Chapitre 1

Enrichissement des modèles qualitatifs : Modèle de logique linéaire temporel et processus de sélection de communauté à façon [*ML* – NE PAS LIRE]

1.1 Introduction

Ce chapitre 1 symbolise le fruit terminal de cette thèse prenant en considération les limites et les avantages du modèle numérique du chapitre **??** et du modèle par raisonnement du chapitre **??**. Rappelons que l'objectif final est de trouver des pistes permettant d'analyser des communautés complexes tout en conservant, précision et explicabilité du résultat. Nous avons vu que le modèle discret (chapitre **??**) a permis l'étude de communautés complexes en redirigeant l'analyse sur les potentiels de coopération et de compétition de communautés bactériennes. Le point critique majeur de notre méthode est l'absence de fonction objective à optimiser, permettant d'affiner les résultats. Dans la littérature, il existe un outil, basé sur le même principe par raisonnement que nous, qui permet d'optimiser différentes fonctions objectives : taille de la communauté et nombre des échanges métaboliques (**?**). Ces optimisations permettent de sélectionner des communautés candidates selon les deux critères précédents. Appliquer et étendre ces choix d'optimisation à notre modèle par raisonnement permettrait de (i) réduire les sorties d'un modèle par raisonnement, (ii) enrichir le modèle en apportant de la souplesse d'utilisation et (iii) laisser les utilisateurs et les utilisatrices définir les concepts de communautés à fort potentiel de coopération ou de compétition. Le second point de critique majeur de notre modèle discret est l'absence de la prise en compte de la temporalité. Au cours du chapitre **??**, nous avons vu que la concentration d'un métabolite au cours du temps n'évolue pas linéairement, mais passe par des phases qualitatives différentes : augmentation (forte ou faible), diminution (forte ou faible) ou encore, absence de changement. Par exemple, sur la Figure **??** l'évolution discrète de la concentration de lactate est : forte augmentation, absence d'évolution puis diminution faible. De plus, lors de l'exploration du modèle de communauté, nous avons vu que le moment où un métabolite est consommé change l'hypothèse d'interaction. Par exemple, le lactose est d'abord consommé par *L. lactis* puis par *P. freudenreichii*, ne permettant pas de définir une hypothèse de compétition entre ces deux espèces. Enfin, nous avons montré la plus-value d'ajouter des contraintes biologiques sur la précision du modèle. Par exemple, le métabolisme des bactéries lactiques reste actif pendant la phase stationnaire permettant de produire de l'acide lactique et ainsi, expliquer les données biologiques. Dans la littérature, d'ajout de la temporalité dans un modèle par raisonnement il existe un formalisme logique introduit pas les travaux de Cabalar (**?**) : la logique linéaire temporelle (LTL). Ce formalisme logique utilise le solveur *telingo* qui combine l'ASP avec des contraintes temporelles. En associant ce formalisme avec le concept IA du raisonnement sur les actions et les changements (RAC), permettant de connaître la cause d'un changement d'un système, Cabalar permet d'expliquer à partir d'inférence de règles logiques temporelles, les observations. En revanche, il existe à notre connaissance, aucune application du un système biologique. En somme, une inférence de la temporalité et de contraintes biologiques dans le modèle de communauté aurait pour conséquence de réduire le nombre de solutions possibles en sélectionnant les modèles qui satisfassent ces contraintes et d'affiner les résultats du modèle logique existant.

Ainsi, nous verrons au cours de ce chapitre comment l'apport d'une souplesse et d'une facilité d'utilisation permet d'enrichir les possibilités de sélection d'une communauté candidate. Puis dans un second temps, nous verrons comment l'ajout de la temporalité en se basant sur le RAC permet d'apporter des connaissances temporelles supplémentaires par rapport au modèle dynamique du chapitre **??**.

1.2 Critères d'optimisation à partir de contraintes biologiques *a priori*

L'une des questions que nous nous sommes posés est la définition de la "meilleure" communauté pour répondre à un objectif. Les travaux de Frioux et ses collègues (**?**) propose de minimiser la taille de la communauté ainsi que les échanges métaboliques conduisant à la production de métabolites cibles. Or, au cours de discussions avec des biologistes, nous avons identifié que les choix de minimisation faits dans MISCOTO ne garantissaient pas l'obtention de la meilleure communauté. Cependant, enlever une optimisation ne permettrait pas de résoudre la combinatoire importante conduit par les interactions bactériennes. Une autre question sous-jacente est donc celle du choix de la fonction objective à optimiser. En complément de celles utilisées dans MISCOTO, nous avons calculés des potentiels de coopération et de compétition ainsi qu'étendu le vocabulaire existant avec les notions de substrats polyopsonistes et du nombre de contributeurs dans un échange. Ces options supplémentaires enrichissent le choix d'optimisation, contribuant à la définition d'une communauté à fort potentiel de coopération ou de compétition. Parmi l'ensemble de ces éléments (nombre d'échanges, substrats polyopsonistes etc.), il existe à notre connaissance, pas de consensus sur la description de la communauté avec de fortes propriétés coopératives/compétitives. Nous avons ainsi développé un prototype laissant l'utilisateur ou l'utilisatrice la possibilité de sélectionner des communautés selon un ensemble de contraintes pertinents pour leur domaine d'application. Ces contraintes se catégorisent en deux parties. La première consiste à filtrer la communauté selon la composition taxonomique de cette dernière. La seconde, consiste à optimiser des concepts du vocabulaire issu de la modélisation discrète des communautés sélectionnés préalablement. Afin de garantir une facilité d'utilisation, nous nous sommes basés sur le format YAML pour définir les contraintes. Dans la suite du chapitre nous allons vous présenter les nouvelles contraintes, comment le prototype s'articule et montrer sa souplesse et facilité d'utilisation.

1.2.1 Méthodologie

Lors du l'élaboration de ce prototype, nous avons consacré du temps à sa facilité d'utilisation. Nous allons donc vous présenter les contraintes supplémentaires, la base de fait créé à partir de ces contraintes et comment la ou le biologiste peut les utiliser. Dans un second temps, l'implémentation de ces nouvelles contraintes sous la représentation mathématique et sous le formalisme ASP.

Contraintes de sélection et interface utilisateur

Puisque les contraintes de modélisation discrète ont été défini dans le chapitre **??**, nous allons uniquement décrire les nouvelles contraintes liées à la composition de la communauté. Afin de faciliter la lisibilité et l'accessibilité, nous avons utilisés le YAML comme une interface claire servant de pont entre les biologistes et l'outil.

Contrainte sur la taille des communautés. L'utilisateur ou l'utilisatrice peut choisir une sélection plus ou moins stringente des tailles de communautés qu'il ou elle souhaite. Ce prototype propose de filtrer les communautés en sélectionnant une taille fixe, c'est à dire, parmi l'ensemble des combinaisons possibles, seules les communautés qui ont exactement cette taille seront sélectionnées. Ou une taille bornée par des valeur minimales

et maximales, seules les communautés dont leurs tailles sont comprises dans l'intervalle seront sélectionnées.

Par exemple, un ou une biologiste aimerait obtenir l'ensemble des communautés qui ont une taille comprise entre **6** et **8**. Le fichier YAML d'entrée se structurerait ainsi :

```yaml
# Size constraints
SIZE:
  size:
  lower bound: 6
  upper bound: 8
```

À partir de la lecture de ce fichier YAML, l'outil génère la liste de fait ASP, permettant de représenter les contraintes sous forme de connaissance :

```
lower_bound_size("6").
upper_bound_size("8").
```

Dans le cas où une taille fixe est demandée par les biologistes, seul le mot clé `size` est concerné. Par exemple, la demande "je ne veux que des communautés de taille **4**" se traduit par :

```yaml
# Size constraints
SIZE:
  size: 4
  lower bound:
  upper bound:
```

Avec la **liste de faits ASP** suivante :

```
lower_bound_size("4").
upper_bound_size("4").
```

Dans le cas où une taille fixe et une taille bornée sont renseignées, l'outil considère seulement la taille fixe.

Contraintes sur la composition taxonomique bactérienne En plus de la taille, la ou le biologiste peut décider de la composition taxonomique, au grain de la souche, des communautés sélectionnées. Nous avons supposé que l'utilisateur ou l'utilisatrice connait un sous ensembles des souches bactériennes. Cette contrainte répond à des demandes biologiques telles que : "Au sein de ma communauté, je souhaite que les souches **XXX** et **YYL** soient présentes car elles permettent la libération de composés d'arômes d'intérêt, mais je ne veux surtout pas la souche **HHJ** et **KM** car elles sont en compétition avec les précédentes souches". Remarque, les raisons biologiques justifiants les intérêts pour ces souches ne sont pas prises en compte au sein de l'outil. Cette contrainte de présence absolue et d'interdiction se traduit respectivement dans le YAML part les clés d'entrée suivante : `always` et `forbid` :

```yaml
# Bacterial composition
constraints
BACTERIAL COMPOSITION:
  always:
    - XXX
    - YYL
  forbid:
```

```
8       - KM
9       - HHJ
```

La liste de faits ASP à partir de ces contraintes sont les suivants :

```
1 always("XXX").
2 always("YYL").
3 forbid("KM").
4 forbid("HHJ").
```

Nous avons également développé des contraintes supplémentaires sur la composition taxonomique de la communauté : associations interdites et obligatoires, nommées respectivement `forbidden asso` et `mandatory asso`. Une demande biologique illustrant ces propos serait : "S'il existe une communauté où une des souches **XV**, **FV** ou **JH** est présente, alors l'ensemble de ces souches doit être présent également. En revanche, tant que l'ensemble des souches **FV**, **PP** et **FT** n'est pas présent ensemble dans la même communauté, mais que des sous-ensembles le sont, je considère la communauté valide." Ces contraintes diffèrent de `always` et de `forbid` du fait des notions d'ensemble, et l'ensemble nul est considéré comme valide du point de vue méthodologie : par exemple, une communauté où ni **XV**, **FV** ou ni **JH** n'est présente, est valide. Ces associations permettent également de définir des différents groupes de souches interdites ou obligatoires. Par exemple, en complétant la demande : "En plus de ces associations, si une des souches **AA** et **ZS** est présente alors l'ensemble doit être présent également. Cependant, l'ensemble **PO** et **ML** ne doit pas être ensemble". Grâce au YAML, ces contraintes sont formalisables comme suit :

```
1  # Bacterial composition
2  constraints
3  BACTERIAL COMPOSITION:
4  forbidden asso:
5      -
6        - FV
7        - PP
8        - FT
9      -
10       - PO
11       - ML
12
13   mandatory asso:
14     -
15       - XV
16       - FV
17       - JH
18     -
19       - AA
20       - ZS
```

Les différents groupes d'ensemble interdits (resp. obligatoires) sont symbolisé par _ de la ligne 5 et 9 (ligne 14 et 18). Ces groupes sont également visibles dans liste de faits ASP générée à partir de ces contraintes et se traduisent par le numéro en amont de la souche et sont définis automatiquement à la lecture du fichier YAML :

```
forbidden_asso("0","FV").
forbidden_asso("0","PP").
forbidden_asso("0","FT").
forbidden_asso("1","PO").
forbidden_asso("1","ML").
```

```
mandatory_asso("0","XV").
mandatory_asso("0","FV").
mandatory_asso("0","JH").
mandatory_asso("1","AA").
mandatory_asso("1","ZS").
```

Flexibilité des fonctions objectives. Laisser le choix à l'utilisateur ou l'utilisatrice d'optimiser une ou plusieurs fonctions objectives correspond à l'enrichissement majeur du modèle logique. Comme nous avons vu précédemment, le choix de la fonction objective est important et sa modularité auprès de l'utilisateur ou l'utilisatrice l'est tout autant. Notre prototype d'outil prend actuellement les choix d'optimisation suivant : maximiser ou minimiser les échanges métaboliques, les métabolites cibles à produire, le scope métabolique et le nombre de substrats polyopsonistes. À partir du fichier d'initialisation YAML, les contraintes de maximisation ou de minimisation sont directement écrits ainsi :

```
# Optimisation
OPTIMISATION:
  - max
  - max

# Objectif
OBJ:
  - target
  - exchanged
```

Les lignes 3 et 4 indiquent le choix de l'optimisation que l'utilisateur ou l'utilisatrice souhaite faire, et les lignes 8 et 9 sont les objectifs possibles décrits plus haut. Dans cet exemple, le ou la biologiste veut en premier maximiser la production d'un ensemble de cibles, puis, dans un second temps, maximiser les métabolites échangeables. L'implémentation de cette contrainte se traduit de la façon suivante en ASP :

```
#maximize {1@1,M:target(M), pscope(M,_,_) }; 0@1}.
#maximize {1@2,S:exchanged(S); 0@2}.
```

Les priorités d'optimisation sont traduites par `1@1` et `1@2`, dans lequel la ligne 1 maximise le nombre de métabolites cibles présents dans le scope communautaire. [*ML – Adapter selon le nouveau formalisme développé dans CCMC ? ou mettre des noms plus communs pour aider la compréhension ?*]

Implémentation des nouvelles contraintes

Nous venons de voir comment nous avons construit notre base de faits utiles pour formaliser les règles et les contraintes, conduisant à la sélection des communautés. Dans la suite de ce chapitre, nous allons formaliser les différentes règles et contraintes en ASP et expliciter formalisme mathématique.

Sélection par la taille de communauté Au sein du programme ASP, nous avons sélectionner les communautés par taille de la façon suivante :

```
lower_bound_size {chosen_bacteria(X):bacteria(X)} upper_bound
  _size.
```

Le listing 1.2.1 sélectionne un sous ensemble de taille minimal `lower_bound_size.` et de taille maximal `upper_bound_size.` parmi l'ensemble `chosen_bacteria(X)`, où X est dans le domaine défini par l'ensemble des bactéries (`bacteria`). Cela revient à sélectionner au hasard des bactéries (`chosen_bacteria(X)`) parmi l'ensemble des bactéries initialement défini `bacteria`, où le nombre de bactéries sélectionner ne peut-être plus grand que `upper_bound_size.` et plus petit que `lower_bound_size.`

Mathématique, soit t un sous-ensemble de taxon appartenant à l'ensemble de taxon $\mathcal{T}$, où un taxon correspond à une bactérie, la cardinalité $|t|$ doit être compris entre les valeurs min et max :

$$\forall t \in \mathcal{T}$$
$$min < |t| < max$$

Sélection par la composition taxonomique au grain de la souche. Nous avons laissé la possibilité à l'utilisateur et à l'utilisatrice de filtrer sur la composition taxonomique de la communauté, avec les prédicats `always`,`forbid`,`mandatory_asso` et `forbidden_asso`.

Le prédicat `always`, représentant un taxon qui doit être présent dans toutes les communautés possibles est implémenté en ASP comme suit :

```
1 :- always(X); not chosen_bacteria(X).
```

Cette règle ne dispose pas de tête, on parle alors de contrainte d'intégrité. Elle se lit comme suit : parmi l'ensemble de bactérie initial, nous ne voulons pas les communautés où la bactérie définit par le prédicat `always(X)` n'est pas présente. La double négation étant lourde, nous pouvons traduire ainsi : sélectionne l'ensemble des communautés où la bactérie présumé `always` est présente.

Mathématiquement, soit A l'ensemble de bactéries obligatoirement présent dans la communauté :

$$\forall t \in \mathcal{T} \text{ si } \forall A \in t$$

Concernant les bactéries qui ne doivent pas être présent dans la communauté, le code ASP ainsi que le formalisme mathématique est très ressemblant :

```
1 :- chosen_bacteria(X); forbid(X) : chosen_bacteria(X).
```

Cela se traduit par : les communautés qui possèdent un ensemble de bactéries interdites (`forbid(X)`) parmi l'ensemble des bactéries tirées au sort ne doivent pas être sélectionnées. Mathématiquement, on appelle l'ensemble de bactérie interdit F, et est défini comme suit :

$$\forall t \in \mathcal{T} \text{ si } \forall F \notin t$$

Nous avons également défini des associations obligatoires (`mandatory_asso`) et interdites (`forbidden_asso`). Pour rappel, l'utilisateur ou l'utilisatrice pouvait définir des groupes d'association, nommé `group_forbidden(G)` dans le cas des associations interdites. Ce prédicat regroupe l'ensemble des groupes (`G`) que la ou le biologiste a défini dans le fichier de configuration YAML. Nous avons défini les associations obligatoires en 3 parties.

Dans le listing 1.1, les communautés sélectionnées (`selected_bacteria(X)`) ont obligatoirement toutes les bactéries de l'ensemble des associations obligatoires parmi l'ensemble de bactéries préalablement choisis (`chosen_bacteria(X1) : mandatory_asso(_,X1).`).

```
selected_bacteria(X) :- chosen_bacteria(X);
                        chosen_bacteria(X1) : mandatory_asso(_,X1).
```

Listing 1.1 – Règle numéro 1 pour définir les associations obligatoires

Il faut ensuite rajouter les communautés où aucune des bactéries des associations obligatoires n'est présent parmi l'ensemble de bactéries préalablement défini (`not chosen_bacteria(X1) : mandatory_asso(_,X1).`). Ce calcul est illustré par le code dans le llisting 1.2

```
selected_bacteria(X) :- chosen_bacteria(X);
                  not chosen_bacteria(X1) : mandatory_asso(_,X1).
```

Listing 1.2 – Règle numéro 2 pour définir les associations obligatoires

Enfin, pour une question de visualisation des solutions, nous avons ajouté la contrainte d'intégrité 1.3 qui ne montre que les communautés filtrées par les associations obligatoires.

```
:- chosen_bacteria(X); not selected_bacteria(X).
```

Listing 1.3 – Règle numéro 3 pour définir les associations obligatoires

Mathématiquement, soit O l'ensemble des bactéries des associations obligatoires, et $o_i \forall i \in O$. Les associations obligatoires sont définies comme :

$$\forall O \in \mathcal{T} \text{ si } o_i = t \text{ alors } \forall o_i \in t$$

Enfin, pour définir les associations interdites au sein d'une communauté (voir listing 1.4, les communautés sélectionnées ne doivent pas posséder l'ensemble des bactéries interdites parmi les communautés déjà sélectionnées (`chosen_bacteria(X2) : forbidden_asso(G,X2).`)

```
:- chosen_bacteria(X); group_forbidden(G);
                  chosen_bacteria(X2) : forbidden_asso(G,X2).
```

Listing 1.4 – Règle définissant les associations interdites

Mathématiquement, soit I l'ensemble des bactéries des associations interdites et $I_i \forall i \in I$. Les associations interdites sont alors définies comme :

$$\forall I \in \mathcal{T} \text{ si } \forall I_i \notin t \text{ alors } I_i \in t$$

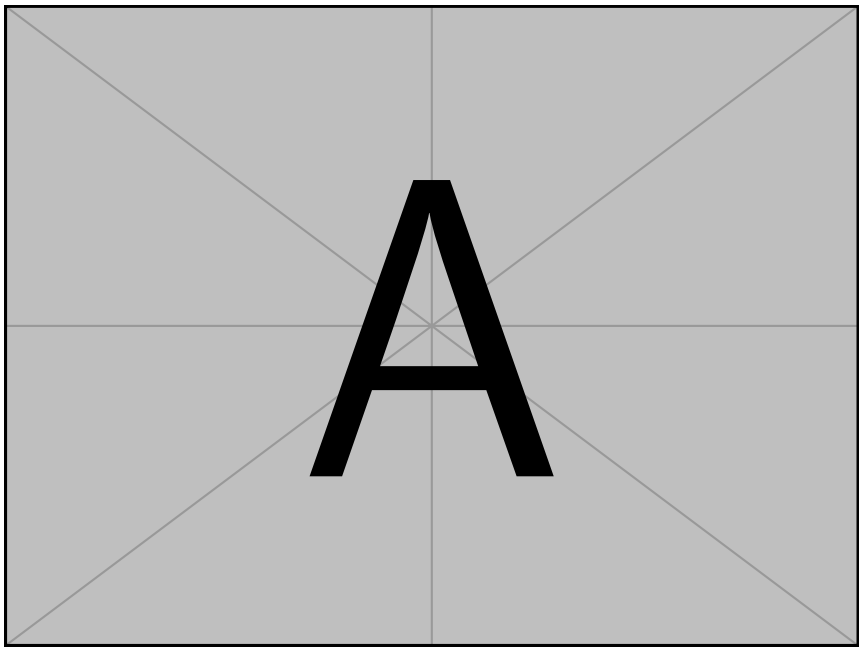

FIGURE 1.1 – Montrer en image le workflow complet en illustration. Montrer la facilité d'utilisation pour les biologistes et les liens entre chaque modification faite par l'utilisateur et la contrainte formalisés.

1.2.2 Résultats préliminaires

Nous venons de décrire et détailler la base de connaissance nécessaire pour sélectionner des communautés à façon. Dans cette sous-section, nous allons dans un premier temps identifier des questions biologiques provenant d'un retour d'expérience sur les précédents chapitres ainsi que de discussions avec les biologistes, et comment l'outil de sélection permet d'y répondre. Puis dans un second temps, nous allons appliquer notre méthode de sélection de communauté à façon sur le jeu de données du papier M2M (**?**) afin de montrer son efficacité à passer à l'échelle.

Exemples

Toutes les questions ci-dessous vont chercher à identifier des ensembles de communautés candidates pour des objectifs métaboliques supposés connus. Les questions seront présentées des plus larges au plus précises. Nous utiliserons les réseaux métaboliques reconstruits à l'échelle du génome de Weiss (**?**). Ils ont été reconstruits avec l'outil GapSeq(**?**) et un consortium bactérien permettant une analyse des interactions métabolique de l'intestin de souris a été créé.

Pour une communauté de taille 5, existe-t-il une communauté qui métaboliste l'ensemble des acides aminés ? Cette question nous permet d'identifier les deux contraintes à utiliser au sein de l'outil : la liste des composés que l'on doit produire, *i.e.* les acides aminés (AA), ainsi que la taille de la communauté finale, *i.e.* 5.

Le fichier initial est donc constitué des informations suivantes :

```
# Size constraints
SIZE:
  size: 5
```

N'ayant précisé aucune optimisation, l'ensemble des cibles, que sont les acides aminés, devront être dans le scope de la communauté. Le résultat pour cette situation est **unsatisfiable**. Cela signifie qu'il n'existe pas de modèles permettant de satisfaire les contraintes de l'utilisateur ou de l'utilisatrice, et donc aucune communauté de taille n'a été trouvé comme étant capable de métaboliser l'ensemble des acides aminés. Le premier réflexe de l'utilisateur ou de l'utilisatrice est de formuler la demande de façon à relâcher les contraintes. La contrainte la plus forte ici consiste à obtenir au moins une communauté permettant de l'ensemble des **métaboliser les acides aminés**. Une reformulation possible serait : métaboliser un **maximum** d'acide aminé. En appliquant cette modification nous obtenons une communauté composée des souches *I48*, *KB1m*, *YL44*, *YL45*, *YL27*.

Pour une communauté composée de 6 taxa, laquelle permet la plus grande production métabolique sans les échanges métaboliques ? Nous avons 2 types de contraintes : 2 optimisations, que sont la maximisation du scope communautaire et la minimisation des échanges métaboliques, et taille, et se traduit comme suit dans le fichier d'initialisation :

```
# Size constraints
SIZE:
  size: 5

  # Optimisation
OPTIMISATION:
  - max
  - min

# Objectif
OBJ:
  - scope
  - exchanged
```

La meilleure communauté, selon le solveur, possède 797 échanges et 1010 métabolites sont productibles en communauté. En plus de connaître la meilleur composition taxonomique répondant à ces contraintes, le vocabulaire contrôlé développé dans le chapitre **??** est également accessible : les métabolites échangeables, les consommateurs et producteurs intervenant dans l'échange, les substrats polyopsonistes, les potentiels de coopération et de compétition.

A partir d'une communauté de 150 réseaux métaboliques de l'intestin, j'aimerai une étudier une communauté de composé de 6 à 8 taxons garantissant une productibilité d'un maximum des 23 composés d'intérêts En plus de ces contraintes, il y a des contraintes de composition taxonomique que sont : *les réseaux métaboliques GCA_003433685, GCA_003433675 et GCA_003433665 doivent-être présent, GCA_003433745, GCA_003433755 et GCA_003433765 doivent jamais apparaître dans la ou les communautés. Les ensembles GCA_003433795, GCA_003433825 et GCA_003433845 ou GCA_003433855 et GCA_003433865 ne doivent pas être présents. En revanche, les ensembles GCA_003433905, GCA_003433925 et GCA_003433945 ou GCA_003433955 et GCA_003433985 doivent être présent.* Cette demande factice test le passage à l'échelle de la méthode. Nous avons donc utilisé 150 réseaux métaboliques reconstruits dans le chapitre **??** :

contrainte d'entrée :

```
# Size constraints
SIZE:
  size:
  lower bound: 6
  upper bound: 8

# Bacterial composition
 constraints
BACTERIAL COMPOSITION:
  in:
    - GCA_003433685
    - GCA_003433675
    - GCA_003433665
  out:
    - GCA_003433745
    - GCA_003433755
    - GCA_003433765
  forbidden asso:
    -
      - GCA_003433795
      - GCA_003433825
      - GCA_003433845
    -
      - GCA_003433855
      - GCA_003433865

  mandatory asso:
    -
      - GCA_003433905
      - GCA_003433925
      - GCA_003433945
    -
      - GCA_003433955
      - GCA_003433985

# Optimisation
OPTIMISATION:
  - max

# Objectif
OBJ:
  - target
```

sortie :

```
Answer: 1

GCA_003433925.
GCA_003433985.
GCA_003433685.
GCA_003433675.
GCA_00343390.
GCA_003433945.
GCA_003433665.
GCA_003433955.
```

Performance :
Models : 1
Time : 197.120s
CPU Time : 196.781s

En maximisant une liste de composés d'intérêt (23) et à partir des paramètres ci-dessus, nous avons trouvé une communauté candidate en environ 3 minutes et 30 secondes.

1.3 Discussion et conclusion

Cette première exploration d'enrichissement d'un modèle logique a répondu aux remarques majeures que nous avons énoncées plus haut : filtrer le nombre de solutions d'un modèle logique et optimisation de fonctions objectives. En effet, l'outil développé dans le chapitre **??** permet de donner une indication des potentiels de coopération et de compétition de communautés microbiennes mais ne permet pas d'identifier les critères permettant de l'affirmer. Avec ce prototype, nous avons proposé un moyen de sélectionner la meilleure communauté selon les critères de l'utilisateur ou de l'utilisatrice, et par extension, de définir la communauté avec un fort potentiel de coopération ou de compétition. Cela a été possible en ajoutant une optimisation d'une ou de plusieurs fonctions objectives. La

plus-value par rapport au logiciel MISCOTO est que l'utilisateur ou l'utilisatrice choisi la fonction objective qu'il ou elle souhaite optimiser.

Cependant, aucune validation expérimentale n'a été faite à ce jour. Au regard de la littérature concernant la création de communauté à façon, une validation expérimentale est nécessaire pour créer une bonne communauté synthétique (**??**). Par exemple, Macchi et ses collègues (**?**) ont construit une communauté bactérienne à partir de 6 génomes isolés pour la dégradation du phénanthrène.

Vis à vis des interactions bactériennes, ce prototype permet de se concentrer essentiellement sur le potentiel de coopération d'une communauté bactérienne. Concernant le potentiel de compétition uniquement défini par les substrats polyopsonistes, c'est à dire un nutriment qui est co-consommé, l'ajouté de la temporalité semble être un bon moyen pour améliorer la prédiction de la compétition. En effet, dans le chapitre **??**, le lactose est bien co-consommé mais pas au même moment dans le temps, empêchant de définir un lien de compétition pour ce substrat. Afin d'essayer de prendre en compte une pseudo-temporalité au sein des modèles logiques, nous allons utiliser la logique temporelle de Cabalar (**?**).

1.4 Intégration d'une temporalité réactionnelle

La logique linéaire temporelle (LTL), à notre connaissance, n'a jamais été appliquée dans un contexte biologique. Pourtant, dans son domaine d'application IA, l'inférence de règles temporelles permet d'affiner les prédictions en proposant des explications séquentielles. Brièvement, Cabalar se base sur le problème de raisonnement sur les actions et le changement (RAC). Il en existe 5 : (i) la **simulation** ; étant donné un état initial du système et une suite d'actions, la simulation permet de prédire l'état du système à un temps futur. (ii) L'**explication** ; étant donné un état initial et un état futur du système, l'explication propose une séquence d'état intermédiaires du système menant de l'état initial à l'état final. (iii) **Planification** ; étant donné un état initial du système et un objectif, la planification propose une séquence d'actions, éventuellement non-déterministes, garantissant l'objectif. (iv) **Diagnostique** ; à partir d'une définition de transitions d'état normal et anormal ainsi qu'un ensemble d'états observés (normal et anormal), le diagnostique propose un ensemble minimal de transitions expliquant les observations anormales. (v) **Vérification**, validation d'une propriété temporelle du système (comme sûreté, vivacité ou équité) ou une propriété de représentations (comme l'équivalence de modèles).

Le premier défi est l'application de ces concepts RAC à la biologie des systèmes. Nous avons donc associé le système à une communauté de GEMs et les actions à des activations de réactions. Ainsi, chaque nouvelle activation d'une réaction, selon l'algorithme d'expansion du réseaux métabolique d'Ebenhöh, consistera un nouvel état du système. En prenant un exemple simple de la communauté fromagère du chapitre **??**, nous savons que le propionate est produit à partir du lactate. Remarquons ici que nous ne précisons pas les organismes pour le moment. À partir de cette information nous pouvons déduire *a minima* une relation de dépendance logique temporelle, qui est : pour qu'*a un moment de la cinétique de fermentation* le propionate soit productible, il faut que le lactate soit productible *avant*. En appliquant la LTL à des réseaux métaboliques, les notions temporelles *un moment* et *avant* seront définies selon la longueur du chemin menant à la production du composé : propionate. Au sein de la LTL, le solveur, *telingo*, cherche à minimiser la taille du chemin. Ainsi, à partir d'un ensemble de nutriments disponibles dans le milieu extracellulaire, un ensemble fini de voies métaboliques mène à la production d'un composé cible, et *telingo* priorise le chemin le plus court, c'est à dire le nombre réaction le plus faible. De ce fait, la temporalité n'est pas représentée comme une dimension, tel que présent dans le dFBA par exemple, mais par une séquence d'actions, activation ou non d'une réaction : on parle de *temporalité réactionnelle*. Dans la suite de ce chapitre nous allons définir les cas d'utilisation de Cabalar et les appliquer à la communauté du chapitre **??**.

Inférence temporelle dans la programmation logique

Solveur *telingo* Nous avons vu que les modèles numériques, contrairement aux modèles logiques, pouvaient intégrer une dimension temporelle et ainsi analyser d'autres propriétés du système biologique. Il existe un formalisme informatique, nommé *logique temporelle linéaire* ou LTL, qui permet d'introduire une notion de temporalité au sein des modèles logiques.

Ce modèle de logique par raisonnement se base sur le même formalisme que celui décrit dans le chapitre **??** : une base de faits, des règles et contraintes doivent-être générées. *telingo*permet d'intégrer des contraintes temporelles et les règles et contraintes utilise les

opérateurs logiques temporels décrits dans la Figure 1.2.

`&initial`	$\mathbf{I}$	*initial*	`&final`	$\mathbb{F}$	*final*
`'p`	$\bullet p$	*previous*	`p'`	$\circ p$	*next*
`<`	$\bullet$	*previous*	`>`	$\circ$	*next*
`<?`	$\mathbf{S}$	*since*	`>?`	$\mathbb{U}$	*until*
`<*`	$\mathbf{T}$	*trigger*	`>*`	$\mathbb{R}$	*release*
`<?`	$\blacklozenge$	*eventually before*	`>?`	$\lozenge$	*eventually afterward*
`<*`	$\blacksquare$	*always before*	`>*`	$\square$	*always afterward*
`<:`	$\widehat{\bullet}$	*weak previous*	`>:`	$\widehat{\circ}$	*weak next*

FIGURE 1.2 – Opérateurs temporels du futur et du passé en *telingo*et logique temporelle issue de (**?**)

Dans l'exemple ci-dessous, nous avons défini une proposition vraie tout le temps : Quelque chose est *chargé* si **au temps précédent** il a été *chargé* et qu'il n'est pas *déchargé*.

```
1    charge :- □ (• charge; not decharge)
```

À partir de cette nouvelle synthaxe, nous avons créé une communauté jouet du chapitre **??** afin d'appliquer la LTL et d'illustrer la temporalité réactionnelle au moyen de 3 concepts RAC : **planification**, **vérification** et **explication**.

1.4.1 Démarche à suivre

Ce formalisme logique est une extension du formalisme décrit dans le chapitre **??** mais n'ayant jamais été appliqué sur des réseaux métaboliques, nous avons représenté les réseaux métaboliques de *P. freudenreichii*, *L. plantarum* et *L. lactis* de manière simplifiée et minimaliste. Ce choix méthodologique a été imposé pour deux raisons : éviter dans un premier temps la problématique du passage à l'échelle et dans un second temps, permettre de résoudre les défis rencontrés lors de l'application des concepts de Cabalar. Dans les sous-sections qui suivent, nous présenterons les différents cas d'utilisation de Cabalar ainsi que leur résolution à partir d'une communauté factice, puis, l'application de ces concepts sur les réseaux métaboliques du chapitre **??**.

1.4.2 Explication des concepts de Cabalar

Ensemble de règles en *telingo*pour la modélisation d'une communauté de bactéries

Scope Nous avons redéfini le vocabulaire exprimé dans le chapitre **??** en intégrant le vocabulaire temporel de la LTL. Ainsi, les notions de réactions activées et de composés productibles deviennent :

```
1 0{isActivated(Rea,Org)}1 :- product(Compound,Rea,Org,_) ;
2 'producible(Reactant,Org) : reactant(Reactant,Rea,Org,_).
```

Listing 1.5 – Règle d'activation d'une réaction

```
producible(Compound,Org):-  product(Compound,Rea,Org,_),
isActivated(Rea,Org).
```

Listing 1.6 – Règle de productibilité d'un composé

Les contraintes exprimées dans les listings 1.5 et **??** signifient qu'une réaction est activée si l'ensemble des réactants ont été produits *avant* (représenté par l'apostrophe '). Un composé est considéré comme productible si c'est un produit d'une réaction et qu'elle est activée.

L'évolution de la concentration d'un métabolite au cours du temps n'étant pas inférée dans le modèle, nous ne pouvons déterminer quand une réaction n'est plus activée. Nous assumons alors que si une réaction est activée une fois elle ne peut l'être une seconde fois. Cette règle s'exprime sous la proposition logique suivante :

```
not isActivated(Rea,Org) :- not not &tel{ <? isActivated(Rea,
   Org)},
 reaction(Rea,Org).
```

Listing 1.7 – Règle interdisant qu'une réaction soit activée deux fois

Disponibilité des nutriments nouvellement productibles Avec cette notion temporelle à l'échelle d'une réaction, un composé produit en amont doit-être disponible pour une réaction qui se situerait en aval. La Figure 1.3 illustre ce concept que nous avons développé. A partir du composé extracellulaire `Ae`, le composé C est productible au temps réactionnel 2. Il est ainsi disponible pour la réaction situé 2 temps réactionnel en aval, la réaction R4.

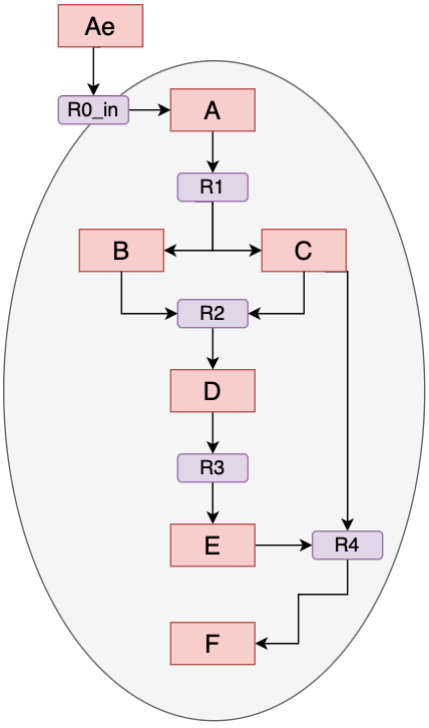

FIGURE 1.3 – Réutilisation du composé C par la réaction R4 où C était productible 2 étapes avant

Avec cette nouvelle propriété, la définition de réaction activée est redéfinie ainsi : Une réaction est activée si tous les réactants de la réaction ont été productible avant (représenté par l'apostrophe à ligne 2) :

```
1 disponible(Compound,Org) :- producible(Compound,Org).
2 disponible(Compound,Org) :- 'disponible(Compound,Org).
```

Bien entendu, au sein d'une communauté bactérienne, nous observons des échanges métaboliques. Les métabolites échangés ont été considérés comme productibles ils deviennent alors disponibles pour l'ensemble des membres de la communauté.

1.4.3 Application de *telingo* sur des organismes jouets

Nous venons de voir les règles essentielles permettant de modéliser une communauté bactérienne avec la synthaxe temporelle. Dans les sections suivantes nous verrons l'applications du formalisme *telingo* sur le des communautés jouets afin de vérifier si les concepts RAC s'adaptent bien sûr à la biologie des systèmes. Les règles et les contraintes utilisées pour modéliser les concepts RAC ne seront pas expliquées, seul l'aspect temporel sera mis en avant.

Planification

Nous avons dès à présent les éléments nécessaires pour modéliser les cas d'application de Cabalar. Pour cela nous avons utilisé les réseaux métaboliques illustré par les Figures 1.4 et 1.5. Dans le cas de la planification, l'objectif à atteindre était de déterminer la communauté minimale pour la production du composé Ne. Dans la figure 1.4, ce composé peut être produit soit par l'organisme 1 seulement et par l'organisme 2 par l'intermédiaire de 2 échanges métaboliques. La sortie du modèle est indiquée par le chemin en vert validant ainsi la prédiction.

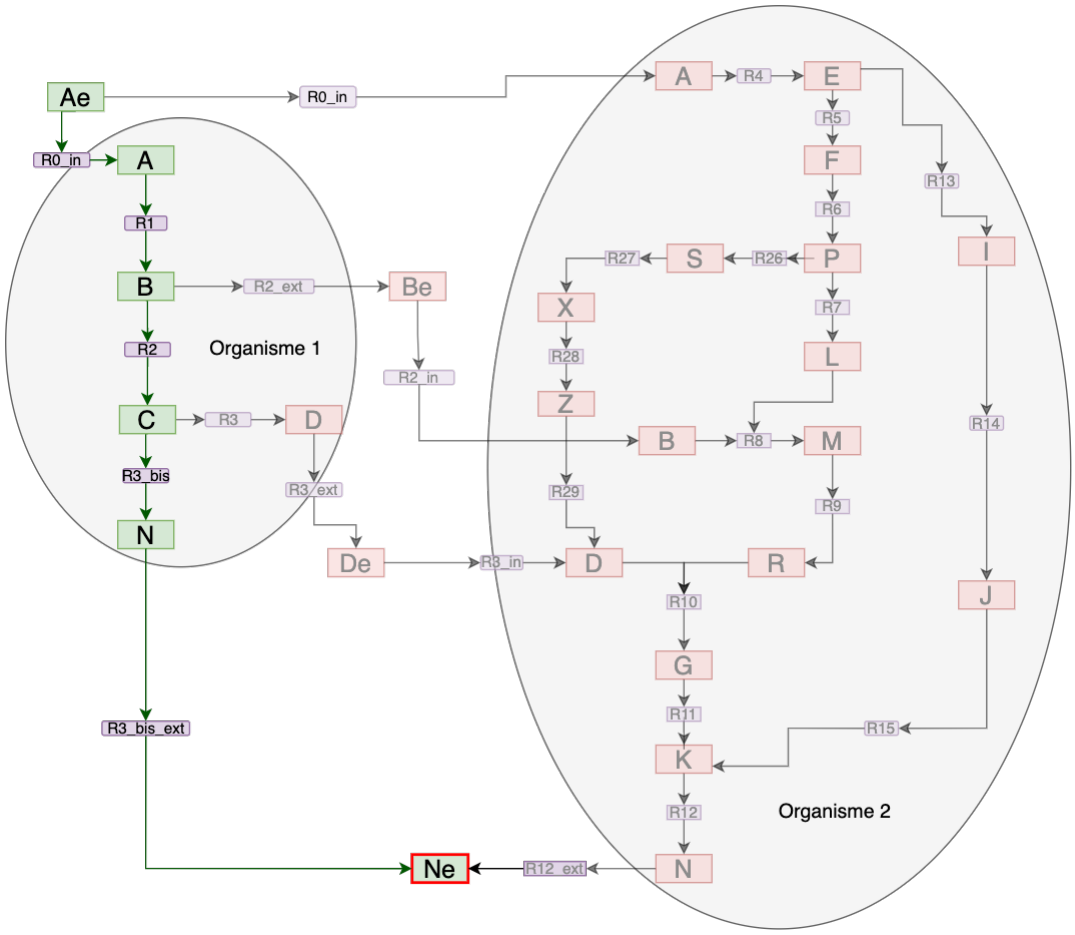

FIGURE 1.4 – Schéma didactique permettant de valider la solution donnée par le modèle dans le cas de la planification en montrant que le composé Ne est productible à partir d'un seul organisme via le chemin vert.

Aucune règle stipulant qu'une communauté est minimum composé de 2 taxons, *telingo* montre que seul l'organisme 1 peut produire le composé `Ne`, comme indiqué par le listing 1.8.

```
Answer: 1
 State 0:
 State 1:
  isActivated("R0_in","Org1")
  producible("A","Org1")
 State 2:
  isActivated("R1","Org1")
  producible("B","Org1")
 State 3:
  isActivated("R2","Org1")
  producible("C","Org1")
 State 4:
  isActivated("R3_bis","Org1")
  producible("N","Org1")
 State 5:
  isActivated("R3_bis_ext","Org1")
```

```
17   producible("Ne","Org1")
18 Optimization: 1 5
19 OPTIMUM FOUND
20
21 Models       : 1
22   Optimum    : yes
23 Optimization : 1 5
24 Calls        : 6
25 Time         : 0.265s (Solving: 0.00s 1st Model: 0.00s Unsat:
       0.00s)
26 CPU Time     : 0.197s
```

Listing 1.8 – Résultat de la planification obtenue après application des règles sur la base de faits

Les séquences des actions montrant la production de `Ne` est représenté par "`State`" dans la sortie du modèle. Ici, Au temps réactionnel 0, aucune réaction est activée. Seul le composé extracellulaire `Ae` serait marqué comme étant `disponible`. Au temps réactionnel suivant, `State 1`, la réaction important le composé `Ae` dans le milieu intracellulaire est activé par la réaction `R0_in` (voir ligne 4 du listing), et le composé `A` est `productible` et `disponible` (ligne 5). Au temps réactionnel suivant, `B` est `productible` et la réaction `R1` est activée. En procédant ainsi, Le composé `Ne` est `productible` et `disponible` pour l'ensemble de la communauté en 5 étapes, correspondant au temps réactionnel 5.

Vérification

Le cas de vérification consiste simplement à vérifier qu'il existe une séquence de réactions activables permettant de produire un composé. Ainsi, nous avons vérifié la productibilité du composé `Ne`(voir listing 1.8) :

Nous pouvons voir qu'à chaque pas de temps, marqué par "`State`", il existe au moins une réaction qui est activée, permettant à terme, de produire le composé `Ne`.

Explication

Dans le cas de l'explication, nous cherchons à déterminer le chemin réactionnel permettant la production d'un composé sachant que l'on a observé la production d'un autre composé à un moment dans le temps. Brièvement, l'objectif est de savoir s'il existe une relation entre une observation faite à un moment dans le temps et l'objectif final. Dans le cas de notre exemple de l'explication représenté par la Figure 1.5, nous avons cherché à expliquer la production du composé `Ne` sachant que le composé `De` a été observé. Le résultat du modèle est indiqué par le tracé en vert. Le résultat du modèle est indiqué par le listing 1.9 :

```
1 State 0:
2 State 1:
3   isActivated("R0_in","Org1") isActivated("R0_in","Org2")
4   producible("A","Org1") producible("A","Org2")
5 State 2:
6   isActivated("R1","Org1") isActivated("R4","Org2")
7   producible("B","Org1") producible("E","Org2")
8 State 3:
```

```
9    isActivated("R2","Org1") isActivated("R2_ext","Org1")
       isActivated("R5","Org2")
10   producible("Be","Org1") producible("C","Org1") producible("
       F","Org2")
11  State 4:
12   isActivated("R2_in","Org2") isActivated("R3","Org1")
       isActivated("R6","Org2")
13   producible("B","Org2") producible("D","Org1") producible("P
       ","Org2")
14  State 5:
15   isActivated("R7","Org2")
16   producible("L","Org2")
17  State 6:
18   isActivated("R3_ext","Org1") isActivated("R8","Org2")
19   producible("De","Org1") producible("M","Org2")
20  State 7:
21   isActivated("R3_in","Org2") isActivated("R9","Org2")
22   prod_explain("D","Org2")
23   producible("D","Org2") producible("R","Org2")
24  State 8:
25   isActivated("R10","Org2")
26   prod_activate("G","Org2")
27   prod_activate_cof("G","Org2")
28   producible("G","Org2")
29  State 9:
30   isActivated("R11","Org2")
31   prod_activate("K","Org2")
32   prod_activate_cof("K","Org2")
33   producible("K","Org2")
34  State 10:
35   isActivated("R12","Org2")
36   prod_activate("N","Org2")
37   prod_activate_cof("N","Org2")
38   producible("N","Org2")
39 Optimization: 3 17
40 OPTIMUM FOUND
41 Models       : 4
42   Optimum    : yes
43 Optimization : 3 17
44 Calls        : 11
45 Time         : 0.366s (Solving: 0.02s 1st Model: 0.00s Unsat:
       0.00s)
46 CPU Time     : 0.350s
```

Listing 1.9 – Résultat de l'explication obtenue après application des règles sur la base de faits didactique

Au temps réactionnel 6 (ligne 19), nous voyons que l'observation est expliquée, et qu'il faut attendre le temps réactionnel 10 pour que le composé final `Ne` soit productible. Nous pouvons également remarquer que le modèle explique en même temps la production du

composé `Be` pour permettre la production de `Ne` : on dit alors que `Be` est un métabolite essentiel à la production de `Ne`.

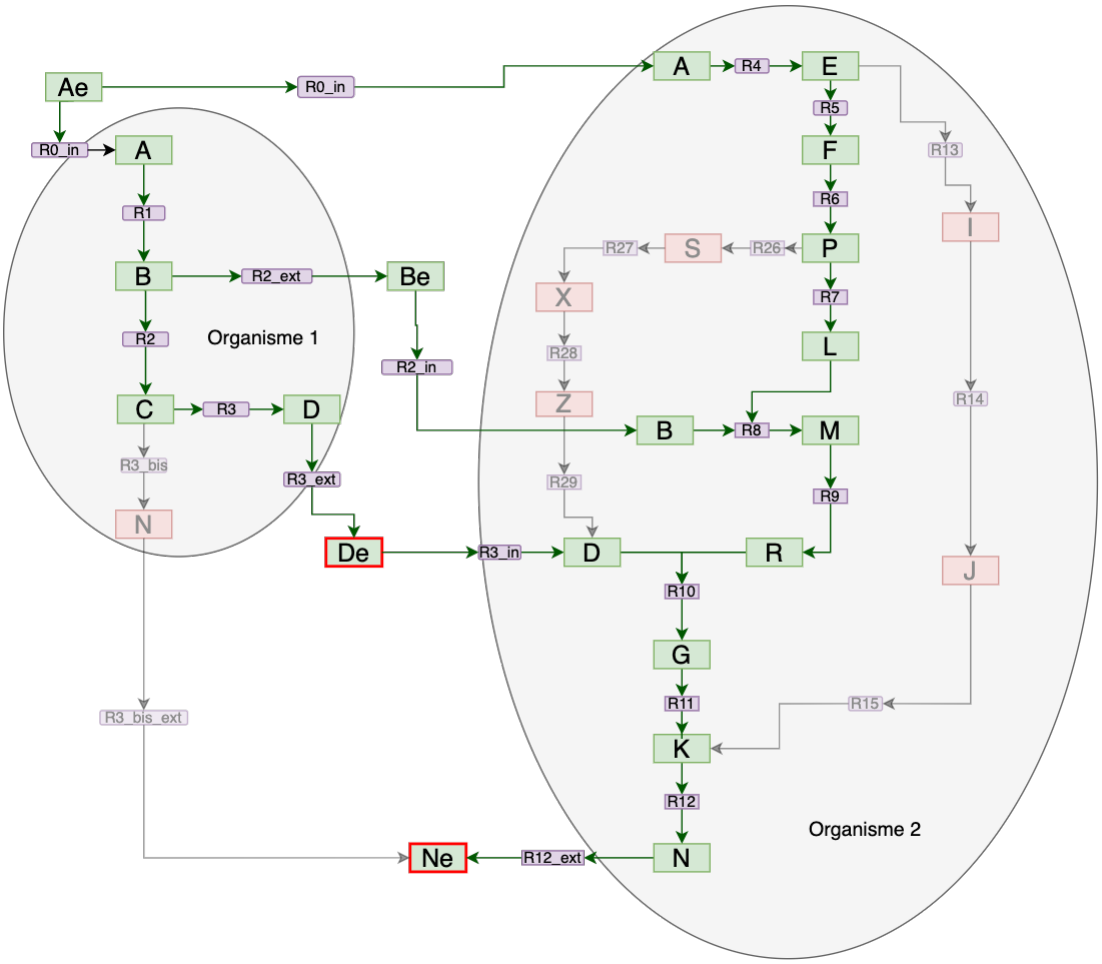

FIGURE 1.5 – Schéma didactique permettant de valider la solution donnée par le modèle dans le cas de l'explication en montrant que le composé Ne est productible à partir de l'élément observable De.

Cette implémentation temporelle, sur un schéma didactique, met en avant deux points intéressants par rapport à un dFBA. Le premier est la possibilité d'avoir la suite réactionnelle conduisant à un ou plusieurs objectifs finaux de façon direct. Cette représentation en temps réactionnel, permet donc de générer des hypothèses testables sur l'activation de ou non de voies métaboliques d'intérêt à travers des méthodes de knockout enzymatique par exemple. La seconde est la mise en avant indirect d'une coopération entre ces deux organismes. Pour rappel, la connaissance initiale montrait une observation dans le milieu extracellulaire de `De` et non un échange entre ces deux organismes. De plus, ce modèle a permis de montrer qu'un autre composé est échangé pour permettre de réaliser l'objectif final.

1.5 Application à une communauté simplifié du chapitre ??

Nous avons repris le milieu du lait ainsi que les réseaux métaboliques du chapitre **??**, et généré la nouvelle base de faits applicable à *telingo*. Nous présenterons que 2 cas : planification et explication.

1.5.1 Cas de la planification

Dans le cas de la planification, le système initial est composé des 3 bactéries, du milieu nutritionnel du lait. L'objectif à attendre est la production de propionate, note `M_ppa_e` dans la sortie du modèle représenté par 1.10 :

Listing 1.10 – Résultat de la planification obtenue après application des règles sur la base de faits issue des données réelles

```
 State 0:
 State 1:
  isActivated("R_ADK1_rev","lactis") isActivated("R_ARGDr","
     lactis")
  producible("M_amp_c","lactis") producible("M_atp_c","lactis
     ") producible("M_citr__L_c","lactis") producible("M_nh4_
     c","lactis")
 State 2:
  isActivated("R_NTD7","lactis") isActivated("R_OCBT_rev","
     lactis") isActivated("R_THRD_L","lactis")
  producible("M_2obut_c","lactis") producible("M_adn_c","
     lactis") producible("M_cbp_c","lactis") producible("M_nh
     4_c","lactis") producible("M_orn_c","lactis") producible
     ("M_pi_c","lactis")
...
...
...
 State 17:
  isActivated("R_PPAKr","lactis")
  producible("M_atp_c","lactis") producible("M_ppa_c","lactis
     ")
 State 18:
  isActivated("R_DM_ppa_c","lactis")
  producible("M_ppa_p","lactis")
 State 19:
  isActivated("R_DM_ppa_p","lactis")
  producible("M_ppa_e","lactis")
Optimization: 1 26
OPTIMUM FOUND
Models       : 44
  Optimum    : yes
Optimization : 1 26
Calls        : 20
```

```
Time             : 65.678s (Solving: 15.03s 1st Model: 0.08s
   Unsat: 0.10s)
CPU Time         : 57.789s
```

Un premier résultat intéressant consiste à observer que la méthode semble passer à l'échelle d'une communauté bactérienne de taille 3 (65 secondes) composé de plusieurs milliers de réactions. Un second point surprenant concerne l'organisme qui produit le propionate : *L. lactis*. En effet, nous savons que seul la bactérie propionique (*P. freudenreichii*) est capable de produire ce composé. Enfin, parmi les 44 modèles, aucun ne prédit d'échanges métaboliques au sein de la communauté.

1.5.2 Cas d'explication

Nous avons également vu dans le chapitre **??**, que le propionate est productible à partir du lactate produit par les bactéries lactiques *L. lactis* et *L. plantarum*. Ainsi, expliquer la production de propionate sachant le lactate observé et consommé par *P. freudenreichii* constitue un exemple d'utilisation pour le cas d'explication. Nous avons représenté dans la Figure 1.6, une simplification des voies métaboliques utilisé par le modèle pour produire le propionate sachant le lactate observé.

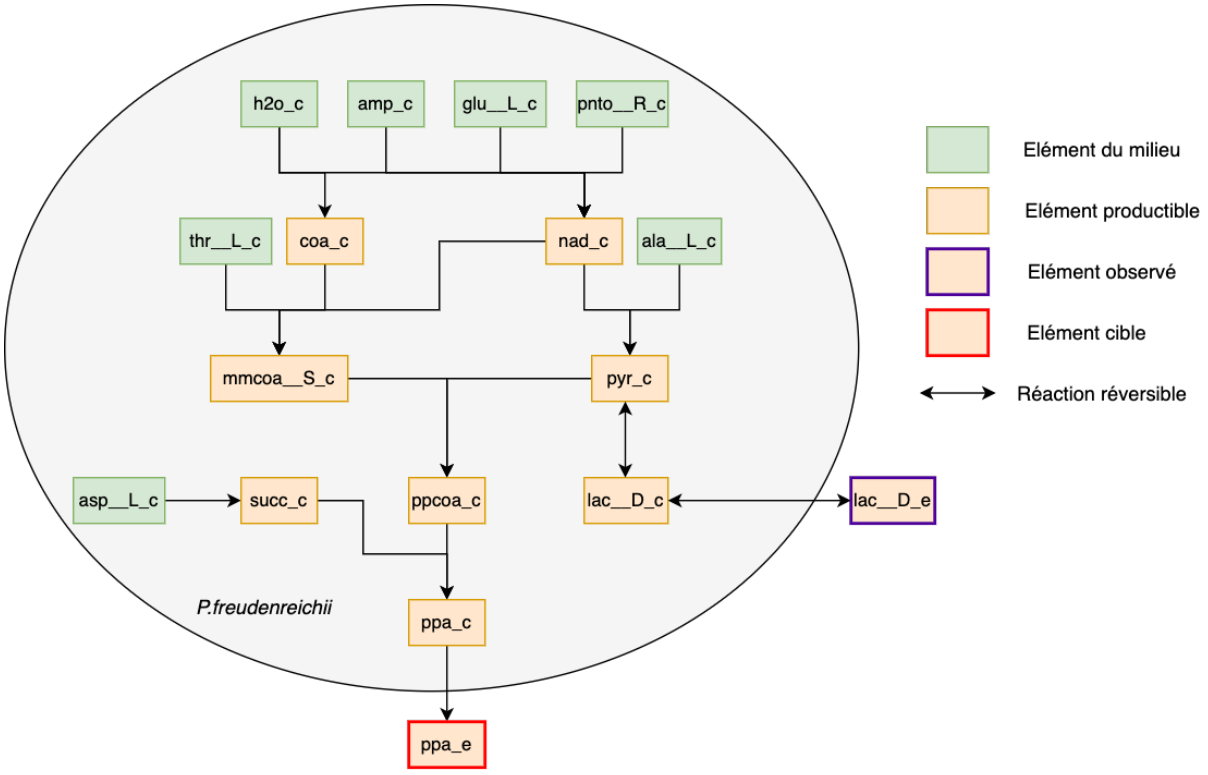

FIGURE 1.6 – Schéma permettant de visualiser le résultat simplifié du modèle d'explication sur les données réelles

Contrairement au cas de planification, il a fallu 23 étapes pour résoudre expliquer la production de propionate à partir du lactate. Malgré l'ajout d'une connaissance *a priori*

(consommation du lactate par *P. freudenreichii*) le modèle ne permet toujours pas d'identifier les interactions entre les bactéries lactiques et la bactérie propionique. Cependant, nous pouvons reconnaître un cycle important pour la production du propionate : cycle de Wood-Werkman.

1.6 Discussion et conclusion

Nous avons montré une application à la biologie des systèmes d'un concept d'IA : le raisonnement sur les actions et le changement. Nous avons également vu l'efficacité de la temporalité sur les exemples didactiques : des hypothèses sur des interactions bactériennes , explications métaboliques d'observations et rapidité d'exécution. Tout comme dans le chapitre **??**, aucune fonction objective n'est nécessaire. Malgré cela, cette approche permet une description réactionnelle d'une communauté bactérienne.

Lors du passage à l'échelle d'un jeu de données réel, nous observons les limites d'une modélisation basée sur la topologie du réseau : production du propionate permise sans échanges métaboliques. Cependant, en sélectionnant les bonnes connaissances *a priori* à inférer dans le modèle, cette modélisation topologique réactionnelle permettrait d'analyser et générer des explications rapidement sur des observations relevées lors des expériences.