long du stage.

Je remercie également vivement Pr. Laurent Simon pour sa contribution au sujet, son soutien et son expertise informatique au cours du stage et de nos réunions.

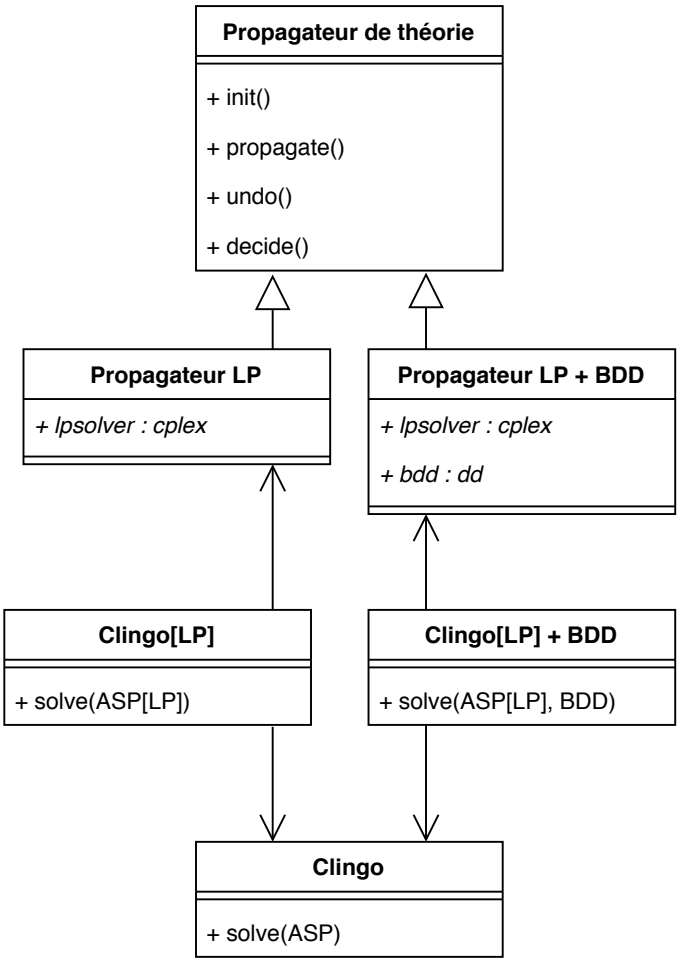

Figure 10: Schéma UML simplifié des outils utilisés. A gauche, calcul des EFMs avec ASP[LP] à partir de l'interface clingo[LP] existante, à droite, calcul des EFMs avec ASP[LP] + BDD à partir de notre nouvelle interface.

Glossaire

ASP Answer Set Programming, ou programmation par ensembles réponses.
Langage de programmation logique.

ASP[LP] Abbréviation de ASP couplé à la programmation linéaire. Syntaxe de clingo[LP].

BDD Binary Decision Diagram, ou Diagramme de décision binaire. Graphe qui représente toutes les affectations possibles de variables booléennes d'une formule booléenne. Désigne par abus de langage un ROBDD : un diagramme de décision binaire réduit et ordonné.

clause Conjonction ou disjonction de littéraux (ex: $a \vee \neg b \vee c$).

clingo Solveur de programmation logique ASP.

clingo[LP] Extension du solveur ASP clingo, interface avec la programmation linéaire.

count Opération permettant de compter le nombre de solutions dans un BDD. En temps polynômial.

cplex Solveur de programmation linéaire.

dd Librairie Python implémentant les BDD.

EFM Elementary Flux modes, ou modes élémentaires de flux.
Vecteurs élémentaires de flux métaboliques à l'état stationnaire.

FBA Flux Balance Analysis. Méthode d'analyse de flux métaboliques à l'état stationnaire à base de programmation linéaire.

let, restrict Opération permettant d'affecter des variables booléennes dans un BDD et obtenir le sous-BDD correspondant. En temps polynômial.

littéral Un littéral est un atome x (littéral positif) ou la négation d'un atome $\neg x$ (littéral négatif).

LP Linear Programming, ou programmation linéaire. Méthode de résolution numérique en temps polynômial.

MILP Mixed Integer Linear Programming, programmation linéaire mixte. Peut contenir des variables entières ou booléennes, ce qui rend le problème dans la classe de complexité NP-difficile.

nogood Clause booléenne négative. Ajoutée lors de la résolution afin de bloquer une solution incompatible avec la régulation ou déjà trouvée.

propagateur Propagateur de théorie. Interface proposée par clingo.
Structure externe à clingo capable d'inspecter chaque affectation et ajouter des nogoods pendant la résolution ASP.

SAT Problème de la satisfiabilité booléenne. Méthode de résolution logique. Problème NP-complet connu.

TRN Transcriptional Regulation Network, réseau de régulation transcriptionnel.

Réferences

[1] J. Orth, R. Fleming, and B. O. Palsson. Reconstruction and use of microbial metabolic networks: the core escherichia coli metabolic model as an educational guide. *EcoSal Plus*, 2010.

[2] A. Varma and B. O. Palsson. Stoichiometric flux balance models quantitatively predict growth and metabolic by-product secretion in wild-type escherichia coli w3110. *Applied and environmental microbiology*, 60(10):3724–3731, Oct 1994. 7986045[pmid].

[3] Stefan Schuster and Claus Hilgetag. On elementary flux modes in biochemical reaction systems at steady state. *J. Biol. Syst*, 2:165–182, 06 1994.

[4] Komei Fukuda and Alain Prodon. Double description method revisited. In *Combinatorics and Computer Science*, 1995.

[5] Zachary A. King, Justin Lu, Andreas Dräger, Philip Miller, Stephen Federowicz, Joshua A. Lerman, Ali Ebrahim, Bernhard O. Palsson, and Nathan E. Lewis. Bigg models: A platform for integrating, standardizing and sharing genome-scale models. *Nucleic acids research*, 44(D1):D515–D522, Jan 2016.

[6] T Pfeiffer, I Sanchez-Valdenebro, J C Nuno, F Montero, and S Schuster. METATOOL: for studying metabolic networks. *Bioinformatics*, 15(3):251–257, 03 1999.

[7] Marco Terzer and Jörg Stelling. Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, 24(19):2229–2235, 08 2008.

[8] Christian Jungreuthmayer, David E. Ruckerbauer, and Jürgen Zanghellini. regefmtool: Speeding up elementary flux mode calculation using transcriptional regulatory rules in the form of three-state logic. *Biosystems*, 113(1):37 – 39, 2013.

[9] Sabine Peres, Mario Jolicœur, Cécile Moulin, Philippe Dague, and Stefan Schuster. How important is thermodynamics for identifying elementary flux modes? *PLOS ONE*, 12(2):1–20, 02 2017.

[10] Vicente Acuña, Flavio Chierichetti, Vincent Lacroix, Alberto Marchetti-Spaccamela, Marie-France Sagot, and Leen Stougie. Modes and cuts in metabolic networks: Complexity and algorithms. *Biosystems*, 95(1):51 – 60, 2009.

[11] Vicente Acuña, Alberto Marchetti-Spaccamela, Marie-France Sagot, and Leen Stougie. A note on the complexity of finding and enumerating elementary modes. *Biosystems*, 99(3):210 – 214, 2010.

[12] Luis F. de Figueiredo, Adam Podhorski, Angel Rubio, Christoph Kaleta, John E. Beasley, Stefan Schuster, and Francisco J. Planes. Computing the shortest elementary flux modes in genome-scale metabolic networks. *Bioinformatics*, 25(23):3158–3165, 09 2009.

[13] Jon Pey and Francisco J. Planes. Direct calculation of elementary flux modes satisfying several biological constraints in genome-scale metabolic networks. *Bioinformatics*, 30(15):2197–2203, 04 2014.

[14] Vítor Vieira and Miguel Rocha. CoBAMP: a Python framework for metabolic pathway analysis in constraint-based models. *Bioinformatics*, 35(24):5361–5362, 07 2019.

[15] A. Rezola, L. F. de Figueiredo, M. Brock, J. Pey, A. Podhorski, C. Wittmann, S. Schuster, A. Bockmayr, and F. J. Planes. Exploring metabolic pathways in genome-scale networks via generating flux modes. *Bioinformatics*, 27(4):534–540, 12 2010.

[16] Axel von Kamp and Steffen Klamt. Enumeration of smallest intervention strategies in genome-scale metabolic networks. *PLOS Computational Biology*, 10(1):1–13, 01 2014.

[17] S. Peres, M. Morterol, and L. Simon. Sat-based metabolics pathways analysis without compilation. In J.O. Dada P. Mendes and K. Smallbone, editors, *Lecture Note in Bioinformatics*, volume 8859, pages 20–31. Springer International Publishing, 2014.

[18] M. Morterol, P. Dague, S. Peres, and L. Simon. Minimality of metabolic flux modes under boolean regulation constraints. In *Proceedings of the 12th International Workshop on Constraint-Based Methods for Bioinformatics (WCB'16)*, pages 65–81, 2016.

[19] Jeffrey D. Orth, Ines Thiele, and Bernhard Ø Palsson. What is flux balance analysis? *Nature Biotechnology*, 28(3):245–248, 2010.

[20] Bernhard Palsson. The challenges of in silico biology. *Nature Biotechnology*, 18(11):1147–1150, November 2000.

[21] Markus W. Covert, Christophe H. Schilling, and Bernhard O. Palsson. Regulation of gene expression in flux balance models of metabolism. *Journal of Theoretical Biology*, 213(1):73–88, November 2001.

[22] Markus W. Covert and Bernhard Ø. Palsson. Transcriptional regulation in constraints-based metabolic models ofEscherichia coli. *Journal of Biological Chemistry*, 277(31):28058–28064, May 2002.

[23] Markus W. Covert, Eric M. Knight, Jennifer L. Reed, Markus J. Herrgard, and Bernhard O. Palsson. Integrating high-throughput and computational data elucidates bacterial networks. *Nature*, 429(6987):92–96, May 2004.

[24] Rahuman S Malik-Sheriff, Mihai Glont, Tung V N Nguyen, Krishna Tiwari, Matthew G Roberts, Ashley Xavier, Manh T Vu, Jinghao Men, Matthieu Maire, Sarubini Kananathan, Emma L Fairbanks, Johannes P Meyer, Chinmay Arankalle, Thawfeek M Varusai, Vincent Knight-Schrijver, Lu Li, Corina Duenas-Roca, Gaurhari Dass, Sarah M Keating, Young M Park, Nicola Buso, Nicolas Rodriguez, Michael Hucka, and Henning Hermjakob. BioModels—15 years of sharing computational models in life science. *Nucleic Acids Research*, November 2019.

[25] 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 Novère, 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, J. Wang, and the rest of the SBML Forum. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 03 2003.

[26] Brett G. Olivier and Frank T. Bergmann. Sbml level 3 package: Flux balance constraints version 2. *Journal of Integrative Bioinformatics*, 15(1):1–39, 3 2018.

[27] Markus W. Covert and Bernhard O. Palsson. Constraints-based models: Regulation of gene expression reduces the steady-state solution space. *Journal of Theoretical Biology*, 221(3):309–325, April 2003.

[28] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory Solving Made Easy with Clingo 5. In Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos, editors, *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*, volume 52 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[29] Tomi Janhunen, Roland Kaminski, Max Ostrowski, Torsten Schaub, Sebastian Schellhorn, and Philipp Wanko. Clingo goes linear constraints over reals and integers. *CoRR*, abs/1707.04053, 2017.

[30] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, Sven Thiele, and Philipp Wanko. *Potassco User Guide*.

[31] R. Carlson and F. Srienc. Fundamental Escherichia coli biochemical pathways for biomass and energy production: identification of reactions. *Biotechnol. Bioeng.*, 85(1):1–19, Jan 2004.

[32] A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.

[33] Henrik Reif Andersen. An introduction to binary decision diagrams. *Lecture notes for Efficient Algorithms and Programs*, Fall 1999.

[34] Saheed Imam, Sascha Schäuble, Aaron N. Brooks, Nitin S. Baliga, and Nathan D. Price. Data-driven integration of genome-scale regulatory and metabolic network models. *Frontiers in Microbiology*, 6, May 2015.