Comparaison visuelle des réseaux métaboliques après inférence des relations d'orthologie entre gènes

Auteurs :
Peter BOCK
Guillamaury DEBRAS
Mercia NGOMA-KOMB
Cécilia OSTERTAG
Franck SOUBES

Clients :
Patricia THÉBAULT
Raluca URICARU

Université de Bordeaux

Rapport de Projet soumis pour l'évaluation principale du Master 1 Bio-informatique

2017

Table des matières

Résumé

Un réseau métabolique constitue l'ensemble des processus métaboliques et physiques qui déterminent les propriétés physiologiques et biochimiques d'une cellule. Ces réseaux comprennent les réactions chimiques du métabolisme, les voies métaboliques, ainsi que les interactions régulatrices qui guident ces réactions. Avec l'avancée du séquençage complet des génomes, il est maintenant possible de reconstituer le réseau de réactions biochimiques dans de nombreux organismes, des bactéries aux humains et ainsi pouvoir les comparer entres eux. Ce projet se base sur des outils existants qui permettent la manipulation informatique des réseaux métaboliques, tel que le format *SBML* et le logiciel de visualisation Cystoscape. Un nouveau programme qui inclut ces outils permet la comparaison de réseaux métaboliques se basant sur des relations d'orthologie entre les enzymes mises en jeu dans ces réseaux.

Remerciements

Nous tenons à remercier toutes les personnes qui nous ont prodigué leurs conseils, fait part de leurs idées et nous ont consacré du temps. Sans cette aide précieuse, nous n'aurions pas pu avancer autant sur notre projet.

— Patricia Thébault et Raluca Uricaru, nos deux encadrantes (clients). Lors de plusieurs réunions, notre projet a fait l'objet de nombreux échanges qui nous ont permis d'éclaircir divers points. Ces rencontres ont contribué à répondre à leurs besoins de manière plus adaptée.
— Florian Lefebvre, stagiaire M2 bio-informatique au LaBRI, nous a aidé à comprendre quelles parties du format SBML étaient utiles à notre projet et nous a proposé des solutions pour la manipulation de ces fichiers.
— Marie Beurton-Aimar, nous a fait bénéficier de ses connaissances sur les réseaux métaboliques, même si parfois ils allaient bien au-delà de ce que nous étions en mesure de réaliser dans les délais impartis, mais nous y avons tout de même puiser beaucoup d'inspiration.
— Floreal Morandat pour nous avoir donnée de son temps et nous avoir montrée comment utiliser Maven pour gérer les dépendances de notre programme.
— Nicolas Rodriguez (Ebi) et Sarah Keating (Caltech), qui ont apporté leur soutien au groupe *sbml-flux* nous permettant ainsi de comprendre comment utiliser libSBML correctement, notamment au niveau de l'utilisation des plugins pour parser les fichiers SBML level 3.
— Todd DeLuca, pour la mise à disposition de son implémentation de l'algorithme RSD sous licence libre, sur laquelle est basé en grande partie notre logiciel.
— et bien d'autres...

Glossaire

API Application Programming Interface.p7

BDBH Bidirectionnal Best Hit.p10

BLAST Basic Local Alignment Search Tool.p4

FBC Flux Balance Constraints.p13

Jeutils The Entrez Programming Utilities for Java.p7

JVM Java Virtual Machine.p7

Kegg Kyoto Encyclopedia of Genes and Genomes.p7

LaBRI Laboratoire Bordelais de Recherche en Informatique.p6

MIRIAM Minimum Information Required In the Annotation of Models.p16

PAML Phylogenetic Analysis by Maximum Likelihood.

RSD Reciprocal Smallest Distance.p4

SBML The System Biology Markup Language.p6

URI Uniform Resource Identifier.p16

UML Unified Modeling Language.

XGMML the eXtensible Graph Markup and Modeling Language.p26

XML Extensible Markup Language.p10

XSD XML Schema Definition.p11

Table des figures

Introduction

La modélisation des processus moléculaires à l'échelle de la cellule connaît un essor très marqué depuis l'apparition des approches globales ("omics"). Cette thématique est au centre d'une nouvelle démarche, la biologie des systèmes, dont l'objectif est d'utiliser des outils mathématiques et informatiques afin de comprendre les relations entre génotype et phénotype. La comparaison des simples répertoires des gènes de deux organismes ne prend pas en compte la combinaison de l'ensemble des interactions entre les molécules. Elle reste insuffisante pour appréhender le fonctionnement cellulaire dans son ensemble. La visualisation des similitudes et différences entre deux réseaux est plus informative. Il devient alors possible d'identifier les alternatives (de voies métaboliques) pour la synthèse d'un même métabolite dans deux bactéries différentes. Aussi, la reconstruction des réseaux et l'annotation des gènes est un problème majeur.

Actuellement, la production de données de séquençage évolue de manière exponentielle et est largement automatisée, cependant l'annotation du génome se fait encore manuellement. Ceci étant, il est impératif d'arriver à automatiser l'annotation du génome afin de rendre les études de génomiques comparatives plus aisées. Le but primaire est la comparaison visuelle de réseaux métaboliques par l'intermédiaire de la recherche de relations d'orthologie entre les gènes codant pour les enzymes de deux bactéries.

Dans un premier temps, nous ferons une analyse du sujet qui sera suivi d'un état de l'art. Ces derniers nous permettrons d'énumérer les éléments disponibles et ainsi déterminer les outils les plus appropriés à l'avancement du projet. Par la suite, nous décrirons les étapes d'acheminement des flux menant à l'obtention d'un résultat visuel sous forme d'un *workflow*. Enfin, nous détaillerons l'implémentation du code jusqu'à l'obtention des paires d'orthologues [1] et la visualisation des ces derniers sur Cytoscape [2].

Accès au code du projet

https ://github.com/CeciliaOstertag/pathway_comparison_project

Pour plus d'informations sur le fonctionnement de notre programme, il est possible de se référer au README et/ou à la javadoc associés à ce projet.

Chapitre 1

Analyse du sujet

1.1 Contexte

Dès que le génome d'un nouvel organisme est séquencé, l'étape suivante est son annotation structurale et fonctionnelle. Cette dernière consiste à prédire la fonction de chaque gène. Elle s'appuie sur la recherche d'homologies entre ces nouvelles séquences et celles des gènes d'organismes modèles. Cependant, la simple comparaison de séquences d'ADN peut se révéler insuffisante pour déterminer correctement la fonction d'un gène dans un organisme donné. Pour remédier à cela, une des stratégies utilisées est de s'appuyer sur la meilleure similarité réciproque entre l'ensemble des gènes de deux organismes pour la recherche de gènes orthologues. En effet, deux gènes orthologues, c'est-à-dire hérités d'un ancêtre commun après un événement de spéciation, ont tendance à conserver des fonctions similaires au cours de l'évolution [3]. Ces annotations fonctionnelles apportent des précisions sur les processus moléculaires en œuvre dans les organismes étudiés. Cependant,avec l'entrée dans l'ère des *Next-Generation Sequencing* et l'absence d'outils d'annotations automatiques, les chercheurs rencontrent des difficultés dans l'annotation des génomes [4]. Il est donc nécessaire de développer de nouveaux outils bio-informatiques, adaptés aux spécificités des organismes étudiés, ainsi qu'aux types de données sur lesquels on souhaite baser l'analyse.

Le but du projet sera, d'une part, d'identifier les relations d'orthologies existantes entre les gènes de notre organisme modèle *Escherichia coli* [5] et celles d'une seconde bactérie, de proximité phylogénétique variable. D'autre part, nous travaillerons sur la visualisation de ces résultats, sous la forme d'un graphe représentant la fusion des réseaux métaboliques des deux bactéries, dans lequel un code couleur permettra de distinguer aisément les enzymes orthologues des enzymes non-orthologues. Les résultats que nous obtiendrons, pourront servir de base à de futures études de comparaisons, qui rendront plus faciles l'annotation des gènes étudiés.

1.2 État de l'art

1.2.1 Algorithmes d'alignement de séquence

L'algorithme BLAST (*Basic Local Alignment Search Tool*) [6], développé par le NCBI, est un algorithme heuristique utilisé pour trouver des régions similaires entre une séquence nucléique ou protéique, et d'autres séquences du même type présentes dans une banque de données. BLAST recherche des segments qui sont localement homologues à une séquence-test fournie par l'utilisateur (un *query sequence*). BLAST utilise une matrice de similarité pour calculer des scores d'alignement. Il fournit un score pour chaque alignement trouvé et utilise ce score pour donner une évaluation statistique de la pertinence de cet alignement (probabilité qu'il soit dû au hasard). Le principe de fonctionnement de BLAST, peut se décomposer en trois étapes :

— La décomposition de la séquence test en segments de longueur k (k-tuplets) chevauchants et la recherche pour chacun d'eux de tous les k-tuplets possibles ayant un score d'homologie supérieur à un seuil donné. BLAST constitue ainsi un dictionnaire de tous les k-tuplets donnant une homologie locale minimum.

— Le parcours de la banque avec le dictionnaire est ainsi effectué. Chaque fois que BLAST identifie une correspondance dans la banque, il tente d'étendre l'homologie en amont et en aval du k-tuplet initialement trouvé.

— Après extension de l'homologie, il évalue à partir du score obtenu la probabilité que celle-ci soit due au hasard (ou plus exactement, son espérance mathématique).

Pour la recherche d'orthologues, une approche classiquement utilisée avec les génomes bactériens se basent sur *Bidirectionnal Best Hit (BDBH)*, et se basent sur les résultats de BLAST. L'algorithme de BDBH que nous utiliserons dans notre projet est le *Reciprocal Smallest Distance algorithm (RSD)* [7], qui fonctionne selon le schéma suivant :

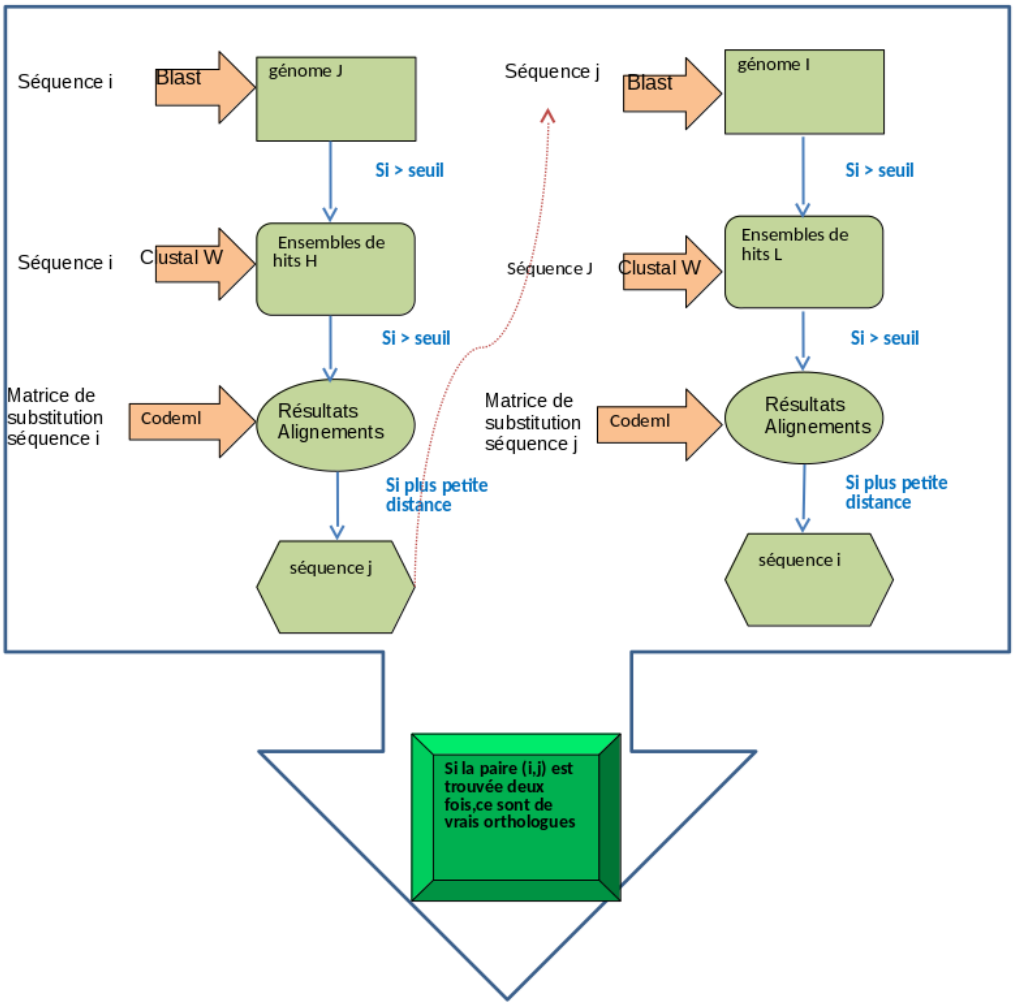

FIGURE 1.1: Détail de l'algorithme RSD

La figure 1.1 représente le fonctionnement de l'algorithme de BDBH. L'algorithme se base sur l'utilisation d'un BLAST initial pour une comparaison initiale des séquences. S'ensuit l'utilisation de la méthode clustalW pour l'alignement de séquences afin d'obtenir les meilleurs *Hits*, enfin l'utilisation de codeml pour l'estimation du calcul de maximum de vraisemblance. La même opération est réalisée cette fois en prenant comme séquence de référence celle trouvé en fin de processus. Cela revient à comparer deux protéomes complet l'un contre l'autre et extraire les meilleurs *hits* lorsque ceux-ci sont réciproques entre deux protéines.

1.2.2 Logiciels de visualisation de réseaux métaboliques

Tulip [8] développé depuis 2001 par le LaBRI est un système de visualisation d'informations dédié à l'analyse et à la visualisation de graphes.
Le logiciel conseillé par nos clients, Cytoscape [2], est quant à lui un logiciel de visualisation et traitement de graphes en langage JAVA. Ce dernier dispose d'une facilité de prise en main avec de nombreux plugins facilement installables permettant l'import de données sous différents formats, l'analyse et la visualisation de graphes. Cytoscape permet notamment d'explorer les interactions d'un groupe de gènes, d'interpréter des résultats de *microarray* ou de modéliser des régulations génétiques ou métaboliques.
Pour ce projet plusieurs logiciels sont disponibles (Gephi, Tulip, Cytoscape) mais afin de répondre à la demande du client nous utiliserons Cytoscape. Nous utiliserons également plusieurs *plugins* spécifiques à Cytoscape, adaptés à nos données et aux questionnements biologiques soulevéés par ce projet.

Cy3sbml

Cy3sbml [9] est une application Cytoscape fournissant des fonctionnalités avancés pour l'importation et le travail avec des modèles encodés dans le format SBML, permettant ainsi d'importer directement des fichiers possédant l'extension *.sbml*. Ceci permet une lecture de fichier *SBML* et de s'en servir pour jouer sur l'affichage dans Cytoscape. En outre, cela permet une vérification de notre parser SBML, pour s'assurer que ce dernier ne présente pas des erreurs syntaxiques. Ce dernier est basé sur la bibliothèque JSBML.

ClusterMaker

ClusterMaker [10] est un *plugin* Cytoscape qui implémente plusieurs algorithmes de *clustering* et fournit des vues de réseaux, de dendrogrammes et de cartes thermiques de résultats. Il est le premier *plugin* Cytoscape qui ajoute une telle variété d'algorithmes et de visualisations de *clusters* (sous-graphes), y compris les seules implémentations de méthodes hiérarchiques, AutoSOME, k-medoïdes et k-Means pour l'expression groupée ou les données génétiques ; et AutoSOME, MCL, TransClust, SPCS, Affinity Propagation et MCODE pour regrouper des réseaux de similarités pour rechercher des familles de protéines. L'application va donc être utile pour rechercher des complexes possibles, des familles de protéines, des relations fonctionnelles en lien avec un contexte biologique précis.

1.2.3 Bases de données

Bases NCBI : Bases de données de séquences

A l'ère du numérique, les informations issues d'analyses biologiques sont disséminées dans une multitude de banques de données. La GenBank est une banque de séquences d'ADN, comprenant toutes les séquences de nucléotides publiquement disponibles et leur traduction en protéines. L'annotation des génomes est un processus à plusieurs niveaux comprenant des gènes codant pour des protéines, ainsi que d'autres unités génomiques fonctionnelles telles que les ARN structuraux, les ARNt, les petits ARN, les pseudo-gènes, les régions de contrôle, les répétitions directes et inversées, les séquences d'insertion, les transposons et d'autres éléments. L'information contenue dans la GenBank peut être très utile pour récupérer automatiquement les gènes, le nom de leur produit, et la séquence correspondante afin de récupérer un fichier multiFASTA. Les deux plus grands centres de bio-informatique du monde sont l'Institut Européen de bio-informatique (EBI) [11], et le *National Center for Biotechnology Information (NCBI)* [12], qui rassemblent la plupart des banques de données.

KEGG : Base de données traitant les orthologues

Le projet de base de données KEGG a été lancé en 1995 par Minoru Kanehisa, professeur à l'Institut de recherche chimique de l'Université de Kyoto, dans le cadre du programme en cours du Génome humain japonais . KEGG est une "représentation par ordinateur" du système biologique.
Plusieurs mois après le lancement du projet KEGG en 1995, le premier rapport du génome bactérien complètement séquencé a été publié. Depuis lors, tous les génomes complets publiés sont accumulés dans le KEGG pour les eucaryotes et les procaryotes. La base de données KEGG GENES contient des informations sur le niveau de gène / protéine et la base de données KEGG GENOME contient des informations sur l'organisme pour ces génomes. La base de données KEGG GENES se compose d'ensembles de gènes pour les génomes complets, et les gènes de chaque ensemble reçoivent des annotations afin d'établir des correspondances aux schémas de câblage des cartes KEGG, des modules KEGG et des hiérarchies BRITE. Ces correspondances sont faites en utilisant le concept **d'orthologues**. Tous les gènes de la base de données KEGG GENES sont regroupés dans la base de données KEGG ORTHOLOGY (KO). Étant donné que les nœuds (produits génétiques) des cartes de chemins KEGG, ainsi que les modules KEGG et les hiérarchies BRITE, reçoivent des identifiants KO, les correspondances sont établies une fois que les gènes du génome sont annotés avec des identifiants KO par la procédure d'annotation du génome dans KEGG [13]

1.2.4 Langage de programmation utilisé

Le langage de programmation qui sera utilisé dans le cadre de ce projet est le langage Java. Java est un langage orienté objet, avec interpréteur portable, créé par Sun Microsystems (aujourd'hui racheté par Oracle). Une des grandes forces de Java est sa portabilité : une fois un programme créé, il peut fonctionner automatiquement sous différents systèmes d'exploitations tels que Windows,Mac, et Linux. Les applications Java peuvent être exécutées sur tous les systèmes d'exploitation pour lesquels a été développée une plate-forme Java, dont le nom est JRE (Java Runtime Environment - Environnement d'exécution Java). Cette dernière est constituée d'une JVM , le programme interprète le code Java et le convertit en code natif. Mais le JRE est surtout constitué d'une bibliothèque standard à partir de laquelle on peut développer la grande majorité des programmes en Java. Pour ce projet, nous avons choisi de programmer en Java afin de pouvoir créer plus facilement un *plugin* Cytoscape par la suite, étant donné que Cytoscape a été codé en Java.

En plus de la bibliothèque standard, des bibliothèques spécialisées dans le domaine de la biologie existent, et seront mises à profit pour mener à bien notre projet.

BioJava

Biojava est un projet open-source qui a pour but de fournir des outils pour analyser des données biologiques. Ce projet a été mis au point par Thomas Down et Matthew Pocock. BioJava est un ensemble de bibliothèques composées des fonctions écrites dans le langage de programmation Java afin de manipuler des séquences, des structures de protéines, ou encore parcourir des fichiers et en extraire des données spécifiques.

Jeutils

JeUtils est un utilitaire de ligne de commande open source pour accéder aux bases de données de l'Institut national d'information sur la biotechnologie (NCBI). Le but de JeUtils est de permettre l'automatisation des requêtes auprès de NCBI via leur API eUtils et leurs services de recherches de séquences BLAST.

libSBML

libSBML est une bibliothèque de logiciels *open-source* fournissant une interface de programmation d'applications (API) pour le format SBML. La puissance de cette bibliothèque est la vérification et l'exactitude des modèles SBML d'une manière qui va au-delà de la simple validation basée sur des schémas. De plus, libSBML facilite la création et la manipulation d'annotations SBML rejoignant ainsi la dynamique du projet.

1.3 Analyse des besoins

Afin de répondre aux exigences de nos clients, les différentes requêtes et traitements seront organisés de la façon suivante :
La première étape est la récupération dans la banque de données NCBI [12] des fichiers GenBank correspondants à *Escherichia coli* et à notre bactérie d'intérêt.

A partir de ces fichiers, un premier parseur nous permettra d'obtenir un fichier multifasta contenant l'intégration des séquences nucléiques ou protéiques de chacune des enzymes pour chaque bactérie. L'utilisation d'un algorithme de *bidirectional best hit (BDBH)* permettra une comparaison par paires de ces séquences, afin d'identifier les relations d'orthologies entre ces séquences. Nous obtiendrons donc une liste de gènes ou de protéines orthologues.

Les fichiers GenBank seront également soumis à la banque de données
RAST/SEED [14], pour obtenir une reconstruction des réseaux métaboliques des deux bactéries, qui seront récupérées au format SBML.
Une fois l'inférence des relations d'orthologies effectuée, l'objectif de la dernière étape sera de fusionner les deux fichiers SBML, en tenant compte des relations entre enzymes.

Enfin, l'utilisation du logiciel Cytoscape, additionnée avec certains *plugins* pour l'importation des modèles encodés en SBML, sera nécessaire pour comparer les réseaux métaboliques et faciliter leur visualisation.

La figure ci-dessous représente, de façon simplifiée, le *pipeline* attendu :

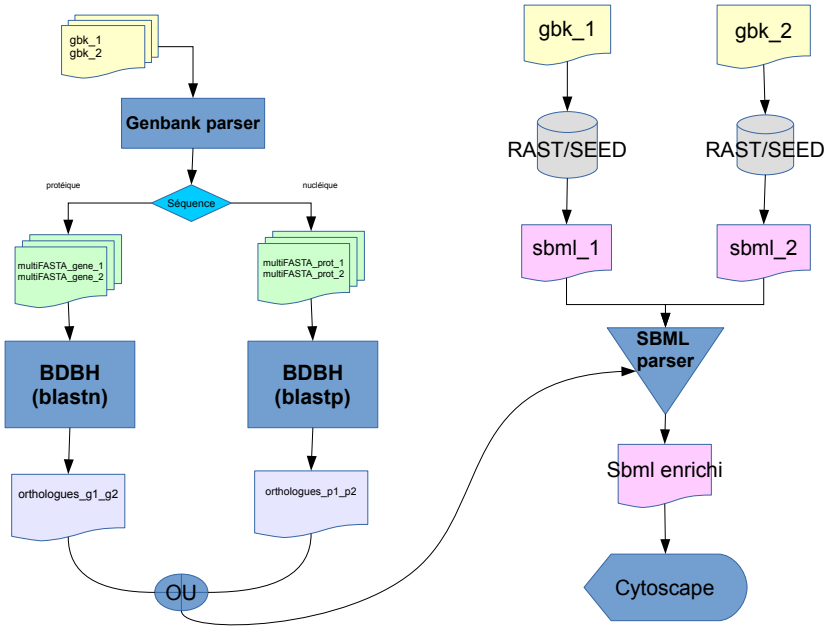

FIGURE 1.2: Fonctionnement du pipeline attendu par nos clients

Le bon fonctionnement de ce *pipeline* repose sur l'obtention de fichiers SBML à partir desquels il est possible de faire le lien avec le génome dont les séquences correspondantes aux enzymes sont issues. Cependant, après plusieurs tests sur la banque RAST/SEED, il apparaît que cette banque est d'une part difficile d'accès et encore en cours de développement, et d'autre part que nous n'obtenons pas d'identifiants permettant de récupérer simplement les données des enzymes au NCBI. Après discussion avec nos clients, une nouvelle banque de données nous a été indiquée : il s'agit de la base de données de modèles métaboliques *BiGG Models* [15], dans laquelle les fichiers SBML fournis contiennent les identifiants des *gene locus tag* référencés au NCBI. Nous utiliserons donc des données issues de cette banque, et nous effectuerons directement des requêtes au NCBI au lieu de télécharger puis parser des fichiers GenBank. Notre *pipeline* devient alors :

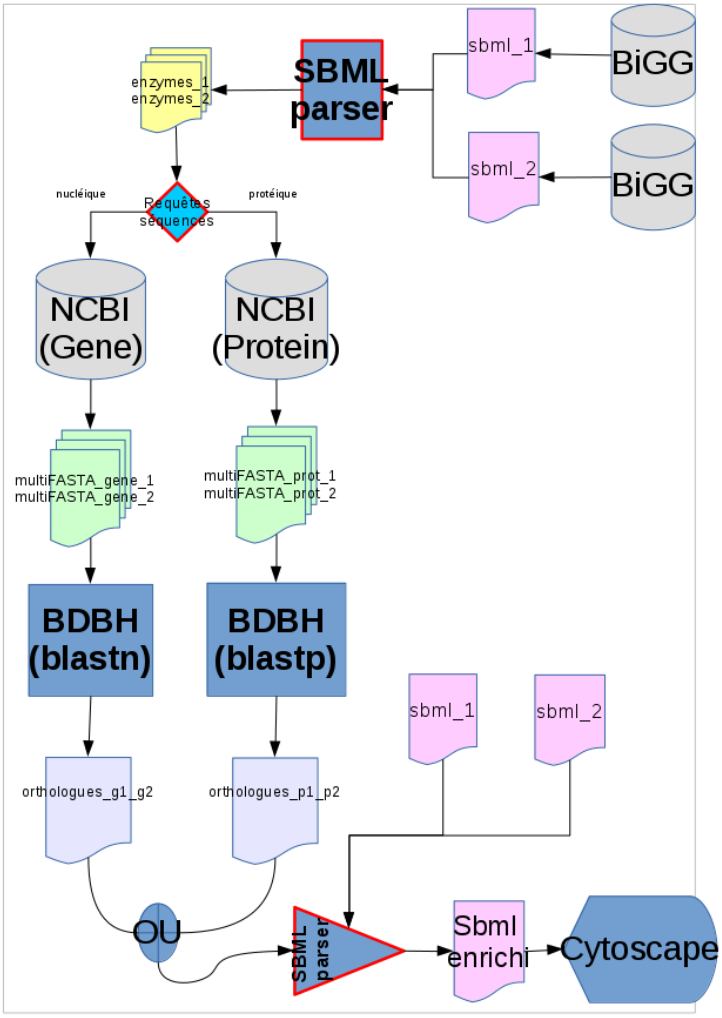

FIGURE 1.3: Fonctionnement du pipeline utilisant les données de BiGG

1.3.1 Besoins fonctionnels

Les fonctionnalités attendues lors de la réalisation de ce projet sont les suivantes :

— Adaptation d'un algorithme de BDBH déjà créé pour la recherche de relations d'orthologies entre séquences nucléiques ou protéiques, selon le choix de l'utilisateur final.

— Création d'un parseur pour fichier SBML, adapté au format de la banque de données BiGG, qui retournera la liste des identifiants de toutes les enzymes présents. Le *Parser* utilisera les fichiers de schémas XSD [16] associés aux fichiers de type SBML pour parser correctement les données.

— Obtention des séquences nucléiques et protéiques associées à chaque enzyme, au format FASTA, à l'aide de requêtes NCBI automatisées.
— Création d'un parseur permettant de créer deux nouveaux fichier SBML qui ajouteront un attribut représentant la présence ou absence de relations d'orthologies entre deux enzymes basés sur le résultat de l'algorithme de BDBH.
— Obtention d'un graphe final visualisé grâce au logiciel Cytoscape, avec un code couleur représentant les relations d'orthologies entre les enzymes des deux bactéries.

1.3.2 Besoins non fonctionnels

En plus des besoins fonctionnels, notre projet devra respecter les règles suivantes :
— Utilisation du langage Java ; ceci est un choix personnel qui nous permet d'avoir une uniformité de langage dans l'optique de réunir ces fonctionnalités en un nouveau *plugin* ou application Cytoscape.

1.4 Formats des données

1.4.1 Format des réseaux métaboliques

Les réseaux métaboliques que nous étudions sont enregistrés au format SBML. Le SBML (*System Biology Markup Language*) est un langage à balises, basé sur le langage XML. Un document SBML est découpé en éléments structurés hiérarchiquement à partir d'un élément racine. Syntaxiquement, le fichier SBML est découpé en paires de balises ouvrantes et fermantes. Il est couramment utilisé pour modéliser des réseaux biochimiques, métaboliques, *etc.*
De plus, ce modèle dispose de nombreuses connexions avec des bases de données (*Biomodels*) et logiciels (Cytoscape, copasi, cell designer) rendant ce format approprié pour son utilisation dans ce projet.

La structure d'un fichier XML (comme le SBML) ainsi que le type de données pouvant être stockés dans chaque tag, sont répertoriés dans un fichier *.XSD* [16] associé, donnant l'ensemble des tags disponibles au fichier SBML, ainsi que leur organisation. La majorité des parseurs de XML utilise les fichiers XSD pour vérifier que la structure du fichier XML est correcte.

La Version 1 Niveau 1 de SBML sortie en 2001 n'a eu de cesse d'évoluer, de nombreuses fonctionnalités étant rajoutées au fur et à mesure. La dernière version du SBML niveau 3 version 1 sur laquelle nous travaillons est l'une des plus récentes datant de 2010. Cependant, il est à noter que chaque SBML diffère suivant son niveau (1,2,3) impliquant des règles restrictives plus élaborées pour les dernières versions des différents niveaux. De plus, il est a préciser que le niveau 2 reste mis à jour car sa version 5 a été publiée en 2015. Afin de synthétiser une réaction comme dit précédemment, SBML utilise donc des balises spécifiques pour chaque composant participant à la réaction de manière astucieuse.

La balise *geneProduct* prend en entrée 3 identifiants les liant à des banques de données (Kegg,Bigg...)
La balise *Species* prend en compte plusieurs entrées qui sont liées à des banques de données par des identifiants. On y retrouve les métabolites (réactifs,produits).
La balise *Reaction* est une déclaration décrivant une transformation, le transport ou le processus de liaison qui peut modifier la quantité d'une ou plusieurs espèces. Par exemple, une réaction peut décrire comment certaines entités (réactifs) sont transformées en certaines autres entités (produits). Elle est subdivisée en plusieurs sous-balises.
— La balise *geneProductAssociation* englobe le ou les gènes codant pour les protéines enzymatiques catalysant la réaction.
— la balise *listOfReactants* caractérise les réactifs (*Species*) intervenant dans la réaction.
— la balise *listOfProducts* inclut les produits de la réaction.

1.4.2 Format des séquences

Le format FASTA est un format de fichier texte utilisé pour stocker des séquences biologiques de nature nucléique ou protéique. Un fichier FASTA est composé de deux lignes. La première ligne décrit la séquence en commençant par le symbole « > » suivi immédiatement de l'identifiant. La ligne suivante est constituée des lettres représentant les acides nucléiques ou les acides aminés de la séquence. Le format multiFASTA consiste simplement en la succession de séquences au format FASTA.

1.5 Jeu d'essai

Le but de ce projet étant de comparer les voies métaboliques de bactéries peu connues avec celles d'*Escherichia Coli*, notre jeu d'essai sera constitué des génomes des quatre organismes suivants :

— Le génome de référence d'*Escherichia coli* str. K-12 substr. MG1655, d'une taille de 4641652 paires de base. [5]
— Le génome de *Bacillus subtilis subsp. subtilis* str 168 [17]
— Le génome de *Helicobacter pylori 26695* [18]
— Le génome de *Salmonella enterica subsp. enterica serovar Typhimurium* str. LT2 [19]

1.6 Bilan

Au terme de cette analyse nous avons pu illustrer les différents besoins de nos clients. Une liste d'outils adéquats, un langage uniformisé et la création d'un *pipeline* spécifique permettront de mener à bien ce projet. Il est important de distinguer les outils que nous allons devoir créer et ceux déjà à notre disposition. Nous réaliserons un *pipeline* complexe à l'aide du langage Java. Il inclura la création des parseurs définis préalablement ainsi que la génération et le traitement de fichiers multifasta. Une partie du pipeline se basera sur le lancement du programme de BDBH déjà existants à l'aide de commandes bash. Enfin, nous obtiendrons des fichiers SBML annotés par notre programme, qui pourront être importés dans le logiciel Cytoscape afin de visualiser le résultat de ces annotations.

Chapitre 2

Conception & Réalisation

Les besoins que nous avons défini précédemment doivent à présent avoir une traduction en terme de concepts clairs qui seront par la suite traduits au moyen du langage Java. De plus, pour mener à bien ce projet, nous avons dû élaborer une architecture logicielle cohérente avec l'objectif à atteindre. Nous avons ainsi subdivisé les fonctionnalités de notre programme en plusieurs classes. (Voir Annexe VI : Diagramme de classes).

2.1 Récupération des séquences correspondant aux enzymes intervenant dans les réseaux métaboliques

2.1.1 Objectifs

Recherche des enzymes dans les fichiers SBML

Les fichiers SBML que nous utilisons dans le cadre de ce projet sont conformes aux spécifications *SBML Level 3* [20], et utilisent le *package Flux Balance Constraints* (fbc) [21] .

Dans ces réseaux métaboliques, les seuls éléments qui nous intéressent ici sont les enzymes qui entrent en jeu lors des différentes réactions. D'après l'organisation de nos fichiers SBML, les éléments correspondant aux enzymes sont identifiés par la balise *fbc :GeneProduct* (voir Annexe 2.17)

De plus, comme notre objectif est d'interroger par la suite les bases de données du NCBI, nous cherchons à récupérer un identifiant qui puisse identifier de façon unique chaque enzyme dans ces bases de données. Or, entre les balises *annotation* de chaque *GeneProduct*, la balise *rdf :ressource* peut nous permettre de récupérer l'identifiant GI de chaque enzyme, qui identifie de façon unique une protéine dans la base de données *NCBI Protein*.
À la fin de cette première étape, nous obtenons le début du *workflow* décrit dans la partie analyse.

Récupération des séquences correspondant aux enzymes

Pour lancer les requêtes de façon automatisées à partir d'un programme local, nous utiliserons le service *Entrez Programming Utilities (E-utilities)* [22], qui permet d'accéder à toutes les bases de données du NCBI. Vu que ce service est utilisé par de très nombreux chercheurs, certaines contraintes existent. La contrainte principale étant une limitation du nombre de requêtes à 3 par secondes. Pour éviter que l'IP de la machine exécutant l'application ne soit temporairement bannie des services du NCBI, il faut respecter cette limite, ainsi qu'essayer d'exécuter les scripts les plus lourds durant les périodes de faibles affluences de la base de données

Une fois les identifiants GI récupérés, il est relativement facile de récupérer les séquences protéiques correspondant auprès de la base de données *Protein*, étant donné que l'identifiant GI est unique. Il est

donc possible de télécharger ces séquences au format FASTA.

En revanche, la recherche des séquences nucléiques correspondant à ces enzymes est plus ardue. En effet, la plupart des séquences ne constituent pas des entrées spécifiques dans la base de données *Gene*, mais font référence à une partie de la séquence du génome de l'organisme correspondant, délimitée par les positions de début et de fin du gène. Ainsi, une recherche manuelle de l'identifiant GI dans la base de données *Gene* permet de sélectionner la portion du génome correspondant et de la télécharger au format FASTA, mais cela n'est pas réalisable en utilisant *E-Utilities*. Pour cette raison, nos clients ont préféré que nous nous concentrions sur l'utilisation des séquences protéiques pour la recherche d'orthologues.

2.1.2 Implémentation

EnzymeFinder

Notre première classe, *EnzymeFinder* permet, en utilisant *libSBML*, de parser les fichiers SBML et de récupérer de façon précise l'attribut GI de chaque *GeneProduct*. Cette classe va donc renvoyer la liste d'identifiants GI des enzymes.

Premièrement, La classe fait appel à *libSBML* pour lire le fichier SBML et créer un objet contenant un modèle du SBML.
Après cela, on crée un nouvel objet modèle dans lequel sont chargées des fonctionnalités supplémentaires, grâce au plugin *fbc*, en plus des fonctionnalités de base du premier. Cela permet au parseur de *libSBML* de reconnaître les informations supplémentaires apportées par le *package Flux Balance Constraints*.
Dans ces nouvelles fonctionnalités se trouve la capacité d'extraire la liste contenant l'ensemble des *fbc :GeneProducts* dans le fichier SBML, et d'extraire pour chaque élément les données d'annotations correspondantes.
Pour chaque *fbc :GeneProduct*, une chaîne de caractères représentant son annotation est extraite, et l'on peut alors extraire de cette chaîne de caractères les éléments *rdf*. L'un de ces éléments *rdf* contient un URL contenant le GI de l'enzyme dans la base de données, et il suffit alors d'extraire la partie de cette URL correspondant à l'identifiant GI et de le stocker dans une liste.

Query

La classe *Query* fait appel à la bibliothèque JEutils pour lancer une requête par enzyme, au NCBI. Cette requête s'effectue en deux étapes : l'identifiant GI est d'abord recherché dans la base de données *Protein*, puis si un résultat est trouvé, une *url* est générée, qui permet de télécharger la séquence au format FASTA dans un fichier TXT. Etant donné la grande quantité de séquences à télécharger pour chaque bactérie (1260 pour *Escherichia coli* par exemple), ces fichiers sont téléchargés dans un dossier temporaire, qui sera supprimé une fois les séquences réunies en un fichier multifasta.

Afin d'éviter l'apparition de bugs, plusieurs fonctions provenant des classes de Jeutils ont été rajoutées dans la classe Query : la fonction *parseTextFromLine* qui retourne une chaîne de caractères sous format String si validation d'une expression régulière et la fonction *parseID* qui retourne une chaîne de caractères représentant un identifiant. La fonction *parseMove* est utilisée afin de générer un nouveau lien url à partir des retours des fonctions précédentes. La fonction *execute* est ensuite utilisée pour effectuer une requête sur les identifiants trouvés pour chaque enzyme et les confronter à la banque de données protéiques du NCBI. Si un identifiant est trouvé, on télécharge le fichier contenant la séquence protéique sous format FASTA par l'appel à la fonction *download*.

2.2 Inférence des relations d'orthologie et annotation des fichiers SBML

2.2.1 Objectifs

Inférence des séquences orthologues

La recherche des séquences orthologues entre les deux bactéries reposant entièrement sur l'algorithme RSD que nous avons cité précédemment, il nous reste uniquement à fournir les fichiers d'entrées à ce programme, et à s'assurer de son exécution correcte avant de passer à l'étape suivante de notre *pipeline*.
L'algorithme RSD peut être exécuté avec des options permettant de filtrer ses résultats, notamment au niveau de la distance évolutive et de la valeur de e-value nécessaire pour qu'un alignement soit considérée comme acceptable. L'option pour changer les seuils de divergence et de e-value est –de, comme expliqué dans cet exemple :

```
rsd_search −q genome1 −−subject−genome=genome2 −o orthologs.txt
−−de divergence_threshold evalue_threshold
```

Les requêtes effectuées préalablement nous permettent d'obtenir une suite de fichiers contenant des séquences uniques au format FASTA. Nous devons donc réunir toutes ces séquences en un fichier multifasta par bactérie, que nous fournirons au programme RSD. Ce programme sera ensuite exécuté automatiquement, et nous pourrons récupérer le fichier de sortie.

Ajout de l'information d'orthologie dans les fichiers SBML

Après avoir obtenu nos paires d'orthologues, plusieurs possibilités sont envisageables. La première se base sur la création d'un nouveau fichier, alors que la seconde fait appel aux fonctionnalités du logiciel Cytoscape pour gérer et afficher les attributs des graphes.

La première, qui est celle envisagée par nos clients, consiste à fusionner deux fichiers SBML provenant des deux bactéries différentes, afin d'obtenir un fichier SBML dit "enrichi" en ajoutant à chaque *GeneProduct* un attribut représentant la présence ou absence de relation d'orthologie entre les deux organismes. Cette méthode présente à la fois des avantages et des inconvénients. En effet, en obtenant un unique fichier, on s'affranchit d'une étape de fusion des réseaux métaboliques par Cytoscape lui-même. Cela facilite donc les opérations de visualisations directes dans Cytoscape. Cependant, telle qu'elle est, cette méthode ne respecte pas *stricto sensu* la syntaxe SBML (c'est-à- dire que le fichier SBML résultant ne correspondrait à aucune spécification SBML existante). De plus, la réalisation de cette fusion de fichiers en utilisant des bibliothèques peut s'avérer pénible et délicate, la raison principale étant que le niveau d'annotations entre chaque bactéries n'est pas identique, *Escherichia coli* étant bien plus annoté que *Helicobacter pylori* par exemple.

Une deuxième méthode de traitement des fichiers SBML est de créer une annotation spécifique aux informations d'orthologie. Cette annotation serait identique dans les deux fichiers SBML pour chaque couple d'enzymes orthologues, et unique pour les enzymes non-orthologues. Cette méthode est basée sur une recherche en amont sur Cytoscape et la découverte de la fonctionnalité *merge*, qui permettent de visualiser simultanément deux réseaux métaboliques possédant des éléments en commun, sur la base de la valeur d'un attribut. Nous pourrions ainsi utiliser les valeurs de l'attribut orthologue ajouté pour effectuer une fusion des deux réseaux lors de leur visualisation. Cette technique permet de déléguer entièrement la visualisation au logiciel Cytoscape. De plus, elle permet de conserver toute l'information des fichiers existants, pouvant être utile aux clients ultérieurement.

La question se pose alors de l'attribut à modifier ou à rajouter à chaque élément *fbc :GeneProduct*, pour inclure cette information d'orthologie.

— La première solution étant d'utiliser l'attribut *name*, décrit comme optionnel dans la spécification SBML, et d'y ajouter le premier identifiant NCBI suivi du deuxième orthologue en les séparant par un caractère quelconque. De cette façon il est possible dans Cytoscape, en utilisant une expression régulière, de les sélectionner et d'appliquer un code couleur et une forme géométrique qui leur soient propres. Ceci serait acceptable car l'attribut *name* est un label optionnel, qui fait très souvent écho au contenu de l'attribut *label* par simplicité. Nous pouvons donc remplacer son contenu sans pour autant perdre trop d'informations qui seraient utiles dans de futures analyses.

```
<fbc:geneProduct metaid="G_b2779" fbc:id="G_b2779" fbc:name="ortho:NP_417259.1/NP_206953.1" fbc:
    label="b2779">
```

FIGURE 2.1: Exemple de début d'un GeneProduct ou le label *name* représente désormais les relations d'orthologie.

— Une autre solution aurait été d'utiliser la balise <*annotation*>, qui permet de faire référence à des bases de données extérieures. Malheureusement l'utilisation de cette balise, et en particulier de l'élément *rdf :ressource* est très encadré. En effet, le système d'annotation encodé dans les fichiers SBML est basée sur les principes MIRIAM. Les ressources externes d'annotation référencées dans un fichier SBML doivent, pour suivre ces principes, être constituées d'un URI avec 2 parties obligatoires. Un *Namespace* identifiant une collection ainsi qu'un identifiant pour l'objet annoté dans cette collection. Un exemple valide serait http ://identifiers.org/pubmed/100, liant l'objet annoté à l'item 100 de la base de données PubMed.

Les deux méthodes décrites ci-dessus ont le même but, qui est l'ajout d'une information concernant l'orthologie. La difficulté dans les deux cas est donc de réussir à placer dans un fichier *SBML Level 3 Version 1* notre liste de paires d'orthologues, et que cette information soit reconnaissable et utilisable au travers des fonctionnalités de Cytoscape.

2.2.2 Implémentation

RsdResultsParser

La classe *RsdResultsParser* nous permet de faire le lien entre les résultats produits par le programme Python qui exécute l'algorithme de recherche d'orthologues, et notre programme Java. Elle consiste en un parseur, qui va récupérer la liste des orthologues dans le fichier produit par l'algorithme RSD.

SbmlAnnotator

La classe *SbmlAnnotator* utilise également *libSBML*, pour annoter un fichier SBML. Cette annotation consiste en la modification de l'attribut *name* de chaque élément *fbc :GeneProduct*. Dans le cas d'une enzyme correspondant à un couple d'orthologues, l'attribut *name* de chaque élément du couple est remplacé par le texte suivant : "ortho :<Id NCBI de l'enzyme dans le génome de référence>/<Id NCBI de l'enzyme dans le second génome>". Dans le cas d'une enzyme non orthologue, il sera remplacé par : "ref :<Id NCBI>" (s'il s'agit du fichier SBML de référence), ou "query :<Id NCBI> (s'il s'agit du second fichier SBML).

2.3 Fonctionnement du pipeline et visualisation des résultats dans Cytoscape

2.3.1 Fonctionnement du pipeline

La classe principale, *PathwayComparisonProject*, nous permet d'exécuter le programme (Voir Annexe VIII pour le workflow). Pour plus de clarté, nous présenterons ici un exemple d'exécution : la comparaison du réseau métabolique de *Helicobacter pylori* avec celui d'*Escherichia coli* (e-value par défaut pour la recherche d'orthologues). Nous nous intéresserons en particulier dans cet exemple à la visualisation du début de la glycolyse, voie métabolique très conservée au cours de l'évolution (voir Figure 2.8).

Cette exécution commence par les entrées utilisateur :

- Chemin absolu du fichier SBML de l'organisme de référence
- Chemin absolu du fichier SBML du second organisme
- Seuil de divergence évolutive (0.8 par default)
- Seuil de e-value pour le BLAST

Les deux fichiers SBML sont ensuite parsés, et la liste des identifiants d'enzymes est récupérée pour chaque bactérie, grâce à l'appel à la classe *EnzymeFinder* par la fonction *findEnzymes*. (Voir Annexe IV pour les enzymes impliqués dans le début de la glycolyse)

Pour l'ensemble de notre jeu d'essai, nous obtenons les résultats suivants :

Bactérie	Enzymes
E.coli	1259
H.pylori	335
S.enterica	1251
B.subtilis	841

TABLE 2.1: Nombre d'enzymes récupérées pour chaque organisme de notre jeu d'essai

La fonction *enzymesQuery* est ensuite appelée pour chacune des deux listes d'enzymes. Elle fonctionne de la façon suivante : pour chaque enzyme, la classe *Query* est instanciée et la séquence correspondante est téléchargée. Pour chaque bactérie, l'ensemble des séquences de tous les fichiers FASTA est lu et rassemblé en un unique fichier multifasta, grâce à la fonction *makeMultifasta*, qui fait appel aux méthodes de la bibliothèque BioJava.

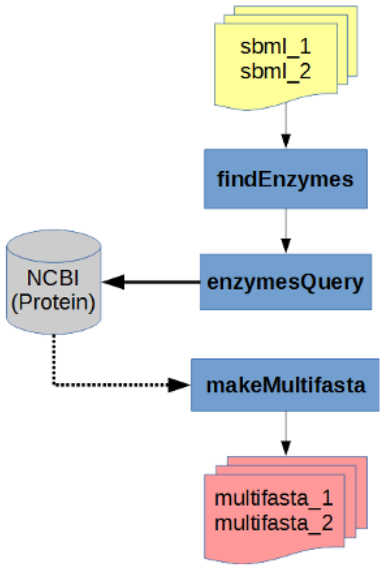

FIGURE 2.2: Première partie du pipeline représentant l'obtention de deux fichiers multiFASTA à partir des deux fichiers SBML

Les deux fichiers multifasta obtenus servent de fichiers d'entrée à l'algorithme RSD. Comme cet algorithme est codé en Python, nous avons utilisé la classe *Runtime* de Java, afin d'exécuter ce programme (commande *rsd_search*) à l'aide d'une commande Bash. La fonction permettant d'exécuter cet algorithme est la fonction *findOrthologs*. A cette étape du programme, l'utilisateur doit reprendre la main et choisir soit les paramètres par défaut de l'algorithme, soit des paramètres spécifiques. Étant donné que les fichiers multifasta sont accessibles à l'utilisateur, il est également possible de faire une vérification manuelle des séquences qui ont été récupérées par notre programme. Après exécution du programme RSD, le fichier de sortie produit est un fichier tabulé donnant dans l'ordre :

- identifiant NCBI de l'enzyme de l'organisme de référence
- identifiant NCBI de l'enzyme du second organisme
- une estimation de la distance de maximum de vraisemblance

(Voir Annexe IV pour les couples d'enzymes orthologues impliqués dans le début de la glycolyse)

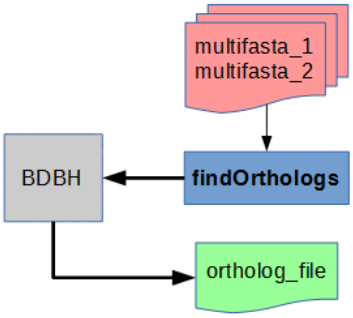

FIGURE 2.3: Deuxième partie du Pipeline représentant l'obtention de deux fichiers orthologues à partir de deux fichiers multiFASTA

Pour finir, la fonction *addOrthologyInfo* permet pour chaque fichier SBML d'ajouter l'information d'orthologie au niveau de l'attribut "name" de chaque *fbc :GeneProduct*. Cela se fait en deux étapes. Pre-

mièrement, le fichier contenant la liste des séquences orthologues est parsé, à l'aide de la classe *RsdResultsParser*, pour récupérer deux listes d'enzymes orthologues de tailles identiques. Chaque partie d'un couple d'orthologues a donc le même indice dans chacune des deux listes, ce qui permettra de les ré-associer. Ensuite, les deux fichiers SBML initiaux sont parsés une seconde fois, cette fois-ci en ajoutant l'information d'orthologie ou d'absence d'orthologie pour chaque enzyme, en utilisant la classe *SbmlAnnotator*.

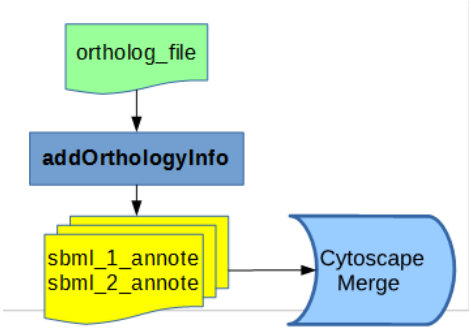

FIGURE 2.4: Avant-dernière étape du Pipeline qui à partir d'une liste d'orthologues et des deux fichiers SBML retourne deux fichiers SBML modifiés avec l'ajout de l'élément orthologue

Comme nous ajoutons ici nos propres annotations à ces fichiers SBML, ces nouvelles informations ne remplacent pas celles des fichiers originaux, mais sont effectuées sur des copies de ces fichiers SBML. Ainsi, nos clients pourront avoir accès à toutes les informations des fichiers SBML téléchargés sur la base de données BiGG, auxquelles s'ajoutent les informations d'orthologie calculées par l'algorithme RSD.(Voir Annexe IV pour les enzymes impliquées dans le début de la glycolyse)

Après avoir appliqué notre programme à notre jeu d'essai, avec les paramètres par défaut, nous avons pu obtenir les résultats suivants (nous rappelons que certaines enzymes peuvent manquer, soit à cause de l'absence d'annotation faisant référence à leur identifiant GI, soit à cause d'erreurs lors des requêtes au NCBI) :

	Orthologues	Bactérie 1	Bactérie 2
E.coli/H.pylori	259	999	76
E.coli/S.enterica	1058	202	193
E.coli/B.subtilis	466	793	375

TABLE 2.2: Nombre d'enzymes en fonction des relations d'orthologie, dans les trois couples de bactéries constituant notre jeu d'essai

En conclusion, nous pouvons voir que les couples de bactéries qui sont proches, d'un point de vue évolutif, partagent beaucoup plus d'enzymes orthologues d'après les résultats de notre programme, ce qui est logique. Ceci est visible sur la différence de nombre d'orthologues entre *E.Coli/S.Enterica* (proches) et *E.Coli/H.pylori* (plus distant). Les résultats de notre programmes sont donc plausibles, et devraient pouvoir être utilisable pour des analyses plus poussés.

2.3.2 Importation des fichiers SBML et information sur les réseaux métaboliques avec Cytoscape

L'importation des fichiers SBML est réalisée à l'aide d'un plugin Cy3sbml *reader* qui est disponible sur Cytoscape après avoir installé ce dernier dans le menu (*Apps* > *App Manager* > *Install Apps*

> Cy3sbml dans la barre de recherche). L'ajout de ce plugin rajoute des styles qui lui sont propres (cy3sbml et cy3sbml-dark). Les styles utilisés ont un intérêt particulier car ils définissent la couleur, la forme, la taille des nœuds et des arêtes. Ces informations sont relativement importantes en théorie des graphes pour l'application d'algorithme ayant pour but de parcourir des graphes. Cy3sbml est le style par défaut que nous utilisons pour montrer des parties de graphes. Il est avantageux de définir ce style par défaut dans Cytoscape (*Control Panel* > *Make current Styles Default*). Ce dernier est ajouté sous forme de XML dans le dossier *CytoscapeConfiguration* avec pour nom *default_vizmap.xml*. Il est donc ainsi possible de modifier le style directement dans le fichier XML sans passer par l'interface Cytoscape. L'intérêt des styles est de garder un repère propre pour chaque type de noeuds *reaction,species,fbc_or,fbc_and,fbc_geneProduct*.(voir Annexe 2.11)
En utilisant le *Control Panel*, il est possible d'obtenir des données quantitatives sur les gènes, nœuds et arêtes présentées ci-dessous.

	gènes	nœuds	arêtes
E.coli	1261	6221	13998
S.enterica	1271	6562	14777
H.pylori	340	1429	2825
B.subtilis	844	3461	7321

TABLE 2.3: Information sur les différents réseaux métaboliques après importation sur Cytoscape

2.3.3 Visualisation avec Cytoscape

Fusion des réseaux à la visualisation

Pour fusionner deux réseaux métaboliques sur Cytoscape, il est nécessaire d'avoir l'application Cy3sbml disponible (*Apps* > *App Manager* > *install* dans le menu). Une fois les deux SBML importés, la fusion sera réalisée à l'aide de l'outil *Merge* présent à partir de *Tools* dans le menu Cytoscape. (*Tools* > *Merge*).

La fusion s'effectue avec l'outil *Union* au niveau de la fenêtre affichée à l'écran puis s'assurer que les réseaux à fusionner sont bien dans la rubrique *Networks to Merge* en utilisant *add to select*.

La fusion doit se faire au niveau des attributs *name*, il est donc important de vérifier au niveau des options avancées qu'apparaisse *name* dans les deux colonnes *Matching Columns*.

La fusion est finalisée en cliquant sur *Merge*.

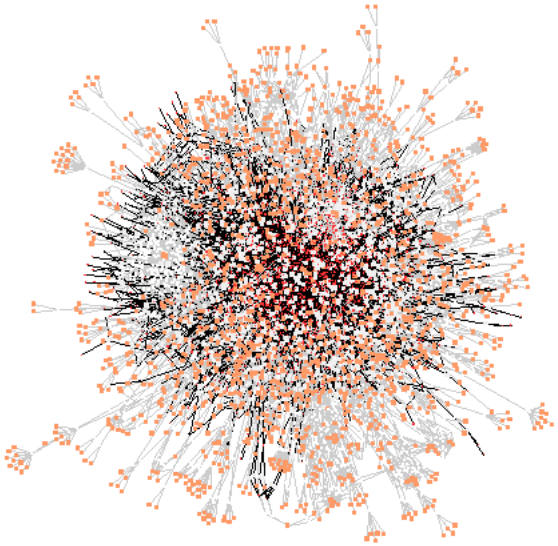

FIGURE 2.5: Visualisation avec Cytoscape d'un réseau métabolique après utilisation de la fonction *merge* entre deux bactéries *H.pylori* et *E.coli*

Une fois la fusion entre deux réseaux métaboliques effectués nous obtenons de nouvelles données quantitatives présentées ci-dessous avec les couples orthologues ajoutés.

	nœuds	arêtes	orthologues
E.coli & S.enterica	7086	18895	1058
E.coli & B.subtilis	7243	18988	466
E.coli & H.pylori	5895	15070	259

TABLE 2.4: Information sur les différents réseaux métaboliques après utilisation de l'outil *merge* sur Cytoscape et après ajout de l'information d'orthologie

Distinction visuelle des groupes d'enzymes en fonction des relations d'orthologie

Pour visualiser les enzymes orthologues et différencier deux bactéries entre elles (le style étant uniforme), différents styles doivent être appliqués (utilisation de *Control Panel - Select et Style* - situé dans la partie gauche de Cytoscape).

L'outil *Select* agit comme une expression régulière et va venir parser le sbml pour différents labels présents au niveau de ce dernier (name,ncbigi ...). En utilisant l'outil *Select* du *Control Panel* et en sélectionnant l'attribut *shared name*(+ > *column filter* > *node : shared name*), il est possible de sélectionner précisément les nœuds orthologues et les nœuds des bactéries (*ref\query*).

Exemple :(En utilisant le style Cy3sbml disponible) Pour appliquer un style unique aux orthologues, il faut tout d'abord sélectionner les nœuds représentant les orthologues, en remplissant « ortho » dans la barre de recherche puis *apply*. Ensuite, il est nécessaire de sélectionner l'outil style et plusieurs *properties* sont affichées. Afin de modifier la couleur des nœuds sélectionnés, il faut cliquer dans le cadrant Byp vide (*Set bypass*). Une fenêtre s'affiche pour choisir un panel de couleurs puis cliquer sur ok.
La tâche sera effectuée dans la *sous-properties Fill Color*.

La même opération est à répéter pour la forme du nœud dans la *sous-properties Shape* (pour valider cliquer au niveau du réseau métabolique). Pour différencier deux bactéries, l'opération précédente est à répéter. Il faut taper « query » dans la barre de recherche dans le but de lui appliquer un nouveau style afin de différencier les nœuds entre les deux bactéries.
Après avoir modifié le style, nous obtenons la visualisation suivante :

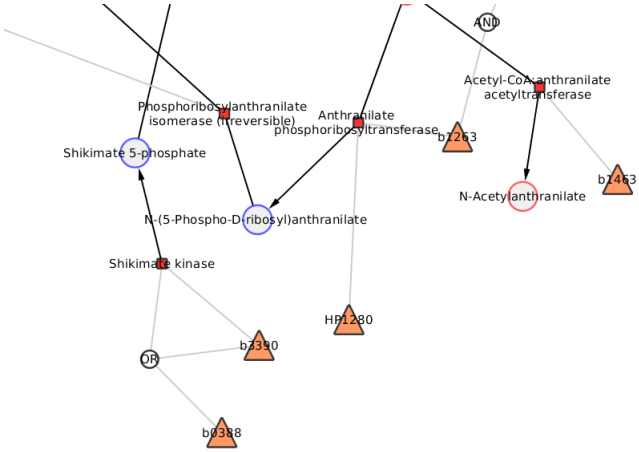

FIGURE 2.6: Visualisation avec Cytoscape sans modification de style après fusion entre *E.coli & H.pylori*

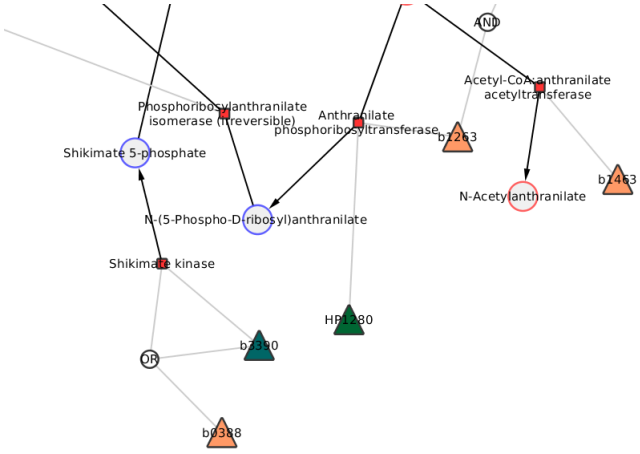

FIGURE 2.7: Visualisation avec Cytoscape après modification du style.Triangles Vert->query, triangles orange->ref, triangles bleus->orthologues

L'indication *query* est propres aux bactéries comparé à *E.coli* c'est à dire *H.pylori,B.subtilis,S.enterica. E.coli* est quand à lui là référence (*ref*)
Pour bien distinguer les *geneproducts* des autres noeuds, Le choix d'une modification des couleurs à été réalisé. Ainsi le code couleur choisis pour les visualisations suivantes est le suivant :

	Code couleur
H.pylori/S.enterica/B.subtilis	Vert
E.coli	Orange
Orthologues	Bleu

TABLE 2.5: Information sur le code couleur employé après utilisation de l'outil *merge* sur Cytoscape et après ajout de l'information d'orthologie

Extraction d'informations correspondantes à des voies métaboliques sur Cytoscape

Une fois le *merge* et la distinction des orthologues entre les deux bactéries effectués. Par la suite, il serait intéressant d'extraire des *pathsway* et visualiser l'inférence des relations d'orthologie sur différentes voies métaboliques communes aux bactéries ou propres à chacune d'entre elles. On va donc chercher à extraire et visualiser des *pathway* sans qu'ils soient perdus dans un trop plein d'informations. Pour ce faire, il faut connaître le nom de chaque enzyme participant à la voie métabolique étudiée. En utilisant BioCyc on peut parvenir à identifier les *pathway* uniques ou communs à un groupe de bactéries. De plus, BioCyc est un outil permettant de récupérer le nom des enzymes pour une multitudes de *pathway* tels que la biosynthèse (d'acides aminés, de sucres, d'hormones ...) ou la dégradation (de protéines, d'acides aminés ...). La comparaison dans BioCyc est réalisée dans la fenêtre (*Analysis* > *Comparative Analysis*) en y indiquant les bactéries à comparer.(voir Annexe 2.17)

Pour extraire un sous-réseau à partir de Cytoscape, Il faut ce servir du *Control Pannel* et de la partie *Select* de cet outil qui permet de définir des expressions régulières. Pour chaque nœud correspondant à une enzyme dans le *pathway* il faudra ajouter (en cliquant sur + >Node : *name, contains* le nom de l'enzyme de manière précise dans l'élément de recherche (voir Annexe 2.13) puis *apply* pour sélectionner les nœuds correspondants aux enzymes. Afin d'éviter de ressaisir à chaque lancement de Cytoscape le nom des enzymes pour un même *pathway* il est possible d'extraire le filtre et de l'importer. Le choix des enzymes afin d'extraire des voies métaboliques est expliqué par le graphe. En effet, le nœud correspondant au nom de l'enzyme est central dans une réaction.
La deuxième étape correspond à la sélection des arêtes adjacentes à nos nœuds afin de compléter le *pathway*. Cette action est effectuée en cliquant au niveau du menu sur la case indiquant deux maisons (*First Neighbors of Selected Nodes*). On évitera de cliquer plusieurs fois sur cet outil afin de ne pas se retrouver avec un graphe trop complexe.
La dernière étape consiste à extraire le graphe dans un nouveau *Network* en cliquant sur l'item à gauche des deux maisons(*New network from selection all edges*). On peut à l'aide de cette méthode établir une relation de comparaison entre les bactéries visuellement. (exemple Glycolyse voir Annexe 2.14).

Signification des nœuds et arêtes sur Cytoscape

En analysant la figure suivante, nous avons un graphe particulier. En effet les arrêtes ne sont qu'un lien et les nœuds peuvent être une réaction ou une enzyme. Avec le style par défaut de Cy3sbml et le code couleur que nous avons défini pour l'orthologie, les différents éléments sont les suivants :

- Triangle : fbc :GeneProduct (gène codant pour une enzyme ou une sous-unité d'une enzyme)
- Cercle : Specie (métabolite)
- Carré : Reaction (réaction enzymatique, portant le nom de l'enzyme associée)

Aussi, deux types d'arêtes peuvent apparaître, des arrêtes grises qui symbolisent une connexion vers un gène codant pour une protéine et des arrêtes noires qui symbolisent un lien entre espèces.

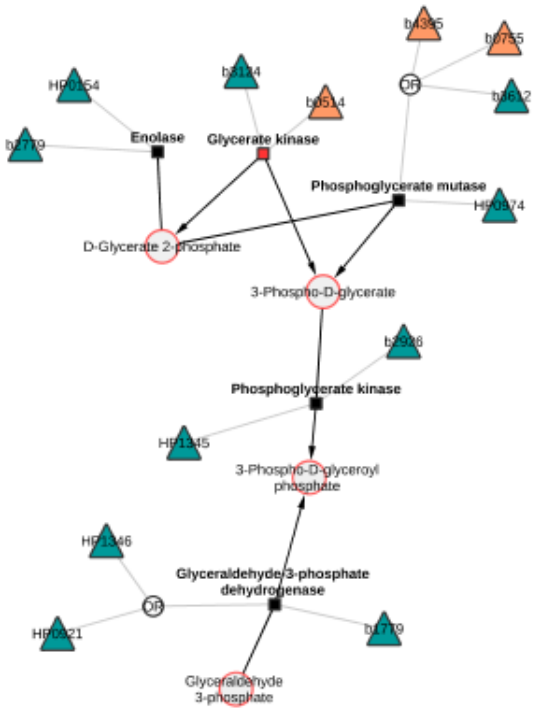

FIGURE 2.8: Visualisation avec Cytoscape des étapes de formation du 3-Phospho-D-Glyceroyl Phosphate via la fusion des réseaux métaboliques de *E.coli* et *H.pylori*
Triange bleu -> gène codant orthologue, triangle orange -> gène codant d'*E.coli*, carré -> réaction enzymatique, cercle -> métabolite, cercle (OR) -> un des gènes lié au nœud peut coder pour une protéine et intervenir dans la réaction

Prenons le cas de la formation du produit *3-Phospho-D-glyceroyl phosphate*. La réaction permettant d'obtenir le produit est l'enzyme *Glyceraldehyde-3-phosphate dehydrogenase* représenté par un petit carré noir. Ce nœud est de degré 4 dont 3 arêtes non-orientées et 1 arête orientée. Les triangles bleus sont des sommets codant pour la protéine *Glyceraldehyde-3-phosphate dehydrogenase*. Ils sont plusieurs car le sommet de droite correspond à un gène propre à *E.coli* et les deux sommets gauches sont propres à *H.pylori*.La couleur de ces sommets est bleue car nous sommes en présence de noeuds orthologues entre les deux bactéries. La dernière arête non-orientée a pour sommet un cercle correspondant au réactif *Glyceraldehyde 3-phosphate* impliqué dans la réaction. Les 3 arêtes non-orientées se rejoignent au niveau du nœud carré et à partir de ce nœud une nouvelle arête orientée forme le produit. En outre on observe quelques enzymes spécifiques à *E.coli* représenté par un triangle orange,une majorité de triangles bleus qui représentent des enzymes orthologues entre *E.coli* et *H.pylori* qui participent à la formation de ce produit, ce résultat semble cohérent car cette voie est très conservée parmi les espèces(autre exemple voir Annexe 2.15).

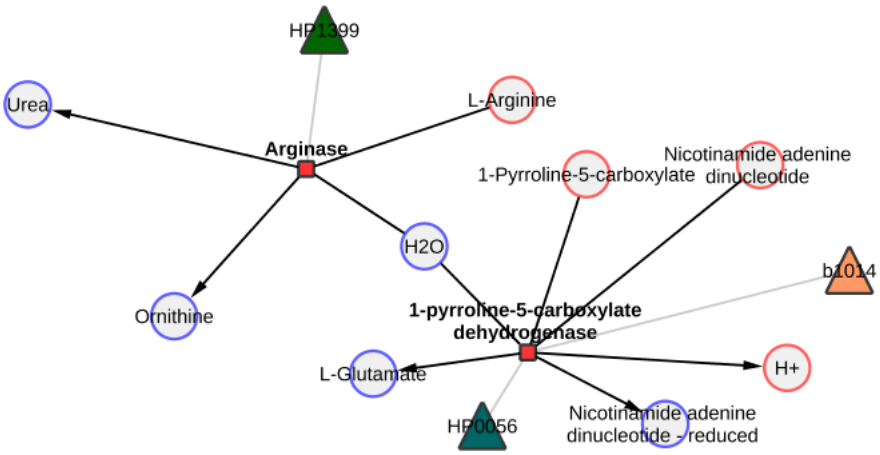

FIGURE 2.9: Visualisation avec Cytoscape de la L-arginine degradation (arginase pathway) fusionnée de *H.pylori* et *E.coli*.
Triangle bleu -> gène codant orthologue, triangle orange -> gène codant d'*E.coli*, carré -> réaction enzymatique, cercle -> métabolite

Sur cette figure on peut visualiser le réseau fusionné entre *H.pylori* et *E.coli* de dégradation de la L-arginine. Les triangles vert représentent des métabolites propres à *H.pylori* et les triangles oranges ceux de *E.coli*. On peut voir que au niveau de l'arginase un seul élément vert est relié, ce qui traduit que l'arginase est propre à *H.pylori*. Un deuxième élément à relever est l'enzyme dehydrogenase qui es relié à un métabolite orthologue représenté par un triangle bleu et un deuxième triangle orange référence à *E.coli*, cela signifie que l'enzyme dehydrogénase est propre aux deux bactéries.

2.4 Perspectives : Plugin Cytoscape dédié aux relations d'orthologie

Dans l'état actuel de notre programme, nous obtenons à la fin de l'exécution de notre *pipeline* deux fichiers aux formats SBML dans lesquels figure l'information d'orthologie. Ces fichiers sont importables dans le logiciel Cytoscape, grâce à l'utilisation du *plugin* Cy3sbml. Une étape de manipulation manuelle des réseaux est donc nécessaire pour que l'utilisateur visualise les deux graphes comparativement avec le code couleur représentant les relations d'orthologie.

Afin d'automatiser l'étape de visualisation, une solution serait d'utiliser les fonctionnalités de l'API de Cytoscape afin d'englober notre programme dans un *plugin* exécutable automatiquement. Cependant, plutôt que de repartir de zéro pour l'importation des fichiers SBML dans Cytoscape, il serait plus judicieux d'apporter des modifications au *plugin* Cy3sbml existant, afin de lui ajouter une fonctionnalité permettant d'inférer ces relations d'orthologie pour un couple de fichiers SBML préalablement importés. L'API de Cytoscape ainsi que le code source de Cy3sbml étant complexes, cela nécessiterait,en plus de temps supplémentaire, de nombreuses discussions avec l'équipe de développement de Cy3sbml.

De plus, le développement d'un *plugin* qui en fonction d'un *pathway* retournerait le résultat d'un filtre avec le nom de chaque enzyme impliquée dans cette voie métabolique serait un gain de temps pour l'analyse et comparaison de réseaux métaboliques.

Conclusion

L'établissement de règles, la réflexion sur papier et la conception au moyen d'un langage informatique sont les trois étapes nécessaires pour aboutir à un logiciel fonctionnel.

Notre projet a eu un début de développement rapide mais l'apprentissage de bibliothèques pour réaliser les différentes tâches du logiciel l'a ralenti. Cependant nous avons su nous adapter aux demandes du client afin d'y répondre dans les délais fournis.

Le code a été pensé et écrit en suivant les bonnes pratiques de programmation.

En outre, notre projet permet à partir de deux fichiers SBML une visualisation comparative de réseaux métaboliques par le bais de la création de fichiers qui peuvent être lus sur Cytoscape et manipulés à notre guise pour réaliser des fusions et des recherches de réseaux. Notre logiciel se comporte comme un *pipeline*, dont la première étape est la récupération d'identifiants des enzymes listées dans chaque fichier SBML. Il s'ensuit une étape de récupération des séquences protéiques des enzymes trouvées, sous la forme d'un fichier multifasta par bactérie. Ensuite, l'utilisation d'un algorithme de type *BDBH* permet de rechercher les couples d'enzymes orthologues parmi les deux listes d'enzymes. Une fois ces couples connus, nous pouvons enfin différencier dans les fichiers SBML les enzymes orthologues des enzymes non-orthologues de par l'utilisation d'un dernier *Parser* qui modifie le label "name" de chaque enzyme de chaque fichier *SBML* afin de différencier les enzymes spécifiques à la première espèce, celles spécifiques à la deuxième espèce, et les enzymes orthologues pour les deux espèces comparées.

Une fois la création de ces 2 nouveaux fichiers *SBML* effectuée, une étape manuelle sur le logiciel Cytoscape permet d'effectuer une fusion de ces 2 fichiers et ainsi obtenir un graphe comparatif des réseaux métaboliques. Un fichier au format XGMML peut être généré à partir de cette fusion et être sauvegardé pour des analyses ultérieures.

En ce qui concerne les modifications qui peuvent être apportées à notre code, de nombreuses implémentations sont possibles. Pour exemple nous pouvons citer l'automatisation de commandes Cytoscape, ce qui permettrait d'éviter les étapes manuelles de fusion. Une autre possibilité serait d'implémenter notre logiciel sur un serveur web ; le client pourrait ainsi obtenir une visualisation de fusion de réseaux métaboliques depuis n'importe quelle machine, en supposant qu'il ait en sa possession deux fichiers sbml. Enfin une dernière perspective serait de modifier notre programme en le transformant en un *plugin* adapté à Cytoscape.

Bibliographie

[1] Kristensen et al. Computational methods for Gene Orthology inference. *Briefings in Bioinformatics*, 12(5) :379–391, September 2011.

[2] Melissa S. Cline, Michael Smoot, Ethan Cerami, Allan Kuchinsky, Nerius Landys, Chris Workman, Rowan Christmas, Iliana Avila-Campilo, Michael Creech, Benjamin Gross, Kristina Hanspers, Ruth Isserlin, Ryan Kelley, Sarah Killcoyne, Samad Lotia, Steven Maere, John Morris, Keiichiro Ono, Vuk Pavlovic, Alexander R. Pico, Aditya Vailaya, Peng-Liang Wang, Annette Adler, Bruce R. Conklin, Leroy Hood, Martin Kuiper, Chris Sander, Ilya Schmulevich, Benno Schwikowski, Guy J. Warner, Trey Ideker, and Gary D. Bader. Integration of biological networks and gene expression data using Cytoscape. *Nature Protocols*, 2(10) :2366–2382, 2007.

[3] Peterson et al. Evolutionary constraints on structural similarity in orthologs and paralogs. *Protein Science : A Publication of the Protein Society*, 18(6) :1306–1315, June 2009.

[4] Vers une méthodologie d'annotation des entités nommées en corpus ?

[5] Escherichia coli str. K-12 substr. MG1655, complete genome. August 2016.

[6] BLAST : Basic Local Alignment Search Tool. https ://blast.ncbi.nlm.nih.gov/Blast.cgi.

[7] Wall et al. Cloud computing for comparative genomics. *BMC Bioinformatics*, 11 :259, 2010.

[8] Patrick Mary David Auber. Data Visualization Software. http ://tulip.labri.fr/TulipDrupal/.

[9] Matthias König, Andreas Dräger, and Hermann-Georg Holzhütter. CySBML : a Cytoscape plugin for SBML. *Bioinformatics (Oxford, England)*, 28(18) :2402–2403, September 2012.

[10] John H. Morris, Leonard Apeltsin, Aaron M. Newman, Jan Baumbach, Tobias Wittkop, Gang Su, Gary D. Bader, and Thomas E. Ferrin. clusterMaker : a multi-algorithm clustering plugin for Cytoscape. *BMC bioinformatics*, 12 :436, November 2011.

[11] EMBL European Bioinformatics Institute. http ://www.ebi.ac.uk/.

[12] National Center for Biotechnology et al. National Center for Biotechnology Information. https ://www.ncbi.nlm.nih.gov/.

[13] Minoru Kanehisa, Susumu Goto, Yoko Sato, Masayuki Kawashima, Miho Furumichi, and Mao Tanabe. Data, information, knowledge and principle : back to metabolism in KEGG. *Nucleic Acids Research*, 42(Database issue) :D199–D205, January 2014.

[14] Overbeek et al. The SEED and the Rapid Annotation of microbial genomes using Subsystems Technology (RAST). *Nucleic Acids Research*, 42(Database issue) :D206–D214, January 2014.

[15] 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, 2016.

[16] XML Schema (W3c), February 2017. Page Version ID : 763143988.

[17] Bacillus subtilis subsp. subtilis str. 168 complete genome. February 2015. { :itemType : dataset} { :version : 3}.

[18] Helicobacter pylori 26695 chromosome, complete genome. August 2016. { :itemType : dataset}.

[19] Salmonella enterica subsp. enterica serovar Typhimurium str. LT2, complete genome. January 2014. { :itemType : dataset}.

[20] Michael Hucka and Lucian P. Smith. SBML Level 3 package : Groups, Version 1 Release 1. *Journal of Integrative Bioinformatics*, 13(3) :290, December 2016.

[21] Brett G. Olivier and Frank T. Bergmann. The Systems Biology Markup Language (SBML) Level 3 Package : Flux Balance Constraints. *Journal of Integrative Bioinformatics*, 12(2) :269, September 2015.

[22] Eric Sayers. *E-utilities Quick Start*. National Center for Biotechnology Information (US), August 2013.

[23] GitHub - todddeluca/reciprocal_smallest_distance : Reciprocal Smallest Distance (RSD) is a pairwise orthology algorithm that uses global sequence alignment and maximum likelihood evolutionary distance between sequences to accurately detects orthologs between genomes. https ://github.com/todddeluca/reciprocal_smallest_distance.

[24] Welcome to Python.org. https ://www.python.org/.

[25] Phylogenetic analysis by maximum likelihood (PAML). http ://abacus.gene.ucl.ac.uk/software/paml.html.

[26] msa.sbc.su.se. http ://msa.sbc.su.se/cgi-bin/msa.cgi.

[27] Timo Lassmann and Erik LL Sonnhammer. Kalign – an accurate and fast multiple sequence alignment algorithm. *BMC Bioinformatics*, 6 :298, 2005.

[28] JeUtils - NCBI eUtils Database Utility in Java. http ://www.algosome.com/.

[29] Andreas Prlić, Andrew Yates, Spencer E. Bliven, Peter W. Rose, Julius Jacobsen, Peter V. Troshin, Mark Chapman, Jianjiong Gao, Chuan Hock Koh, Sylvain Foisy, Richard Holland, Gediminas Rimsa, Michael L. Heuer, H. Brandstätter-Müller, Philip E. Bourne, and Scooter Willis. BioJava : an open-source framework for bioinformatics in 2012. *Bioinformatics (Oxford, England)*, 28(20) :2693–2695, October 2012.

[30] libSBML : A library for SBML. http ://sbml.org/Software/libSBML.

Annexe I : Extrait du fichier SBML d'*Escherichia coli* sans et avec l'ajout de la mention orthologue

```
<fbc:geneProduct fbc:id="G_b2835" fbc:label="b2835" fbc:name="lpLt" metaid="G_b2835">
  <annotation>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:bqbiol="http://biomodels.net/biology-qualifiers/">
      <rdf:Description rdf:about="#G_b2835">
        <bqbiol:is>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/uniprot/P39196" />
          </rdf:Bag>
        </bqbiol:is>
        <bqbiol:isEncodedBy>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/ncbigi/GI:16130739" />
            <rdf:li rdf:resource="http://identifiers.org/ncbigene/947317" />
            <rdf:li rdf:resource="http://identifiers.org/asap/ABE-0009300" />
            <rdf:li rdf:resource="http://identifiers.org/ecogene/EG12455" />
            <rdf:li rdf:resource="http://identifiers.org/kegg.orthology/k001" />
          </rdf:Bag>
        </bqbiol:isEncodedBy>
      </rdf:Description>
    </rdf:RDF>
  </annotation>
</fbc:geneProduct>
```

FIGURE 2.10: Organisation d'un élément fbc :GeneProduct

```
<fbc:geneProduct metaid="G_b4033" fbc:id="G_b4033" fbc:name="ortho:NP_418457.1/CAB15420.1" fbc:label="b4033">
  <annotation>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:bqbiol="http://biomodels.net/biology-qualifiers/">
      <rdf:Description rdf:about="#G_b4033">
        <bqbiol:is>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/uniprot/P02916"/>
          </rdf:Bag>
        </bqbiol:is>
        <bqbiol:isEncodedBy>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/ncbigi/GI:16131859"/>
            <rdf:li rdf:resource="http://identifiers.org/ncbigene/948532"/>
            <rdf:li rdf:resource="http://identifiers.org/asap/ABE-0013195"/>
            <rdf:li rdf:resource="http://identifiers.org/ecogene/EG10555"/>
          </rdf:Bag>
        </bqbiol:isEncodedBy>
      </rdf:Description>
    </rdf:RDF>
  </annotation>
</fbc:geneProduct>
```

FIGURE 2.11: Modification de la balise *fbc :name* optionnel pour y ajouter l'information d'orthologie entre deux bactéries

Annexe II : Visualisation sur Cytoscape

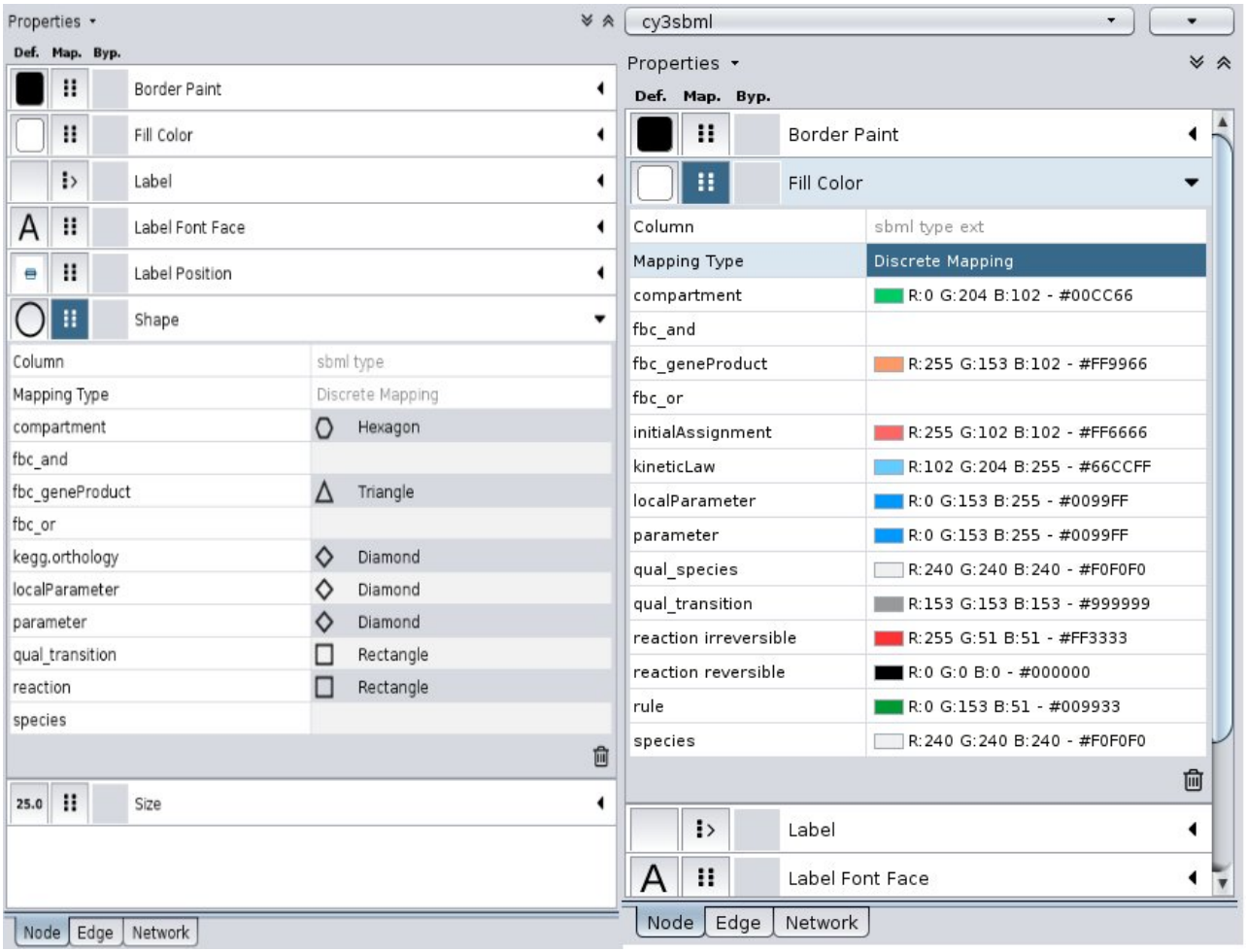

FIGURE 2.12: Organisation de la forme et couleur des nœuds en fonction du style

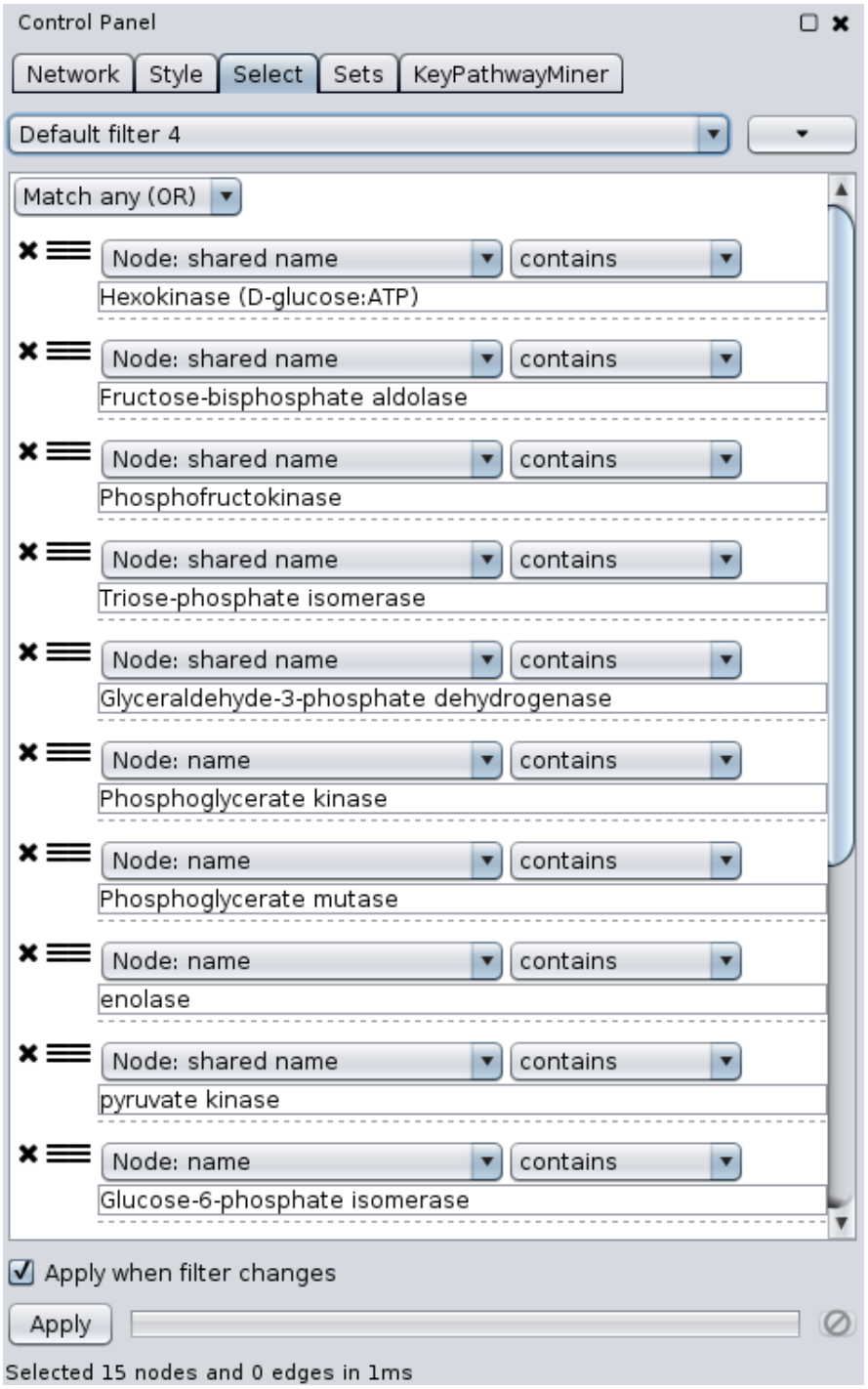

FIGURE 2.13: Exemple d'extraction de pathway sur Cytoscape en utilisant l'outil Select du *Control Panel* avec ajout du nom des enzymes

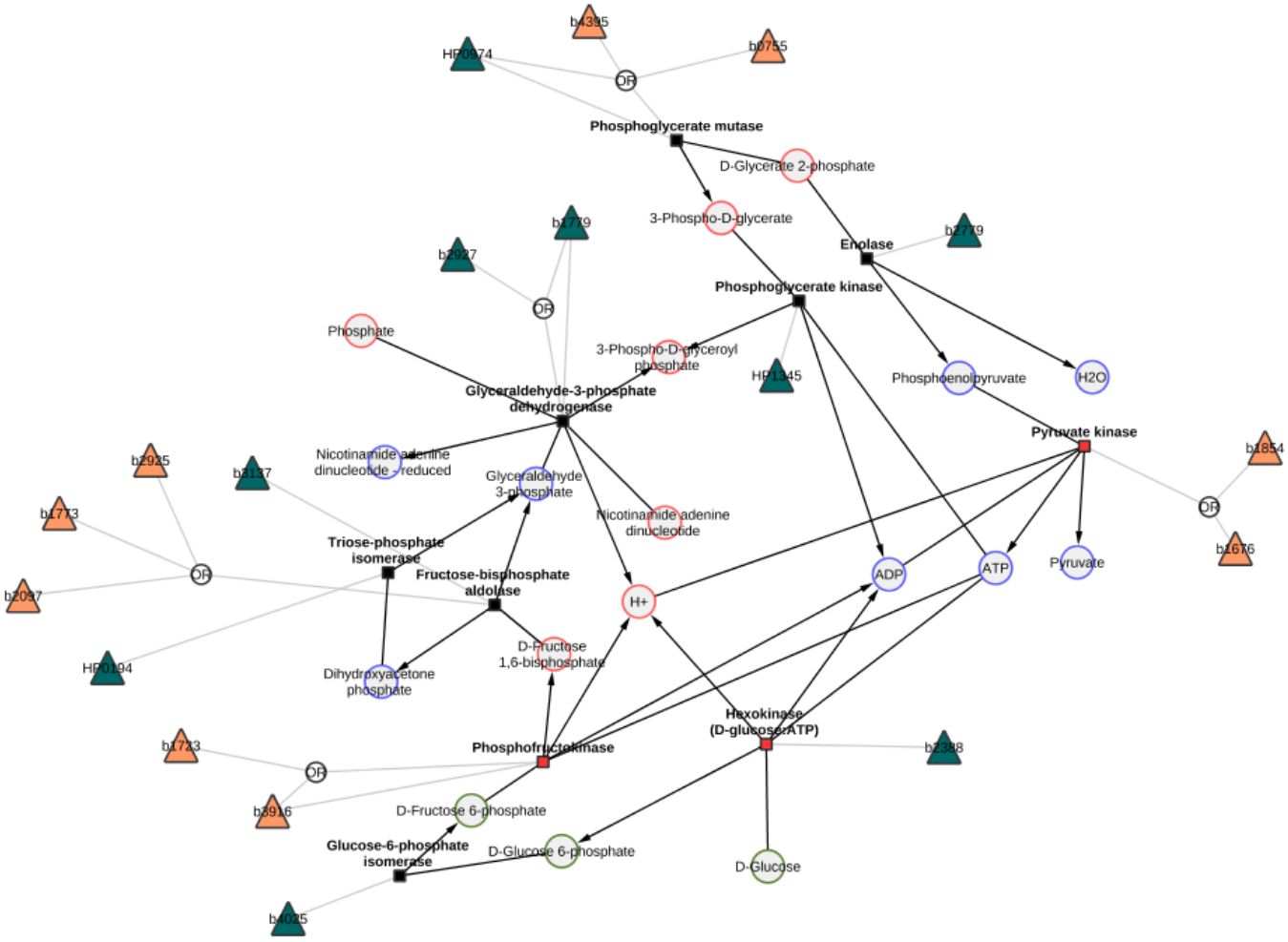

FIGURE 2.14: Visualisation sous Cytoscape du *pathway* de la glycolyse entre *E.coli & H.pylori* les triangles bleus, oranges représentent respectivement les orthologues et les gènes propres à *E.coli*

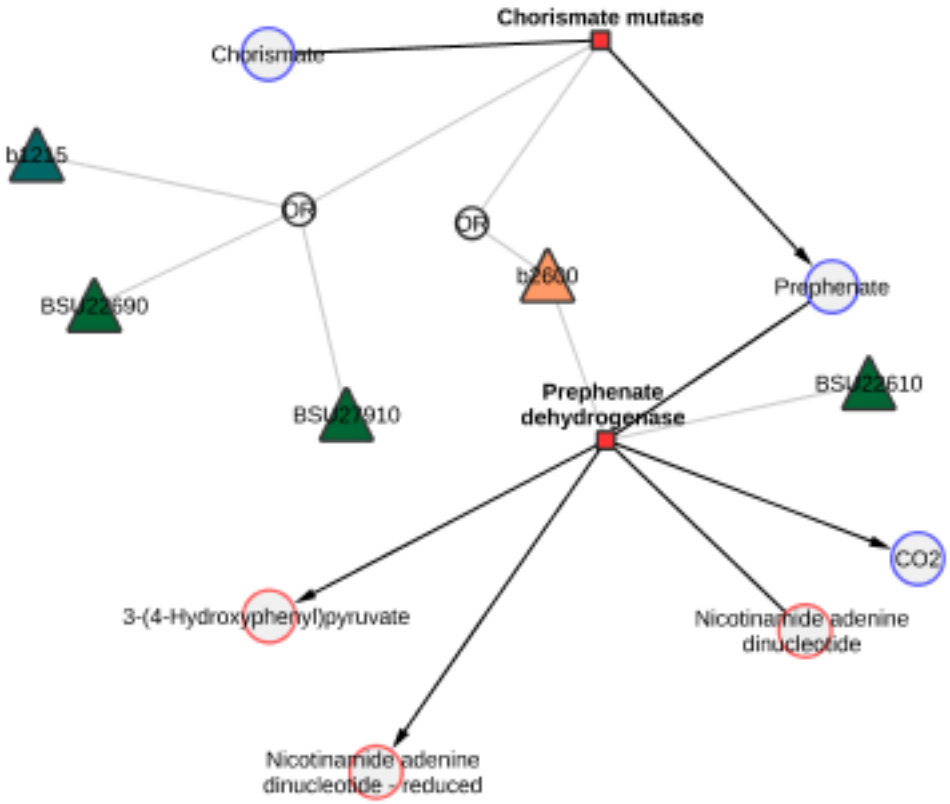

FIGURE 2.15: Pathway partiel de la L-phenylalanine biosynthesis I, dans la visualisation comparative entre *E.Coli & B.subtilis*, chorismate mutase voie conservé entre les deux bactéries

Annexe III : Liste des programmes utilisés pour la réalisation du projet

Les programmes que nous avons utilisés pour réaliser le projet sont les suivantes :

- L'algorithme RSD que nous utilisons [23], utilise la version 2.7 de Python [24] (Nous avons choisi d'utiliser cette version car elle est sous licence MIT, c'est-à-dire libre et *opensource*)
- La version 2.6.0 du BLAST
- La version 4.9 de PAML [25]
- la version 2.04 de Kalign [26] (algorithme d'alignement multiple 10 fois plus rapide que ClustalW [27]).
- Jeutils (The Entrez Programming Utilities for Java) [28]
- La version 4.2.4 de BioJava [29]
- La version 5.15.0 de libSBML [30]
- La version 3.5 de Cytoscape

Annexe IV : Exemple d'utilisation du Pipeline pour étudier la glycolyse

cette partie détaille un exemple concret, la glycolyse, en montrant le traitement d'un fbc :GeneProduct résultant étape par étape.

GeneProduct du SBML E.Coli

```
<fbc:geneProduct metaid="G_b2779" fbc:id="G_b2779" fbc:name="b2779"
   fbc:label="b2779">
  <annotation>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:bqbiol="http://biomodels.net/biology-qualifiers/">
     <rdf:Description rdf:about="#G_b2779">
       <bqbiol:is>
         <rdf:Bag>
           <rdf:li rdf:resource="http://identifiers.org/uniprot/P0A6P9
              "/>
         </rdf:Bag>
       </bqbiol:is>
       <bqbiol:isEncodedBy>
         <rdf:Bag>
           <rdf:li rdf:resource="http://identifiers.org/ncbigi/GI
              :16130686"/>
           <rdf:li rdf:resource="http://identifiers.org/ncbigene
              /945032"/>
           <rdf:li rdf:resource="http://identifiers.org/asap/ABE
              -0009110"/>
           <rdf:li rdf:resource="http://identifiers.org/ecogene/
              EG10258"/>
         </rdf:Bag>
       </bqbiol:isEncodedBy>
     </rdf:Description>
    </rdf:RDF>
  </annotation>
</fbc:geneProduct>
```

Listing 2.1: Retrieval of Individuals

GeneProduct du SBML Heliobacter

```
<fbc:geneProduct metaid="G_HP0154" fbc:id="G_HP0154" fbc:name="HP0154" fbc:label="HP0154">
  <annotation>
```

```
    <rdf:RDF xmlns:rdf="http :// www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:bqbiol="http://biomodels.
        net/biology- qualifiers /">
      <rdf: Description  rdf:about="#G_HP0154">
        <bqbiol:isEncodedBy>
          <rdf:Bag>
            <rdf: li  rdf:resource="http ://  identifiers  .org/ ncbigi /GI:15644783"/>
            <rdf: li  rdf:resource="http ://  identifiers  .org/ncbigene/898806"/>
          </rdf:Bag>
        </bqbiol:isEncodedBy>
      </rdf: Description>
    </rdf:RDF>
  </annotation>
</fbc:geneProduct>
```

GI extrait pour les deux SBML

E.Coli

16130686

on trouve l'identifiant correspondant *NP_417259.1* dans GeneBank

Heliobacter

15644783

on trouve l'identifiant correspondant *NP_206953.1* dans GeneBank

Resultat Orthologie a partir de DB proteique

```
PA  escherichia_coli . fasta    helicobacter_pylori . fasta   0.8   1e-05
OR NP_417259.1 NP_206953.1 0.9736
```

Modification du SBML pour mémoriser l'orthologie

Geneproduct E.coli modifié

```
<fbc:geneProduct  metaid="G_b2779" fbc:id="G_b2779" fbc:name="ortho:NP_417259.1/NP_206953.1" fbc:
    label="b2779">
  <annotation>
    <rdf:RDF xmlns:rdf="http :// www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:bqbiol="http://biomodels.
        net/biology- qualifiers /">
      <rdf: Description  rdf:about="#G_b2779">
        <bqbiol: is>
          <rdf:Bag>
            <rdf: li  rdf:resource="http ://  identifiers  .org/ uniprot /P0A6P9"/>
          </rdf:Bag>
        </bqbiol: is>
        <bqbiol:isEncodedBy>
          <rdf:Bag>
            <rdf: li  rdf:resource="http ://  identifiers  .org/ ncbigi /GI:16130686"/>
            <rdf: li  rdf:resource="http ://  identifiers  .org/ncbigene/945032"/>
            <rdf: li  rdf:resource="http ://  identifiers  .org/asap/ABE-0009110"/>
            <rdf: li  rdf:resource="http ://  identifiers  .org/ecogene/EG10258"/>
          </rdf:Bag>
        </bqbiol:isEncodedBy>
      </rdf: Description>
    </rdf:RDF>
  </annotation>
</fbc:geneProduct>
```

Geneproduct Heliobacter modifié

```
<fbc:geneProduct  metaid="G_HP0154" fbc:id="G_HP0154" fbc:name="ortho:NP_417259.1/NP_206953.1"
    fbc:label="HP0154">
```

```
  <annotation>
    <rdf:RDF xmlns:rdf="http :// www.w3.org/1999/02/22−rdf−syntax−ns#" xmlns:bqbiol="http://biomodels.
        net/biology− qualifiers /">
      <rdf: Description  rdf:about="#G_HP0154">
        <bqbiol:isEncodedBy>
          <rdf:Bag>
            <rdf: li  rdf:resource="http ://  identifiers  .org/ ncbigi /GI:15644783"/>
            <rdf: li  rdf:resource="http ://  identifiers  .org/ncbigene/898806"/>
          </rdf:Bag>
        </bqbiol:isEncodedBy>
      </rdf: Description>
    </rdf:RDF>
  </annotation>
</fbc:geneProduct>
```

Importation dans Cytoscape des SBML complets, annotés, *Merge* sur les label *name*, puis *PathFinder* pour trouver un *pathway* partiel de la glycolyse avec les orthologues.

Annexe V : Code source du programme PathwayComparisonProject

PathwayComparisonProject

```
package com.github.pathway_comparison_project;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ConnectException;
import java.net.UnknownHostException;
import java.util.Map.Entry;
import java.util.*;

import org.biojava.nbio.core.sequence.ProteinSequence;
import org.biojava.nbio.core.sequence.io.FastaReaderHelper;
import org.biojava.nbio.core.sequence.io.FastaWriterHelper;

/**
 * Main program
 *
 * @see Query
 * @see EnzymeFinder
 * @see RsdResultsParser
 * @see SbmlAnnotator
 *
 * @author Peter Bock, Guillamaury Debras, Mercia Ngoma-Komb, Cecilia Ostertag,
 *         Franck Soubes
 */

public class PathwayComparisonProject {

    /**
     * Chargement de la bibliotheque native libsbmlj
     */
    static {
        try {
            System.loadLibrary("sbmlj");
            Class.forName("org.sbml.libsbml.libsbml");
        } catch (UnsatisfiedLinkError e) {
            System.err.println("Error encountered while attempting to load libSBML:");
            System.err
                    .println("Please check the value of your "
                            + (System.getProperty("os.name").startsWith("Mac OS") ? "
                                DYLD_LIBRARY_PATH"
```

```
                    : "LD_LIBRARY_PATH")
                + " environment variable and/or your" + " 'java.library.path' system property (
                    depending on"
                + " which one you are using) to make sure it list the" + " directories needed
                    to find the "
                + System.mapLibraryName("sbmlj") + " library file and"
                + " libraries it depends upon (e.g., the XML parser).");
            System.exit(1);
        } catch (ClassNotFoundException e) {
            System.err.println("Error: unable to load the file 'libsbmlj.jar'."
                    + " It is likely that your -classpath command line "
                    + " setting or your CLASSPATH environment variable " + " do not include the file '
                        libsbmlj.jar'.");
            e.printStackTrace();

            System.exit(1);
        } catch (SecurityException e) {
            System.err.println("Error encountered while attempting to load libSBML:");
            e.printStackTrace();
            System.err.println("Could not load the libSBML library files due to a" + " security
                exception.\n");
            System.exit(1);
        }
    }

    public static void main(String[] args) throws IOException {

        String fastaDirName = "./FastaFiles";
        String tmpDirName = "./Tmp";
        String refName = "ref_sbml.xml";
        String queryName = "query_sbml.xml";
        Hashtable<String, String> corresp1 = new Hashtable<>(); // correspondances
                                                                // ids bigg (GI)
                                                                // /
                                                                // ids ncbi
                                                                // bacterie1
        Hashtable<String, String> corresp2 = new Hashtable<>(); // correspondances
                                                                // ids bigg (GI)
                                                                // /
                                                                // ids ncbi
                                                                // bacterie2
        ArrayList<String> fileNames1 = new ArrayList<String>();
        ArrayList<String> fileNames2 = new ArrayList<String>();
        String sbml1 = null;
        String sbml2 = null;
        String multifasta1 = null;
        String multifasta2 = null;
        String orthologFile = null;

        System.out.println("\tPATHWAY COMPARISON PROJECT\n\n");

        while (true) {
            System.out.println("Chemin du fichier SBML de refeerence : ");
            sbml1 = string_input();
            if (!new File(sbml1).isFile()) {
                System.out.println("Erreur, chemin invalide");
            } else {
                break;
```

```
    }
}
System.out. println ("Nom de l'organisme de reference : ");
String organism1 = string_input ();

while (true) {
    System.out. println ("Chemin du second fichier SBML : ");
    sbml2 = string_input ();
    if (!new File(sbml2). isFile ()) {
        System.out. println ("Erreur, chemin invalide ");
    } else {
        break;
    }
}
System.out. println ("Nom du second organisme : ");
String organism2 = string_input ();

ArrayList<String> biggIdsList1 = findEnzymes(sbml1);

File tmp = new File(tmpDirName);
tmp.mkdir();

long startTime1 = System. currentTimeMillis ();
fileNames1 = enzymesQuery(organism1, biggIdsList1, corresp1, "Tmp/results1");
long estimatedTime1 = System. currentTimeMillis () - startTime1;
System.out. println ("Temps (ms) : " + estimatedTime1);

File dir = new File(fastaDirName);
dir .mkdir();
String outputName1 = "FastaFiles /" + organism1. replaceAll (" ", "_") + ". fasta ";
multifasta1 = makeMultifasta(fileNames1, outputName1);

ArrayList<String> biggIdsList2 = findEnzymes(sbml2);

long startTime2 = System. currentTimeMillis ();
fileNames2 = enzymesQuery(organism2, biggIdsList2, corresp2, "Tmp/results2");
long estimatedTime2 = System. currentTimeMillis () - startTime2;
System.out. println ("Temps (ms) : " + estimatedTime2);

String outputName2 = "FastaFiles /" + organism2. replaceAll (" ", "_") + ". fasta ";
multifasta2 = makeMultifasta(fileNames2, outputName2);

removeDirectory(tmp);

while (true) {
    System.out. println (
            "Voulez-vous utiliser un seuil de divergence et une evalue differentes des valeurs
                par defaut (divergence = 0.8 et evalue = 1e-5) ? (O/N)");
    String ans2 = string_input ();
    if (ans2. equals ("O")) {
        System.out. println ("Valeurs de divergence et evalue (suivre la syntaxe suivante : 0.8 1e
            -5) ");
        String de = string_input ();
        long startTime3 = System. currentTimeMillis ();
        orthologFile = findOrthologs ( multifasta1 , multifasta2 , de);
        long estimatedTime3 = System. currentTimeMillis () - startTime3;
        System.out. println ("Temps (ms) : " + estimatedTime3);
        break;
```

```
        } else if (ans2.equals("N")) {
            long startTime3 = System.currentTimeMillis();
            orthologFile = findOrthologs(multifasta1, multifasta2);
            long estimatedTime3 = System.currentTimeMillis() - startTime3;
            System.out.println("Temps (ms) : " + estimatedTime3);
            break;
        } else {
            System.out.println("Erreur");
        }
    }

    addOrthologyInfo(sbml1, orthologFile, corresp1, refName);
    addOrthologyInfo(sbml2, orthologFile, corresp2, queryName);

    System.out.println("Syntaxe de l'annotation, pour chaque fbc:GeneProduct : \n - Pour les
        orthologues : "
        + "ortho:<Id NCBI de l'enzyme du genome de reference/<Id NCBI de l'enzyme du second
            genome> "
        + "\n- Pour les non-orthologues : [ref|query]:<Id NCBI de l'enzyme du genome courant>");
    System.out.println("Vous pouvez a present visualiser et comparer ces deux reseaux avec
        Cytoscape"
        + " (operation Merge, sur la base de l'attribut 'name' ");
}

/**
 * Removes a non empty directory
 *
 * @param dir
 *            : directory name
 **/
public static void removeDirectory(File dir) {
    if (dir.isDirectory()) {
        File[] files = dir.listFiles();
        if (files != null && files.length > 0) {
            for (File aFile : files) {
                removeDirectory(aFile);
            }
        }
        dir.delete();
    } else {
        dir.delete();
    }
}

/**
 * String user input
 *
 * @return string (String)
 **/
public static String string_input() {

    try {
        BufferedReader buff = new BufferedReader(new InputStreamReader(System.in));
        String string = buff.readLine();
        return string;
    } catch (IOException e) {
        System.out.println(e);
        return null;
```

```
    }
}

/**
 * Parses an sbml file and returns a list of all enzymes (fbc:GeneProduct)
 * NCBI GI ids found in the file
 *
 * @param sbmlFile
 *              : path to the sbml file
 * @return enzymeList : list of all enzymes NCBI GI ids
 **/
public static ArrayList<String> findEnzymes(String sbmlFile) {
    EnzymeFinder finder = new EnzymeFinder(sbmlFile);
     finder.find();
    ArrayList<String> enzymeList = finder.getEnzymeList();
    return enzymeList;
}

/**
 * Reads a fasta file containing proteic sequences
 *
 * @param filename
 *              : name of the fasta file to read from
 * @param seqList
 *              : list of sequences found in the file
 **/
public static void readProtFasta(String filename, ArrayList<ProteinSequence> seqList) {

    LinkedHashMap<String, ProteinSequence> helper;
    try {
        File file = new File(filename);
        helper = FastaReaderHelper.readFastaProteinSequence(file);
        for (Entry<String, ProteinSequence> entry : helper.entrySet()) {
            seqList.add(entry.getValue());
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * Writes a fasta file containing proteic sequences
 *
 * @param filename
 *              : name of the output fasta file
 * @param seqs
 *              : list of sequences that will be written
 **/
public static void writeProtFasta(String filename, ArrayList<ProteinSequence> seqs) {

    try {
        File file = new File(filename);
        FastaWriterHelper.writeProteinSequence(file, seqs);

    } catch (IOException e) {
        e.printStackTrace();
    } catch (Exception e) {
```

```
            System.out.println(e);
        }
    }

    /**
     * Makes a multifasta file from a list of fasta files
     *
     * @param fileNames
     *            : list of fasta files names
     * @param outputName
     *            : name of the resulting multifasta file
     * @return outputName : name of the resulting multifasta file
     **/
    public static String makeMultifasta(ArrayList<String> fileNames, String outputName) {
        ArrayList<ProteinSequence> seqList = new ArrayList<>();
        for (int i = 0; i < fileNames.size(); i++) {
            String name = fileNames.get(i);
            try {
                readProtFasta(name, seqList);
            } catch (Exception e) {
                System.out.println(e);
                System.out.println(name);
            }
        }
        writeProtFasta(outputName, seqList);

        System.out.println("Le fichier multifasta a ete cree : " + outputName);
        return outputName;

    }

    /**
     * Uses RSD algorithm to find ortholog sequences between two sets of proteic
     * sequences in fasta format
     *
     * @param multifasta1
     *            : name of first multifasta file
     * @param multifasta2
     *            : name of first multifasta file
     * @return orthologFile : name of the resulting text file containing the
     *           list of orthologs
     **/
    public static String findOrthologs(String multifasta1, String multifasta2) {

        String orthologFile = "./orthologs.txt";
        Runtime runtime = Runtime.getRuntime();
        String commande = "rsd_search -q " + multifasta1 + " --subject-genome=" + multifasta2 + " -o "
                + orthologFile;
        System.out.println("Recherche d'orthologues en cours ...");
        try {
            Process p = runtime.exec(commande);
            p.waitFor();
            System.out.println("Recherche d'orthologues terminee, voir fichier " + orthologFile);
        } catch (IOException e) {
            System.out.println(e);
        } catch (InterruptedException e) {
            System.out.println(e);
        }
```

```
    return orthologFile;

}

/**
 * Uses RSD algorithm to find ortholog sequences between two sets of proteic
 * sequences in fasta format, with a non default evalue
 *
 * @param multifasta1
 *            : name of first multifasta file
 * @param multifasta2
 *            : name of first multifasta file
 * @param de
 *            : value of divergence and evalue thresholds
 * @return orthologFile : name of the resulting text file containing the
 *           list of orthologs
 **/
public static String findOrthologs(String multifasta1, String multifasta2, String de) {

    String orthologFile = "orthologs.txt";
    Runtime runtime = Runtime.getRuntime();
    String commande = "rsd_search -q " + multifasta1 + " --subject-genome=" + multifasta2 + " -o "
            + "./"
            + orthologFile + " --de " + de;
    try {
        Process p = runtime.exec(commande);
        BufferedReader output = new BufferedReader(new InputStreamReader(p.getInputStream()));
        BufferedReader error = new BufferedReader(new InputStreamReader(p.getErrorStream()));
        String line = "";

        while ((line = output.readLine()) != null) {
            System.out.println(line);
        }

        while ((line = error.readLine()) != null) {
            System.out.println(line);
        }

        p.waitFor();
        System.out.println("Fin de l'execution de l'algorithme RSD. Voir fichier " + orthologFile
                + " si le programme s'est execute correctement");
    } catch (IOException e) {
        System.out.println(e);
    } catch (InterruptedException e) {
        System.out.println(e);
    }
    return orthologFile;
}

/**
 * Makes query at NCBI databank to retrive fasta files associated with NCBI
 * Protein GI corresponding to enzymes
 *
 * @param biggIdsList
 *            : list of BiGG ids (Protein GI) of one bacteria
 * @param corresp
 *            : correspondance between requested BiGG ids and found NCBI ids
 * @param fastaName
```

```
     *                : used to define the names of downloaded fasta files
     * @return fileNames : list of downloaded fasta files ' names
     * @throws IOException
     **/
    public static ArrayList<String> enzymesQuery(String organism, ArrayList<String> biggIdsList,
            Hashtable<String, String> corresp, String fastaName) throws IOException {
        ArrayList<String> fileNames = new ArrayList<String>();
        for (int i = 0; i < biggIdsList.size(); i++) {
            String newFastaName = fastaName + "_" + i + ".fasta";
            String enzymeBiggId = biggIdsList.get(i);
            Query query = new Query(enzymeBiggId);
            try {
                query.execute(newFastaName);
                String enzymeNcbiId = query.getNcbiId();
                String fileName = query.getFileName();
                corresp.put(enzymeBiggId, enzymeNcbiId);
                fileNames.add(fileName);
            } catch (StringIndexOutOfBoundsException e) {
                e.printStackTrace();
            } catch (IllegalArgumentException e) {
                System.out.println("Id non trouve");
                continue;
            } catch (UnknownHostException e) {
                System.out.println("Base de donnees innaccessible, recommencez plus tard");
                System.out.println(e);
                System.exit(1);
            } catch (ConnectException e) {
                System.out.println("Base de donnees innaccessible, recommencez plus tard");
                System.out.println(e);
                System.exit(1);
            } catch (NullPointerException e) {
                System.out.println("Erreur lors de la requete");
                System.out.println(e);
                continue;
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        System.out.println("Toutes les requetes ont ete effectuees");
        return fileNames;
    }

    /**
     * Add the information about orthology in the attribute "name" of each
     * fbc:GeneProduct in the original sbml file, then saves the result in a new
     * sbml file
     *
     * @param sbmlFile
     *                : path to the original sbml file
     * @param orthologFile
     *                : name of the text file containing the list of ortholog
     *                proteins (result of RSD algorithm)
     * @param corresp
     *                : correspondance between requested BiGG ids (GI) and found
     *                NCBI ids
     * @param outputName
     *                : name of the new sbml file
     **/
```

```
    public static void addOrthologyInfo( String sbmlFile, String orthologFile , Hashtable<String,
        String> corresp,
          String outputName) {

        RsdResultsParser parser = new RsdResultsParser( orthologFile );
        parser . findOrthologList ();
        ArrayList<String> lref = parser . getOrthtologListRef ();
        ArrayList<String> lquery = parser . getOrthtologListQuery ();

        SbmlAnnotator annotator = new SbmlAnnotator(sbmlFile, lref , lquery , corresp );
        annotator .annotateName(outputName);

        System.out . println (" Fichier SBML annote");
    }
}
```

EnzymeFinder

```
package com.github.pathway_comparison_project;

import org.sbml.libsbml.FbcModelPlugin;
import org.sbml.libsbml.GeneProduct;
import org.sbml.libsbml.ListOfGeneProducts;
import org.sbml.libsbml.Model;
import org.sbml.libsbml.SBMLDocument;
import org.sbml.libsbml.SBMLReader;

import java.util.ArrayList;

/**
 * This class is used to find a list of all enzymes (fbc:GeneProducts) in a SBML file
 *
 * @see PathwayComparisonProject
 * @see Query
 * @see RsdResultsParser
 * @see SbmlAnnotator
 *
 * @author Peter Bock, Guillamaury Debras, Mercia Ngoma-Komb, Cecilia Ostertag, Franck Soubes
 */

public class EnzymeFinder {

    /*
    * Path to the sbml file
    */
    protected String sbmlFile;

    /*
    * List of NCBI GI ids of all fbc:GeneProduct found in the sbml file
    */
    protected ArrayList<String> enzymeList = new ArrayList<>();

    public EnzymeFinder(String sbmlFile) {
        this.sbmlFile = sbmlFile;
    }

    /*
    * Returns the list of NCBI GI ids of all fbc:GeneProduct found in the sbml file
    * @param enzymeList : List of NCBI GI ids of all fbc:GeneProduct found in the sbml file
    */

    public ArrayList<String> getEnzymeList() {
        return enzymeList;
    }

    /**
     * Parses the SBML file and gets all occurrences of NCBI Protein GI
     *
     **/

    public void find() {

        SBMLReader reader = new SBMLReader();
        SBMLDocument doc = reader.readSBML(sbmlFile);
```

```
        Model model = doc.getModel();
        FbcModelPlugin fbc = (FbcModelPlugin) model.getPlugin("fbc");
        ListOfGeneProducts gpList = fbc.getListOfGeneProducts();

        for (int i = 0; i < gpList.getNumGeneProducts(); i++) {
            GeneProduct gp = gpList.get(i);
            String annotationString = gp.getAnnotationString();
            int indexNcbiGI = annotationString.indexOf("ncbigi/") + 10;
            int indexEnd = annotationString.indexOf("\"", indexNcbiGI);
            if (indexNcbiGI != -1 && indexEnd != -1) {
                String ncbigi = annotationString.substring(indexNcbiGI, indexEnd);
                enzymeList.add(ncbigi);
            } else {
                System.out.println("Le GI n'a pas ete trouve ... ");
            }

        }
        System.out.println(enzymeList.size() + " enzymes ont ete trouvees");
    }

}
```

Query

```
package com.github.pathway_comparison_project;

import org.apache.log4j.BasicConfigurator;
import org.biojava.nbio.core.sequence.ProteinSequence;
import org.biojava.nbio.core.sequence.io.FastaReaderHelper;

import com.algosome.eutils.EntrezSearch;
import com.algosome.eutils.EntrezFetch;
import java.util.*;
import java.util.Map.Entry;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.io.*;
import java.net.*;
import com.algosome.eutils.io.*;

/**
 * This class is used to query NCBI Protein DB for enzyme sequences in fasta
 * format
 *
 * @see PathwayComparisonProject
 * @see EnzymeFinder
 * @see RsdResultsParser
 * @see SbmlAnnotator
 *
 * @author Peter Bock, Guillamaury Debras, Mercia Ngoma-Komb, Cecilia Ostertag,
 *         Franck Soubes
 */

public class Query {

/*
* Enzyme EntrezSearch id
*/
    protected String idstring = "";

    /*
* Url used to download the text file
*/
    protected String finalUrl = "";

    /*
* Enzyme NCBI GI id (called BiggId because the sbml file comes from BiGG database)
*/
    protected String enzymeBiggId;

    /*
* Enzyme NCBI Protein id
*/
    protected String enzymeNcbiId;

    /*
* NCBI database to query
*/
    protected String dataBase;
```

```
    /*
* Name to give to the fasta file when downloading
*/
    protected String fastaFile;

    /**
     * Automated query to NCBI Protein database. Uses a Protein GI id to search
     * the corresponding sequence, and download the file in fasta format
     *
     * @param enzymeBiggId
     *            : Protein GI id corresponding to an enzyme
     **/

    public Query(String enzymeBiggId) {
        this.enzymeBiggId = enzymeBiggId;
        dataBase = EntrezSearch.DB_PROTEIN;
    }

    /**
     * Copied from jeutils EntrezSearch class to avoid bugs. Parses out an XML
     * value tag out of line using a regular expression.
     *
     * @param line
     *            A line of text to parse.
     * @param tag
     *            The XML tag identifier.
     * @return A string representation of the text between the given tags, or an
     *         empty string if nothing was found.
     */
    protected String parseTextFromLine(String line, String tag) {
        Pattern pattern = Pattern.compile(tag + ">(.+)</" + tag);
        Matcher matcher = pattern.matcher(line);
        matcher.find();
        try {
            return matcher.group(1);
        } catch (IllegalStateException ise) {
            return "";
        }
    }

    /**
     * Copied from jeutils EntrezSearch class to avoid bugs. Parses an ID
     * identifier out of the give line.
     *
     * @param line
     *            A line of text to parse.
     * @return A string representation of the ID. This parses any text between
     *         <Id> and </Id>
     * @see com.algosome.eutils.io.InputStreamParser
     */
    protected String parseID(String line) {
        return parseTextFromLine(line, "Id");
    }

    /**
     * Parses an new hyperlink reference out of the give line, in case the
     * document was moved to another url.
     *
```

```
     * @param line
     *                A line of text to parse.
     * @return A string representation of the new url.
     */
    protected String parseMove(String line, String tag) {
        Pattern pattern = Pattern.compile(tag + ">The document has moved <a href=\"(.+)\">here</a>.</"
                + tag);
        Matcher matcher = pattern.matcher(line);
        matcher.find();
        try {
            return matcher.group(1);
        } catch (IllegalStateException ise) {
            return "";
        }
    }

    /**
     * Execution of the query to NCBI Protein databank.
     *
     * @param fastaName
     *                : Name of the downloaded fasta file
     **/

    public void execute(String fastaName) throws Exception {

        // BasicConfigurator.configure();

        EntrezSearch search = new EntrezSearch();

        search.setDatabase(dataBase);

        search.setTerm(enzymeBiggId);

        search.setMaxRetrieval(1);

        InputStreamParser myInputStreamParser1 = new InputStreamParser() {

            public void parseFrom(int start) {
            }

            public void parseTo(int end) {
            }

            public void parseInput(InputStream is) throws IOException {

                String url = "";

                BufferedReader line_reader = new BufferedReader(new InputStreamReader(is));
                String line = null;
                while ((line = line_reader.readLine()) != null) {
                    if (line.indexOf("<Id>") != -1) {
                        String id = parseID(line);
                        if (id != null) {
                            idstring += id + " ";
                        }
                    }
                    // si l'url a change, on remplace la premiere url par la
                    // deuxieme
```

```
            else if (line.indexOf("<p>") != -1) {

                url = parseMove(line, "p");
                is = new URL(url).openStream();
                BufferedReader line_reader2 = new BufferedReader(new InputStreamReader(is));
                String line2 = null;
                while ((line2 = line_reader2.readLine()) != null) {
                    if (line2.indexOf("<Id>") != -1) {
                        String id = parseID(line2);
                        if (id != null) {
                            idstring += id + " ";
                        }
                    }
                }
            }
        }
    }
};
if ((myInputStreamParser1 != null))
    search.doQuery(myInputStreamParser1);
else
    System.out.println("myInputStreamParser1 NULL!");

if (idstring != "") {
    search.setIds(idstring); // on donne directement les ids trouves par
                             // le
    // parseur pour contourner le bug

    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        e.printStackTrace();
        throw new Exception(e);
    }
    try {
        EntrezFetch fetch = new EntrezFetch(search);
        fetch.setRetType("fasta");
        fetch.setRetMode("text");

        InputStreamParser myInputStreamParser2 = new InputStreamParser() {
            public void parseFrom(int start) {
            }

            public void parseTo(int end) {
            }

            public void parseInput(InputStream is) throws IOException {
                BufferedReader br = new BufferedReader(new InputStreamReader(is));
                String line = null;
                while ((line = br.readLine()) != null) {
                    if (line.indexOf("<p>") != -1) {
                        finalUrl = parseMove(line, "p");
                    }
                }
            }
        };

        if ((myInputStreamParser2 != null))
```

```
                fetch.doQuery(myInputStreamParser2);
            else
                System.out.println("myInputStreamParser2 NULL!");

            fastaFile = download(finalUrl, fastaName);
            System.out.println(fastaFile);
            readProtFasta(fastaFile);
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else {
        throw new IllegalArgumentException();
    }
}

/**
 * Download the file in fasta format
 *
 * @param url
 *              : Name of the efetch url used to download the file
 * @param fastaName
 *              : Name of the downloaded fasta file
 **/

public String download(String url, String fastaName) {
    Runtime runtime = Runtime.getRuntime();
    String commande = "wget -O " + fastaName + " " + url;
    try {
        Process p = runtime.exec(commande);
        p.waitFor();
    } catch (IOException e) {
        System.out.println(e);
    } catch (InterruptedException e) {
        System.out.println(e);
    }
    return fastaName;
}

public String getFileName() {
    return fastaFile;
}

/**
 * Reads a fasta file containing proteic sequences
 *
 * @param filename
 *              : name of the fasta file to read from
 * @param seqList
 *              : list of sequences found in the file
 **/

public void readProtFasta(String filename) {
    LinkedHashMap<String, ProteinSequence> helper;
    try {
        File file = new File(filename);
        helper = FastaReaderHelper.readFastaProteinSequence(file);
        for (Entry<String, ProteinSequence> entry : helper.entrySet()) {
            enzymeNcbiId = entry.getValue().getAccession().toString();
```

```
                String [] fields = enzymeNcbiId.split(" ");
                enzymeNcbiId = fields[0];
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Returns the corresponding NCBI id
     *
     * @return enzymeNcbiId : NCBI id
     **/

    public String getNcbiId() {
        return enzymeNcbiId;
    }

}
```

RsdResultsParser

```
package com.github.pathway_comparison_project;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/**
 * This class is used to parse the output file containing the results of the RSD algorithm used to
     find orthologs between two sets of sequences
 *
 * @see Query
 * @see EnzymeFinder
 * @see PathwayComparisonProject
 * @see SbmlAnnotator
 *
 * @author Peter Bock, Guillamaury Debras, Mercia Ngoma-Komb, Cécilia Ostertag, Franck Soubes
 */

public class RsdResultsParser {

/*
* Name of the file containing the results of the RSD algorithm
*/
    private String orthologFile;

    /*
* List of NCBI ids of enzymes from the reference genome
*/
    private ArrayList<String> orthologListRef = new ArrayList<>();

    /*
* List of NCBI ids of enzymes from the subject genome
*/
    private ArrayList<String> orthologListQuery = new ArrayList<>();

    public RsdResultsParser(String orthologFile) {
        this.orthologFile = orthologFile;
    }

    /**
     * Parses the pairs of orthologs written in the orthology results file, and
     * saves them in two ArrayList : one for the reference genome, and the other
     * for the subject genome
     *
     **/
    public void findOrthologList() {
        try {
            BufferedReader buff = new BufferedReader(new FileReader(orthologFile));
            String line = null;
            while ((line = buff.readLine()) != null) {
                String[] fields = line.split("\t");
                if (fields.length > 3) {
                    orthologListRef.add(fields[1]);
                    orthologListQuery.add(fields[2]);
```

```
                }
            }
            buff.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Returns the list of ortholog proteins from the reference genome
     *
     * @return orthologListRef : list of ortholog proteins from the reference genome
     *
     **/

    public ArrayList<String> getOrthtologListRef() {
        return orthologListRef;
    }

    /**
     * Returns the list of ortholog proteins from the subject genome
     *
     * @return orthologListQuery : list of ortholog proteins from the subjecte genome
     *
     **/
    public ArrayList<String> getOrthtologListQuery() {
        return orthologListQuery;
    }

}
```

---

h

SbmlAnnotator

```
package com.github.pathway_comparison_project;

import java.util.ArrayList;
import java.util.Hashtable;

import org.sbml.libsbml.FbcModelPlugin;
import org.sbml.libsbml.GeneProduct;
import org.sbml.libsbml.ListOfGeneProducts;
import org.sbml.libsbml.Model;
import org.sbml.libsbml.SBMLDocument;
import org.sbml.libsbml.SBMLReader;
import org.sbml.libsbml.SBMLWriter;

/**
 * This class is used to add enzyme orthology information to a given sbml file.
 * The information is added in the attribute "name" of each fbc:GeneProduct
 * element of the model. The annotation syntax is as follows : - for orthologs :
 * ortho:<NCBI Id of enzyme from reference genome>/<NCBI Id of enzyme from
 * subject genome> - for non-orthologs : [ref|query]:<NCBI Id of enzyme from
 * current genome>
 *
 * @see Query
 * @see EnzymeFinder
 * @see RsdResultsParser
 * @see PathwayComparisonProject
 *
 * @author Peter Bock, Guillamaury Debras, Mercia Ngoma-Komb, Cécilia Ostertag,
 *         Franck Soubes
 */
public class SbmlAnnotator {

/*
* Path to the sbml file to annotate
*/
    protected String sbmlFile;

    /*
* List of NCBI ids of enzymes from the reference genome
*/
    protected ArrayList<String> orthologListRef = new ArrayList<>();

    /*
* List of NCBI ids of enzymes from the subject genome
*/
    protected ArrayList<String> orthologListQuery = new ArrayList<>();

    /*
* Correspondance between NCBI GI ids of all fbc:GeneProduct and corresponding NCBI ids
*/
    protected Hashtable<String, String> corresp = new Hashtable<>();

    public SbmlAnnotator(String sbmlFile, ArrayList<String> orthologListRef, ArrayList<String>
        orthologListQuery,
          Hashtable<String, String> corresp) {
        this.sbmlFile = sbmlFile;
        this.orthologListRef = orthologListRef;
```

```
        this.orthologListQuery = orthologListQuery;
        this.corresp = corresp;
    }

    /**
     * Parses the sbml file and add annotation to each fbc:GeneProduct, then
     * saves these informations in a copy of the original sbml file
     *
     **/

    public void annotateName(String outputName) {

        String reference = outputName.split("_")[0];
        SBMLReader reader = new SBMLReader();
        SBMLWriter writer = new SBMLWriter();
        SBMLDocument doc = reader.readSBML(sbmlFile);
        Model model = doc.getModel();
        FbcModelPlugin fbc = (FbcModelPlugin) model.getPlugin("fbc");
        ListOfGeneProducts gpList = fbc.getListOfGeneProducts();

        int nb_ortho = 0;
        int nb_other = 0;
        for (int i = 0; i < gpList.getNumGeneProducts(); i++) {
            GeneProduct gp = gpList.get(i);
            String annotationString = gp.getAnnotationString();
            int indexNcbiGI = annotationString.indexOf("ncbigi/") + 10;
            int indexEnd = annotationString.indexOf("\"", indexNcbiGI);
            if (indexNcbiGI != -1 && indexEnd != -1) {
                String ncbigi = annotationString.substring(indexNcbiGI, indexEnd);
                String ncbiId = corresp.get(ncbigi);
                if (orthologListRef.contains(ncbiId)) {
                    nb_ortho++;
                    int index = orthologListRef.indexOf(ncbiId);
                    gp.setName("ortho:" + orthologListRef.get(index) + "/" + orthologListQuery.get(index)
                            );
                } else if (orthologListQuery.contains(ncbiId)) {
                    nb_ortho++;
                    int index = orthologListQuery.indexOf(ncbiId);
                    gp.setName("ortho:" + orthologListRef.get(index) + "/" + orthologListQuery.get(index)
                            );
                } else if (ncbiId != null) {
                    nb_other++;
                    gp.setName(reference + ":" + ncbiId);
                }
            } else {
                nb_other++;
            }
        }
        model.setName(outputName);
        System.out.println("Nombre d'enzymes orthologues : " + nb_ortho + "\nReste : " + nb_other);
        writer.writeSBML(doc, outputName);
    }

}
```

Annexe VI : Diagramme de classes

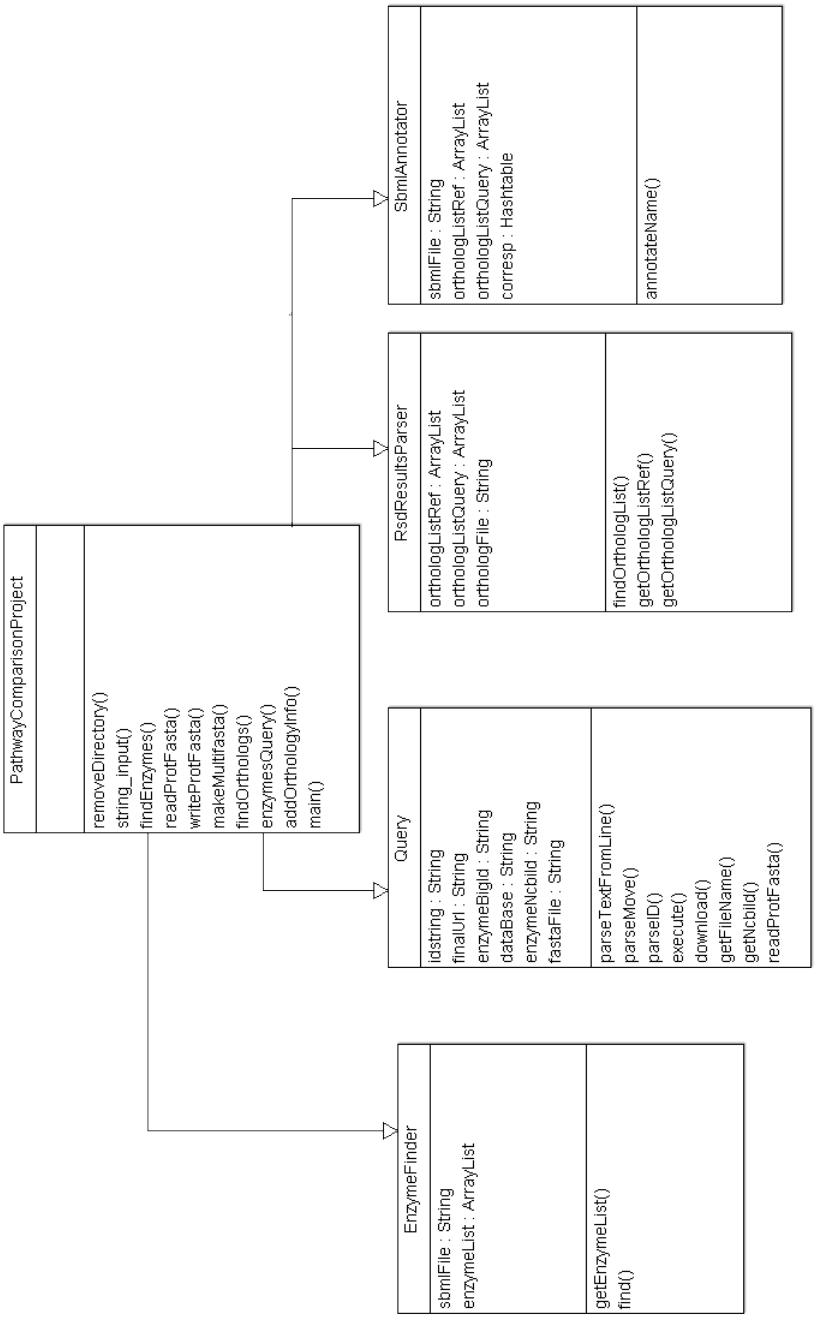

FIGURE 2.16: Diagramme de classe correspondant à notre programme

Annexe VII : BioCyc

Pathway Class: Biosynthesis - Amino Acids Biosynthesis	*E. coli* K-12 substr. MG1655star	*H. pylori* 26695
β-alanine biosynthesis III	X	X
alanine biosynthesis I	X	
alanine biosynthesis III	X	
arginine biosynthesis II (acetyl cycle)	X	
arginine biosynthesis IV (archaebacteria)	X	
arginine degradation I (arginase pathway)		X
arginine degradation VI (arginase 2 pathway)		X
asparagine biosynthesis I	X	
asparagine biosynthesis II	X	
asparagine biosynthesis III (tRNA-dependent)		X
aspartate biosynthesis	X	
cysteine biosynthesis I	X	X
glutamate biosynthesis I	X	
glutamate biosynthesis II	X	X
glutamate biosynthesis III	X	X
glutamate biosynthesis IV	X	
glutamate biosynthesis V	X	
glutamate degradation II	X	X
glutamine biosynthesis I	X	X
glycine biosynthesis I	X	X
glycine biosynthesis IV	X	

FIGURE 2.17: Exemple d'utilisation de BioCyc afin de chercher et comparer des voies métaboliques entre bactéries ici est comparé la biosynthèse d'acides aminés entre *E.coli & H.pylori*

Annexe VIII : Workflow de notre programme

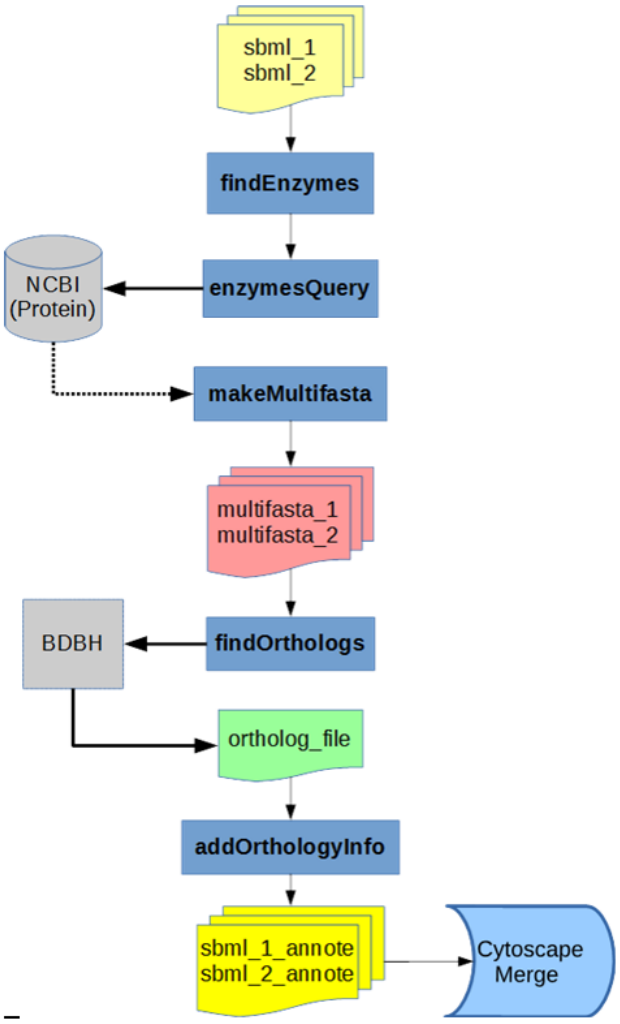

FIGURE 2.18: Workflow présentant les fonctions principales de notre classe principale, PathwayComparisonProject