

ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS  
Département Informatique  
64 avenue Jean Portalis  
37200 Tours, France  
Tél. +33 (0)2 47 36 14 14  
[polytech.univ-tours.fr](http://polytech.univ-tours.fr)

**Projet Recherche & Développement**  
**2017-2018**

**Étude algorithmique d'un problème de  
comparaison de réseaux biologique**



**POLYTECH<sup>®</sup>**  
**TOURS**

**Tuteurs académiques**  
Emmanuel NÉRON  
Ameur SOUKHAL

**Étudiant**  
Yuexin DAI (DI5)

2 avril 2018



## Liste des intervenants

Nom	Email	Qualité
Yuexin DAI	<a href="mailto:yuexin.dai@etu.univ-tours.fr">yuexin.dai@etu.univ-tours.fr</a>	Étudiant DI5
Emmanuel NÉRON	<a href="mailto:emmanuel.neron@univ-tours.fr">emmanuel.neron@univ-tours.fr</a>	Tuteur académique, Département Informatique
Ameur SOUKHAL	<a href="mailto:ameur.soukhal@univ-tours.fr">ameur.soukhal@univ-tours.fr</a>	Tuteur académique, Département Informatique



# Avertissement

Ce document a été rédigé par Yuexin DAI susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Emmanuel NÉRON et Ameer SOUKHAL susnommés les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



## Pour citer ce document

Yuexin DAI, *Étude algorithmique d'un problème de comparaison de réseaux biologique*,  
Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais  
de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
author={DAI, Yuexin},
title={Étude algorithmique d'un problème de comparaison de réseaux biologique},
type={Projet Recherche \& Développement},
school={Ecole Polytechnique de l'Université François Rabelais de Tours},
address={Tours, France},
year={2017-2018}
}
```

# Table des matières

Liste des intervenants	<b>a</b>
Avertissement	<b>b</b>
Pour citer ce document	<b>c</b>
Table des matières	<b>i</b>
Table des figures	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1 Les acteurs du projet sont :.....	1
2 Contexte de la réalisation.....	2
2.1 Contexte .....	2
2.2 Description du problème.....	2
2.3 Objectifs .....	4
2.4 Bases méthodologiques.....	5
<b>2 Description générale</b>	<b>6</b>
1 Environnement du projet .....	6
2 Caractéristiques des utilisateurs .....	6
3 Fonctionnalités et structure générale du système .....	6
<b>3 Etat de l'art/veille</b>	<b>9</b>
1 Définition de l'algorithme .....	9
2 Notion complémentaire.....	9
3 Explication heuristique ALGOH.....	10
4 Explication heuristique ALGOBB .....	15

5	Complexité de ONE-TO-ONE SKEWGRAM.....	16
<b>4</b>	<b>Analyse et conception</b>	<b>18</b>
1	Génération de jeux de données .....	18
2	Automatisation des tests .....	19
3	Amélioration de l'algorithme .....	19
<b>5</b>	<b>Mise en oeuvre</b>	<b>20</b>
1	Fichier de sortie .....	20
2	Modification de ALGOBB .....	20
3	Données difficiles .....	21
<b>6</b>	<b>Conclusion</b>	<b>25</b>
1	Bilan de le semestre 9.....	25
2	Planning de le semestre 10.....	25
	<b>Annexes</b>	<b>27</b>
<b>A</b>	<b>Planification</b>	<b>28</b>
1	Découpage du projet en tâches(S9) .....	28
1.1	Tâche 1 : Étude du problème biologique.....	28
1.2	Tâche 2 : Étude de l'Heuristique .....	28
1.3	Tâche 3 : Étude du code existant .....	29
1.4	Tâche 4 : Rédaction du cahier de spécification .....	29
1.5	Tâche 5 : Validation le cahier de spécification. ....	29
1.6	Tâche 6 : Rédaction du rapport de PRD(S9) .....	29
1.7	Tâche 7 : Préparation de la soutenance de S9 .....	30
2	Découpage du projet en tâches(S10) .....	30
2.1	Tâche 1 : Génération de jeux de données .....	30
2.2	Tâche 2 : Automatisation des tests .....	30
2.3	Tâche 3 : Analyse les résultats du test.....	30
2.4	Tâche 4 : Validation les résultats du test.....	31
2.5	Tâche 5 : Nouvelles fonctions évaluation(pmax) .....	31
2.6	Tâche 6 : Nouvelles fonctions pour choisir extension .....	31
2.7	Tâche 7 : Analyse et comparaison les résultats de l'amélioration de l'algorithme.....	31
2.8	Tâche 8 : Rédaction du rapport final.....	31
2.9	Tâche 9 : Préparation de la soutenance .....	32
3	Le diagramme de Gantt.....	32

<b>B</b>	<b>Spécifications fonctionnelles</b>	<b>34</b>
1	Définition de la fonction 1 : $ccc(G1, G2)$ .....	34
2	Définition de la fonction 2 : $ccc2(G1, G2, L)$ .....	35
3	Définition de la fonction 3 : $est\_pont(G1, G2, L, r)$ .....	36
4	Définition de la fonction 4 : $getCovetSet(D, G, L)$ .....	37
5	Définition de la fonction 5 : $s\_t\_Graph(D, L)$ .....	40
6	Définition de la fonction 6 : $algoH(D, G, L)$ .....	40
<b>C</b>	<b>Spécifications non fonctionnelles</b>	<b>44</b>
1	Contraintes de développement et conception .....	44
2	Contraintes de fonctionnement et d'exploitation .....	44
<b>D</b>	<b>Documentation d'installation</b>	<b>45</b>
<b>E</b>	<b>Cahier développeur</b>	<b>46</b>
1	Description des projets.....	46
2	Fonctionnement Du projet .....	46
3	Description de l'algorithme original .....	47
4	Description de l'algorithme améliorée.....	48
5	Les paramètres pour exécuter ce projet .....	48
6	Les autres fonctions .....	49
<b>F</b>	<b>Cahier de Test</b>	<b>50</b>
	<b>Webographie</b>	<b>55</b>
	<b>Bibliographie</b>	<b>56</b>

# Table des figures

## 1 Introduction

1	Logo du Laboratoire Informatique de Tours.....	1
2	Définition de SKEWGRAM .....	3
3	Exemple des chemins (D, G) - consistants .....	3
4	Définition de ONE-TO-ONE SKEWGRAM .....	4
5	Exemple de ONE-TO-ONE SKEWGRAM avec D et G original .....	4
6	Exemple de ONE-TO-ONE SKEWGRAM avec D et G calculé .....	4

## 2 Description générale

1	Diagramme de cas d'utilisateur .....	7
2	Diagramme d'activité .....	7
3	Les fonctions de partie 1 .....	8
4	Les fonctions de partie 2 .....	8

## 3 Etat de l'art/veille

1	L'algorithme GETCOVERSET.....	11
2	Fonction1 $ccc(G1, G2)$ .....	11
3	Fonction2 $ccc2(G1, G2, L)$ .....	12
4	Fonction3 $est\_pont(G1, G2, L, r)$ .....	12
5	Fonction4 $getCoverSet(D, G, L)$ .....	13
6	L'algorithme ALGOH.....	13
7	L'algorithme ALGOH_1.....	14
8	L'algorithme ALGOH_2.....	14
9	L'algorithme ALGOH_3.....	15



10	L'algorithme de algoBB.....	16
11	La structure de données de l'arbre généré.....	16
12	Complexité de l'algorithme ALGOH .....	17
<b>4</b>	<b>Analyse et conception</b>	
1	Exemple de grapheG de type « Graphe aléatoire » avec N=10 .....	18
2	Exemple de grapheD de type « Graphe aléatoire » avec N=10 .....	18
<b>5</b>	<b>Mise en oeuvre</b>	
1	Comparaison de trois versions .....	21
2	Exemples sans solutions .....	21
3	Exemples sans solutions .....	21
4	Les résultats de parcourir tous les arcs .....	22
5	Les instances qui nécessitent des calculs complexes .....	22
6	Le résultat de la comparaison de deux groups .....	22
7	Les fichiers générés.....	23
8	Les paramètres stockés pour analyser.....	23
9	Le nuage de points de le sommet de "Out-degree" de j et "In-degree de i .....	23
10	Le nuage de points qui ajoute les instances moins de 100 secondes.....	24
11	Le nuage de points qui ajoute les instances moins de 100 secondes.....	24
<b>A</b>	<b>Planification</b>	
1	Le diagramme de Gantt de S9 .....	32
2	Le diagramme de Gantt de S10 .....	32
3	Tableau de synthèse S9.....	33
4	Tableau de synthèse S10.....	33
<b>B</b>	<b>Spécifications fonctionnelles</b>	
1	Exemple de la fonction 1_1 .....	34
2	Exemple de la fonction 1_2 .....	35
3	Exemple de la fonction 1_3 .....	35
4	Exemple de la fonction 2 .....	36
5	Exemple de la fonction 3_1 .....	36
6	Exemple de la fonction 3_2 .....	37
7	Exemple de la fonction 4_1 .....	37
8	Exemple de la fonction 4_2 .....	38
9	Exemple de la fonction 4_3 .....	38
10	Exemple de la fonction 4_4 .....	38

11	Exemple de la fonction 4_5 .....	39
12	Exemple de la fonction 4_6 .....	39
13	Exemple de la fonction 4_7 .....	39
14	Exemple de la fonction 4_8 .....	40
15	Exemple de la fonction 5 .....	40
16	Exemple de la fonction 6_1 .....	41
17	Exemple de la fonction 6_2 .....	41
18	Exemple de la fonction 6_3 .....	42
19	Exemple de la fonction 6_4 .....	42
20	Exemple de la fonction 6_5 .....	42
21	Exemple de la fonction 6_6 .....	43
22	Exemple de la fonction 6_7 .....	43
23	Exemple de la fonction 6_8 .....	43
 <b>E Cahier développeur</b>		
1	L'algorithme de algoH .....	47
2	L'algorithme de algoBB.....	48
 <b>F Cahier de Test</b>		
1	Test unitaire .....	50
2	Résultat de test unitaire .....	50
3	Résultat de algoH et algoBB .....	51
4	Comparaison de trois versions .....	51
5	Exemples sans solutions .....	51
6	Exemples sans solutions .....	52
7	Les fichiers dans 10_1 .....	52
8	Les paramètres stockés par save_log .....	52
9	Les résultats de parcourir tous les arcs .....	53
10	Les instances qui nécessitent des calculs complexes .....	53
11	Le résultat de la comparaison .....	53
12	Les fichiers générés.....	54
13	Les paramètres stockés pour analyser.....	54

# 1

## Introduction

Ce rapport est pour le PROJET RECHERCHE & DÉVELOPPEMENT réalisée dans le cadre de la 5ème année du cycle d'ingénieur à l'École Polytechnique Tours. Le PR&D se déroule du 20 septembre 2017 au 6 avril 2018 à raison de 2 jours par semaine à temps plein. Ce rapport contient le cahier de spécification, l'état de l'art/veille, l'analyse et conception, et la planification de deux semestre.

Le sujet dans ce projet est « Étude algorithmique d'un problème de comparaison de réseaux biologique ». L'expression du besoin de ce projet est réalisée par M. Emmanuel Néron. La supervision de l'avancement de ce projet est réalisée par M. Emmanuel NÉRON et M. Ameer SOUKHAL.

### 1 Les acteurs du projet sont :

#### La maîtrise d'œuvre (MOE) :

DAI Yuexin, étudiante en 5ième année de l'École d'Ingénieur Polytech Tours au sein du département Informatique, dans le cadre du PR&D ainsi que ses encadrants Emmanuel Néron de l'équipe de recherche ROOT ERL-CNRS 6305 du Laboratoire Informatique de Tours.



Figure 1 – Logo du Laboratoire Informatique de Tours

Les recherches menées au sein de cette équipe portent sur l'étude et la résolution des problèmes d'ordonnancement et de planification, que ce soit dans le cadre de problématiques « industrielles » ou de problèmes théoriques. Il s'agit donc de problèmes d'optimisation, bien souvent combinatoires, pour lesquels des techniques de Recherche Opérationnelle et d'Aide à la Décision sont mises en œuvre. ([www.li.univ-tours.fr](http://www.li.univ-tours.fr)).

**La maitrise d'ouvrage (MOA) :** Monsieur Emmanuel Néron qui est Directeur de Polytech Tours (emmanuel.neron@univ-tours.fr) Ce document a été rédigé par DAI Yuexin et sera validé par les différents acteurs du projet.

## 2 Contexte de la réalisation

### 2.1 Contexte

Un réseau biologique est une représentation abstraite d'un système biologique. L'objectif principal de la modélisation par des réseaux est d'étudier les relations entre les composants de ces réseaux pour analyser ou prédire des fonctions biologiques. Les études récentes montrent qu'un système biologique ne peut être compris en se basant uniquement sur la fonction de chacun de ses composants, beaucoup des fonctions étant réalisées par un groupe de composants (par exemple les complexes des protéines, les opérons, etc.)

En bio-informatique, les réseaux biologiques sont souvent modélisés par des graphes dont les sommets sont les composants biologiques et les arêtes représentent leurs interactions. En fonction de la nature de ces réseaux, leurs graphes correspondants peuvent être orientés (par exemple réseaux métaboliques), non orientés (par exemple réseaux d'interaction des protéines) ou mixtes (par exemple réseaux d'interaction protéine-protéine et protéine-ADN)[2].

La comparaison de réseaux biologiques est actuellement l'une des approches les plus prometteuses pour aider à la compréhension du fonctionnement des organismes vivants. Elle apparaît comme la suite attendue de la comparaison de séquences biologiques, dont l'étude a permis le développement de concepts nouveaux et une réelle avancée scientifique, mais qui ne représente en réalité qu'un aspect (l'aspect dit génomique) des informations manipulées par les biologistes.

Nous nous intéressons dans ce projet à une classe particulière des problèmes de comparer les réseaux hétérogènes pour trouver le plus long chemin passant par un arc « xy ». Ces problèmes s'avèrent être NP- complet.

### 2.2 Description du problème

Ce problème est soulevé par Monsieur Hafedh MOHAMED BABOU. L'algorithme que j'ai étudié a également été fait par lui. Donc c'est un projet de coopération avec université de Nantes. Cet algorithme est décrit dans un thèse [4]. Je vais introduire cet algorithme dans les chapitres suivants. Dans ce projet, nous voulons comparer deux réseaux hétérogènes : un réseau principal et un réseau guide. Le premier réseau est représenté par un graphe orienté D (appelé graphe principal) et le deuxième est représenté par un graphe non-orienté G (appelé graphe guide). On suppose qu'il y a une correspondance entre les sommets de D et ceux de G. Cette correspondance est représentée par la fonction suivante.

$$f : V(D) \rightarrow 2^{V(G)} \quad (1)$$

Intuitivement, la fonction  $f$  permet de représenter une similarité entre les sommets selon un critère donné (par exemple similarité entre les séquences de protéines). Exemple : Si D est le graphe des réactions, et G est le graphe d'interaction protéine-protéine pour le même organisme, alors pour tout  $R \in V(D)$ ,  $f(R)$  désigne l'ensemble des protéines dans  $V(G)$  qui catalysent la réaction R.

Notre problème de comparaison est un problème de maximisation appelé Skew SubGraph Mining (abrégé SkewGraM). Le problème SkewGraM est défini comme suit.

**SKEWGRAM**

**Instance :** Un graphe orienté  $D$ , un graphe non-orienté  $G$ ,  
une fonction  $f : V(D) \mapsto 2^{V(G)}$ .

**Solution :** Un chemin  $(D, G, f)$ -consistant.

**Mesure :** La longueur du chemin  $(D, G, f)$ -consistant.

Figure 2 – Définition de SKEWGRAM

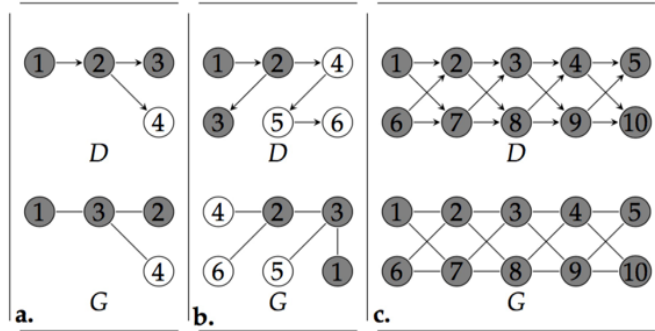
Le problème SkewGraM peut être vu comme étant un problème de recherche des motifs multidimensionnels. Ici le motif est un chemin dans  $D$  et un sous-graphe connexe dans  $G$  tels que chaque sommet du chemin possède au moins un sommet correspondant (par la fonction  $f$ ) dans le sous-graphe connexe. Donc ce problème peut être généralisé en modifiant les motifs de manière appropriée et/ou en introduisant une fonction objective qui mesure l'intérêt des motifs recherchés.

**Définition : Chemin  $(D, G, f)$  - consistant** Soient  $D$  un graphe orienté acyclique (DAG),  $G$  un graphe non-orienté et une fonction de correspondance

$$f : V(D) \rightarrow 2^{V(G)}. \quad (2)$$

Un chemin  $P$  dans  $D$  est dit  $(D, G, f)$  - consistant s'il existe  $X, X \subseteq V(G)$ , tel que :

1.  $X \subseteq \bigcup_{v \in V(P)} f(v)$ ;
2.  $f(v) \cap X \neq \emptyset$  pour tout  $v \in V(P)$ ;
3.  $G[X]$  est connexe.

Figure 3 – Exemple des chemins  $(D, G)$  - consistants**Exemples des chemins  $(D, G)$  - consistants.**

(a) Le plus long chemin  $(D, G)$  -consistant est le chemin  $1 \rightarrow 2 \rightarrow 3$ . (b) Le plus long chemin  $(D, G)$ -consistant est aussi le chemin  $1 \rightarrow 2 \rightarrow 3$ . En revanche, le plus long chemin dans  $D$  est le chemin  $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6$ . Ce chemin n'est pas  $(D, G)$ -consistant, parce que le sous-graphe induit dans  $G$  par l'ensemble de sommets  $1, 2, 4, 5, 6$  n'est pas connexe. (c) Il y a 32 chemins  $(D, G)$ -consistants de longueur 4.

**Le problème ONE-TO-ONE SKEWGRAM**

Pour étudier le problème SkewGraM, nous allons commencer par sa version la plus simple, à savoir le cas où la fonction de correspondance  $f$  est une bijection de l'ensemble  $V(D)$  vers l'ensemble  $v : v \in V(G)$ . Cette version est appelée One-to-One SkewGraM. Dans cette version, nous pouvons, sans perte de généralité, supposer que  $V(D) = V(G) = 1, 2, \dots, n$  et que  $f$  vérifie :  $f(v) = v$ , pour tout  $v \in V(D)$ . Par convention, nous considérons que  $f$  est la fonction identité, c'est-à-dire nous écrivons  $f(v) = v$ , au lieu de  $f(v) = v$ . Dans ce cas, un chemin  $(D, G, f)$  -consistant est dit  $(D, G)$  -consistant.

**ONE-TO-ONE SKEWGRAM**

**Instance :** Un DAG  $D$ , un graphe non-orienté  $G$  ayant le même ensemble de sommets  $\{1, 2, \dots, n\}$ .

**Solution :** Un chemin  $(D, G)$ -consistant.

**Mesure :** La longueur du chemin  $(D, G)$ -consistant.

Figure 4 – Définition de ONE-TO-ONE SKEWGRAM

$D = (V, A)$ ,  $G = (V, E)$ , Si  $f$  vérifie :  $f(v) = v$ , pour tout  $v \in V(D)$  ( $f = \text{id}$ ) Chemin  $(D, G)$ -consistant : chemin  $P$  dans  $D$  tel que  $G[V(P)]$  est connexe.

L'algorithmique sera capable de résoudre Le problème ONE-TO-ONE SKEWGRAM. Un graphe orienté  $D = (V, A)$ , un graphe non-orienté  $G = (V, E)$ .

La question de ce problème est trouver le plus long chemin  $(D, G)$  – consistant.

### Exemple du problème ONE-TO-ONE SKEWGRAM

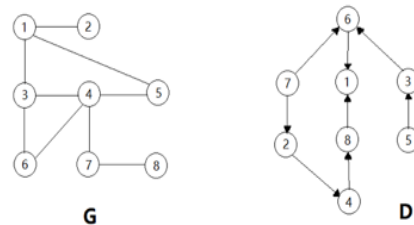


Figure 5 – Exemple de ONE-TO-ONE SKEWGRAM avec  $D$  et  $G$  original

Le plus long chemin dans  $D$  est  $7 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 1$  mais ce n'est pas connexe dans graphe  $G$ . Donc on peut trouver le plus long chemin  $(D, G)$  – consistant dans  $D$  est  $5 \rightarrow 3 \rightarrow 6 \rightarrow 1$

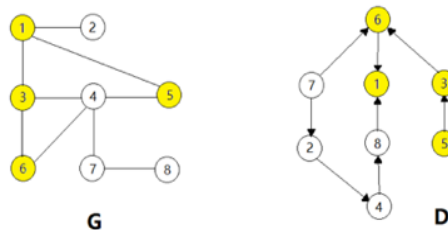


Figure 6 – Exemple de ONE-TO-ONE SKEWGRAM avec  $D$  et  $G$  calculé

## 2.3 Objectifs

L'objectif de ce projet sont :

- Étudier le problème de comparaison de réseaux biologique et comment les traduire graphiquement (ONE-TO-ONE SKEWGRAM).
- Étudier et analyser l'algorithme (« AlgoH » et « AlgoBB ») existant et le code existant pour résoudre le problème « ONE-TO-ONE SKEWGRAM ». (Trouver le plus long chemin passant par un arc «  $xy$  »).
- Écrire du code pour réduire le temps de calcul de l'algorithme « AlgoBB ».

« ALGOH » est l'algorithme de Heuristique, et « ALGOBB » est l'algorithme de séparation et évaluation (branch-and-bound).

## 2.4 Bases méthodologiques

Pour ce projet, le rapport de projet est réalisé à l'aide de Latex.

Dans le cas du développement, on a besoin d'utiliser Python 3.6 avec un package « networkx-V 1.11 »

« networkx »[\[WWW1\]](#) est un package de langage Python pour la création, la manipulation et l'étude de la structure, de la dynamique et de la fonction de réseaux complexes.

Avec NetworkX, vous pouvez charger et stocker des réseaux dans des formats de données standard et non standard, générer de nombreux types de réseaux aléatoires et classiques, analyser la structure du réseau, créer des modèles réseau, concevoir de nouveaux algorithmes réseau, dessiner des réseaux et bien plus encore.

Pour gérer et conduire le projet, j'ai utilisé « GanttProject » pour réaliser la planification.

En terme de méthodologie, le modèle de chute d'eau sera utilisé.

# 2

## Description générale

### 1 Environnement du projet

Lors de la réalisation de ce projet, l'algorithmique (AlgoH) sera mis en place par Python 3.6. Donc ce projet ne dépend d'aucun matériel, seulement l'environnement de Python 3.6. Mais nous avons utilisé plusieurs des fonctions d'un package « networkx-V 1.11 ». Donc ce paquet est obligatoire. Nous devons le télécharger et importer.

Pour l'IDE de Python, j'ai utilisé « PyCharm ». Mais vous pouvez utiliser n'importe quelle IDE.

### 2 Caractéristiques des utilisateurs

La MOA sera l'utilisatrice pour valider ce l'algorithme. Il possède de bonnes connaissances en informatique et en ordonnancement. Donc un seul type d'utilisateur est identifié.

### 3 Fonctionnalités et structure générale du système

Les fonctionnalités dans ce projet sont très simple.

Ce projet n'a que deux fonctions.

La fonction principale consiste à résoudre le problème « ONE – TO – ONE SKEWGRAM ». On peut le dire aussi de trouver le plus long chemin (D, G) – consistant.

Pour résoudre ce problème, il y a deux algorithmes « AlgoH » et « AlgoBB ». En plus, j'ai ajouté un petit algorithme pour trouver les instances difficiles à résoudre pour analyser ce problème.

Les données nécessaires pour ce projet ont un graphique orienté D, un graphe non-orienté G, et un chemin L. Les données peuvent être générées et par l'utilisateur. Et puis, on peut lire les graphes et chemin L par les fichiers générés.

Et pour le résultat, on a le plus long chemin (D, G) – consistant que on a trouvé et aussi les paramètres pour analyser. On peut l'afficher les résultats dans la console ou sortir les résultats dans un fichier.



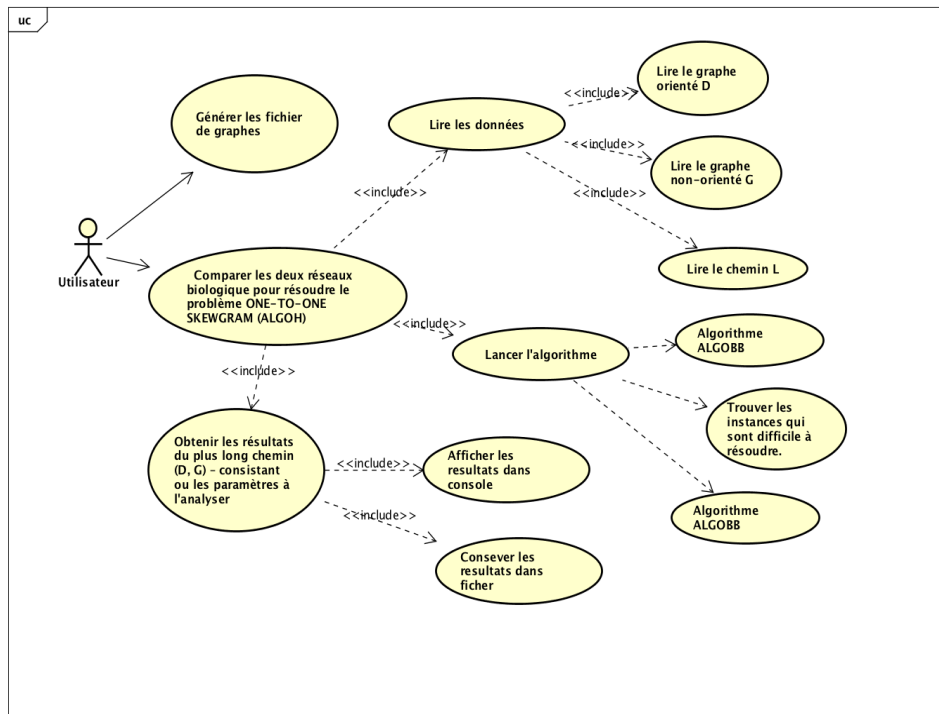


Figure 1 – Diagramme de cas d'utilisateur

Ce projet consiste à étudier et optimiser l'algorithme, et il utilise la bibliothèque « networkx ». On doit utiliser les classes et les méthodes dans « networkx ». Donc, pour savoir les détails du modèle de graphe orienté, le modèle de graphe non-orienté, le chemin L, et les fonctions, on peut trouver sur [WWW2].

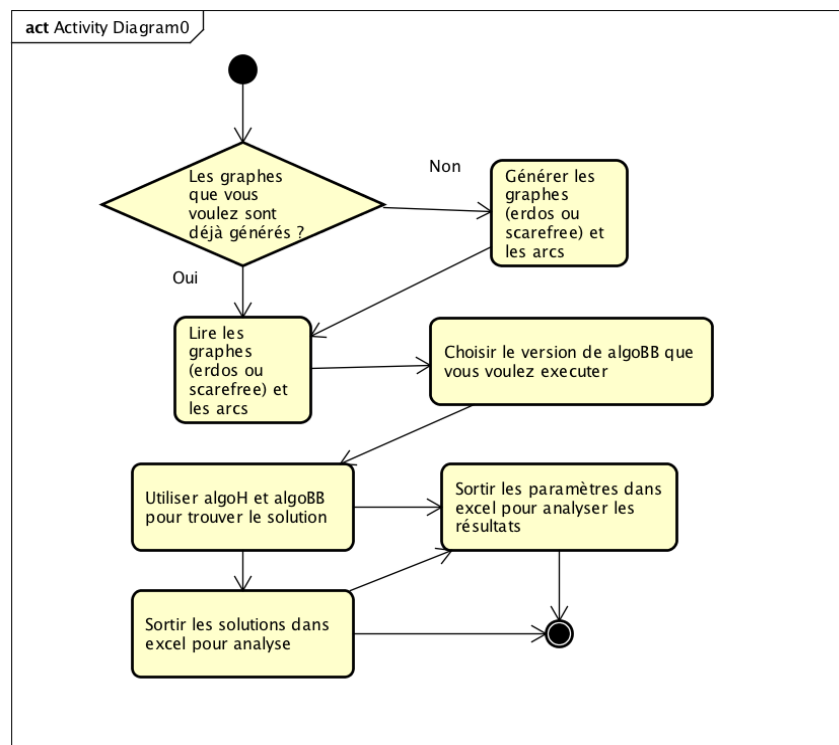


Figure 2 – Diagramme d'activité

Il y a deux types de graphes dans ce projet, « erdos\_renyi » et « scarefree ».

Grâce à l'analyse, nous avons décidé d'améliorer l'algorithme « ALGOBB » pour le graphe de type « erdos\_renyi ».

Après améliorer, il existe trois versions, la version originale et deux versions optimisées.

Dans ce projet, il y a sept fichiers de python. Cet algorithme est composé de 2 partie, et six fonctions. Partie 1 dans ce l'algorithme est « getCoverSet », cette section a quatre fonctions comme ci-dessous.

### **Part 1 getCoverSet**

- CCC (G1, G2)
- CCC2 (G1, G2, L)
- est\_pont (G1, G2, L, r)
- getCoverSet(D, G, L)

**Figure 3** – *Les fonctions de partie 1*

Partie 2 dans ce l'algorithme est « AlgoH », cette section a deux fonctions comme ci-dessous.

### **Part 2 AlgoH(D, G, L)**

- s\_t\_Graph(D, L)
- algoH

**Figure 4** – *Les fonctions de partie 2*

Et je vais continuer à introduire ces fonctions dans la partie « 3.3 Explication heuristique ALGOH » et le partie « 5 Spécification fonctionnelle ».

# 3

## Etat de l'art/veille

Comme j'ai dit dans la partie 1.2.2 « Description du problème », je travaille sur l'algorithme « Heuristique » pour résoudre le problème ONE-TO-ONE SKEWGRAM. Cette partie a donc comme objectif de décrire l'algorithme « ALOGOH ».

Le code existant est également programmé par l'auteur d'un thèse [4], monsieur Hafedh MOHAMED BABOU.

### 1 Définition de l'algorithme

L'idée de cette heuristique est de construire un chemin  $(D, G)$  - consistant de manière progressive, en prolongeant l'arc  $xy$  donné en argument. Nous allons d'abord définir un ensemble couvrant d'un chemin avec l'algorithme « GETCOVERSET ». Ensuite, nous présentons l'algorithme « ALGOH » qui calcule les ensembles couvrants et les utilise pour réduire les graphes  $D$  (graphe orienté) et  $G$  (graphe non-orienté) en supprimant des sommets qui ne peuvent pas prolonger un chemin  $(D, G)$  - consistant.

### 2 Notion complémentaire

Il y a deux réseaux différents : réseaux homogènes et réseaux hétérogènes

Réseaux homogènes : Même type, même type de graphe, espèces différentes  
Réseaux hétérogènes : Type différent, type de graphe peut être différent (e.g. un graphe orienté vs un graphe non-orienté), même espèce.

Un graphe non-orienté  $G$  est un couple formé de deux ensembles : un ensemble  $V$  (noté  $V(G)$ ) dont les éléments sont appelés sommets (vertices, en anglais) et un ensemble  $E$  de paires non-orientées (noté  $E(G)$ ),  $E \subseteq V \times V$ , dont les éléments sont appelés arêtes (edges, en anglais). Le graphe non-orienté  $G$  est noté  $G = (V, E)$ .

Un graphe non-orienté  $G = (V, E)$  est dit connexe si pour tous sommets  $u$  et  $v$ , il existe un chemin reliant  $u$  et  $v$ .

Un graphe orienté  $G$  est un couple formé de deux ensembles : un ensemble  $V$  (noté  $V(G)$ ) dont les éléments sont appelés sommets et un ensemble de paires orientées  $A$  (noté  $A(G)$ ),  $A \subseteq uv : u, v \in V$ , dont les éléments sont appelés arcs. Le graphe orienté  $G$  est noté  $G = (V, A)$ .

Un graphe orienté  $G = (V, A)$  est dit fortement connexe si pour tout  $uv \in V \times V$ , il existe dans  $G$  un chemin de  $u$  vers  $v$  et un chemin de  $v$  vers  $u$ . Le graphe  $G$  est dit faiblement connexe si  $G^*$  est connexe.

Un graphe hamiltonien est un graphe possédant au moins un cycle passant par tous les sommets une fois et une seule. Un tel cycle élémentaire est alors appelé cycle hamiltonien.

**Définition 1 :** Étant donnés deux graphes non-orientés  $G_1 = (U, E_1)$  et  $G_2 = (U, E_2)$ , une composante connexe commune entre  $G_1$  et  $G_2$  est un ensemble maximal  $X \subseteq U$  tel que  $G_1[X]$  et  $G_2[X]$  sont connexes.

**Notion 1 :** On note par  $i \rightsquigarrow^D j$  un chemin dans  $D$  partant du sommet  $i$  vers un sommet  $j$ . Quand  $ij$  est un arc de  $D$ , le chemin  $i \rightsquigarrow^D j$  est noté  $i \rightarrow^D j$ .

**Notion 2 :** On note par  $CCC(D^*, G, i \rightsquigarrow^D j)$  la composante connexe commune entre  $D^*$  et  $G$  qui contient tous les sommets du chemin  $i \rightsquigarrow^D j$ , si une telle composante existe. Si une telle composante n'existe pas, on note  $CCC(D^*, G, i \rightsquigarrow^D j) = \emptyset$ .

**Notion 3 :** On note par  $S_i^+$  l'ensemble de sommets  $j$  dans  $D$  tel qu'il existe un chemin  $i \rightsquigarrow^D j$ .

On note par  $S_i^-$  l'ensemble de sommets  $j$  dans  $D$  tel qu'il existe un chemin  $j \rightsquigarrow^D i$ .

**Définition 2 :** Soit  $r \in V$ . Le sommet  $r$  est dit pont de par rapport à  $G$ , s'il n'y a aucune composante connexe commune entre  $D^*[V-\{r\}]$  et  $G[V-\{r\}]$  contenant tous les sommets du chemin, c'est-à-dire  $CCC(D^*[V-\{r\}], G[V-\{r\}], i \rightsquigarrow^D j) = \emptyset$ .

**Définition 3 :** L'ensemble couvrant d'un chemin  $i \rightsquigarrow^D j$ , noté  $CoverSet(D, G, i \rightsquigarrow^D j)$ , est l'ensemble  $X$  vérifiant les conditions suivantes :

1.  $V(i \rightsquigarrow^D j) \in X \in S_i^- \cup S_j^+ \cup V(i \rightsquigarrow^D j)$ ;
2.  $D^*[X]$  et  $G[X]$  sont connexes;
3. Si  $r$  est un pont de  $i \rightsquigarrow^{D[X]} j$  par rapport à  $G[X]$  alors  $X \in S_r^- \cup S_r^+ \cup \{r\}$ ;
4.  $X$  est maximal (par rapport à l'inclusion) avec les conditions 1., 2. et 3. Si, pour un chemin, il n'y a aucun ensemble  $X$  satisfaisant les conditions 1., 2. et 3., alors par convention  $CoverSet(D, G, i \rightsquigarrow^D j) = \emptyset$ .

L'ensemble couvrant d'un chemin donné est unique, et peut se calculer en un temps polynomial.

### 3 Explication heuristique ALGOH

#### Partie 1 : L'algorithme GETCOVERSET (ensemble couvrant d'un chemin)

Cet algorithme a présenté dans la thèse [4]

Il y a trois règle dans ce l'algorithme :

- Règle 1 : Connexité dans  $D$  (accessibilité);
- Règle 2 : Connexité dans  $G$  (ponts);
- Règle 3 : Connexité dans  $D$  et  $G$  simultanément (composantes connexes communes)

L'algorithme, appelé « GETCOVERSET » est pour calculer l'ensemble couvrant d'un chemin donné.

---

**Algorithme 1**  $\text{GETCOVERSET}(D, G, i \rightsquigarrow^D j)$

---

**Entrées :** Un DAG  $D = (V, A(D))$ , un graphe non-orienté  $G = (V, E(G))$ , un chemin  $i \rightsquigarrow^D j$ .

**But :** Calcul de l'ensemble couvrant de  $i \rightsquigarrow^D j$ .

```

1:  $S := S_i^- \cup S_j^+ \cup V(i \rightsquigarrow^D j)$ ;
2:  $S := \text{CCC}(D^*[S], G[S], i \rightsquigarrow^{D[S]} j)$ ;  $\text{STOP} := \text{faux}$ ;
3: tantque  $((\text{STOP} = \text{faux}) \text{ et } (S \neq \emptyset))$  faire
4:    $S_{\text{tmp}} := S$ ; /* Les chemins  $i \rightsquigarrow^{D[S]} j$  et  $i \rightsquigarrow^D j$  sont identiques */
5:   pour tout pont  $r$  de  $i \rightsquigarrow^D j$  par rapport à  $G$  faire
6:      $S_{\text{tmp}} := S_{\text{tmp}} \cap (\{r\} \cup S_r^- \cup S_r^+)$ ;
7:   fin pour
8:   si  $(S = S_{\text{tmp}})$  alors
9:      $\text{STOP} := \text{vrai}$ ;
10:  sinon
11:     $S := S_{\text{tmp}}$ ;  $S := \text{CCC}(D^*[S], G[S], i \rightsquigarrow^{D[S]} j)$ ;
12:  finsi
13: fin tantque
14: return  $S$ 

```

---

Figure 1 – L'algorithme  $\text{GETCOVERSET}$ .

Pour calculer  $\text{CCC}(D^*[S], G[S], i \rightsquigarrow^D j)$  (lignes 2 et 11), nous utilisons l'algorithme « GEN-PARTREFINEMENT », il présenté dans la thèse [3].

En étudiant le code, j'ai trouvé que le code existant pour cet algorithme est différent de ce qui est écrit dans le document.

Afin de tester et d'améliorer les performances de cet algorithme, l'algorithme de ce projet sera basé sur le code existant.

Donc, l'algorithme de ce rapport sera présenté basé sur le code existant et l'algorithme dans la thèse. Je vais présenter brièvement les deux parties requises pour ce projet dans cette section.

Une description détaillée avec les exemples de ces six fonctions sera écrite en la section « Annexes : B : Spécification fonctionnelle ».

#### Fonction 1 $\text{ccc}(G1, G2)$

```

# composantes connexes communes entre deux graphes
def ccc(G1, G2):
    res = []
    if (nx.is_connected(G1) and nx.is_connected(G2)):
        s = set(G1.nodes())
        res = [s]
        # print("res", res)
        return res
    elif (not nx.is_connected(G2)):
        L2 = nx.connected_components(G2);
        # print("L2", list(L2))

        for u in list(L2):
            # print("u", u)
            res = res + ccc(nx.subgraph(G1, list(u)), nx.subgraph(G2, list(u)))
    else:
        L1 = nx.connected_components(G1);
        for u in list(L1):
            # print("u2", u)
            res = res + ccc(nx.subgraph(G1, list(u)), nx.subgraph(G2, list(u)))
        # print("res", res)
    return res

```

Si G1 et G2 connectent

Si G2 ne connecte pas

Toutes les composantes connexes

Récuratif - Rappeler  $\text{ccc}(G1(u), G2(u))$

G1 ne connecte pas

Figure 2 – Fonction 1  $\text{ccc}(G1, G2)$ 

Cette fonction permet de trouver les composantes connexes communes entre deux graphes non-orienté.

## Fonction 2 ccc2 (G1, G2, L)

```

# composante connexe commune contenant les sommets de L
def ccc2(G1, G2, L):
    C = ccc(G1, G2);
    # print("res1:", C)
    res = {}
    for u in list(C):
        v = set(L)
        if (v.issubset(u)):
            res = u
            break
    # print("res 2", res)
    return res

```

Appeler la méthode ccc pour trouver la liste de connexe commune de G1, G2  
Par exemple.  
res [{1, 4, 5, 6, 8, 11, 12, 15}, {7}, {16}]

Pour toutes les composantes connexes commune vérifier si elle contient L ou pas

Par exemple, si L est {11, 8} Res {1, 4, 5, 6, 8, 11, 12, 15}

Figure 3 – Fonction2 ccc2(G1, G2, L)

Cette fonction permet de trouver la composante connexe commune de deux graphes non-orienté contenant les sommets de L.

Il appelle la fonction 1. Et puis, il va retourner le set de la composante connexe qui content toutes les sommets dans L.

## Fonction 3 est\_pont (G1, G2, L, r)

```

def est_pont(G1, G2, L, r):
    # print("verif est_pont ", r)

    H1 = nx.Graph(G1)
    H2 = nx.Graph(G2)

    H1.remove_node(r)
    H2.remove_node(r)
    a = ccc2(H1, H2, L)
    if (len(a) == 0):
        return True
    else:
        return False

```

Supprimer le nœud r de G1 et G2 -> H1 H2

Vérifier si la composante connexe commune contenant les sommets de L (H1, H2, L) existe ou pas.

S'il n'existe pas la composante connexe retourner « True »

Figure 4 – Fonction3 est\_pont (G1, G2, L, r)

Cette fonction permet de vérifier est-ce que le sommet r est le pont du chemin L entre G2 et G2.

Si le sommet r est le pont, après avoir supprimé le sommet r, on n'a pas la composante connexe commune contenant toutes les sommets de L.

Après nous avons bien compris les trois premières fonctions. Nous pouvons enfin obtenir un ensemble couvrant d'un chemin (coverset) à travers la fonction 4 « getCoverSet ».

**Fonction 4 getCoverSet(D, G, L)**

Cette fonction permet de trouver la composante connexe d'un graphe orienté D et un graphe non-orienté G contenant tous les sommets de L.

D'abord, nous appelons la fonction « ccc2 » pour obtenir S.

$D1 = D[S]$ ,  $G1 = G[S]$ .

Après on a le code comme ci-dessous :

```

while (not STOP and len(S) > 0):
    St = S
    for r in list(S.difference(set(L))):
        x = est_pont(D1.to_undirected(), G1, L, r)
        if x:
            A = {r}
            A = A.union(set(nx.ancestors(D1, r)))
            A = A.union(set(nx.descendants(D1, r)))
            St = St.intersection(A)
    if (St == S):
        STOP = True
    else:
        S = St
        G1 = G1.subgraph(list(S))
        D1 = D1.subgraph(list(S))
        S = ccc2(D1.to_undirected(), G1, L)
return S

```

Pour toutes les sommets dans le liste S sauf L({11, 8}), déterminer si les sommets est pont ou pas avec le méthode .

Si le sommet r est le pont, A est l'ensemble de ancêtres et descendants de r et r

St égal à l'intersection de S et A.

Si St égal à S on arrête, sinon S égal à St, on les refaire.

Figure 5 – Fonction4 getCoverSet(D, G, L)

On a utilisé les deux fonctions de paquet « networks » comme ci-dessous :

« ancestors (G, source) » : Renvoie tous les nœuds ayant un chemin vers « source » dans G.

« descendants (G, source) » : Renvoie tous les nœuds accessibles depuis « source » dans G

Le résultat de cette fonction est le « coverset » S.

**Partie 2 : L'algorithme ALGOH**


---

**Algorithme 2 ALGOH(D, G, xy)**

---

**Entrées :** Un DAG  $D = (V, A(D))$ , un graphe non-orienté  $G = (V, E(G))$ , un arc  $xy \in A(D)$ .

**But :** Calcul d'un ensemble  $S \subseteq V$  tels que  $D[S]$  est un chemin  $(D, G)$ -consistant passant par  $xy$ , ou  $S = \emptyset$ .

1: /\* msc : meilleure solution courante; cc : chemin courant \*/

2: /\* L<sub>ext</sub> : la liste des chemins dans D qui sont obtenus en prolongeant le chemin cc, dans D, par un seul sommet \*/

3: /\* DEC : les sous-graphes de D induits par les ensembles couvrants des chemins dans L<sub>ext</sub> \*/

4: /\* HEC : les sous-graphes Hamiltoniens induits par les ensembles couvrants des chemins dans L<sub>ext</sub> \*/

/\* s (resp. t) est la source (resp. le terminal) du chemin courant cc \*/

5: msc :=  $\emptyset$ ; cc :=  $x \rightarrow^D y$ ; s := x; t := y; STOP := faux;

6: tantque (STOP = faux) faire

7: S := GETCOVERSET(D, G, cc); D := D[S]; G := G[S];

8: si (S =  $\emptyset$  ou D est Hamiltonien) alors

9: STOP := vrai;

10: si |msc| > |S| alors S := msc finsi

11: sinon

12: L<sub>ext</sub> :=  $\emptyset$ ; /\* extension du chemin courant cc =  $s \rightsquigarrow^D t$  \*/

13: pour tout v qui est un prédécesseur de s ou successeur de t faire

14: p = EXTEND(cc, v); /\* extension de cc par v \*/

15: L<sub>ext</sub> := L<sub>ext</sub>  $\cup$  {p};

16: fin pour

17: DEC := {D[COVERSET(D, G, p)] : p  $\in$  L<sub>ext</sub>};

18: HEC := {d  $\in$  DEC : d est Hamiltonien};

19: Soit h<sub>max</sub>  $\in$  HEC t.q. |V(h<sub>max</sub>)| = max{|V(h)| : h  $\in$  HEC}

20: si |msc| < |V(h<sub>max</sub>)| alors msc := V(h<sub>max</sub>) finsi

21: Soit p<sub>max</sub> = s<sub>max</sub>  $\rightsquigarrow^D$  t<sub>max</sub> t.q. value(p<sub>max</sub>) = max{value(p) : p  $\in$  L<sub>ext</sub>};

22: si |msc|  $\geq$  value(p<sub>max</sub>) alors

23: /\* il n'existe aucun chemin (D, G)-consistant passant par xy et de longueur supérieure à msc \*/

24: S := msc; STOP := vrai;

25: sinon

26: cc := p<sub>max</sub>; /\* choix de l'extension la plus prometteuse \*/

27: s := s<sub>max</sub>; t := t<sub>max</sub>

28: finsi

29: finsi

30: fin tantque

31: return S

---

Figure 6 – L'algorithme ALGOH



Propriété : Soit  $p$  un chemin dans  $D$ . Si  $D[\text{CoverSet}(p)]$  est Hamiltonien alors son chemin Hamiltonien est le plus long chemin  $(D, G)$ -consistant contenant  $p$ .

Algorithme 2 ALGOH( $D, G, xy$ )

Entrées : Un DAG  $D = (V, A(D))$ , un graphe non-orienté  $G = (V, E(G))$ , un arc  $xy \in A(D)$ .  
 But : Calcul d'un ensemble  $S \subseteq V$  tels que  $D[S]$  est un chemin  $(D, G)$ -consistant passant par  $xy$ , ou  $S = \emptyset$ .  
 1: /\*  $msc$  : meilleure solution courante;  $cc$  : chemin courant \*/  
 2: /\*  $L_{ext}$  : la liste des chemins dans  $D$  qui sont obtenus en prolongeant le chemin  $cc$ , dans  $D$ , par un seul sommet \*/  
 3: /\*  $DEC$  : les sous-graphes de  $D$  induits par les ensembles couvrants des chemins dans  $L_{ext}$  \*/  
 4: /\*  $HEC$  : les sous-graphes Hamiltoniens induits par les ensembles couvrants des chemins dans  $L_{ext}$  \*/  
 /\*  $s$  (resp.  $t$ ) est la source (resp. le terminal) du chemin courant  $cc$  \*/  
 5:  $msc := \emptyset$ ;  $cc := x \rightarrow^D y$ ;  $s := x$ ;  $t := y$ ;  $STOP := faux$ ;  
 6: tantque ( $STOP = faux$ ) faire  
 7:    $S := \text{GETCOVERSET}(D, G, cc)$ ;  $D := D[S]$ ;  $G := G[S]$ ;  
 8:   si ( $S = \emptyset$  ou  $D$  est Hamiltonien) alors  
 9:      $STOP := vrai$ ;  
 10:   si  $|msc| > |S|$  alors  $S := msc$  fini  
 11:   sinon  
 12:      $L_{ext} := \emptyset$ ; /\* extension du chemin courant  $cc = s \rightsquigarrow^D t$  \*/  
 13:     pour tout  $v$  qui est un prédécesseur de  $s$  ou successeur de  $t$  faire  
 14:        $p = \text{EXTEND}(cc, v)$ ; /\* extension de  $cc$  par  $v$  \*/  
 15:        $L_{ext} := L_{ext} \cup \{p\}$ ;  
 16:     fin pour  
 17:      $DEC := \{D[\text{COVERSET}(D, G, p)] : p \in L_{ext}\}$ ;  
 18:      $HEC := \{d \in DEC : d \text{ est Hamiltonien}\}$ ;  
 19:     Soit  $h_{max} \in HEC$  tq.  $|V(h_{max})| = \max\{|V(h)| : h \in HEC\}$   
 20:     si  $|msc| < |V(h_{max})|$  alors  $msc := V(h_{max})$  fini  
 21:     Soit  $p_{max} = s_{max} \rightsquigarrow^D t_{max}$  tq.  
 22:      $value(p_{max}) = \max\{value(p) : p \in L_{ext}\}$ ;  
 23:     si  $|msc| \geq value(p_{max})$  alors  
 24:       /\* il n'existe aucun chemin  $(D, G)$ -consistant passant par  $xy$  et de longueur supérieure à  $msc$  \*/  
 25:        $S := msc$ ;  $STOP := vrai$ ;

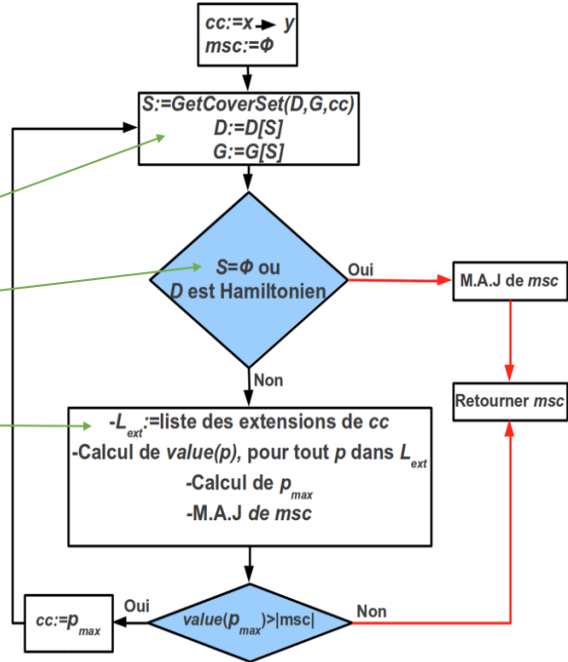


Figure 7 – L'algorithme ALGOH\_1

$V(D) = \emptyset$ .

Dans ce cas, l'algorithme « ALGOH » ne trouve aucun chemin  $(D, G)$ -consistant passant par chemin «  $xy$  ».  $D$  est Hamiltonien. Dans ce cas l'algorithme retourne la meilleure solution courante (l'ensemble  $msc$ ).

$V(D) \neq \emptyset$  et  $D$  n'est pas Hamiltonien.

Dans ce cas, on commence d'abord par calculer la liste  $L_{ext}$  (lignes 12-16) des chemins obtenus en prolongeant le chemin courant  $cc = s \rightarrow t$  par un seul sommet (un prédécesseur de  $s$  ou un successeur de  $t$ ).

```

9:  STOP := vrai;
10: si |msc| > |S| alors S := msc fini
11: sinon
12:   L_ext := ∅; /* extension du chemin courant cc = s ~>^D t */
13:   pour tout v qui est un prédécesseur de s ou successeur de t faire
14:     p = EXTEND(cc, v); /* extension de cc par v */
15:     L_ext := L_ext ∪ {p};
16:   fin pour
17:   DEC := {D[COVERSET(D, G, p)] : p ∈ L_ext};
18:   HEC := {d ∈ DEC : d est Hamiltonien};

```

```

D_ext = s_t_Graph( D_c, ext);
R = ccc2( D_ext.to_undirected(), G_c, ext)
D_ext = D_ext(R)
G_ext = G_c(R)

```

Figure 8 – L'algorithme ALGOH\_2

A la ligne 17, on calcule l'ensemble  $DEC$  des sous-graphes induits dans  $D$  par les chemins de la liste  $L_{ext}$ .

Les graphes de l'ensemble  $HEC$ ,  $HEC \subseteq DEC$ , sont des graphes Hamiltoniens et par conséquent sont des chemins  $(D, G)$ -consistants passant par  $xy$ ; ils sont donc utilisés pour mettre à jour la meilleure solution courante  $msc$ . Le calcul de ces graphes est effectué à la ligne 18, et la mise à jour de la solution courante est effectuée aux lignes 19- 20.

Maintenant, pour choisir une extension parmi celles dans  $L_{ext}$ , on utilise la fonction  $value$  définie comme suit : pour tout  $p \in L_{ext}$ ,  $value(p)$  est la longueur du plus long chemin dans le sous-graphe  $D[\text{CoverSet}(D, G, p)]$ .



L'extension "la plus prometteuse",  $p_{max}$ , (calculée à la ligne 21) est celle qui correspond à un maximum pour la fonction value. Ici également on distingue deux sous-cas :

1. La valeur de l'extension la plus prometteuse est supérieure à la longueur de la meilleure solution courante (lignes 25-28), alors  $cc := p_{max}$ . Ensuite, on passe à la prochaine itération de la boucle « tantque ». C'est à ce moment-là que l'heuristique risque de perdre la solution optimale. Car nous pouvons choisir une extension  $p_{max}$  qui a une très grande valeur, alors qu'il n'y a aucun chemin  $(D, G)$  -consistant qui contient  $p_{max}$  comme sous-chemin.
2. La valeur de l'extension la plus prometteuse est inférieure à la longueur de la meilleure solution courante. Dans ce cas, il n'y aura aucun chemin  $(D, G)$  -consistant passant par  $xy$  dont la longueur est supérieure à longueur de la meilleure solution courante (lignes 22-24). Donc l'algorithme s'arrête et retourne la meilleure solution courante.

Pour calculer le valeur  $p$ ,  $p_{max}$  et  $m_{sc}$ , j'ai étudié par les codes existants comme figure 9.

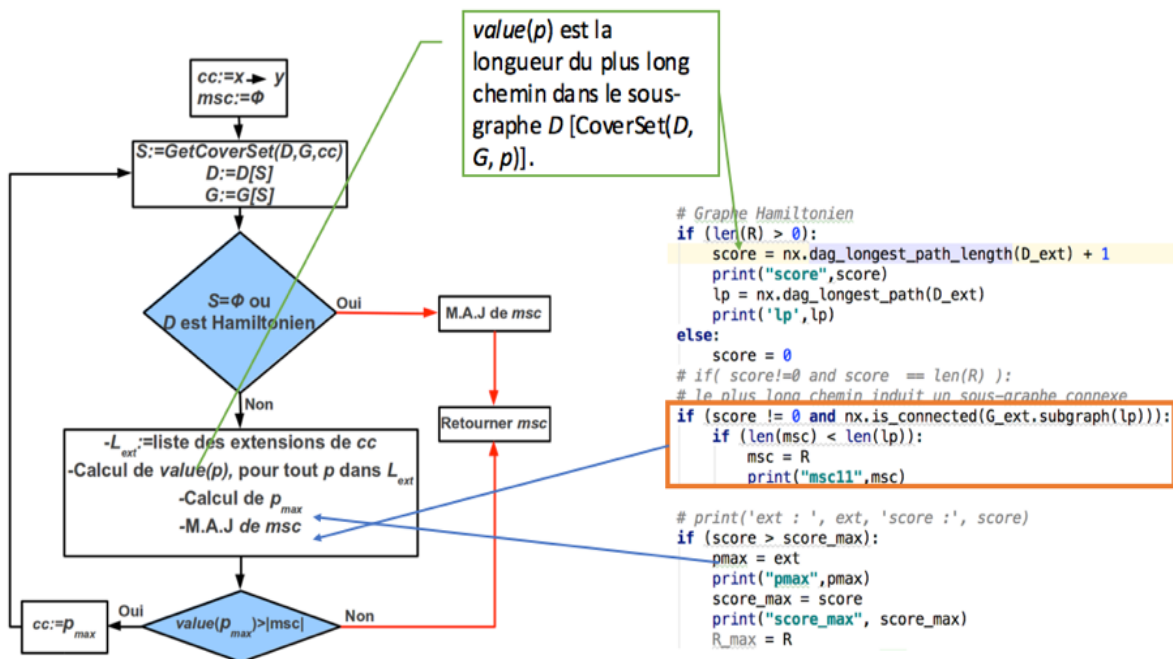


Figure 9 – L'algorithme ALGOH\_3

« value(p) » est égal à « score » est le longueur du plus long chemin. L'implémentation du code nécessite les fonctions de bibliothèque suivantes.

« dag\_longest\_path\_length (G) : »

Renvoie la longueur de chemin la plus longue dans un graphe G.

« dag\_longest\_path (G) : »

Renvoie le chemin le plus long dans un DAG (liste).

« is\_connected (G) : »

Retourne « True » si le graphique est connecté, sinon retourne « false »

## 4 Explication heuristique ALGOBB

L'algorithme de « algoBB » :

Évaluation (Règle 1).

L'algorithme `ALGOBB` est défini comme suit.

1. Créer la racine de l'arbre  $TS$  associée à l'arc  $xy$ .
2. **Tant que** ( $TS$  contient un sommet  $s$  à explorer) **faire**
  - (a) évaluer  $TS$  (**Règle 1**);
  - (b) séparer  $TS$  par rapport au sommet  $s$ ;
  - (c) élaguer  $TS$  (**Règle 2**);
3. Soit  $s_{max}$  une feuille de  $TS$  vérifiant :  $BBvalue(s_{max}) \geq BBvalue(s)$ , pour toute feuille  $s$  de  $TS$ . Soit  $p_{max} = p(s_{max})$ .
4. Si  $p_{max}$  est  $(D, G)$ -consistant, alors  $p_{max}$  est le plus long chemin  $(D, G)$ -consistant passant par  $xy$ . Sinon, il n'y a aucun chemin  $(D, G)$ -consistant passant par  $xy$ .

Figure 10 – L'algorithme de `algoBB`

Soit  $s_1, s_2, \dots, s_k$  l'ensemble des sommets à explorer. Nous choisissons le sommet  $s$  tel que  $BBvalue(s) = \max BBvalue(s_i) : 1 \leq i \leq k$ . Dans le cas où il y a plusieurs sommets  $s_i$  dont  $BBvalue(s_i)$  est maximum, on en choisit arbitrairement un parmi eux.

Élagage (Règle 2).

Soit  $S_{max}$  un sommet de  $TS$  satisfaisant les conditions suivantes :

(i)  $s_{max}$  a été déjà exploré, et (ii) pour tout sommet  $s$  de  $TS$ , si  $s$  a été déjà exploré, alors  $BBvalue(s_{max}) \geq BBvalue(s)$ . On supprime dans  $TS$  toutes les feuilles  $s$  tel que  $BBvalue(s) < BBvalue(s_{max})$ . Cette suppression est appliquée récursivement sur les sommets (sauf  $S_{max}$ ) qui deviennent feuilles après la suppression de leurs fils.

Pour savoir le plus d'information regarder l'article [4] « Comparaison de réseaux biologiques » de monsieur Hamed Mohamed Babou.

Dans « `algoBB` », tout d'abord, nous voulons ajouter un sommet initial (numéro 1).

La structure de données de l'arbre généré dans « `algoBB` » est :

```
T.add_node(1)
T.node[numNode]['exp'] = 0
T.node[numNode]['chemin'] = L
T.node[numNode]['score'] = 0
T.node[numNode]['sol'] = False
T.node[numNode]['chemin_sol'] = []
```

Figure 11 – La structure de données de l'arbre généré

- ['exp'] = 0 : Ce sommet n'a pas encore été exploré.
- ['exp'] = 1 : Ce sommet a été exploré.
- ['chemin'] : La liste de sommets avec extensions (La valeur initiale est  $L$ ).
- ['score'] : La longueur du plus long chemin dans le graphe  $D_{ext}$ .
- ['sol'] : False : C'est pas la vraie solution. True : C'est la solution de ce problème.
- ['chemin\_sol'] : La solution de ce problème dans ce cas.

## 5 Complexité de ONE-TO-ONE SKEWGRAM

$D^*$  : le graphe support de  $D$

Théorème de ONE-TO-ONE SKEWGRAM :

ONE-TO-ONE SKEWGRAM est NP-complet, même si  $D^*$  est planaire extérieur et  $G$  est un arbre de diamètre 4. Il peut réduire au problème de  $MAX\ 2S_{AT}$ .

$G \backslash D^*$	Tree	Outerplanar	General graph
Chordless path or cycle, (bi-)star	P [Lem. 1]		
Tree with diameter 4	P [Lem. 1]	NPC [Thm. 1]	NPC [Thm. 1]
General graph	P [Lem. 1]	NPC [Thm. 1]	NPC [Thm. 1] APX-hard [Thm. 2]

Figure 12 – Complexité de l'algorithme ALGOH

Soit  $C = \{C_1, \dots, C_p\}$  et  $X_n = \{X_1, \dots, X_n\}$  une instance de MAX 2S<sub>AT</sub>. Le graphe orienté  $D$  est construit sur  $2p+2n+2$  niveaux.

# 4

## Analyse et conception

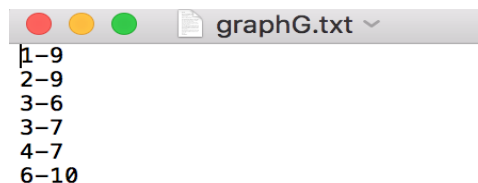
### 1 Génération de jeux de données

Pour tester la performance de cet algorithme, nous devons générer les jeux de données. Dans ce projet, nous étudions deux types de graphes. « Graphe aléatoire » et « Réseau invariant d'échelle »[1].

Donc, j'ai écrit les fonctions pour générer les graphiques D, G et L.


N est le nombre de sommets dans un graphe. (Je choisis habituellement  $N = 360$  pour la recherche) On peut définir N et le nombre d'instances pour chaque N.

Les fichiers d'arc L et les fichiers de graphe D sont donc des correspondances biunivoques.



```
graphG.txt
1-9
2-9
3-6
3-7
4-7
6-10
```

Figure 1 – Exemple de grapheG de type « Graphe aléatoire » avec  $N=10$



```
graphD.txt
N:10
a:9
b:10
1-8
1-6
2-1
2-10
2-6
3-6
4-10
4-3
5-8
7-3
8-6
9-10
9-5
9-6
```

Figure 2 – Exemple de grapheD de type « Graphe aléatoire » avec  $N=10$

## 2 Automatisation des tests

Une fois les jeux de données générées, nous pouvons commencer à tester les performances de cet algorithme.

Les mesures de performance qui nous intéressent sont :

1. Le temps de calculer (algoH et algoBB).
2. Le nombre de boucle
3. La longueur de solution (le plus long chemin) (algoH et algoBB)
4. Le nombre de boucle (algoH)
5. Le nombre total d'extensions calculées (algoH)
6. Les extension(ext) qui a le même score avec « pmax »
7. Le nombre de sommets explorés (algoBB)
8. Écart de longueur de solution entre « l'algoBB » et « l'algoH »
9. Écart de temps entre « l'algoBB » et « l'algoH »

Et j'ai créé trois tableaux dans le même fichier (Excel) pour stocker les informations.

Dans le premier tableau, je stocke toutes les informations.

Dans le deuxième tableau, je supprime toutes les lignes qui n'ont pas trouvé de solution dans « algoBB ».

Dans le troisième tableau, je calcule la moyenne de chaque instance de N, la moyenne de chaque N, et le taux d'écart de longueur de solution entre « algoBB » et « algoH ».

## 3 Amélioration de l'algorithme

Après avoir généré le graphique et fait les tests automatiques, nous trouvons que l'algorithme ALGOH ne réduit pas beaucoup de temps que l'algorithme « algoBB », mais le taux d'écart de longueur de solution entre « algoBB » et « algoH » est vraiment grand.

Donc, nous avons décidé d'optimiser l'algorithme « algoBB » au lieu d'optimiser l'algorithme « algoH » comme prévu.

# 5

## Mise en oeuvre

### 1 Fichier de sortie

Il y a trois tableaux dans un fichier comme le sortie.

« data » :

Stocker toutes les données de sortie

« data\_filtered » :

Dans le deuxième tableau, on va supprimer les données qui n'a pas le solution pour « algoBB » sur la base du tableau « data ».

« data\_moyen »

Dans le troisième tableau, on va calculer le moyen de chaque numéro de instances, et aussi le moyen de chaque nombre de sommet.

### 2 Modification de ALGOBB

Il y a deux versions de modification de « algoBB » dans ce projet.

AlgoBB Version2 :

Dans ce version, je vais améliorer algoBB(branch and bound) pour réduire le coût de temps. Et l'idée centrale de l'optimisation est la recherche "heuristique". Pour chaque branche, nous explorons seulement  $2(\alpha)$  sommets qui a le plus grand score, pas tous les sommets.

AlgoBB Version3 :

Dans ce version, je vais continuer à améliorer algoBB(branch and bound) pour réduire le coût de temps. Et l'idée centrale de l'optimisation est la recherche "heuristique" aussi. La différence entre AlgoBB Version3 et AlgoBB Version 2 est :

Dans Version2, nous explorons seulement  $2(\alpha)$  sommets qui a le plus grand score par chaque branche, pas tous les sommets. Dans Version3, nous explorons seulement  $2(\alpha)$  sommets qui a le plus grand score par chaque niveau, pas tous les sommets.

Pour comparer les trois versions de « algoBB », j'ai comparé ces fichiers exportés.

nb_nodes	nb_instance	nb_L	L	Version original				Version Résultat 1				Version Résultat 2			
				longueur	sol	bb	nb_branch	time_opt	bb	longueur	sol	bb	nb_branch	time_opt	bb
360	1	1	[26, 148]	21	[89, 323, 325, 1]	3	1.430148			21	[89, 323, 325]	2	1.158704		
360	1	2	[197, 72]	15	[197, 72, 122, 7]	4	1.661768			15	[197, 72, 122]	4	1.718629		
360	1	3	[197, 76]	10	[197, 76, 287, 2]	9	1.196278			10	[197, 76, 287]	9	1.252667		
360	1	4	[207, 143]	16	[89, 323, 325, 1]	3	1.131808			16	[89, 323, 325]	3	0.871636		
360	1	5	[328, 300]	21	[89, 323, 325, 1]	4	2.109904			21	[89, 323, 325]	4	3.46629		
360	1	6	[23, 345]	12	[89, 21, 223, 35]	7	0.984988			12	[89, 21, 223]	7	1.418744		
360	1	7	[52, 201]	16	[89, 21, 223, 35]	13	2.506689			16	[89, 21, 223]	12	3.25402		
360	1	8	[35, 243]	10	[323, 325, 251]	18	1.77536			10	[323, 325, 251]	18	2.615627		
360	1	9	[261, 7]	5	[89, 21, 251, 26]	3	0.331623			5	[89, 21, 251]	3	0.411846		
360	1	10	[89, 208]	8	[89, 208, 259, 3]	10	1.05918			8	[89, 208, 259]	10	1.686002		
360	1	11	[332, 16]	19	[89, 323, 325, 1]	126	23.550172			19	[89, 323, 325]	113	21.881421		
360	1	12	[168, 100]	10	[89, 21, 223, 25]	4	0.458868			10	[89, 21, 223]	7	0.716085		
360	1	13	[186, 16]	10	[323, 325, 179]	24	2.36444			10	[323, 325, 179]	19	1.709112		
360	1	14	[169, 56]	14	[89, 21, 223, 35]	24	3.361731			14	[89, 21, 223]	19	2.644541		
360	1	15	[35, 243]	10	[323, 325, 251]	18	1.7912			10	[323, 325, 251]	18	1.885428		
360	1	16	[52, 94]	17	[89, 21, 223, 35]	11	2.567646			17	[89, 21, 223]	11	2.575724		
360	1	17	[198, 353]	16	[89, 323, 325, 1]	5	1.319625			16	[89, 21, 223]	45	6.896191		
360	1	18	[279, 130]	15	[89, 21, 223, 25]	51	6.894988			15	[89, 21, 223]	47	6.306522		
360	1	19	[86, 201]	9	[89, 323, 325, 8]	26	3.725504			9	[89, 323, 325]	24	2.157633		
360	2	1	[347, 34]	19	[242, 339, 100]	58	20.16807			19	[242, 339, 100]	44	12.415097		
360	2	2	[15, 170]	12	[15, 170, 148, 9]	3	0.604444			12	[15, 170, 148]	3	0.487011		
360	2	3	[306, 332]	8	[242, 123, 136]	34	3.541284			8	[242, 123, 136]	10	0.964194		
360	2	4	[76, 210]	23	[261, 249, 351]	6	4.978218			23	[261, 249, 351]	6	4.626748		
360	2	5	[183, 200]	11	[261, 249, 351]	11	1.351917			11	[261, 249, 351]	11	1.25524		
360	2	6	[91, 246]	28	[242, 339, 100]	26	25.823156			28	[242, 339, 100]	16	15.415666		
360	2	7	[126, 258]	13	[126, 258, 165]	4	0.686185			13	[126, 258, 165]	4	0.664989		
360	2	8	[223, 74]	16	[261, 249, 351]	10	2.059405			16	[261, 249, 351]	7	1.53676		
360	2	9	[250, 222]	24	[261, 249, 351]	8	6.14579			24	[261, 249, 351]	5	4.279242		

Figure 1 – Comparaison de trois versions

On peut trouver que pour la plupart des données, les résultats sont les mêmes. Pour les instances qui a beaucoup de branches(plus que 10), le temps est évidemment réduit.

Mais parce que c'est « heuristique », donc pour très peu de cas, nous ne trouverons pas la réponse.

Comme la partie rouge de la table ci-dessous :

360	4	16	[195, 20]	15	[9, 353, 63, 22]	48	4.842554			15	[9, 353, 63, 22]	33	3.351024			15	[9, 353, 63, 22]	21	2.206006
360	4	17	[215, 305]	22	[9, 353, 63, 22]	109	17.318491			22	[9, 353, 63, 22]	109	15.699645			22	[9, 353, 63, 22]	33	4.79585
360	4	18	[342, 305]	19	[9, 353, 63, 22]	85	9.98297			19	[9, 353, 63, 22]	85	10.647829			19	[9, 353, 63, 22]	27	3.457335
360	4	19	[353, 76]	0	[]	4	0.373078			0	[]	3	0.314933			0	[]	3	0.296088
360	4	20	[259, 60]	7	[9, 353, 63, 75]	4	0.394435			7	[9, 353, 63, 75]	4	0.413149			7	[9, 353, 63, 75]	4	0.389801
360	5	1	[38, 125]	18	[214, 260, 8]	20	3.119504			18	[214, 260, 8]	14	2.388062			18	[214, 260, 8]	15	2.375024
360	5	2	[336, 113]	6	[260, 87, 77, 336]	13	1.049944			0	[]	9	0.772397			0	[]	9	0.719738
360	5	3	[333, 69]	34	[214, 260, 116]	5	6.550667			34	[214, 260, 116]	5	7.072512			34	[214, 260, 116]	5	6.522234
360	5	4	[23, 55]	24	[214, 260, 116]	5	1.924241			24	[214, 260, 116]	7	2.785477			24	[214, 260, 116]	9	3.193516
360	5	5	[156, 63]	21	[214, 260, 87, 2]	36	6.759525			21	[214, 260, 87]	36	6.686486			21	[214, 260, 87]	21	3.76292
360	5	6	[37, 347]	38	[214, 260, 72, 2]	292	190.242444			38	[214, 260, 72]	269	166.72954			38	[214, 260, 72]	65	45.184105
360	5	7	[105, 98]	14	[214, 260, 116]	4	1.053029			14	[214, 260, 116]	4	0.616361			14	[214, 260, 116]	5	0.750374
360	5	8	[286, 350]	28	[214, 260, 87, 2]	163	49.61027			28	[214, 260, 87]	148	41.266699			28	[214, 260, 87]	45	13.186753

Figure 2 – Exemples sans solutions

Pour les instances avec de nombreuses branches, le temps est optimisé de 23 secondes à 6 secondes comme la figure ci-dessous :

360	1	11	[332, 16]	19	[89, 323, 325, 1]	126	23.550172			19	[89, 323, 325]	113	21.881421			19	[89, 323, 325]	27	6.035935
-----	---	----	-----------	----	-------------------	-----	-----------	--	--	----	----------------	-----	-----------	--	--	----	----------------	----	----------

Figure 3 – Exemples sans solutions

### 3 Données difficiles

Grâce aux résultats, nous pouvons voir qu'il y a des cas où il faut beaucoup de temps pour trouver la solution et parfois il va obtenir le solution très rapide.

Nous espérons donc étudier quels types d'instances prendront plus de temps à fonctionner. J'ai donc écrit une fonction « choose\_n\_arc(n) » pour résoudre ce problème.

Nous choisissons 360\_5 comme l'instance à étudier.

On va parcourir tous les arcs dans graphe D (360\_5 erdos) pour trouver les données difficiles(le temps plus que 100 secondes à calculer pour algoBB).

Attention, parce que il y a plus que 10000 arcs à calculer, ça prend beaucoup de temps (environ 100h).

J'ai affiché la réponse sur la console, et je stocke la sortie de la console dans Excel.



```

Run 12998 .....
12999 [360, 281] 35 17.1798100000001453 [214, 260, 116, 360, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 1]
Run 12999 .....
13000 [360, 298] 35 5.0480999999999296 [214, 260, 116, 360, 298, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 1]
Run 13000 .....
13001 [360, 304] 35 35.732920999999798 [214, 260, 116, 360, 304, 28, 127, 340, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339]
Run 13001 .....
13002 [360, 308] 35 0.170782999999504611 [214, 260, 116, 360, 308, 211, 225, 113, 315, 132, 101]
Run 13002 .....
13003 [360, 312] 35 3.08943999999524467 [214, 260, 116, 360, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 101]
Run 13003 .....
13004 [360, 324] 35 25.617601000000538 [214, 260, 116, 360, 324, 148, 62, 267, 190, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 34]
Run 13004 .....
13005 [360, 326] 35 44.322710999998616 [214, 260, 116, 360, 326, 288, 259, 325, 111, 200, 216, 18, 290, 328, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 1]
Run 13005 .....
13006 [360, 329] 35 51.032056000002548 [214, 260, 116, 360, 329, 193, 310, 333, 19, 131, 213, 259, 325, 111, 200, 216, 18, 290, 328, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154]
Run 13006 .....
13007 [360, 331] 35 43.703978000003411 [214, 260, 116, 360, 331, 68, 288, 259, 325, 111, 200, 216, 18, 290, 328, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1]
Run 13007 .....
13008 [360, 348] 35 81.586333999999235 [214, 260, 116, 360, 348, 160, 277, 155, 221, 254, 38, 206, 349, 27, 334, 90, 82, 173, 286, 359, 333, 19, 131, 213, 259, 325, 111, 200, 216, 18, 290, 328, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 1]
Run 13008 .....
13009 [360, 354] 35 0.2414820000412874 [214, 260, 87, 360, 354, 208, 84, 250, 110, 335, 315, 132, 101]

```

Figure 4 – Les résultats de parcourir tous les arcs

Après l'exécution de la fonction « choose\_n\_arc(n) », nous allons sortir un fichier « arcL.txt ».

Et on peut regarder le fichier « arcL.txt » comme ci-dessous :

Figure 5 – Les instances qui nécessitent des calculs complexes

Et j'ai comparé les solutions de trois versions, on peut savoir que pour les données difficiles à calculer, il y a deux groupes.

Dans le premier group, « nb\_branch » est grand, on peut améliorer et réduire le temps.

Dans le deuxième groupe, « nb\_branch » est égal à 1, par le modification de « AlgoBB », le temps ne change pas.

		Donnée original				Résultat1				Résultat2			
nb_nodes	nb_instance	longueur	nb_soi	nb_branch	time_opt_bb	longueur	nb_soi	nb_branch	time_opt_bb	longueur	nb_soi	nb_branch	time_opt_bb
360	5	13 [82, 98]	46	[214, 260, 1]	176	255.807187	46	[214, 260, 1]	176	256.1664	46	[214, 260, 1]	79
360	5	17 [237, 29]	46	[214, 260, 1]	176	221.000453	46	[214, 260, 1]	176	238.2644	46	[214, 260, 1]	79
360	5	12 [82, 63]	45	[214, 260, 1]	143	294.622023	45	[214, 260, 1]	143	185.78399	45	[214, 260, 1]	77
360	5	14 [82, 197]	45	[214, 260, 1]	143	176.745244	45	[214, 260, 1]	143	185.75403	45	[214, 260, 1]	77
360	5	8 [17, 347]	38	[214, 260, 7]	280	173.08986	38	[214, 260, 7]	280	173.80023	38	[214, 260, 8]	69
360	5	23 [253, 285]	45	[214, 260, 1]	112	143.992444	45	[214, 260, 1]	76	108.5054	45	[214, 260, 1]	76
360	5	15 [162, 360]	85	[214, 260, 8]	224	125.69708	85	[214, 260, 8]	254	115.29503	85	[214, 260, 8]	88
360	5	11 [65, 263]	107	[214, 260, 1]	1	115.531908	107	[214, 260, 1]	1	103.43779	107	[214, 260, 1]	1
360	5	16 [214, 260]	107	[214, 260, 7]	1	110.85578	107	[214, 260, 7]	1	119.62947	107	[214, 260, 7]	1
360	5	1 [1, 138]	107	[214, 260, 1]	1	104.742462	107	[214, 260, 1]	1	107.56383	107	[214, 260, 1]	1
360	5	25 [260, 87]	107	[214, 260, 8]	1	104.680287	107	[214, 260, 8]	1	115.48004	107	[214, 260, 8]	1
360	5	36 [260, 116]	107	[214, 260, 1]	1	103.751287	107	[214, 260, 1]	1	118.61086	107	[214, 260, 1]	1
360	5	24 [254, 38]	107	[214, 260, 1]	1	100.764138	107	[214, 260, 1]	1	108.54753	107	[214, 260, 1]	1
360	5	30 [312, 338]	107	[214, 260, 1]	1	99.307557	107	[214, 260, 1]	1	105.36027	107	[214, 260, 1]	1
360	5	33 [325, 111]	107	[214, 260, 1]	1	98.787134	107	[214, 260, 1]	1	100.15476	107	[214, 260, 1]	1
360	5	27 [312, 57]	106	[214, 260, 1]	1	98.683412	106	[214, 260, 1]	1	105.95299	106	[214, 260, 1]	1
360	5	32 [320, 208]	107	[214, 260, 1]	1	97.796342	107	[214, 260, 1]	1	99.038423	107	[214, 260, 1]	1
360	5	21 [251, 238]	107	[214, 260, 1]	1	97.777078	107	[214, 260, 1]	1	112.14498	107	[214, 260, 1]	1
360	5	10 [38, 206]	107	[214, 260, 1]	1	96.743314	107	[214, 260, 1]	1	98.2554	107	[214, 260, 1]	1
360	5	20 [251, 107]	106	[214, 260, 1]	1	96.459607	106	[214, 260, 1]	1	104.696	106	[214, 260, 1]	1
360	5	28 [312, 93]	104	[214, 260, 1]	1	96.103516	104	[214, 260, 1]	1	104.03488	104	[214, 260, 1]	1
360	5	9 [1, 265]	106	[214, 260, 1]	1	95.70383	106	[214, 260, 1]	1	106.2734	106	[214, 260, 1]	1
360	5	22 [252, 154]	107	[214, 260, 1]	1	94.913418	107	[214, 260, 1]	1	101.13434	107	[214, 260, 1]	1
360	5	5 [32, 203]	107	[214, 260, 1]	1	94.702886	107	[214, 260, 1]	1	106.0143	107	[214, 260, 1]	1
360	5	6 [35, 271]	107	[214, 260, 1]	1	93.838262	107	[214, 260, 1]	1	97.79504	107	[214, 260, 1]	1
360	5	7 [37, 210]	106	[214, 260, 1]	1	93.192421	106	[214, 260, 1]	1	104.4977	106	[214, 260, 1]	1
360	5	4 [4, 322]	107	[214, 260, 1]	1	92.940023	107	[214, 260, 1]	1	102.13959	107	[214, 260, 1]	1
360	5	31 [313, 49]	107	[214, 260, 1]	1	92.971283	107	[214, 260, 1]	1	91.988233	107	[214, 260, 1]	1
360	5	29 [312, 343]	103	[214, 260, 1]	1	90.705333	103	[214, 260, 1]	1	99.146283	103	[214, 260, 1]	1
360	5	18 [250, 29]	106	[214, 260, 1]	1	90.623035	106	[214, 260, 1]	1	96.99588	106	[214, 260, 1]	1
360	5	39 [332, 110]	107	[214, 260, 1]	1	90.32044	107	[214, 260, 1]	1	97.135714	107	[214, 260, 1]	1
360	5	35 [332, 248]	104	[214, 260, 1]	1	89.338978	104	[214, 260, 1]	1	97.760414	104	[214, 260, 1]	1
360	5	34 [328, 195]	106	[214, 260, 1]	1	89.272782	106	[214, 260, 1]	1	92.145404	106	[214, 260, 1]	1
360	5	2 [1, 167]	104	[214, 260, 1]	1	88.383137	104	[214, 260, 1]	1	100.28973	104	[214, 260, 1]	1
360	5	9 [38, 27]	105	[214, 260, 1]	1	87.613031	105	[214, 260, 1]	1	96.983313	105	[214, 260, 1]	1

Figure 6 – Le résultat de la comparaison de deux groupes



En plus, j'ai écrit les quatre fonctions ci-dessous dans « Donnee\_analyse » pour analyser les plus détaillée des raisons :  $L : i \rightarrow j$

cal\_degree\_level\_1 :

Cette fonction permet de calculer le sommet de "Out-degree" de  $j$  et "In-degree" de  $i$  dans graphe D

cal\_degree\_level\_2

Cette fonction permet de calculer le somme de "Out-degree" de "Out-degree" de  $j$  et "In-degree" de "In-degree" de  $i$  dans graphe D

cal\_degree(G, LL)

Cette fonction permet de calculer le sommet et le plus grand variable de "Out-degree" de  $j$  et "In-degree" de  $i$  dans graphe G

cal\_shortest\_path(G, LL)

Cette fonction permet de calculer le plus court chemin entre  $i$  et  $j$  dans graphe G

Les fichiers ci-dessous générés :

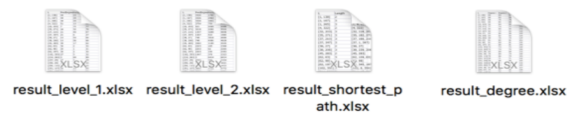


Figure 7 – Les fichiers générés

Je les mets ensemble.

L	time_opt_bb	TotalDegree	TotalDegree Max Degree	TotalDegree Length
13 [62, 98]	255.807187	31	2265	81 150 2
17 [237, 29]	221.920453	29	2204	69 132 3
12 [62, 63]	204.622023	32	2368	81 161 3
14 [62, 197]	176.745246	32	2353	81 149 3
8 [37, 347]	173.089865	27	1980	83 153 3
23 [253, 285]	143.992444	30	2179	75 145 3
15 [162, 350]	123.659708	32	2323	74 148 3
11 [45, 283]	115.521908	62	4545	76 134 2
16 [214, 260]	110.855578	89	6414	74 146 3
1 [1, 138]	104.742462	67	5086	69 137 2
25 [260, 87]	104.690287	79	5803	85 159 3
26 [260, 116]	103.751287	75	5440	74 144 2
24 [254, 38]	100.764136	83	6125	64 127 3
30 [312, 358]	99.307557	77	5583	67 129 3
33 [325, 111]	98.787134	76	5594	75 134 3
27 [312, 57]	98.663412	79	5707	81 148 3
32 [320, 208]	97.796342	85	6148	83 150 3
21 [251, 238]	97.777078	60	4361	86 145 3
10 [38, 206]	96.743314	66	4828	63 125 3
20 [251, 107]	96.459507	65	4772	71 130 3
28 [312, 93]	96.103516	79	5762	67 129 3
3 [1, 265]	95.70393	69	5167	76 144 3
22 [252, 154]	94.913418	76	5501	69 132 3
5 [32, 203]	94.702895	77	5586	84 139 3
6 [35, 271]	93.818292	76	5500	77 151 3
7 [37, 210]	93.192421	67	4846	83 152 3
4 [4, 322]	92.940023	61	4504	80 146 2
31 [313, 49]	90.971351	71	5121	75 148 2
29 [312, 343]	90.720533	76	5516	71 138 3

Figure 8 – Les paramètres stockés pour analyser

On peut faire les analyses par les paramètres ci-dessus, nous pouvons analyser.

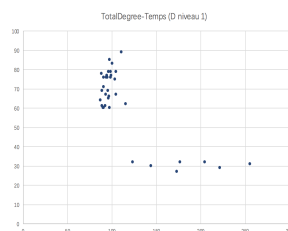
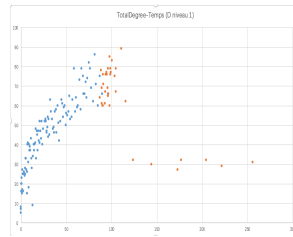


Figure 9 – Le nuage de points de le sommet de "Out-degree" de  $j$  et "In-degree" de  $i$

Nous pouvons clairement trouver deux groupes.

A la fin, j'ai ajouté les instances qui prennent moins de 100 secondes à calculer dans le nuage de points.



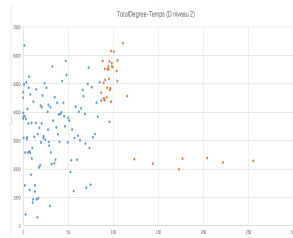
**Figure 10** – Le nuage de points qui ajoute les instances moins de 100 secondes

Après analyse, nous pouvons trouver que le temps de calcul de « AlgoBB » est lié à la somme de "Out-degree" de  $j$  et "In-degree" de  $i$  dans le graphe  $D$ .

Mais pour les instances qui ont exploré de nombreux sommets et ont besoin de beaucoup de temps à calculer, nous ne savons toujours pas pourquoi il est difficile pour lui de calculer.

Pour les autres paramètres à analyser, il n'y a pas de connexion évidente.

Par exemple, le nuage de points qui ajoute les instances dans le nuage de points de somme de "Out-degree" de  $j$  et "In-degree" de  $i$  dans le graphe  $D$  est comme ci-dessous :



**Figure 11** – Le nuage de points qui ajoute les instances moins de 100 secondes

# 6

## Conclusion

### 1 Bilan de le semestre 9

**Les taches suivantes sont déjà finies :**

- Étude du problème biologique
- Étude de l'Heuristique
- Étudier le code existant de l'algorithme « ALGOH ».
- Étude du code existant
- Validation le cahier de spécification.
- Rédaction du rapport de PRD(S9).

**Les tâches à faire :**

- Préparation de la soutenance de S9

La plus grande difficulté que j'ai rencontrée est l'exécution du code existant sur mon ordinateur. Parce que je ne connais pas la version du package requise. Après le contact avec l'auteur du code, le projet s'est déroulé sans problème.

Donc j'ai terminé le projet strictement comme prévu, il n'y avait pas d'avance et de retard.

La planification de S9 est présentée dans la partie « Annexe : 1 (Annexe A)Découpage du projet en tâches(S9) » et la diagramme de Gantt 1 (Annexe A).

### 2 Planning de le semestre 10

J'ai déjà prévu les tâches et leurs durées pour S10.

Les détails sont présentés dans la partie « Annexe : 2 (Annexe A)Découpage du projet en tâches(S10) » et la diagramme de Gantt 2 (Annexe A).

**Les tâches à faire :**

- Génération de jeux de données
- Automatisation des tests
- Analyse des résultats des tests
- Validation les résultats
- Nouvelles fonctions évaluation(pmax)

- Nouvelles fonctions pour choisir extension
- **Modification de « AlgoBB »**
- **Trouver les données difficiles**
- Analyse et comparaison les résultats de l'amélioration de l'algorithme
- Rédaction du rapport final
- Préparation de la soutenance

Au début de S10, j'ai testé l'algorithme « AlgoH » et « AlgoBB ».

Après avoir généré le graphique et fait les tests automatiques, nous trouvons que l'algorithme « AlgoH » ne réduit pas beaucoup de temps que l'algorithme « AlgoBB », mais le taux d'écart de longueur de solution entre « AlgoBB » et « AlgoH » est vraiment grand.

Donc, nous avons décidé d'optimiser l'algorithme « AlgoBB », au lieu d'optimiser l'algorithme « AlgoH » comme prévu.

Dans le S10, j'ai fini tous les tâches comme prévu sauf « Nouvelles fonctions évaluation(pmax) » et « Nouvelles fonctions pour choisir extension ».

Parce que j'ai fait les deux versions de modifications de « AlgoBB », pour le remplacer.

Mais il n'y a pas de réponse à l'analyse des données difficiles.

Pour aller plus loin, nous pouvons continuer à étudier des données difficiles et continuer à optimiser l'algorithme.

## Annexes

# A

## Planification

Pour ce projet, je travaille deux 2 jours par semaine à temps plein. Donc pour l'estimation de charge, deux jours équivalent à une semaine.

### 1 Découpage du projet en tâches(S9)

#### 1.1 Tâche 1 : Étude du problème biologique

##### Description de la tâche

Cette tâche consiste à bien comprendre le problème de comparaison de réseaux biologique. Elle est composée de lecture des documents donnée par MOA, trouver et étudier les thèses concernant mon sujet, et aussi réfléchir les informations fournies dans les réunions entre la MOA et la MOE.

##### Livrables

Un rapport de présentation du problème sera confirmé par MOA. Le contenu de ce rapport sera mis dans le cahier de spécification.

##### Estimation de charge

Cette tâche est estimée à 4 jours.

#### 1.2 Tâche 2 : Étude de l'Heuristique

##### Description de la tâche

Cette tâche permet de comprendre l'algorithme de l'Heuristique. L'algorithme qui peut résoudre notre problème s'appelle « AlgoH ». Elle est composée de lecture des thèses et des documents concernant cet algorithme.

##### Livrables

Un rapport de présentation du problème sera confirmé par MOA. Le contenu de ce rapport sera mis dans le cahier de spécification.

**Estimation de charge**

Cette tâche est estimée à 4 jours.

**1.3 Tâche 3 : Étude du code existant****Description de la tâche**

Cette tâche permet d'étudier le code existant. Le code est lié à l'algorithme « AlgoH », mais ce n'est pas exactement la même chose. Afin d'améliorer l'algorithme, il est nécessaire d'apprendre le code existant.

**Livrables**

Les slides seront présentés dans les réunions entre la MOA et la MOE.

**Estimation de charge**

Cette tâche est estimée à 8 jours.

**1.4 Tâche 4 : Rédaction du cahier de spécification****Description de la tâche**

Cette tâche consiste en la rédaction du cahier de spécification. Celle-ci permet de définir le contexte, le périmètre, les systèmes et leurs fonctionnalités ainsi que le déroulement du projet à l'aide d'un planning.

**Livrables**

Un cahier de spécification sera à rendre.

**Estimation de charge**

Cette tâche est estimée à 4 jours.

**1.5 Tâche 5 : Validation le cahier de spécification.****Description de la tâche**

Une fois la première version du cahier de spécification fait, celui-ci va subir des navettes entre la MOA et la MOE afin que ces deux partis soient en accord.

**Estimation de charge**

Cette tâche est estimée à 1 jour.

**1.6 Tâche 6 : Rédaction du rapport de PRD(S9)****Description de la tâche**

Une fois le cahier de spécification est validé, le rapport de PRD(S9) commence à être écrit.

**Livrables**

Un rapport sera à rendre avant le 10/12/2017.

**Estimation de charge**

Cette tâche est estimée à 3 jours.

## 1.7 Tâche 7 : Préparation de la soutenance de S9

### Description de la tâche

Préparation de la soutenance de S9.

### Livrables

Soutenance avec PowerPoint.

### Estimation de charge

Cette tâche est estimée à 1 jour.

## 2 Découpage du projet en tâches(S10)

### 2.1 Tâche 1 : Génération de jeux de données

#### Description de la tâche

Cette tâche permet de générer des données pour tester la performance de cet algorithme. Dans ce projet, nous allons générer 50 fichiers de graphe D, 50 fichiers de graphe G, et 50 fichiers d'arc L. Pour les fichiers de graphe D et G, nous générons 5 fichiers de graphe avec le nombre de N (N=10, 60, 110, 160, 210) sommets différents. Pour les fichiers d'arc L, nous sélectionnons au hasard 20 arcs dans graphe D.

#### Estimation de charge

Cette tâche est estimée à 1 jours.

### 2.2 Tâche 2 : Automatisation des tests

#### Description de la tâche

Cette tâche permet d'effectuer des tests de performance automatisés en utilisant les données générées.

#### Estimation de charge

Cette tâche est estimée à 2 jours.

### 2.3 Tâche 3 : Analyse les résultats du test

#### Description de la tâche

Cette tâche permet d'analyser les résultats du test.

#### Estimation de charge

Cette tâche est estimée à 2 jours. Les changements aux résultats de l'analyse ne sont pas inclus.



## 2.4 Tâche 4 : Validation les résultats du test

### Description de la tâche

Cette tâche permet de valider les résultats par le MOA.

### Livrables

Une présentation sera à fait avec le MOA.

### Estimation de charge

Cette tâche est estimée à 1 jours.

## 2.5 Tâche 5 : Nouvelles fonctions évaluation(pmax)

### Description de la tâche

Cette tâche permet de trouver les nouvelles fonctions évaluation de « pmax ».

### Estimation de charge

Cette tâche est estimée à 6 jour.

## 2.6 Tâche 6 : Nouvelles fonctions pour choisir extension

### Description de la tâche

Cette tâche permet de trouver les nouvelles fonctions pour choisir extension.

### Estimation de charge

Cette tâche est estimée à 6 jours.

## 2.7 Tâche 7 : Analyse et comparaison les résultats de l'amélioration de l'algorithme

### Description de la tâche

Cette tâche permet d'analyser et comparer les résultats de l'amélioration de l'algorithme

### Estimation de charge

Cette tâche est estimée à 4 jours.

## 2.8 Tâche 8 : Rédaction du rapport final

### Description de la tâche

Cette tâche consiste à la rédaction du rapport final de ce PRD. Celui-ci pourra être réalisé tout au long du projet.

### Livrables

Un rapport sera à rendre.

### Estimation de charge

Cette tâche est estimée à 4 jours.

## 2.9 Tâche 9 : Préparation de la soutenance

### Description de la tâche

Cette tâche consiste à préparer la soutenance de ce PRD avec la réalisation d'un diaporama sur le projet et les résultats obtenus.

### Livrables

Le livrable de cette tâche est le code source, les exécutables du projet et le PowerPoint de la soutenance.

### Estimation de charge

Cette tâche est estimée à 2 jours.

## 3 Le diagramme de Gantt

### Le diagramme de Gantt de S9

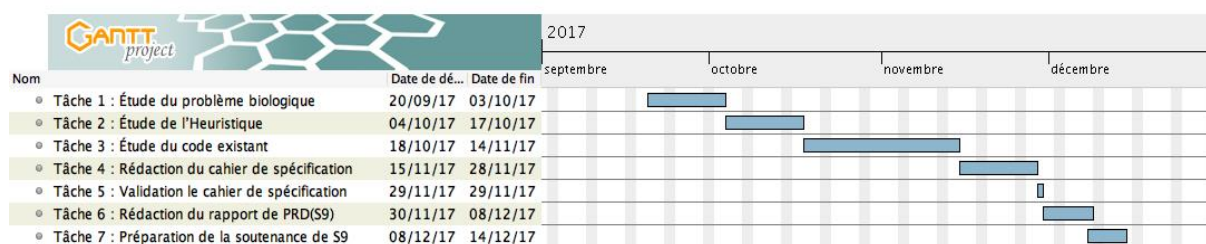


Figure 1 – Le diagramme de Gantt de S9

### Le diagramme de Gantt de S10

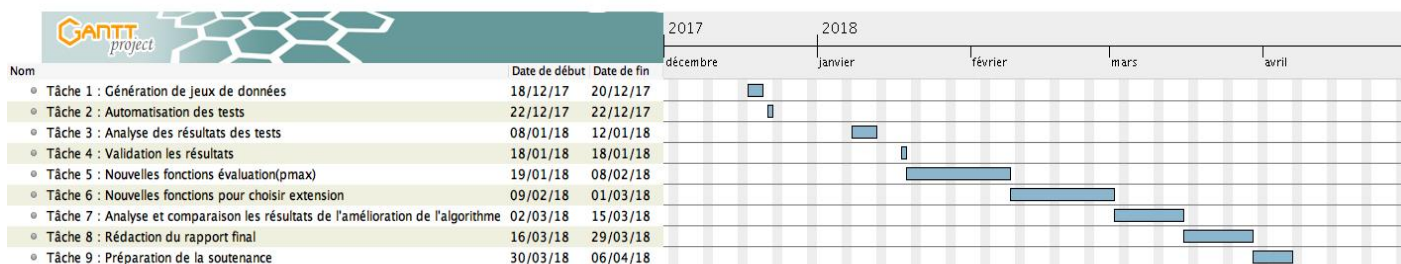


Figure 2 – Le diagramme de Gantt de S10

### Tableau de synthèse S9

### Tableau de synthèse S10

Tableau de synthèse S9	
Tâche	Taux d'achèvement
Étude du problème biologique	100%
Étude de l'Heuristique	100%
Étudier le code existant de l'algorithme « ALGOH ».	100%
Étude du code existant	100%
Validation le cahier de spécification.	100%
Rédaction du rapport de PRD(S9).	100%

Figure 3 – Tableau de synthèse S9

Tableau de synthèse S10	
Tâche	Taux d'achèvement
Génération de jeux de données	100%
Automatisation des tests	100%
Analyse des résultats des tests	100%
Validation les résultats	100%
Nouvelles fonctions évaluation(pmax)	0%
Nouvelles fonctions pour choisir extension	0%
Modification de « AlgoBB »	100%
Trouver les données difficiles	100%
Analyse et comparaison les résultats de l'amélioration de l'algorithme	80%
Rédaction du rapport final	100%
Préparation de la soutenance	100%

Figure 4 – Tableau de synthèse S10

# B

## Spécifications fonctionnelles

Pour comparer les deux réseaux biologiques pour résoudre le problème ONE-TO-ONE SKEWGRAM, nous avons besoin de données d'un graphe orienté D, un graphe non-orienté G et un arc L dans D.

Après l'exécution de l'algorithme, le résultat est affiché sur la console. En plus, nous pouvons également afficher les résultats sous forme de fichier si nécessaire.

### 1 Définition de la fonction 1 : ccc (G1, G2)

#### Identification de la fonction 1

Cette fonction permet de trouver les composantes connexes communes entre deux graphes non-orientés.

#### Description de la fonction 1

Entrée : G1 : graphe non-orienté, G2 : graphe non-orienté

Sortie : Liste des composantes connexes communes entre deux graphes

#### Exemple de la fonction 1

Comme je l'ai présenté avant, on a les deux graphes G1 et G2. L2 est une liste de toutes les composantes connexes de G. Dans cet exemple, il y a trois composantes connexes de G.

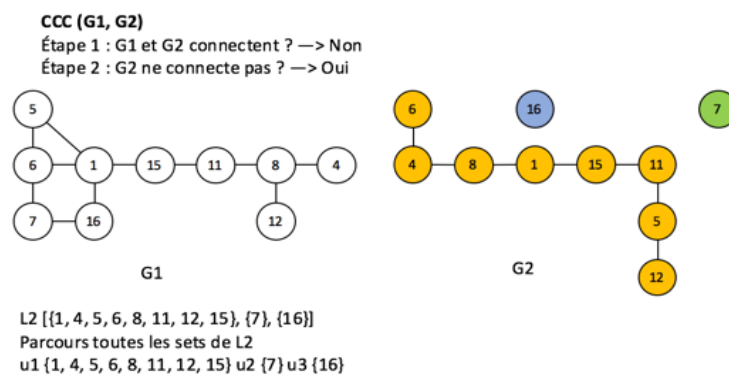


Figure 1 – Exemple de la fonction 1\_1

Cette fonction va parcourir tous les sets de L2. « u1 » est le premier ensemble dans la liste L2. On rappelle le fonction CCC pour trouver si G1[u1] et G2[u1] sont connectent. Si la réponse est « oui », on ajoute « u1 » dans la liste de résultat.

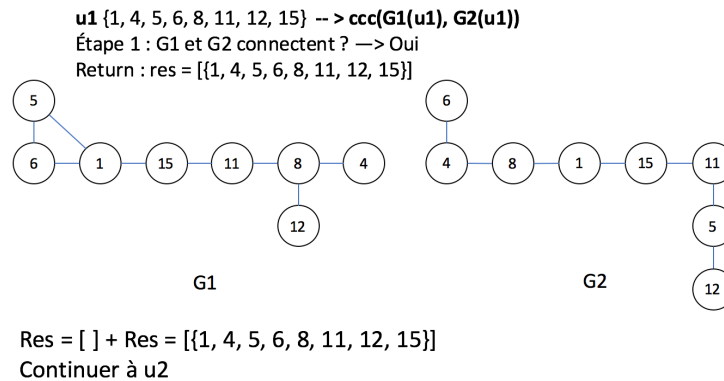


Figure 2 – Exemple de la fonction 1\_2

On fait la même chose pour « u2 ».

**u2 {7} --> ccc(G1, G2)**  
 Étape 1 : G1 et G2 connectent ? --> Oui  
 Return :Res = [{7}]



Res = [ ] + Res = [{1, 4, 5, 6, 8, 11, 12, 15}, {7}]  
 Continuer à u3 comme u2, u3 est également connecté  
 res [{1, 4, 5, 6, 8, 11, 12, 15}, {7}, {16}]  
 → Liste des *composantes connexes communes entre deux graphes*

Figure 3 – Exemple de la fonction 1\_3

A la fin, on a une liste de résultat qui est la liste des composantes connexes communes entre deux graphes.

## 2 Définition de la fonction 2 : ccc2 (G1, G2, L)

### Identification de la fonction 2

Cette fonction permet de trouver la composante connexe commune de deux graphes non-orienté contenant les sommets de L.

### Description de la fonction 2

Entrée : G1 : graphe non-orienté, G2 : graphe non-orienté et chemin L

Sortie : Le set de la composante connexe commune contenant les sommets de L.

### Exemple de la fonction 2

res [1, 4, 5, 6, 8, 11, 12, 15, 7, 16]  
 L [11, 8]

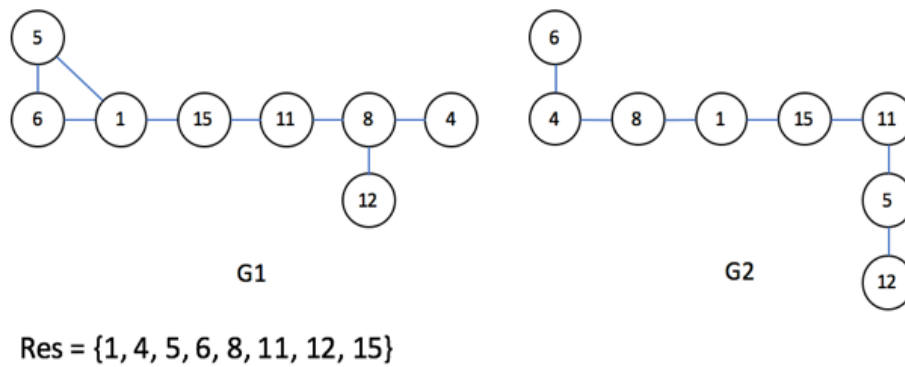


Figure 4 – Exemple de la fonction 2

L'ensemble qui contient L [11,8] est {1, 4, 5, 6, 8, 11, 12, 15}, donc le résultat est {1, 4, 5, 6, 8, 11, 12, 15}.

### 3 Définition de la fonction 3 : est\_pont (G1, G2, L, r)

#### Identification de la fonction 3

Cette fonction permet de vérifier est-ce que le sommet r est le pont du chemin L entre deux graphes.

#### Description de la fonction 3

Entrée : Deux graphes non-orienté, chemin L, sommet r

Sortie : True ou False

#### Exemple de la fonction 3

Par exemple, on va vérifier si le sommet « 1 » est pont ou pas.

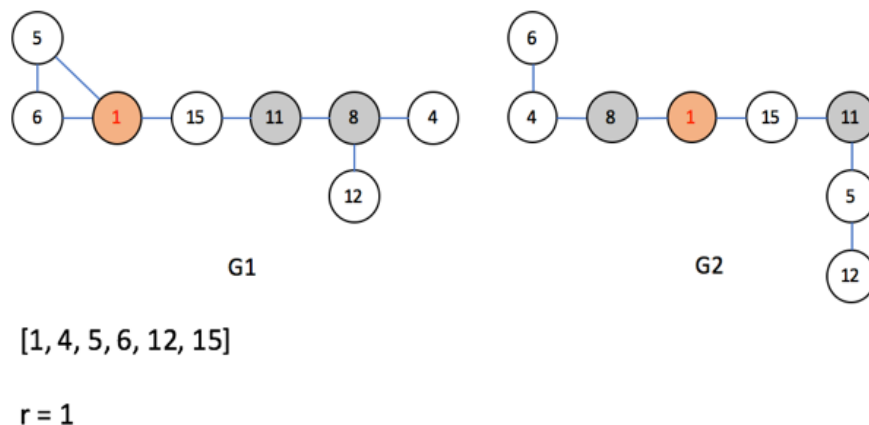


Figure 5 – Exemple de la fonction 3\_1

On le supprime dans les deux graphes. Et nous avons deux graphes H1 et H2.

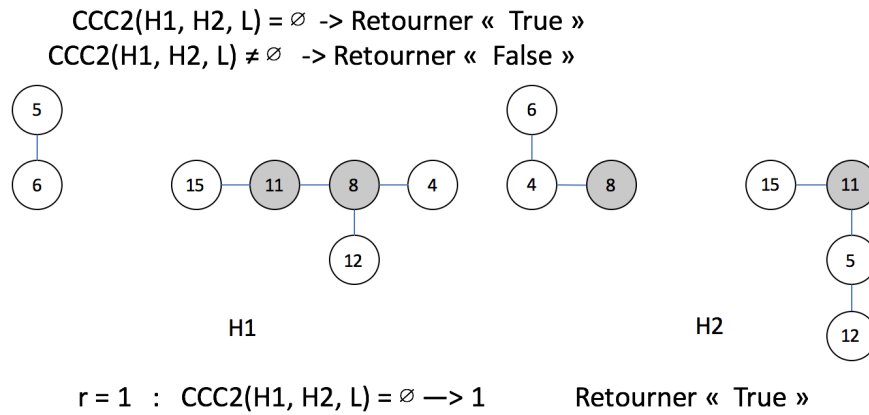


Figure 6 – Exemple de la fonction 3\_2

CCC2(H1, H2, L) est vide, donc le sommet « 1 » est le pont.

#### 4 Définition de la fonction 4 : getCovetSet (D, G, L)

##### Identification de la fonction 4

Cette fonction permet de trouver la composante connexe d'un graphe orienté D et un graphe non-orienté G contenant tous les sommets de L.

##### Description de la fonction 4

Entrée : D : graphe orienté, G : graphe non-orienté, chemin L

Sortie : L'ensemble couvrant d'un chemin donné

##### Exemple de la fonction 4

Dans cet exemple, on a le chemin L, le graphe orienté D, le graphe non-orienté G comme le graphe ci-dessous :

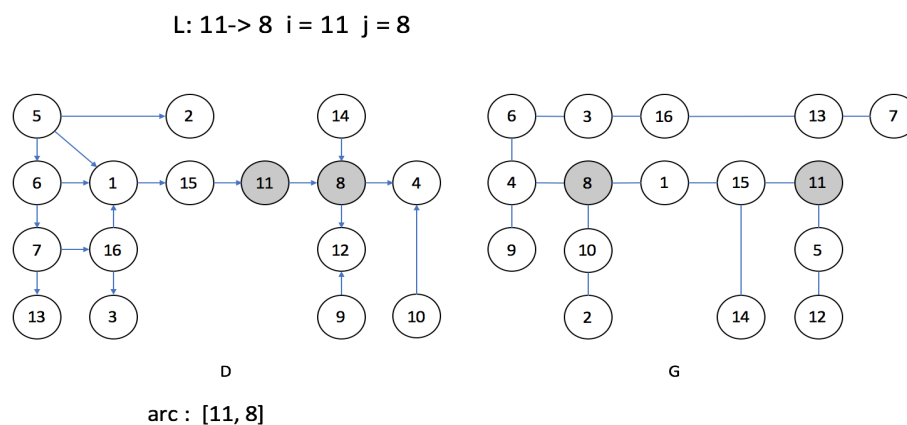


Figure 7 – Exemple de la fonction 4\_1

L'ensemble S1 contient les ascendances de sommet « 11 ».

L'ensemble S2 contient les descendances de sommet « 8 ».

S est l'union d'ensemble S1 et S2.

$s1 \{16, 1, 5, 6, 7, 15\}$  : ancêtres de  $i$   
 $s2 \{4, 12\}$  : descendants de  $j$   
 $S \{1, 4, 5, 6, 7, 8, 11, 12, 15, 16\} : S1 \cup s2 \cup L$

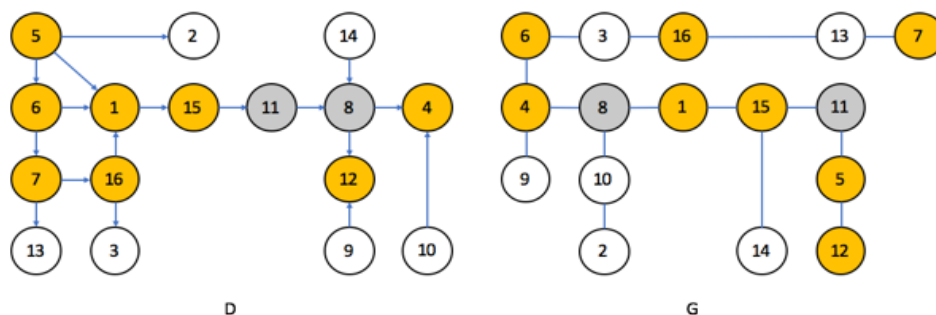
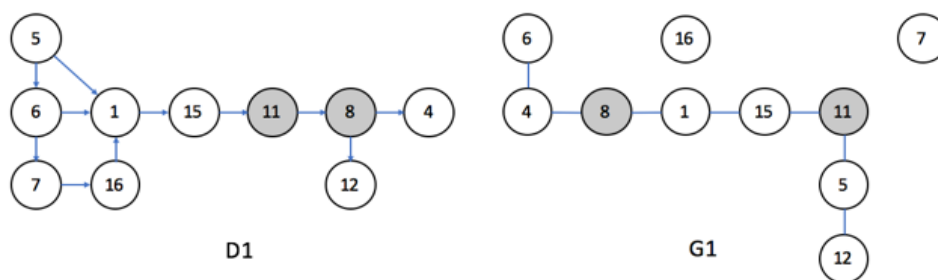


Figure 8 – Exemple de la fonction 4\_2

$D1$  est  $D[S]$ , et  $G2$  est  $G[S]$ .

On appelle le fonction `ccc2` pour trouver la composante connexe.



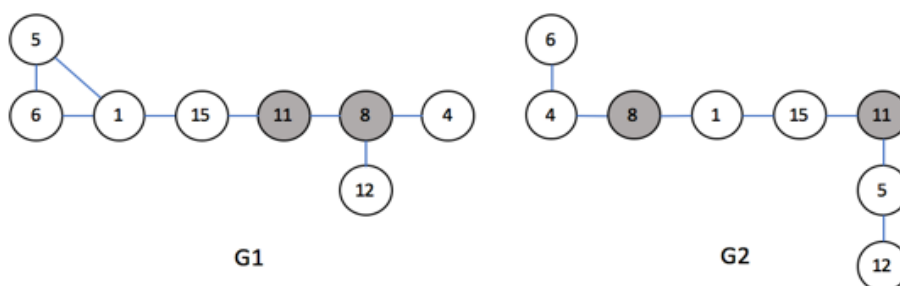
$G1:G[S]$   $D1:D[S]$

—> `ccc2 (D1.to_undirected(), G1, L)`

composante connexe commune contenant les sommets de  $L$

Figure 9 – Exemple de la fonction 4\_3

Nous avons les deux graphes ci-dessous par le fonction `ccc2`.



$S = res = \{1, 4, 5, 6, 8, 11, 12, 15\}$

Figure 10 – Exemple de la fonction 4\_4

Nous convertissons le graphe non-orienté en graphe-orienté.



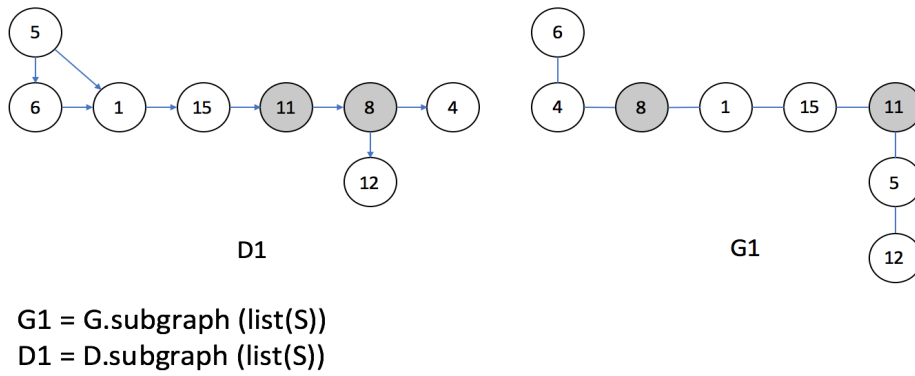
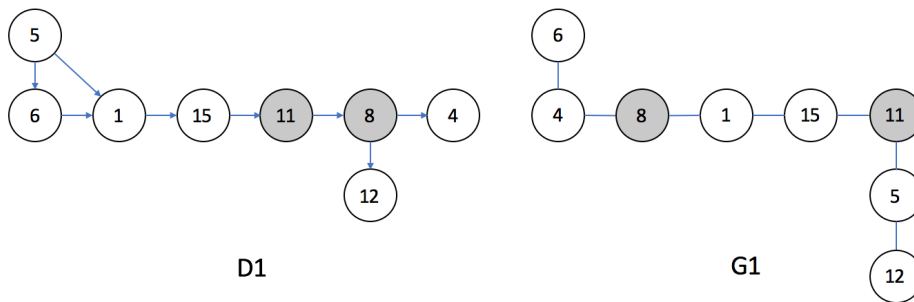


Figure 11 – Exemple de la fonction 4\_5

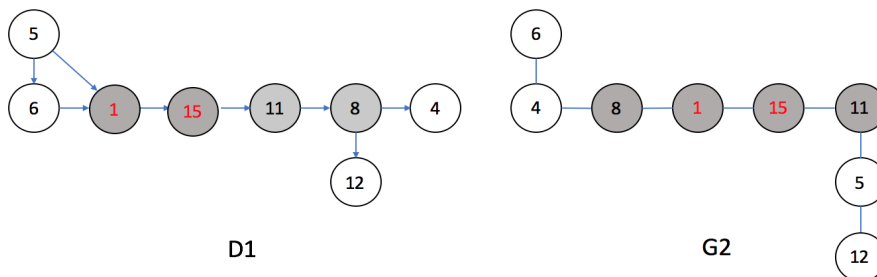
Nous utilisons la fonction « est\_pont » pour trouver toutes les ponts.



→ « est\_pont(D1.to\_undirected(), G1, L, r) »

Figure 12 – Exemple de la fonction 4\_6

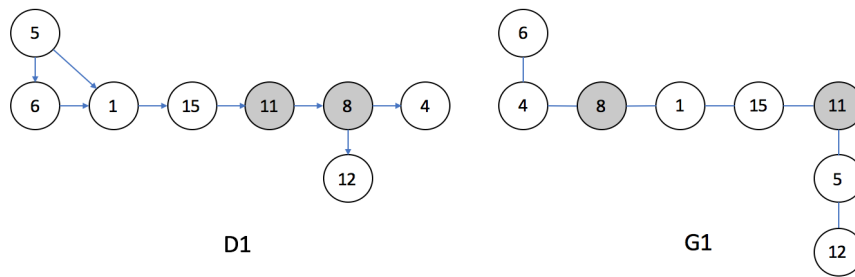
{1,15} sont ponts



Par exemple,  
 $r = 1$   $A = \{1\} \cup \{\text{ancestors de } 1\} \cup \{\text{descendants de } 1\} = \{1, 4, 5, 6, 8, 11, 12, 15\}$   
 $St$  est égal à l'intersection de  $S$  et  $A = \{1, 4, 5, 6, 8, 11, 12, 15\}$ .  
 $St = S \rightarrow \text{Stop}$

Figure 13 – Exemple de la fonction 4\_7

Pour toutes les ponts, nous cherchons l'intersection de « A » et S (St). Parce que St est égal à S, nous arrêtons de calculer. Et nous avons « CoverSet ».



« CoverSet (D, G, 11-> 8) » est égal à {1,4,5,6,11,12,15}

Figure 14 – Exemple de la fonction 4\_8

## 5 Définition de la fonction 5 : s\_t\_Graph (D, L)

### Identification de la fonction 5

Cette fonction permet de trouver le sous-graphe de D qui contient tous les nœuds ayant un chemin vers départ de chemin L et tous les nœuds accessibles depuis fin de chemin L.

### Description de la fonction 5

Entrée : D : graphe orienté

Sortie : D\_ext : graphe orienté

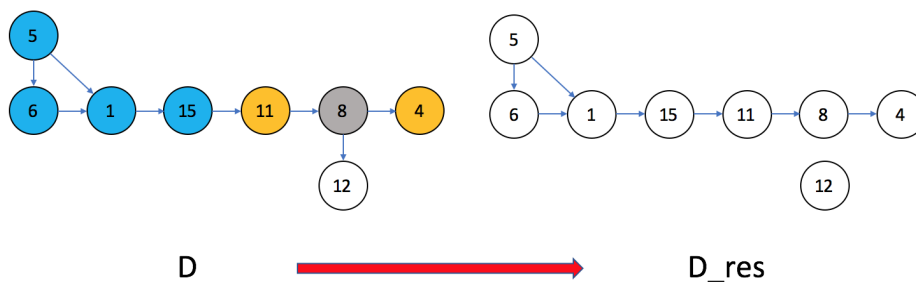
### Exemple de la fonction 5

L'ensemble S1 contient les ascendances de sommet « 11 » plus de « 11 ».

L'ensemble S2 contient les descendances de sommet « 4 » plus de « 4 »

Nous avons le graphe D\_res comme le graphe ci-dessous.

Si  $L = 11 \rightarrow 8 \rightarrow 4$   $i = 11, j = 4$   
 $S1 \{1,5,6,15,11\} : \{ \text{ancestors de } i \} + \{11\}$ .  
 $S2 \{4\} : \{ \text{descendants de } j \} + \{4\}$ .



$$D\_res = (V(S1+S2+V(D)), A(D(S1).edges+D(S2).edges + L))$$

Figure 15 – Exemple de la fonction 5

## 6 Définition de la fonction 6 : algoH (D, G, L)

### Identification de la fonction 6

Cette fonction permet de résoudre le problème ONE-TO-ONE SKEWGRAM. Elle permet de trouver un plus long chemin consistant entre deux graphes (D graphe-orienté et G graphe non-orienté) passant par un arc.

### Description de la fonction 6

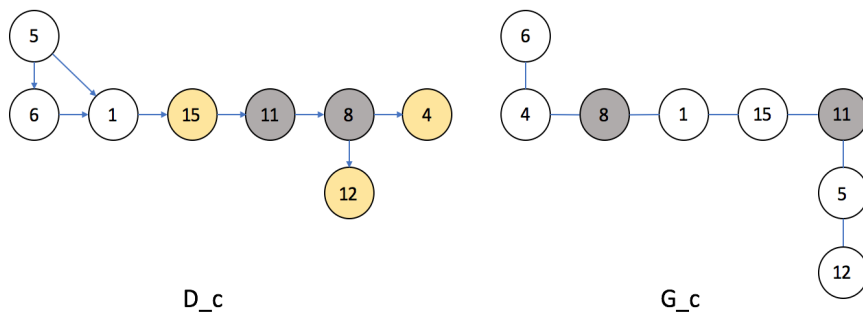
Entrée : D : graphe orienté, G : graphe non orienté, chemin L

Sortie : S : ensemble de les sommets qui a le plus long chemin

### Exemple de la fonction 6

Au début, nous avons un graphe-orienté, un graphe non-orienté et un chemin L. Nous appellerons la fonction « GETCOVERSET » pour obtenir le « CoverSet » S.

CouverSet est {1, 4, 5, 6, 8, 11, 12, 15} Le chemin cc est noté  $s \rightarrow t$  (11→8)



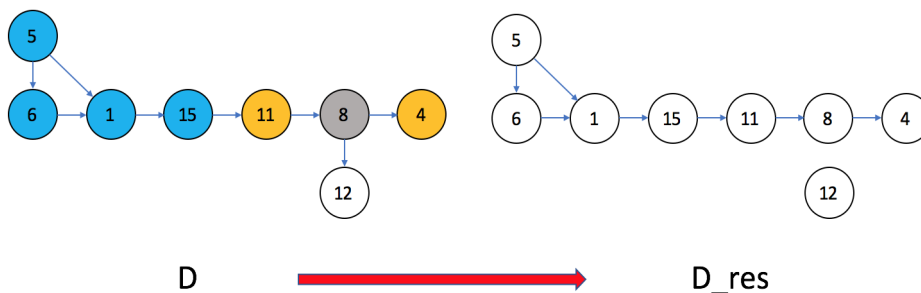
Les prédécesseurs de s est « 15 » et les successeurs de t est « 4, 12 »

Liste des extensions(Lext) : [[11, 8, 4], [11, 8, 12], [15, 11, 8]]

Figure 16 – Exemple de la fonction 6\_1

Extension (ext) est la liste (Par exemple, [11, 8, 4]) qui contient les sommets dans L (11, 8) et un sommet qui est le prédécesseur de s (15) ou le successeur de t (4, 12). Les prédécesseurs et les successeurs de t sont « 15, 4, 12 ». Donc, nous avons la liste des extensions.

Par exemple, ext = [11, 8, 4]  
 $\rightarrow D_{\text{ext}} = s\_t\_Graph(D\_c, \text{ext});$



$$D_{\text{res}} = (V(S_1 + S_2), A(D(S_1).edges + D(S_2).edges + L))$$

Figure 17 – Exemple de la fonction 6\_2

L'algorithme parcourt tous les extensions(ext) dans Lext.

Nous choisissons « ext » est égal à [11, 8, 4] comme un exemple.

Une fois que « ext » est sélectionné, l'algorithme appelle la fonction « s\_t\_Graph » pour obtenir D\_res.

$R = \text{ccc2}(\text{D\_ext.to\_undirected}(), \text{G\_c}, \text{ext})$   
 $\rightarrow R = \{1, 4, 5, 6, 8, 11, 15\}$

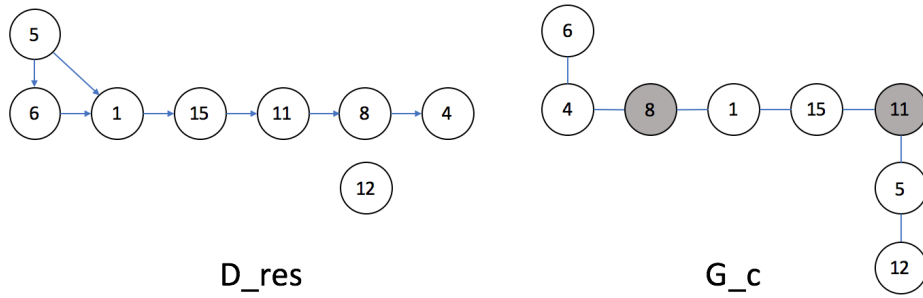


Figure 18 – Exemple de la fonction 6\_3

Nous devons trouver l'ensemble de la composante connexe de D\_ext et G\_c par la fonction « ccc2 ».

$R = \text{ccc2}(\text{D\_ext.to\_undirected}(), \text{G\_c}, \text{ext})$   
 $\rightarrow R = \{1, 4, 5, 6, 8, 11, 15\}$   
 $\rightarrow \text{D\_ext} = \text{D\_ext}(R)$   
 $\text{G\_ext} = \text{G\_c}(R)$

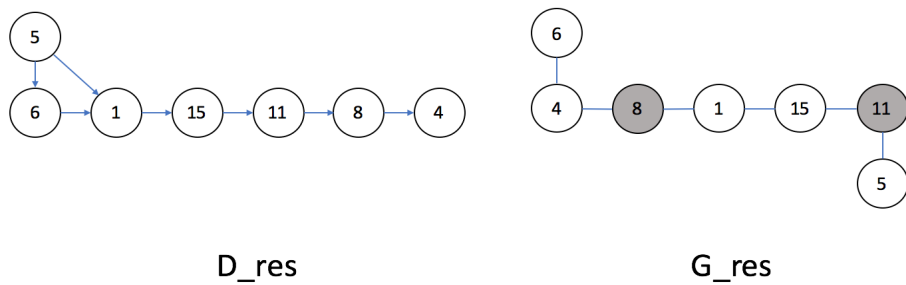


Figure 19 – Exemple de la fonction 6\_4

Nous avons G\_ext.

ext : [11, 8, 4] score 7  
 lp [5, 6, 1, 15, 11, 8, 4] msc {1, 4, 5, 6, 8, 11, 15} pmax [11, 8, 4] score\_max 7

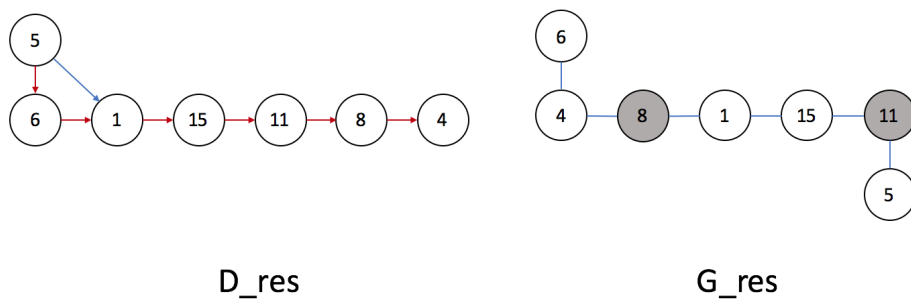


Figure 20 – Exemple de la fonction 6\_5

Si nous avons D\_res et G\_res, nous pouvons obtenir le plus long chemin(lp), le score, pmax et score\_max de ext[11, 8, 4].

ext : [11, 8, 12] R : {1, 5, 8, 11, 12, 15} score = 6 lp [5, 1, 15, 11, 8, 12]

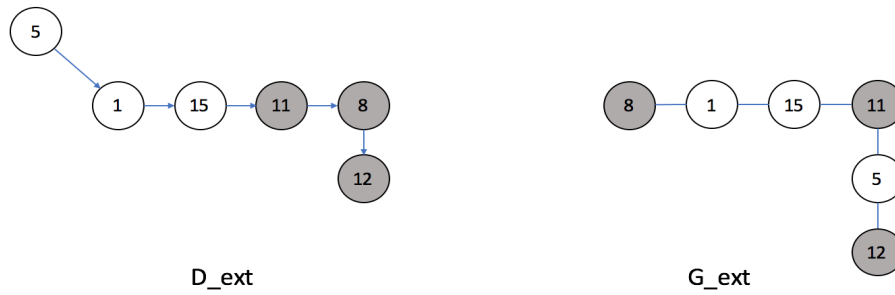


Figure 21 – Exemple de la fonction 6\_6

ext : [15, 11, 8] R : {1, 4, 5, 6, 8, 11, 12, 15} score 7 lp [5, 6, 1, 15, 11, 8, 4]

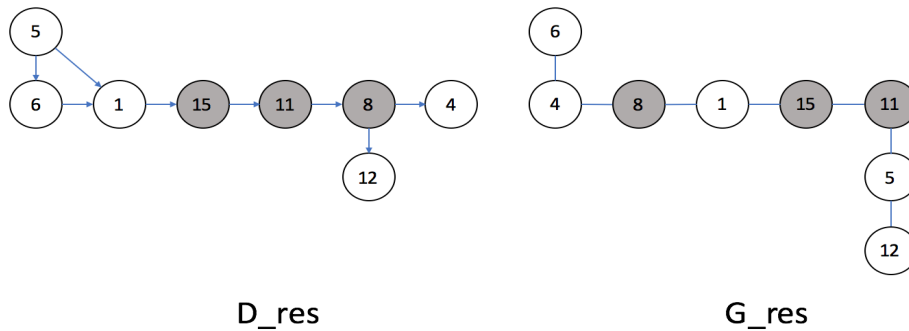


Figure 22 – Exemple de la fonction 6\_7

De même, nous avons également calculé  $\text{ext}[11, 8, 12]$  et  $\text{ext}[11, 8, 15]$ .

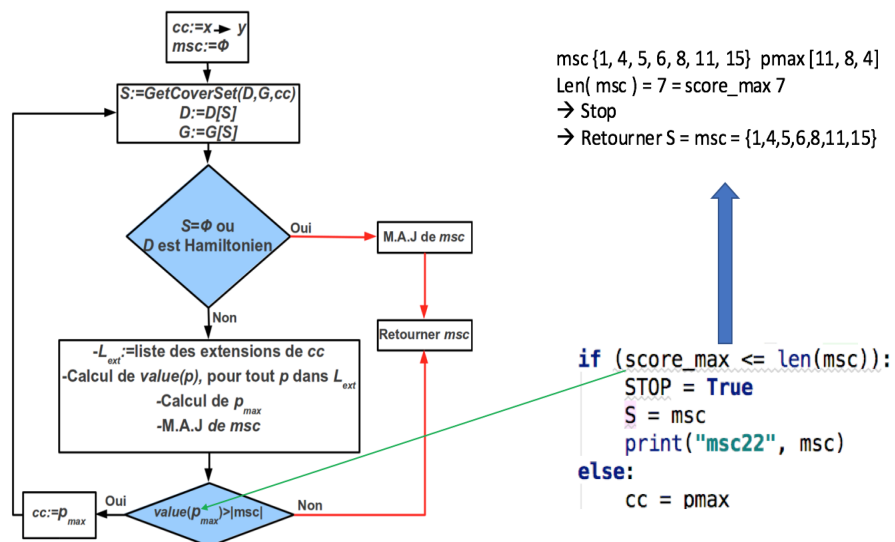


Figure 23 – Exemple de la fonction 6\_8

Donc l'algorithme s'arrête et renvoie A.

# C

# Spécifications non fonctionnelles

## 1 Contraintes de développement et conception

- Langages : Python3.6 ;
- Plateforme optionnelle : PyCharm ;
- Bibliothèques de programmes : Package « networkx-V 1.11 »

## 2 Contraintes de fonctionnement et d'exploitation

Dans ce projet, l'exécution de cet algorithme nécessite des connaissances informatiques. Il y a aucune contrôle d'accès des utilisateurs. La performance est l'objectif que nous voulons améliorer, il n'y a pas de limite.

# D

## Documentation d'installation

Pour installation « python 3.6 », on peut télécharger sur [\[WWW6\]](#)

Choisir la version que vous voulez utiliser.

C'est vraiment simple.

Pour plus d'informations, on peut trouver sur [\[WWW7\]](#)

Pour installer « networkx-V 1.11 » :

Vous pouvez télécharger sur [\[WWW3\]](#)

Les étapes d'installer « networkx 1.11 »

1. Décompressez et changez de répertoire dans le répertoire source (il devrait contenir les fichiers README.txt et setup.py).

2. Lancez « python setup.py install » pour construire et installer

3. (Facultatif) Exécutez « nosetests » pour exécuter les tests si vous avez installé « nose ».

Pour savoir plus de comment installer sur [\[WWW4\]](#)

Pour tous les informations (les documentations des méthodes de « networkx »), on peut trouver sur [\[WWW5\]](#)

## 1 Description des projets

Ce projet consiste à étudier et optimiser l'algorithme pour résoudre le problème de comparaison de réseaux biologique et comment les traduire graphiquement (ONE-TO-ONE SKEWGRAM)

Dans ce projet, les outils et libraires utilisées sont suivant :

- Langages : Python 3.6;
- Plateforme optionnelle : PyCharm;
- Bibliothèques de programmes : Package « networkx-V 1.11 »

Ce projet sera mis en place par Python 3.6. Donc ce projet ne dépend d'aucun matériel, seulement l'environnement de Python 3.6.

Pour l'IDE de Python, j'ai utilisé « PyCharm ». Mais vous pouvez utiliser n'importe quelle IDE.

Nous devons utiliser les classes et plusieurs des fonctions d'un package « networkx-V 1.11 ». Donc ce paquet est obligatoire. Nous devons le télécharger et importer.

Donc, pour savoir les détails du modèle de graphe orienté, le modèle de graphe non-orienté, le chemin L, et les fonctions, on peut trouver sur [[WWW2](#)]

Comment installer l'environnement sera introduit dans la partie « Documentation d'installation ».

## 2 Fonctionnement Du projet

Les fonctionnalités dans ce projet sont très simple. Ce projet n'a que deux fonctions.

La fonction principale consiste à résoudre le problème « ONE – TO – ONE SKEWGRAM ». On peut le dire aussi de trouver le plus long chemin (D, G) – consistant.

Pour résoudre ce problème, il y a deux algorithmes « AlgoH » et « AlgoBB ». En plus, j'ai ajouté un petit algorithme pour trouver les instances difficiles à résoudre pour analyser ce problème.

Pour plus de détails, voir le chapitre 3 et le chapitre 4 du rapport.



### 3 Description de l'algorithme original

L'algorithme de « getcoverset » :

Il y a trois règle dans ce l'algorithme :

Règle 1 : Connexité dans  $D$  (accessibilité);

Règle 2 : Connexité dans  $G$  (ponts);

Règle 3 : Connexité dans  $D$  et  $G$  simultanément (composantes connexes communes)

L'algorithme, appelé « GETCOVERSET » est pour calculer l'ensemble couvrant d'un chemin donné.

L'algorithme de « algoH » :

---

**Algorithme 2**  $\text{ALGOH}(D, G, xy)$

---

**Entrées :** Un DAG  $D = (V, A(D))$ , un graphe non-orienté  $G = (V, E(G))$ , un arc  $xy \in A(D)$ .

**But :** Calcul d'un ensemble  $S \subseteq V$  tels que  $D[S]$  est un chemin  $(D, G)$ -consistant passant par  $xy$ , ou  $S = \emptyset$ .

```

1: /* msc : meilleure solution courante ; cc : chemin courant */
2: /* L_ext : la liste des chemins dans D qui sont obtenus en prolongeant
   le chemin cc, dans D, par un seul sommet */
3: /* DEC : les sous-graphes de D induits par les ensembles couvrants
   des chemins dans L_ext */
4: /* HEC : les sous-graphes Hamiltoniens induits par les ensembles
   couvrants des chemins dans L_ext */
   /* s (resp. t) est la source (resp. le terminal) du chemin courant cc */
5: msc :=  $\emptyset$ ; cc :=  $x \rightarrow^D y$ ; s := x; t := y; STOP := faux;
6: tantque (STOP = faux) faire
7:   S := GETCOVERSET(D, G, cc); D := D[S]; G := G[S];
8:   si (S =  $\emptyset$  ou D est Hamiltonien) alors
9:     STOP := vrai;
10:   si |msc| > |S| alors S := msc finsi
11:   sinon
12:     L_ext :=  $\emptyset$ ; /* extension du chemin courant cc =  $s \rightsquigarrow^D t$  */
13:     pour tout v qui est un prédécesseur de s ou successeur de t faire
14:       p = EXTEND(cc, v); /* extension de cc par v */
15:       L_ext := L_ext  $\cup$  {p};
16:     fin pour
17:     DEC := {D[COVERSET(D, G, p)] : p  $\in$  L_ext};
18:     HEC := {d  $\in$  DEC : d est Hamiltonien};
19:     Soit h_max  $\in$  HEC t.q. |V(h_max)| = max{|V(h)| : h  $\in$  HEC}
20:     si |msc| < |V(h_max)| alors msc := V(h_max) finsi
21:     Soit p_max = s_max  $\rightsquigarrow^D$  t_max t.q.
       value(p_max) = max{value(p) : p  $\in$  L_ext};
22:     si |msc|  $\geq$  value(p_max) alors
23:       /* il n'existe aucun chemin (D, G)-consistant passant par xy et
       de longueur supérieure à msc */
       S := msc; STOP := vrai;
24:     sinon
25:       cc := p_max; /* choix de l'extension la plus prometteuse */
26:       s := s_max; t := t_max
27:     finsi
28:   finsi
29:   finsi
30: fin tantque
31: return S

```

---

Figure 1 – L'algorithme de algoH

L'algorithme de « algoBB » :

Évaluation (Règle 1).

L'algorithme `ALGOBB` est défini comme suit.

1. Créer la racine de l'arbre  $TS$  associée à l'arc  $xy$ .
2. **Tant que** ( $TS$  contient un sommet  $s$  à explorer) **faire**
  - (a) évaluer  $TS$  (**Règle 1**);
  - (b) séparer  $TS$  par rapport au sommet  $s$ ;
  - (c) élaguer  $TS$  (**Règle 2**);
3. Soit  $s_{max}$  une feuille de  $TS$  vérifiant :  $BBvalue(s_{max}) \geq BBvalue(s)$ , pour toute feuille  $s$  de  $TS$ . Soit  $p_{max} = p(s_{max})$ .
4. Si  $p_{max}$  est  $(D, G)$ -consistant, alors  $p_{max}$  est le plus long chemin  $(D, G)$ -consistant passant par  $xy$ . Sinon, il n'y a aucun chemin  $(D, G)$ -consistant passant par  $xy$ .

Figure 2 – L'algorithme de `algoBB`

Soit  $s_1, s_2, \dots, s_k$  l'ensemble des sommets à explorer. Nous choisissons le sommet  $s$  tel que  $BBvalue(s) = \max BBvalue(s_i) : 1 \leq i \leq k$ . Dans le cas où il y a plusieurs sommets  $s_i$  dont  $BBvalue(s_i)$  est maximum, on en choisit arbitrairement un parmi eux.

Élagage (Règle 2).

Soit  $S_{max}$  un sommet de  $TS$  satisfaisant les conditions suivantes :

(i)  $s_{max}$  a été déjà exploré, et (ii) pour tout sommet  $s$  de  $TS$ , si  $s$  a été déjà exploré, alors  $BBvalue(s_{max}) \geq BBvalue(s)$ . On supprime dans  $TS$  toutes les feuilles  $s$  tel que  $BBvalue(s) < BBvalue(s_{max})$ . Cette suppression est appliquée récursivement sur les sommets (sauf  $S_{max}$ ) qui deviennent feuilles après la suppression de leurs fils.

Pour savoir le plus d'information regarder l'article « Comparaison de réseaux biologiques » de monsieur Hamed Mohamed Babou.

## 4 Description de l'algorithme améliorée

AlgoBB Version2 :

Dans ce version, je vais améliorer `algoBB`(branch and bound) pour réduire le coût de temps. Et l'idée centrale de l'optimisation est la recherche "heuristique". Pour chaque branche, nous explorons seulement  $2(\alpha)$  sommets qui a le plus grand score, pas tous les sommets.

AlgoBB Version3 :

Dans ce version, je vais améliorer `algoBB`(branch and bound) pour réduire le coût de temps. Et l'idée centrale de l'optimisation est la recherche "heuristique" aussi. La différence entre AlgoBB Version3 et AlgoBB Version 2 est : Dans Version2, nous explorons seulement  $2(\alpha)$  sommets qui a le plus grand score par chaque branche, pas tous les sommets. Dans Version3, nous explorons seulement  $2(\alpha)$  sommets qui a le plus grand score par chaque niveau, pas tous les sommets. Donc, C'est la troisième version de cet algorithme pour résoudre le problème ONE-TO-ONE SKEWGRAM.

## 5 Les paramètres pour exécuter ce projet

Dans le fonction « main », on va appeler le fonction « execute ».

Les paramètres dans fonction « main » sont : Si `RUN_SOURCE` = 1, on va utiliser `algoBB` de version source. Si `RUN_VERSION1` = 1, on va utiliser `algoBB` de version1. Si `RUN_VERSION2` = 1, on va utiliser `algoBB` de version2. Si `RUN_ERDOS_RENYI` = 1, on va utiliser le graphe de type "ERDOS\_RENYI". Si `RUN_SCALEFREE` = 1, on va utiliser le graphe de type "SCALEFREE".

## 6 Les autres fonctions

Pour savoir les fonctions pour analyser, regarder « Cahier de Test ».

# F

## Cahier de Test

Tout d'abord, j'ai fait les "Test unitaire" de l'algorithme principal.

```
import unittest
import networkx as nx
import Algo_source
import AlgoBB_V2
import AlgoBB_V3
import Gerer_graphe as graphe

class Test(unittest.TestCase):
    def test_ccc(self):...
    def test_ccc2(self):...
    def test_est_pont(self):...
    def test_s_t_Graph(self):...
    def test_getCoverSet(self):...
    def test_algoBB(self):...
    def test_algoH(self):...

if __name__ == '__main__':
    unittest.main()
```

Figure 1 – Test unitaire

Et dans l'algorithme « test\_algoBB », j'ai testé les trois versions de « algoBB ». Le résultat est comme ci-dessous :



Figure 2 – Résultat de test unitaire

On peut savoir qu'il n'y a pas de problème pour les entrées et sorties.

Pour vérifier les paramètres et la solution de l'algoH et l'algoBB. J'ai exécuté 360\_1 de type de graphe « scarefree ».

A1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	nb_nodes	nb_instance	nb_L	L	longueur_sol_h	time_opt_nb_boucle	list_nb_ex	list_ext	longueur_sol_bb	nb_branch	time_opt	ecart_long	ecart_temp(h-bb)								
2	360	1	1 [8, 1]		7 [92, 2, 8, 1]	0.041156	1 [16]	[8, 1, 4]	7 [92, 2, 8, 1]	1	0.064345	0	-0.02319								
3	360	1	2 [2, 116]		3 [64, 2, 116]	0.006384	1 [3]	[64, 2, 116]	3 [64, 2, 116]	1	0.011002	0	-0.00462								
4	360	1	3 [2, 23]		11 [5, 2, 23, 3]	0.291443	1 [7]	[2, 23, 3]	11 [5, 2, 23, 3]	1	0.0551414	0	-0.25997								
5	360	1	4 [174, 182]		0 [ ]	0.008526	1 [ ]	[ ]	0 [ ]	0	0.007321	0	0.001205								
6	360	1	5 [2, 234]		3 [64, 2, 234]	0.007965	1 [3]	[64, 2, 234]	3 [64, 2, 234]	1	0.012512	0	-0.00455								
7	360	1	6 [2, 139]		0 [ ]	0.005378	1 [ ]	[ ]	0 [ ]	0	0.005383	0	-5E-06								
8	360	1	7 [35, 135]		0 [ ]	0.004905	1 [ ]	[ ]	0 [ ]	0	0.004446	0	0.000459								
9	360	1	8 [2, 13]		11 [5, 2, 13, 3]	0.309895	1 [7]	[2, 13, 3]	11 [5, 2, 13, 3]	1	0.0586854	0	-0.27696								
10	360	1	9 [169, 128]		0 [ ]	0.005509	1 [ ]	[ ]	0 [ ]	0	0.005293	0	0.000216								
11	360	1	10 [2, 10]		7 [92, 2, 10]	0.028659	1 [5]	[2, 10, 1]	7 [92, 2, 10]	1	0.054618	0	-0.02596								
12	360	1	11 [213, 21]		0 [ ]	0.004999	1 [ ]	[ ]	0 [ ]	0	0.005977	0	-0.00098								
13	360	1	12 [2, 234]		3 [64, 2, 234]	0.006119	1 [3]	[64, 2, 234]	3 [64, 2, 234]	1	0.010601	0	-0.00448								
14	360	1	13 [36, 152]		4 [92, 2, 36]	0.010163	1 [2]	[28, 36, 1]	5 [92, 2, 28]	1	0.02124	-1	-0.01108								
15	360	1	14 [1, 307]		0 [ ]	0.007476	1 [ ]	[ ]	0 [ ]	0	0.007191	0	0.000285								
16	360	1	15 [2, 168]		0 [ ]	0.004018	1 [ ]	[ ]	0 [ ]	0	0.004322	0	-0.0003								
17	360	1	16 [302, 3]		0 [ ]	0.007527	1 [ ]	[ ]	0 [ ]	0	0.008563	0	-0.00104								
18	360	1	17 [42, 1]		7 [92, 2, 42]	0.050014	1 [16]	[42, 1, 4]	7 [92, 2, 42]	1	0.077395	0	-0.02738								
19	360	1	18 [4, 144]		13 [104, 2, 8]	1.113706	6 [10, 2, 22]	[63, 3, 11]	13 [104, 2, 8]	6	1.846218	0	-0.73251								
20	360	1	19 [150, 168]		0 [ ]	0.004268	1 [ ]	[ ]	0 [ ]	0	0.004471	0	-0.0002								
21	360	1	20 [76, 127]		0 [ ]	0.004039	1 [ ]	[ ]	0 [ ]	0	0.005471	0	-0.00143								

Figure 3 – Résultat de algoH et algoBB

Pour comparer les trois versions de « algoBB », j'ai comparé ces fichiers exportés.

nb_nodes	nb_instance	nb_L	L	Version original				Version Résultat 1				Version Résultat 2			
				longueur_bb	sol_bb	nb_branch	time_opt_bb	longueur_bb	sol_bb	nb_branch	time_opt_bb	longueur_bb	sol_bb	nb_branch	time_opt_bb
360	1	1	[26, 148]	21 [89, 323, 325, 1]	3	1.430148	21 [89, 323, 325]	2	1.158704	21 [89, 323, 325]	3	1.633448			
360	1	2	[197, 72]	15 [197, 72, 122, 7]	4	1.661768	15 [197, 72, 122]	4	1.718625	15 [197, 72, 122]	4	1.807054			
360	1	3	[197, 76]	10 [197, 76, 287, 2]	9	1.196278	10 [197, 76, 287]	9	1.252667	10 [197, 76, 287]	10	1.396969			
360	1	4	[207, 143]	16 [89, 323, 325, 1]	3	1.131808	16 [89, 323, 325]	3	0.871636	16 [89, 323, 325]	3	0.819024			
360	1	5	[328, 300]	21 [89, 323, 325, 1]	4	2.109904	21 [89, 323, 325]	4	3.46629	21 [89, 323, 325]	4	2.239722			
360	1	6	[23, 345]	12 [89, 21, 223, 35]	7	0.984988	12 [89, 21, 223]	7	1.418744	12 [89, 21, 223]	9	1.223077			
360	1	7	[52, 201]	16 [89, 21, 223, 35]	13	2.506689	16 [89, 21, 223]	12	3.25402	16 [89, 21, 223]	11	2.191735			
360	1	8	[35, 243]	10 [323, 325, 251]	18	1.77536	10 [323, 325, 251]	18	2.615627	10 [323, 325, 251]	13	1.384938			
360	1	9	[261, 7]	5 [89, 21, 251, 26]	3	0.331623	5 [89, 21, 251]	3	0.411846	5 [89, 21, 251]	3	0.318183			
360	1	10	[89, 208]	8 [89, 208, 259, 3]	10	1.05918	8 [89, 208, 259]	10	1.686002	8 [89, 208, 259]	9	0.908537			
360	1	11	[332, 16]	19 [89, 323, 325, 1]	126	23.550172	19 [89, 323, 325]	113	21.881421	19 [89, 323, 325]	27	6.035939			
360	1	12	[168, 100]	10 [89, 21, 223, 25]	4	0.459868	10 [89, 21, 223]	7	0.716085	10 [89, 21, 223]	7	0.676212			
360	1	13	[186, 16]	10 [323, 325, 179]	24	2.36444	10 [323, 325, 179]	19	1.709112	10 [323, 325, 179]	13	1.216015			
360	1	14	[169, 56]	14 [89, 21, 223, 35]	24	3.361731	14 [89, 21, 223]	19	2.644541	14 [89, 21, 223]	13	1.846532			
360	1	15	[35, 243]	10 [323, 325, 251]	18	1.7912	10 [323, 325, 251]	18	1.885428	10 [323, 325, 251]	13	1.365214			
360	1	16	[52, 94]	17 [89, 21, 223, 35]	11	2.567646	17 [89, 21, 223]	11	2.575724	17 [89, 21, 223]	19	3.489898			
360	1	17	[198, 353]	16 [89, 323, 325, 1]	5	1.319625	16 [89, 21, 223]	45	6.896191	16 [89, 21, 223]	17	3.235694			
360	1	18	[279, 130]	15 [89, 21, 223, 25]	51	6.894988	15 [89, 21, 223]	47	6.306522	15 [89, 21, 223]	17	2.731497			
360	1	19	[86, 201]	9 [89, 323, 325, 8]	26	3.725504	9 [89, 323, 325]	24	2.157633	9 [89, 323, 325]	11	1.175482			
360	2	1	[347, 34]	19 [242, 339, 10]	58	20.16807	19 [242, 339, 10]	44	12.415097	19 [242, 339, 10]	19	6.828265			
360	2	2	[15, 170]	12 [15, 170, 148, 9]	3	0.604444	12 [15, 170, 148]	3	0.487011	12 [15, 170, 148]	5	0.692956			
360	2	3	[306, 332]	8 [242, 123, 136]	34	3.541284	8 [242, 123, 136]	10	0.964194	8 [242, 123, 136]	7	0.664314			
360	2	4	[76, 210]	23 [261, 249, 35]	6	4.978218	23 [261, 249, 35]	6	4.626748	23 [242, 339, 11]	15	8.04506			
360	2	5	[183, 200]	11 [261, 249, 35]	11	1.351917	11 [261, 249, 35]	11	1.25524	11 [261, 249, 35]	11	1.20347			
360	2	6	[91, 246]	28 [242, 339, 10]	26	25.823156	28 [242, 339, 11]	16	15.415666	28 [242, 339, 11]	11	10.739608			
360	2	7	[126, 258]	13 [126, 258, 165]	4	0.686185	13 [126, 258, 165]	4	0.664989	13 [126, 258, 165]	5	0.741444			
360	2	8	[223, 74]	16 [261, 249, 35]	10	2.059405	16 [261, 249, 35]	7	1.53676	16 [261, 249, 35]	7	1.482644			
360	2	9	[250, 222]	24 [261, 249, 35]	8	6.14579	24 [261, 249, 35]	5	4.279242	24 [261, 249, 35]	5	4.109193			

Figure 4 – Comparaison de trois versions

On peut trouver que pour la plupart des données, les résultats sont les mêmes. Mais parce que c'est « heuristique », donc pour très peu de cas, nous ne trouverons pas la réponse.

Comme la partie rouge de la table ci-dessous :

360	4	16 [195, 20]		15 [9, 353, 63, 22]	48	4.842554		15 [9, 353, 63, 22]	33	3.351024		15 [9, 353, 63, 22]	21	2.206008
360	4	17 [215, 305]		22 [9, 353, 63, 22]	109	17.318491		22 [9, 353, 63, 22]	109	15.699645		22 [9, 353, 63, 22]	33	4.79585
360	4	18 [342, 305]		19 [9, 353, 63, 22]	85	9.95297		19 [9, 353, 63, 22]	85	10.947829		19 [9, 353, 63, 22]	27	3.457335
360	4	19 [353, 76]		0 [ ]	4	0.373078		0 [ ]	3	0.314933		0 [ ]	3	0.296088
360	4	20 [259, 60]		7 [9, 353, 63, 75]	4	0.394439		7 [9, 353, 63, 75]	4	0.413146		7 [9, 353, 63, 75]	4	0.389801
360	5	1 [38, 125]		18 [214, 260, 8]	20	3.119504		18 [214, 260, 8]	14	2.388062		18 [214, 260, 8]	15	2.375024
360	5	2 [336, 113]		6 [260, 87, 77, 334]	13	1.049944		6 [260, 87, 77, 334]	9	0.772397		6 [260, 87, 77, 334]	9	0.719738
360	5	3 [333, 69]		34 [214, 260, 116]	5	6.550667		34 [214, 260, 116]	5	7.072512		34 [214, 260, 116]	5	6.522234
360	5	4 [23, 55]		24 [214, 260, 116]	5	1.924241		24 [214, 260, 116]	7	2.785477		24 [214, 260, 116]	9	3.193516
360	5	5 [156, 63]		21 [214, 260, 8]	36	6.759529		21 [214, 260, 8]	36	6.686496		21 [214, 260, 8]	21	3.76292
360	5	6 [37, 347]		38 [214, 260, 72]	292	190.242444		38 [214, 260, 72]	269	166.72954		38 [214, 260, 72]	65	45.184109
360	5	7 [105, 98]		14 [214, 260, 116]	4	1.053029		14 [214, 260, 116]	4	0.616361		14 [214, 260, 116]	5	0.750374
360	5	8 [286, 350]		28 [214, 260, 8]	163	49.61027		28 [214, 260, 8]	148	41.266699		28 [214, 260, 8]	45	13.186753

360	1	11	[332, 16]	19	[89, 323, 325, 1]	126	23.550172	19	[89, 323, 325]	113	21.881421	19	[89, 323, 325]	27	6.035939
-----	---	----	-----------	----	-------------------	-----	-----------	----	----------------	-----	-----------	----	----------------	----	----------

Figure 6 – Exemples sans solutions

Pour savoir que les fonctions dans « Gerer\_graphe » et les fonctions dans « Gerer\_fichier » sont corrects, on peut regarder les fichiers exportés.

Les dossiers suivant sont générés par « execute ».

Dans chaque répertoire, les fichiers que nous générons sont :

create\_graph\_erdos\_renyi() :

-grapheD.txt

-grapheG.txt

-arcL.txt

-D.gml

-G.gml

save\_log(dirPath, L, indexL, time\_opt\_h, time\_opt\_bb, nb\_boucle, list\_nb\_ext, sol\_h, list\_ext, sol\_bb, nb\_branch, ecart\_longeur, ecart\_temps) :

-solution1.txt->solution20.txt

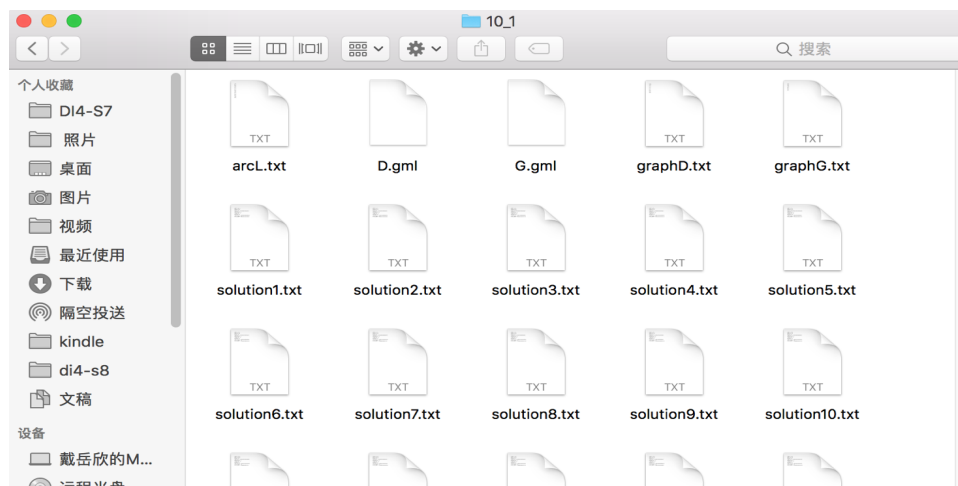


Figure 7 – Les fichiers dans 10\_1

Dans chaque solution, on peut vérifier les paramètres.

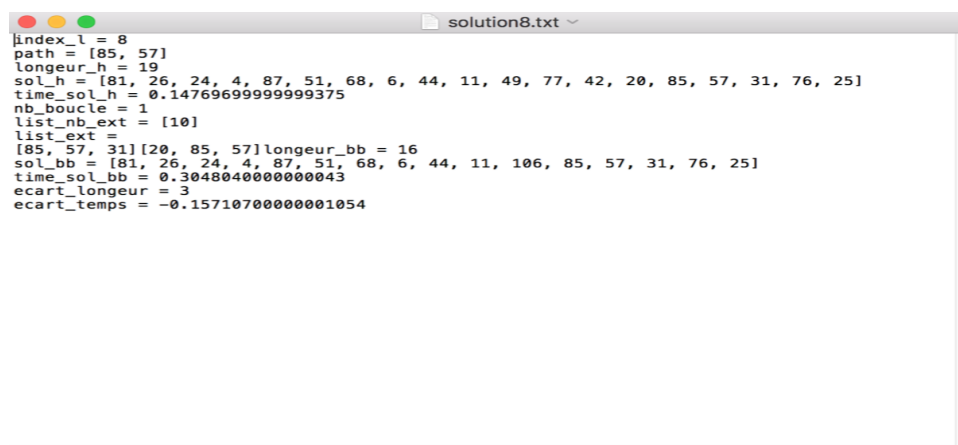


Figure 8 – Les paramètres stockés par save\_log

Pour vérifier la fonction « choose\_n\_arc(n) » dans « Donnee\_analyse », on peut regarder l'image ci-dessous :

```
Run 12998 .....
12999 [360, 281] 35 17.1798100000001453 [214, 260, 116, 360, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 1]
Run 12999 .....
13000 [360, 298] 35 5.0448099999999296 [214, 260, 116, 360, 298, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 1]
Run 13000 .....
13001 [360, 304] 35 35.732920999999798 [214, 260, 116, 360, 304, 28, 127, 340, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 1]
Run 13001 .....
13002 [360, 308] 35 0.170782999999504611 [214, 260, 116, 360, 308, 211, 225, 113, 315, 132, 101]
Run 13002 .....
13003 [360, 312] 35 3.08943999999524467 [214, 260, 116, 360, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 101]
Run 13003 .....
13004 [360, 324] 35 25.617601000000538 [214, 260, 116, 360, 324, 148, 62, 267, 190, 279, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 1]
Run 13004 .....
13005 [360, 326] 35 44.322710999998616 [214, 260, 116, 360, 326, 288, 259, 325, 111, 200, 216, 18, 290, 328, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 1]
Run 13005 .....
13006 [360, 329] 35 51.032036000002548 [214, 260, 116, 360, 329, 193, 310, 333, 19, 131, 213, 259, 325, 111, 200, 216, 18, 290, 328, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 1]
Run 13006 .....
13007 [360, 331] 35 43.703978000003411 [214, 260, 116, 360, 331, 68, 288, 259, 325, 111, 200, 216, 18, 290, 328, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 1]
Run 13007 .....
13008 [360, 348] 35 81.58633399999235 [214, 260, 116, 360, 348, 160, 277, 155, 221, 254, 38, 206, 349, 27, 334, 90, 82, 173, 286, 359, 333, 19, 131, 213, 259, 325, 111, 200, 216, 18, 290, 328, 153, 356, 209, 341, 168, 169, 258, 179, 281, 355, 135, 30, 248, 351, 346, 66, 80, 2, 44, 227, 287, 147, 252, 154, 271, 91, 273, 224, 1, 138, 265, 152, 21, 76, 337, 339, 241, 312, 358, 22, 50, 79, 343, 194, 174, 45, 283, 220, 108, 251, 172, 107, 279, 184, 94, 345, 321, 289, 219, 4, 322, 245, 53, 307, 347, 272, 335, 315, 132, 1]
Run 13008 .....
13009 [360, 354] 35 0.2414820000412874 [214, 260, 87, 360, 354, 208, 84, 250, 110, 335, 315, 132, 101]
```

Figure 9 – Les résultats de parcourir tous les arcs

Parce qu'il faut parcourir tous les arcs dans graphe D (360\_5 erdos), ça prend beaucoup de temps (environ 100h).

Je stocke la sortie dans console en Excel.

Et on peut regarder le fichier « arcL.txt » comme ci-dessous :

Figure 10 – Les instances qui nécessitent des calculs complexes

Et j'ai comparé les solutions de trois versions, on peut savoir que pour les données difficiles à calculer, il y a deux groupes. Dans le premier group, « nb\_branch » est grand, on peut améliorer et réduire le temps.

		Donnée original					Résultat1					Résultat2				
nb_nodes	nb_instance	nb_1	longueur	nb_soi	nb_branch	time_opt	longueur	nb_soi	nb_branch	time_opt	longueur	nb_soi	nb_branch	time_opt		
360	5	13 [82, 96]	48	[214, 260, 1]	176	255.807187	48	[214, 260, 1]	176	255.807187	48	[214, 260, 1]	79	98.187427		
360	5	17 [237, 29]	48	[214, 260, 1]	176	221.000453	48	[214, 260, 1]	176	228.26444	48	[214, 260, 1]	79	103.8527		
360	5	12 [82, 63]	45	[214, 260, 1]	143	204.622023	45	[214, 260, 1]	143	185.78399	45	[214, 260, 1]	77	93.948342		
360	5	14 [82, 197]	45	[214, 260, 1]	143	176.745244	45	[214, 260, 1]	143	185.754023	45	[214, 260, 1]	77	91.023442		
360	5	8 [17, 347]	38	[214, 260, 7]	280	173.089866	38	[214, 260, 7]	280	173.80025	38	[214, 260, 8]	69	92.738908		
360	5	23 [253, 285]	45	[214, 260, 1]	112	143.992444	45	[214, 260, 1]	78	108.5054	45	[214, 260, 1]	77	98.986609		
360	5	15 [162, 350]	85	[214, 260, 8]	224	125.859708	85	[214, 260, 8]	254	115.29503	85	[214, 260, 8]	89	95.126703		
360	5	11 [65, 263]	107	[214, 260, 1]	1	113.521908	107	[214, 260, 1]	1	103.49779	107	[214, 260, 1]	1	99.517611		
360	5	16 [214, 260]	107	[214, 260, 7]	1	110.855578	107	[214, 260, 7]	1	119.62947	107	[214, 260, 7]	1	102.45082		
360	5	1 [1, 138]	107	[214, 260, 1]	1	104.724462	107	[214, 260, 1]	1	107.56383	107	[214, 260, 1]	1	109.900319		
360	5	25 [260, 87]	107	[214, 260, 8]	1	104.682097	107	[214, 260, 8]	1	115.48209	107	[214, 260, 8]	1	100.43004		
360	5	96 [903, 116]	107	[214, 260, 1]	1	103.751287	107	[214, 260, 1]	1	118.61088	107	[214, 260, 1]	1	102.45954		
360	5	24 [254, 38]	107	[214, 260, 1]	1	100.764138	107	[214, 260, 1]	1	108.54753	107	[214, 260, 1]	1	95.246673		
360	5	30 [312, 398]	107	[214, 260, 1]	1	99.307557	107	[214, 260, 1]	1	105.36025	107	[214, 260, 1]	1	97.438309		
360	5	33 [325, 111]	107	[214, 260, 1]	1	98.787134	107	[214, 260, 1]	1	100.75476	107	[214, 260, 1]	1	97.035452		
360	5	27 [312, 57]	106	[214, 260, 1]	1	98.683412	106	[214, 260, 1]	1	105.95299	106	[214, 260, 1]	1	95.801729		
360	5	32 [320, 208]	107	[214, 260, 1]	1	97.796342	107	[214, 260, 1]	1	99.038423	107	[214, 260, 1]	1	95.307419		
360	5	21 [251, 238]	107	[214, 260, 1]	1	97.777078	107	[214, 260, 1]	1	112.41498	107	[214, 260, 1]	1	94.465213		
360	5	10 [38, 206]	107	[214, 260, 1]	1	96.743314	107	[214, 260, 1]	1	98.2554	107	[214, 260, 1]	1	100.62128		
360	5	20 [251, 107]	106	[214, 260, 1]	1	96.459507	106	[214, 260, 1]	1	104.696	106	[214, 260, 1]	1	92.565592		
360	5	28 [312, 93]	104	[214, 260, 1]	1	96.103516	104	[214, 260, 1]	1	104.03488	104	[214, 260, 1]	1	93.0705		
360	5	9 [1, 285]	106	[214, 260, 1]	1	95.70383	106	[214, 260, 1]	1	106.2734	106	[214, 260, 1]	1	108.8007		
360	5	22 [252, 154]	107	[214, 260, 1]	1	94.913418	107	[214, 260, 1]	1	101.13434	107	[214, 260, 1]	1	90.638448		
360	5	5 [32, 203]	107	[214, 260, 1]	1	94.702895	107	[214, 260, 1]	1	106.0143	107	[214, 260, 1]	1	102.53274		
360	5	6 [85, 271]	107	[214, 260, 1]	1	93.818262	107	[214, 260, 1]	1	99.779504	107	[214, 260, 1]	1	93.157758		
360	5	7 [37, 210]	106	[214, 260, 1]	1	93.192421	106	[214, 260, 1]	1	104.4977	106	[214, 260, 1]	1	99.443071		
360	5	4 [4, 322]	107	[214, 260, 1]	1	92.940023	107	[214, 260, 1]	1	102.13959	107	[214, 260, 1]	1	93.431899		
360	5	31 [313, 49]	107	[214, 260, 1]	1	92.971283	107	[214, 260, 1]	1	91.988234	107	[214, 260, 1]	1	89.665897		
360	5	29 [312, 343]	103	[214, 260, 1]	1	90.705533	103	[214, 260, 1]	1	99.146283	103	[214, 260, 1]	1	89.194682		
360	5	18 [250, 29]	106	[214, 260, 1]	1	90.623035	106	[214, 260, 1]	1	96.995588	106	[214, 260, 1]	1	86.778574		
360	5	19 [250, 110]	107	[214, 260, 1]	1	90.33244	107	[214, 260, 1]	1	97.135714	107	[214, 260, 1]	1	89.881866		
360	5	30 [332, 248]	104	[214, 260, 1]	1	89.338778	104	[214, 260, 1]	1	91.750474	104	[214, 260, 1]	1	88.756554		
360	5	34 [328, 195]	106	[214, 260, 1]	1	89.272782	106	[214, 260, 1]	1	92.145404	106	[214, 260, 1]	1	87.571449		
360	5	2 [1, 187]	104	[214, 260, 1]	1	88.383137	104	[214, 260, 1]	1	100.2897	104	[214, 260, 1]	1	86.392953		
360	5	9 [38, 27]	105	[214, 260, 1]	1	87.613035	105	[214, 260, 1]	1	96.983315	105	[214, 260, 1]	1	86.196591		

Figure 11 – Le résultat de la comparaison



Pour analyser pourquoi les données sont difficile à calculer.

On peut regarder les quatre fonctions ci-dessous dans « Donnee\_analyse » :

cal\_degree\_level\_1

cal\_degree\_level\_2

cal\_degree(G, LL)

cal\_shortest\_path(G, LL)

Les fichiers ci-dessous générés :



Figure 12 – Les fichiers générés

Je les mets ensemble.

	L	time_opt_bb	TotalDegree	TotalDegree	Max Degree	TotalDegree	Length
13	[62, 98]	255.807187	31	2265	81	150	2
17	[237, 29]	221.920453	29	2204	69	132	3
12	[62, 63]	204.622023	32	2368	81	161	3
14	[62, 197]	176.745246	32	2353	81	149	3
8	[37, 347]	173.089865	27	1980	83	153	3
23	[253, 285]	143.992444	30	2179	75	145	3
15	[162, 350]	123.659708	32	2323	74	148	3
11	[45, 283]	115.521908	62	4545	76	134	2
16	[214, 260]	110.855578	89	6414	74	146	3
1	[1, 138]	104.742462	67	5086	69	137	2
25	[260, 87]	104.690287	79	5803	85	159	3
26	[260, 116]	103.751287	75	5440	74	144	2
24	[254, 38]	100.764136	83	6125	64	127	3
30	[312, 358]	99.307557	77	5583	67	129	3
33	[325, 111]	98.787134	76	5594	75	134	3
27	[312, 57]	98.663412	79	5707	81	148	3
32	[320, 208]	97.796342	85	6148	83	150	3
21	[251, 238]	97.777078	60	4361	86	145	3
10	[38, 206]	96.743314	66	4828	63	125	3
20	[251, 107]	96.459507	65	4772	71	130	3
28	[312, 93]	96.103516	79	5762	67	129	3
3	[1, 265]	95.70393	69	5167	76	144	3
22	[252, 154]	94.913418	76	5501	69	132	3
5	[32, 203]	94.702895	77	5586	84	139	3
6	[35, 271]	93.818292	76	5500	77	151	3
7	[37, 210]	93.192421	67	4846	83	152	3
4	[4, 322]	92.940023	61	4504	80	146	2
31	[313, 49]	90.971351	71	5121	75	148	2
29	[312, 343]	90.720533	76	5516	71	138	3

Figure 13 – Les paramètres stockés pour analyser

On peut faire les analyses par les paramètres ci-dessus, nous pouvons analyser.





# Webographie

- [WWW1] NETWORKX. *networkx*. URL : <https://networkx.github.io/documentation/networkx-1.10/overview.html> (visité le 09/12/2017).
- [WWW2] NETWORKX. *networkx*. URL : <https://networkx.github.io/documentation/networkx-1.11/tutorial/index.html> (visité le 09/12/2017).
- [WWW3] NETWORKX2. *networkx2*. URL : <https://pypi.python.org/pypi/networkx/1.11> (visité le 09/12/2017).
- [WWW4] NETWORKX3. *networkx3*. URL : <https://networkx.github.io/documentation/networkx-1.11/install.html%20-%20quick-install> (visité le 09/12/2017).
- [WWW5] NETWORKX4. *networkx4*. URL : <https://networkx.github.io/documentation/networkx-1.11/tutorial/index.html> (visité le 09/12/2017).
- [WWW6] PYTHON. *PYTHON*. URL : <https://www.python.org> (visité le 09/12/2017).
- [WWW7] PYTHON. *PYTHON*. URL : <https://wiki.python.org/moin/BeginnersGuide/Download> (visité le 09/12/2017).



## Bibliographie

- [1] Reka ALBERT. « Scale-free networks in cell biology ». In : *Journal of cell science* 118.21 (2005), p. 4947–4957.
- [2] Eric ALM et Adam P ARKIN. « Biological networks ». In : *Current opinion in structural biology* 13.2 (2003), p. 193–202.
- [3] Anh-Tuan GAI, Michel HABIB, Christophe PAUL et Mathieu RAFFINOT. « Identifying common connected components of graphs ». In : (2003).
- [4] Hafedh MOHAMED BABOU. « Comparaison de réseaux biologiques ». Thèse de doct. Nantes, 2012.

# Étude algorithmique d'un problème de comparaison de réseaux biologique

Yuexin DAI

Encadrement : Emmanuel NÉRON et Ameer SOUKHAL

## Objectifs

Les objectifs sont:

- Étudier le problème de comparaison de réseaux biologique.
- Étudier l'algorithmique et le code existant (ALGOH et ALGOBB).
- Améliorer les performances de l'algorithme « ALGOBB ».



## Mise en œuvre

Ce projet est d'étudier et améliorer l'algorithmique et le code existant de « ALGOH » pour résoudre le problème de comparaison de réseaux biologique et comment les traduire graphiquement (ONE-TO-ONE SKEWGRAM). Lors de la réalisation de ce projet, l'algorithmique (AlgoH) sera mis en place par Python 3.6 avec un package « networkx-V 1.11 ».

## Résultats attendus

- Analyse de performance de l'algorithme « ALGOH ».
- Améliorer les performances de l'algorithme « ALGOH ».



# Étude algorithmique d'un problème de comparaison de réseaux biologiques

Yuxin DAI

Encadrement : Emmanuel NÉRON et Ameer SOUKHAL

## Objectifs

Les objectifs sont:

- Étudier le problème de comparaison de réseaux biologique.
- Étudier l'algorithme et le code existant (ALGOH et ALGOBB).
- Améliorer les performances de l'algorithme « ALGOBB ».

## Mise en œuvre

Ce projet est d'étudier et améliorer l'algorithme et le code existant de « ALGOH » pour résoudre le problème de comparaison de réseaux biologique et comment les traduire graphiquement (ONE-TO-ONE SKEWGRAM). Lors de la réalisation de ce projet, l'algorithme (AlgoH) sera mis en place par Python 3.6 avec un package « networkx-V 1.11 ».

## Résultats attendus

- Analyse de performance de l'algorithme « ALGOH ».
- Améliorer les performances de l'algorithme « ALGOH ».



# Étude algorithmique d'un problème de comparaison de réseaux biologique

## Résumé

Ce rapport est pour le PROJET RECHERCHE & DÉVELOPPEMENT réalisée dans le cadre de la 5ème année du cycle d'ingénieur à l'École Polytechnique Tours. Ce rapport contient le cahier de spécification, l'état de l'art/veille, l'analyse et conception, et la planification de deux semestres. Le sujet dans ce projet est « Étude algorithmique d'un problème de comparaison de réseaux biologique ». Donc le plus important dans ce rapport est la présentation de l'algorithme « Heuristique » qui s'appelle « ALGOH ». Cet algorithme est dans le but de résoudre le problème ONE-TO-ONE SKEWGRAM. En d'autres termes, il permet de trouver un plus long chemin entre deux graphes passant par un arc.

## Mots-clés

algorithme, biologique, ONE-TO-ONE SKEWGRAM

## Abstract

This report is for the RESEARCH & DEVELOPMENT PROJECT carried out as part of the 5th year of the engineering cycle at Polytech' Tours. This report contains the requirements specification, state of the art / standby, analysis and design, and the planning of two semesters. The subject in this project is "Algorithms for the comparison of biological networks". So the most important thing in this report is the presentation of the "Heuristic" algorithm called "ALGOH". This algorithm is for the purpose of solving the problem ONE-TO-ONE SKEWGRAM. In other words, it makes it possible to find a longest path between two graphs passing through an arc.

## Keywords

algorithm, biological, ONE-TO-ONE SKEWGRAM

**Tuteurs académiques**  
Emmanuel NÉRON  
Ameur SOUKHAL

**Étudiant**  
Yuexin DAI (DI5)