

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

Projet Recherche & Développement

2018-2019

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



Entreprise

Laboratoire Informatique de Tours (EA630)



Tuteurs entreprise

Emmanuel NÉRON (Directeur de Polytech Tours)

Ronan BOCQUILLON (Enseignant chercheur)

Étudiant

Morad SANBA (DI5)

Tuteurs académiques

Emmanuel NÉRON

Ronan BOCQUILLON

Liste des intervenants

Entreprise

Laboratoire Informatique de Tours (EA630)
64 avenue Jean Portalis
37200 Tours
polytech.univ-tours.fr



Nom	Email	Qualité
Morad SANBA	morad.sanba@etu.univ-tours.fr	Étudiant DI5
Emmanuel NÉRON	emmanuel.neron@etu.univ-tours.fr	Tuteur académique, Département Informatique
Ronan BOCQUILLON	ronan.bocquillon@etu.univ-tours.fr	Tuteur académique, Département Informatique
Emmanuel NÉRON	emmanuel.neron@etu.univ-tours.fr	Tuteur entreprise, Directeur de Polytech Tours
Ronan BOCQUILLON	ronan.bocquillon@etu.univ-tours.fr	Tuteur entreprise, Enseignant chercheur



Avertissement

Ce document a été rédigé par Morad SANBA susnommé l'auteur.

L'entreprise Laboratoire Informatique de Tours (EA630) est représentée par Emmanuel Néron et Ronan BOCQUILLON susnommés les tuteurs entreprise.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Emmanuel NÉRON et Ronan BOCQUILLON 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

Morad SANBA, *Étude algorithmique d'un problème de comparaison de réseaux biologique polytech*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
  author={SANBA, Morad},
  title={Étude algorithmique d'un problème de comparaison de réseaux biologique
    polytech},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2018-2019}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	v
INTRODUCTION	1
REMERCIEMENTS	2
1 INTRODUCTION PROJET	3
1 Contexte de la réalisation	3
1.1 Contexte	3
1.2 Les acteurs du projet.....	4
1.3 Description du problème.....	4
1.4 Objectifs	8
1.5 Hypothèses.....	8
1.6 Bases méthodologiques.....	8
2 Description générale	10
1 Environnement du projet	10
2 Caractéristiques des utilisateurs	10
3 Fonctionnalités et structure générale du système	11
4 Contraintes de développement, d'exploitation et de maintenance	14

5	Description des interfaces externes du logiciel	14
5.1	Les entrées et les sorties.....	15
3	Etat de l'art/veille	16
1	Modélisation de réseaux biologiques	16
1.1	Réseaux homogènes VS réseaux hétérogènes.....	16
2	Théorie des Graphes.....	16
2.1	Définitions et notions de base	16
2.2	Notions de complexité	18
2.2.1	Classe P et NP	18
2.3	Algorithme étudier	20
2.3.1	Algorithme GETCOVERSET	20
2.3.2	Complexité de ONE-TO-ONE SKEWGRAM.....	21
2.3.3	Algorithme ALGOH(D, G, xy) - heuristique	22
2.3.4	Exemple de l'algoH.....	24
2.3.5	Complexité de l'algorithme ALGOH.....	25
2.3.6	Algorithme exact ALGOBB	25
2.3.7	Explication de l'algoBB.....	26
2.4	Programmation Par Contraintes (PPC)	27
2.4.1	Présentation de la PPC	27
2.4.2	Domaines d'application de la PPC.....	28
2.4.3	Comment résoudre un CSP? et Quelques notions	28
2.5	Principes de la PPC et problématique de la modélisation.....	29
2.5.1	Graphe de contraintes	29
2.5.2	Principes de la PPC	30
2.5.3	Algorithme de filtrage	30
2.5.4	Mécanisme de propagation	31
2.5.5	Mécanisme de recherche de solutions.....	31
2.6	Modélisation et Utilisation de graphes en programmation par contraintes .	31
2.6.1	Contraintes globales	32
2.7	Modélisation d'un graphe par des variables	32
2.7.1	Modèle booléen.....	32
2.7.2	Modèle ensembliste	33
2.7.3	Modèle basé sur des variables entières (successeurs)	34
2.8	Les contraintes sur les graphes.....	35
2.8.1	Les contraintes TREE et PROPER-TREE	35
2.8.2	Les contraintes ANTIARBO	36
2.8.3	Les contraintes ARBO.....	36
3	Etude Solveurs	37
3.1	Solveur CHOCO.....	37
3.2	ILOG Solver ou CP Optimizer.....	39

4	Analyse et conception	40
1	Modélisation mathématique	40
1.1	Première étape (Recherche du plus long chemin)	40
1.2	Deuxième étape (connexité)	41
5	Mise en œuvre	43
1	Librairie CHOCO.....	43
2	Framework JUNIT	43
3	Intégration continue (GitLab-CI).....	44
4	Versioning du code (GitLab)	44
5	Revue du code : plugin CheckStyle	45
6	Documentation	45
7	Choix techniques	46
7.1	Choix solveurs.....	46
7.2	Choix langages de programmation	46
8	Identification et gestion des risques.....	46
8.0.1	risques humains.....	47
8.0.2	risques sur les délais.....	47
8.0.3	risques techniques	47
9	Fichier de sortie	47
10	Comparaison des résultats	48
6	Bilan et Conclusion	51
1	Bilan du semestre 9	51
2	Conclusion	52
3	Planning et tâche du semestre 10.....	52
4	Gestion des tâches	53
5	Bilan sur la qualité	54
6	Bilan auto-critique.....	54
	Annexes	56
A	Plannification	57
1	Diagramme de Gantt prévisionnel.....	57
2	Découpage en tâches au semestre 9	57
3	Diagramme de Gantt du semestre 10	61
4	Tableau de synthèse du semestre 9	61
5	Tableau de synthèse du semestre 10	62

B	Spécifications fonctionnelles	63
1	Diagramme de classe	63
2	Classe "Parseur"	64
2.1	Fonction ExistFichierGraphe	64
2.2	Fonction ParserNonOriented	64
2.3	Fonction ParserOriented	65
3	Classe "Graphe"	65
4	Classe "GrapheNonOriente"	66
5	Classe "GrapheOriente"	66
6	Classe DonneesModeleChoco	66
6.1	Fonction ConnexionArcXij	66
6.2	Fonction connexionArreteYij	66
6.3	Fonction connexionArreteBijk	67
6.4	Fonction CheminExistEviterSommet	67
6.5	Fonction LienEntreSommet	67
7	Classe ModeleChoco	67
7.1	Fonction solve	68
8	Solution	68
8.1	Fonction stockerFichier	68
8.2	getSommetSuivant	68
8.3	afficherChemin	68
C	Spécifications non fonctionnelles	69
1	Contraintes de développement et conception	69
2	Contraintes de fonctionnement et d'exploitation	69
2.1	Performances	69
D	Document joint à ce rapport	70
1	Cahier de test	70
2	Manuel d'installation Choco	70
3	Cahier de recette	70
4	Cahier de développeur	70
5	Manuel d'utilisation	70
E	Webographie	71
F	Bibliographie	72
	Comptes rendus hebdomadaires	73

Table des figures

1 INTRODUCTION PROJET

1	figure 1 : Définition de SKEWGRAM.....	5
2	figure 2 : Exemple des chemins (D, G)-consistants	5
3	figure 3 : Définition de ONE-TO-ONE SKEWGRAM	6
4	figure 4 : Exemple de problème ONE-TO-ONE SKEWGRAM	7
5	figure 5 : Le plus long chemin dans D $7 \mapsto 2 \mapsto 4 \mapsto 8 \mapsto 1$ mais pas connexe dans G	7
6	figure 6 : Le plus long chemin dans D : $5 \mapsto 3 \mapsto 6 \mapsto 1$ connexe dans G	7
7	figure 7 : Modèle en cascade	9

2 Description générale

1	figure 8 : Forme du fichier du graphe orienté D.....	11
2	figure 8.1 : Forme du fichier du graphe non orienté G.....	12
3	figure 9 : Forme du fichier sortie	12
4	figure 10 : Diagramme d'utilisation.....	13
5	figure 11 : Diagramme de séquence.....	13
6	figure 12 : Diagramme d'activité.....	14
7	figure 13 : Interface Homme/Machine	15
8	figure 13 : Interaction avec la console	15

3 Etat de l'art/veille

1	figure 14 : Représentation des classes P, NP, NP-difficile et NP-complet)	19
2	figure 15 : Temps nécessaire à l'exécution d'un algorithme en fonction de sa complexité)	19
3	figure 16 : Qualifications usuelles des complexités)	20

4	figure 17 : L'algorithme GETCOVERSET(D, G, i, j).....	20
5	figure 18 : Calcul de CoverSet($D, G, S_i^- \cup S_j^+ \cup \dots$).....	21
6	figure 19 : L'heuristique ALGOH(D, G, xy).....	22
7	figure 20 : Diagramme illustrant l'heuristique ALGOH(D, G, xy).....	23
8	figure 21 : Application de l'algoH sur cet exemple.....	24
9	figure 22 : Algorithme AlgoBB.....	26
10	figure 23 : Application AlgoBB.....	27
11	figure 24 : Exemple de graphe de contrainte.....	29
12	figure 27 : Exemple de graphe G^* à rechercher, tel que $\underline{G} \subseteq G^* \subseteq \overline{G}$	33
13	figure 31 : Quelques exemples de graphes ne pouvant pas être correctement modélisés par la représentation successeurs.....	35
14	figure 32 : Illustration de la contrainte TREE(G, N).....	36
15	figure 36 : Hiérarchie de classes de CHOCO.....	38
 5 Mise en œuvre		
1	figure 37 : Lancement de tous les tests unitaires.....	44
2	figure 38 : Intégration continue GitLab-CI.....	44
3	figure 39 : Versioning du code GitKraken.....	45
4	figure 40 : Revue du code avec checkStyle.....	45
5	figure 41 : Exemple documentation Doxygen.....	46
6	figure 42 : Fichier résultat.....	47
7	figure 43 : Exemple résultat stocker dans le fichier texte.....	48
8	figure 44 : Exemple affichage console.....	48
9	figure 43 : Comparaison résultats AlgoBB et la modélisation.....	49
 6 Bilan et Conclusion		
1	figure 45 : les tâches du semestre 9 et du S10.....	52
2	figure 46 : les tâches du semestre 10.....	53
 A Plannification		
1	figure 47 : Diagramme de Gantt prévisionnelle.....	57
2	figure 39 : Découpage en tâches.....	58
3	figure 40 : Diagramme de Gantt du semestre 10.....	61
4	figure 40 : Diagramme de Gantt du semestre 10.....	61
5	figure 40 : Tableau de synthèse du S9.....	62
6	figure 40 : Tableau de synthèse du S10.....	62

B Spécifications fonctionnelles

1	figure 41 : Diagramme de classe	63
2	figure 41 : Format du fichier graphe non orienté G	65
3	figure 42 : Format du fichier graphe orienté D.....	65



INTRODUCTION

Ce document est le rapport de projet concernant le projet de fin d'études (Recherche et Développement) : *Étude algorithmique d'un problème de comparaison de réseaux biologique*. L'objectif de ce document est de décrire le contexte dans lequel le projet intervient, les divers choix choisis durant ce projet et la façon dont le projet s'est déroulé. La supervision de ce projet est réalisée par Monsieur Emmanuel NERON et Monsieur Ronan BOCQUILLON.

Ce rapport contient le cahier de spécification, l'état de l'art/veille, l'analyse et conception, et sa planification. Il répond à plusieurs questions : Comment ce projet a-t-il été abordé, quels étaient les objectifs initiaux et quels sont les résultats ? Pour pouvoir y répondre, dans un premier temps nous introduisons le projet, son contexte, la description du problème et son déroulement. Dans un second temps nous allons étudier les différents algorithmes qui répondent à cette problématique. Enfin nous étudierons et programmerons ses algorithmes à l'aide de la programmation par contraintes.



REMERCIEMENTS

Je tiens à adresser mes vifs remerciements à mon encadrant : Ronan BOCQUILLON, pour sa disponibilité, pour ses explications et son suivi tout au long de l'année.

Je souhaiterais vivement remercier Monsieur Jean-Yves RAMEL et Monsieur Nicolas RAGOT respectivement enseignants chercheurs informatiques, pour leurs aides et leurs conseils.

J'adresse également mes profonds remerciements à l'ensemble de l'équipe enseignante de Polytech Tours.

1

INTRODUCTION PROJET

Dans cette section nous allons aborder différents points : le contexte de la réalisation du projet, les acteurs du projet, les objectifs, les hypothèses, la description du problème et les bases méthodologiques sur la gestion du projet.

1 Contexte de la réalisation

1.1 Contexte

La reconstruction des réseaux biologiques est de nos jours, une thématique qui prend de l'ampleur pour intégrer et donner du sens aux données obtenues grâce aux nouvelles technologies de séquençage.

Un réseau biologique est une représentation de la circulation d'un certain type d'information dans la cellule. Il en existe plusieurs types. Nous allons nous intéresser aux réseaux d'interaction protéine-protéine. "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"[2]. La bio-informatique a permis de comparer et résoudre de nombreux problèmes, c'est une des méthodes les plus utilisées à présent.

Les méthodes d'analyse bio-informatiques jouent, un rôle très important dans ce processus. Pour modéliser les réseaux biologiques en bio-informatique on utilise les graphes les sommets de ce graphe représentent les composants biologiques et les arêtes entre ses sommets représentent l'interaction de ses composants. Il existe plusieurs types de réseaux biologiques. Ses réseaux se divisent en plusieurs types. Nous nous allons nous intéresser à deux types de réseaux. Des réseaux dits "métaboliques" (un réseau métabolique est l'ensemble des interactions moléculaires qui se produisent dans un organisme). Ses réseaux sont représentés par des graphes orientés. Les réseaux d'interactions sont représentés par des graphes non orientés. [1]

Dans ce projet, nous allons nous focaliser sur la comparaison de deux réseaux de types différents (par exemple un réseau métabolique et un réseau d'interaction des protéines) ces réseaux sont modélisés respectivement par des graphes de types différents des graphes orientés et des graphes non orientés, c'est ce que nous appelons, des réseaux hétérogènes. Donc nous aurons deux réseaux modélisés respectivement par un graphe orienté D (appelé graphe principal) et un graphe non orienté G (appelé graphe guide), ces réseaux ont une fonction f qui établissant la

correspondance entre les sommets. Les seuls points en commun entre ses deux graphes sont les sommets par contre les interactions entre les sommets peuvent être différents dans les deux graphes.

1.2 Les acteurs du projet

La maitrise d'ouvrage (MOA) :

- Monsieur, Emmanuel NÉRON, directeur de Polytech Tours.
- Monsieur, Ronan BOCQUILLON, enseignant chercheur.

Ils font partie de l'équipe de Recherche Opérationnelle, Ordonnancement, Transport ROOT ERL-CNRS 6305 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 oeuvre.

www.li.univ-tours.fr

La maitrise d'oeuvre (MOE) :

SANBA Morad étudiant à l'École Polytechnique de l'université de Tours en 5ème année cycle d'ingénieur en informatique, système d'information.

1.3 Description du problème

Le problème consiste à trouver le plus long chemin dans un graphe orienté D qui induit un sous-graphe connexe dans le graphe non orienté G . Ce problème s'appelle **SkewGram**. Un exemple d'application de ce problème est la recherche de chaînes des réactions successives dans notre cas représenté par des chemins, dans un métabolique d'un même organisme, qui sont catalysés par des protéines. Ce problème appartient à la classe **NP** des problèmes. Cette classe contient les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial.

Ce problème a été soulevé par Monsieur Hamed MOHAMED BABOU dans sa thèse en 2012 [2]. Dans cette thèse Monsieur BABOU a proposé deux algorithmes **AlgoH** et **AlgoBB** que nous allons étudier par la suite.

Il existe deux types de réseaux :

- Les réseaux homogènes représentés par les mêmes types de graphe (comparaison inter-espèces).
- Les réseaux hétérogènes représentés par des graphes différents (un graphe orienté vs un graphe non orienté) c'est ce qu'on appelle "comparaison intra-espèces".

Dans ce problème nous allons comparer deux réseaux hétérogènes : un réseau principal et un réseau guide. Nous allons représenter par un graphe orienté D , le réseau principal et le réseau guide 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 [2] :

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

"La fonction f permet de représenter une similarité entre les sommets selon un critère donné (par exemple similarité entre les séquences de protéines)" [2].

Par exemple : f désigne l'ensemble des protéines qui catalysent la réaction.

Notre problème de comparaison est un problème de **maximisation** appelé Skew SubGraph Mining (abrégé SkewGraM). Le problème SkewGraM a été défini par Monsieur Hamed Mohamed Babou dans sa thèse comme suit [2] :

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 1 – Définition de SKEWGRAM

Le problème SkewGraM est un problème de recherche des motifs multidimensionnels. Le motif que nous aborderons dans ce projet est la recherche du plus long chemin dans D qui induit un sous-graphe connexe dans G .

Monsieur Hamed Mohamed Babou a défini dans sa thèse un chemin consistant comme suit [2] :

Définition : Chemin (D, G, f) -consistant Soient D un graphe orienté, G un graphe non orienté et une fonction de correspondance $f : V(D) \mapsto 2^{V(G)}$.

Un chemin P dans D est (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.

Exemple :

Prenant les exemples suivants [2] :

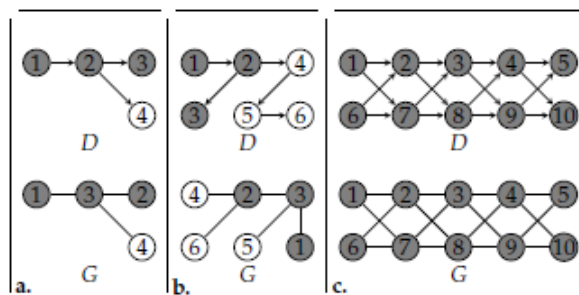


Figure 2 – Exemple des chemins (D, G, f) -consistants

Exemples des chemins (D, G) -consistants.

Dans l'exemple (a) le plus long chemin (D, G) -consistant est le chemin $1 \mapsto 2 \mapsto 3$ ou bien $1 \mapsto 2 \mapsto 4$.

Dans l'exemple (b) le plus long chemin (D, G) -consistant est aussi le chemin $1 \mapsto 2 \mapsto 3$. En revanche, le plus long chemin dans D est le chemin $1 \mapsto 2 \mapsto 4 \mapsto 5 \mapsto 6$. Ce chemin n'est pas $(D,$

G)-consistant, parce que le sous-graphe induit dans G par l'ensemble des sommets 1, 2, 4, 5, 6 n'est pas connexe.

Dans l'exemple (c) on trouve 32 chemins (D,G)-consistants de longueur 4, cela est dû au fait que dans le graphe G tous les sommets sont reliés.

Le problème ONE-TO-ONE SKEWGRAM :

Pour simplifier le problème SkewGraM, nous allons considérer la fonction f une bijection de l'ensemble $V(D)$ vers l'ensemble $v : v \in V(G)$. C'est la version que nous allons étudier il est appelée **One-to-One SkewGraM**. Dans ce problème nous allons supposer que $V(D) = V(G) = 1, 2, \dots, n$ et que f est la fonction identité c'est-à-dire que $f(v) = v$, pour tout $v \in V(D)$.

"Le problème de One-to-One SkewGraM est un problème de *maximisation* qui se présente ainsi" [2] :

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 3 – Définition de ONE-TO-ONE SKEWGRAM

Complexité du problème One-to-One Skew-GraM :

Comme ce que Monsieur Hamed Mohamed Babou à expliquer dans sa thèse [2],

Le problème One-to-One Skew-GraM est un problème NP-complet. Un problème P est dit NP-complet s'il vérifie les deux conditions suivantes :

1. $P \in NP$;
2. $P' \leq_P P$, pour tout $P' \in NP$.

Un problème P est dit NP-difficile si et seulement s'il vérifie la condition 2.

La condition 2, peut être remplacée par la vérification s'il existe au moins un problème NP-difficile qui puisse être transformé, en temps polynomial, vers P .

"Le problème **One-to-One Skew-GraM** peut se résoudre en temps polynomial quand au moins l'une des conditions suivantes est satisfaite" [2] :

- a) D^* est un arbre.
- b) G est un chemin ou cycle élémentaire.
- c) G est une bi-étoile.
- d) G est une étoile.

Exemple du problème ONE-TO-ONE SKEWGRAM :

On considère deux graphes, un graphe orienté $D = (V, A)$ et un graphe non orienté $G = (V, E)$. La question est de trouver le plus long chemin (D, G) -consistant ?

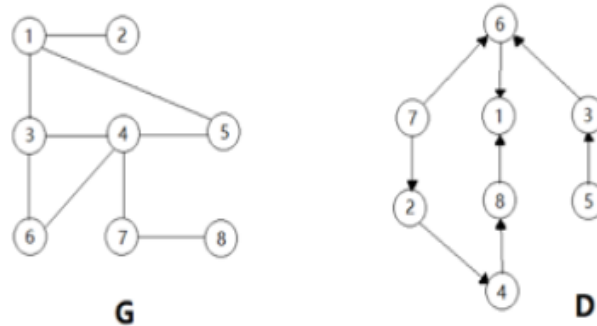


Figure 4 – Exemple de problème ONE-TO-ONE SKEWGRAM

Si on considère cet exemple le plus long chemin dans D est $7 \mapsto 2 \mapsto 4 \mapsto 8 \mapsto 1$ mais on regardant dans le graphe G ce chemin n'est pas connexe dans G .

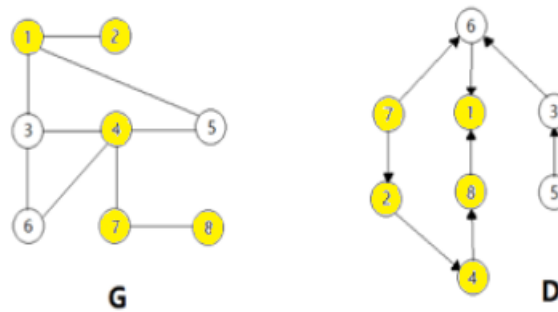


Figure 5 – Le plus long chemin $7 \mapsto 2 \mapsto 4 \mapsto 8 \mapsto 1$ mais pas connexe

Mais par contre le chemin $5 \mapsto 3 \mapsto 6 \mapsto 1$ est le plus long chemin (D, G) -consistant.

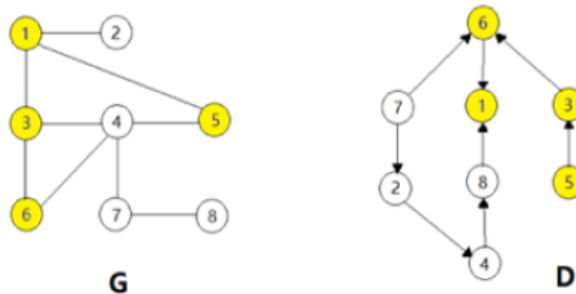


Figure 6 – Le plus long chemin dans D est $5 \mapsto 3 \mapsto 6 \mapsto 1$ connexe dans G

1.4 Objectifs

Le but est d'étudier le problème de comparaison de réseaux biologiques. Dans un premier temps l'objectif est d'étudier les algorithmes : **AlgoH** et **AlgoBB** créée par **Hafedh Mohamed BABOU**. Ses algorithmes permettent de trouver le plus long chemin consistant. Nous allons nous intéresser plus particulièrement à l'algorithme **AlgoBB** car ce dernier permet d'explorer tous les cas possibles contrairement à l'algoH qui est une heuristique.

Dans un deuxième temps je vais étudier la programmation par contraintes (PPC) et plus particulièrement la thèse de Monsieur **Jean-Guillaume FAGES**. Cette dernière va nous permettre de proposer une modélisation à notre problème et le résoudre à l'aide d'un solveur.

Donc au terme de ce projet nous allons proposer une modélisation à notre problème et l'implémenter à l'aide d'un solveur. Ensuite nous allons comparer leurs temps d'exécution et les résultats renvoyer.

1.5 Hypothèses

Mon projet est orienté plus recherche que développement, ce qui va prendre une partie importante de celui-ci. Également je serais amenée à étudier la programmation par contraintes qui va me permettre de modéliser mon problème.

Concernant le solveur à utiliser ce dernier dépend de notre modélisation, car certains solveurs ne supportent pas certaines modélisations. Par exemple on ne peut pas utiliser un modèle ensembliste sur le solveur CHOCO. Donc nous allons faire par la suite une étude de ses différents solveurs qu'on pourra utiliser.

Le langage de programmation que nous avons choisi est JAVA, car les deux principaux solveurs que nous serons amenés à utiliser, à savoir CHOCO et ILOG SOLVER (CP Optimizer) sont codés en JAVA, c'est leur seul langage en commun.

Concernant le type de fichier que le solveur va prendre en entrée, nous avons prévu d'utiliser deux fichiers textes qui contiennent les deux graphes D et G.

1.6 Bases méthodologiques

Le projet sera géré avec le logiciel "GanttProject" et un dépôt sur GitHub sera utilisé pour versionner le code, concernant la gestion des différentes tâches (tickets) j'ai utilisé l'outil en ligne "Trello". Pour la modélisation nous allons utiliser les diagrammes UML.

Le déroulement du projet sera fait similairement à la méthode en cascade **Figure 7**. Cette méthode nécessite de concevoir avant de produire quelque chose. Le principe est simple on ne passe à la phase suivante que lorsque la précédente est validée. Cette méthode permet de sécuriser le planning du projet puisque l'on verrouille chacune des étapes les unes après les autres. Il est bien adapté à mon projet puisque je serais amenée à proposer et valider des modélisations auprès de mon tuteur avant de commencer l'implémentation.

Le projet sera découpé en itération comprenant des phases de recherche, d'analyses, de développements et de tests.

Toutes les documentations seront faites en parallèle du développement et n'appartiendront à aucune itération.

Le solveur à choisir dépendra de la modélisation proposée. Pourquoi cela car cela dépend par exemple de la liste des contraintes définies et certains solveurs n'utilisent pas cette contrainte.

Par exemple On ne peut pas utiliser un modèle ensembliste dans le solveur **CHOCO**. Donc nous allons étudier une liste de solveur par la suite, parmi cette liste il y a deux solveurs que mon encadrant ma proposé **CHOCO Solver** et **ILOG Solver** aussi appelé **CP Optimizer**.

Le langage de développement sera "Java" car les deux solveurs cités précédemment sont implémenter en Java.

Les commentaires respecteront la norme doxygen. Les rapports seront effectués avec **Latex**.

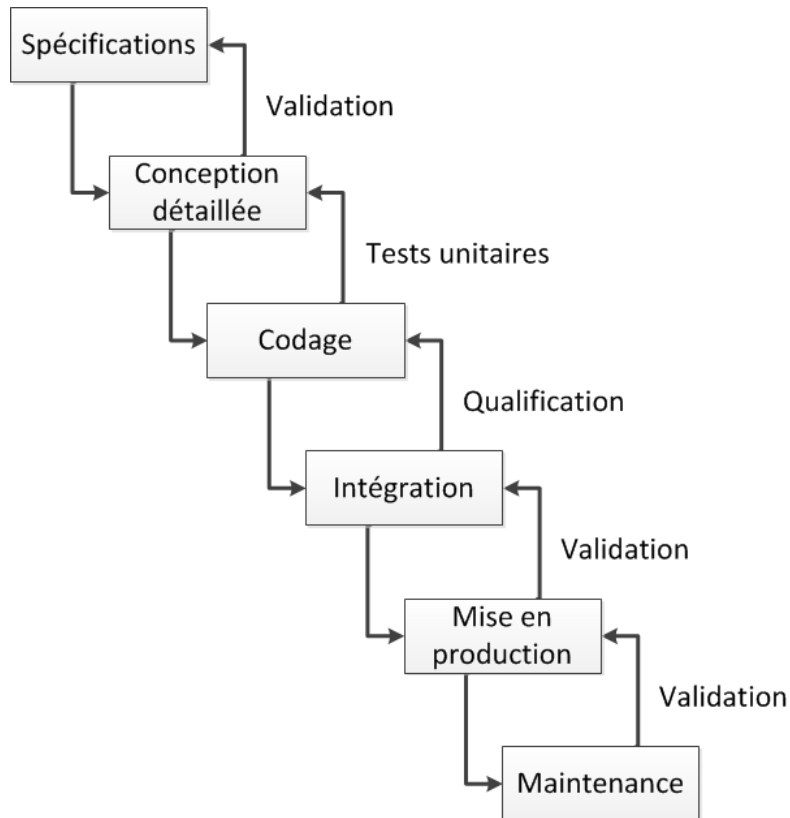


Figure 7 – Modèle en cascade

2

Description générale

Dans ce chapitre nous allons aborder : l'environnement de réalisation du projet, les fonctionnalités et la structure générale du système, les contraintes de développement. Nous allons décomposer notre système en fonctions et expliquer les entrées et sorties de chacune.

1 Environnement du projet

Ce projet prend comme base les deux algorithmes **AlgoH** et **AlgoBB**, nous dans ce projet nous nous intéresserons plus particulièrement à l'algoBB. Le but n'est pas d'implémenter l'algoBB, il va nous permettre de comprendre notre problème.

Notre problème sera modélisé à l'aide de la programmation par contraintes. Il sera programmé avec "Java" Pour l'IDE que nous allons utiliser est "Intellij IDEA 2017". Deux étapes sont importantes :

- La modélisation de notre problème. Ce qui consiste à déterminer un ensemble de variables, leurs domaines de définitions et les contraintes. Nous allons voir cela par la suite.
- Résolution, à l'aide d'un solveur.

Environnement de développement :

- Le système d'exploitation est Windows.
- Le langage de programmation : JAVA.
- L'IDE qui sera utilisé est Intellij.

2 Caractéristiques des utilisateurs

Les utilisateurs finaux de ce logiciel sont les enseignants-chercheurs et l'équipe de Recherches Opérationnelles, Ordonnancement, Transport ROOT. Ces derniers ont de très bonnes connaissances en informatique et plus précisément dans le domaine de la programmation par contraintes.

3 Fonctionnalités et structure générale du système

Dans ce projet il aura une fonctionnalité principale qui est de trouver le plus long chemin (D,G)-consistant. Pour répondre à cette problématique nous allons proposer une modélisation mathématique. Chaque modélisation va représenter une fonction. Pour le moment il est prévu de faire une seule modélisation. Si nous sommes amenés à faire d'autres modélisations nous aurons d'autres fonctions.

Les données nécessaires, sont deux graphes un orienté D, un graphe non orienté G et un chemin L. Ces données seront sous la forme d'un fichier que nous allons parser, donc il aura une fonctionnalité parseur. Ces données me seront fournies par mes encadrants par la suite. Il aura deux fonctionnalités parseurs car les données à parser sont toujours deux graphes un orienté et un autre non orienté, il n'aura alors que le nombre de sommet, le sens des arcs, la relation entre les sommets et le chemin L qui vont varier. Pour l'instant deux fichiers sont prévus pour stocker les instances, il possible par la suite de passer au solveur trois fichiers (grapheD, grapheG, ArcL). Également en sortie pour chaque instance un fichier est créé et contiendra les résultats de cette instance.

La fonction parseur va prendre en entrée les deux fichiers qui contiennent les deux graphes. Ce fichier va respecter un format précis voir **Figure 1** et renvoi en sortie le nombre de noeud les relations entre les arcs des deux graphes et le chemin L. Nous allons également prévus que le parseur vérifie le nombre de sommet entre les graphes est le même, et si le chemin L correspond bien à un chemin avant de parser.

Le fichier d'entrée est prévu comme suit :



1	N:10
2	P:0.2
3	3-10
4	3-7
5	5-3
6	5-6
7	7-1
8	9-7
9	10-6
10	10-2
11	2-4
12	4-8

Figure 1 – Forme du fichier du graphe orienté D

Dans ce fichier de type "txt" on remarque :

- "N" qui correspond au nombre de sommet du graphe G.
- "P :0.2", cette donnée nous ne concerne pas dans notre projet, donc nous allons ignorer cette ligne pendant le passage du fichier.
- Pour finir on retrouve à gauche la liste des sommets source séparé par un split de la liste des sommets arrivés.



```

graphG.txt
1 N:10
2 1-2
3 1-3
4 1-5
5 2-7
6 2-6
7 2-9
8 3-7
9 4-10
10 4-6
11 5-8
12 6-7
13 7-9
14 8-9

```

Figure 2 – Forme du fichier du graphe non orienté G

Également dans ce fichier on retrouve :

- L'entier "N" qui correspond au nombre de sommet du graphe G.
- Ensuite comme le fichier précédent à gauche on retrouve la liste des sommets source séparer par split de la liste des sommets arrivés.

La sortie sera le plus long chemin (D, G)-consistant qui sera affiché sous la forme de fichier texte et dans la console. On affiche également le temps d'exécution, les informations relatives à cette instance par exemple : le nombre de sommet, le chemin de l'instance etc.

Le fichier de sortie du solveur est prévu comme suit :

Modele	Nb_Nœud	Chemin	Temps_execution_heure
1	5	1,2,6,8,7	1,6
2	5	1,2,6,8,7	0.5

Figure 3 – Forme du fichier sortie

Le fichier de sortie sera de type "txt" il contient :

- Une colonne nommée "Modele" et contient les numéros des modèles implémenter.
- Une colonne "Nb_Nœud" nombre de noeud que contient la solution.
- Une colonne "Chemin" qui contient la solution renvoyer c'est-à-dire le chemin plus long chemin consistant.
- Une colonne "Temps_execution_heure" qui contient le temps passer pour trouver la solution.

Pour chacune des tâches prévues nous aurons une durée qu'il faut respecter. Nous allons voir cela plus en détails dans la partie planification.

Le logiciel affiche le plus long chemin (D,G)-consistant. Il fonctionne selon le schéma suivant :

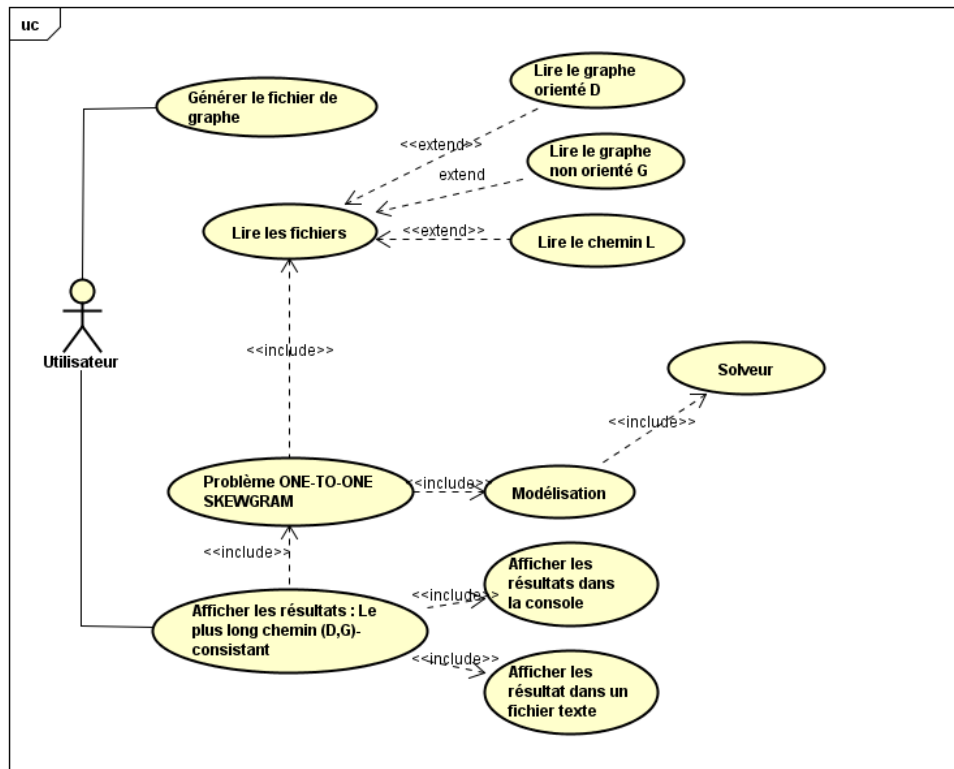


Figure 4 – Diagramme d'utilisation

L'interaction entre le système et l'utilisateur se présente ainsi :

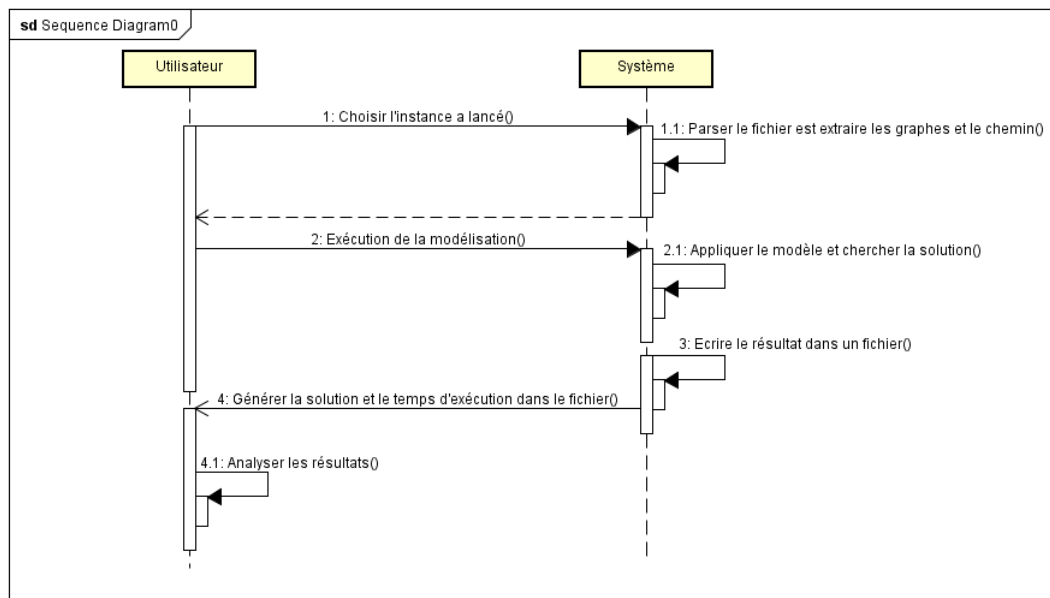


Figure 5 – Diagramme de séquence

Au début nous avons prévu de développer une petite interface, cela ne sera pas effectué vue le manque de temps. L'interaction se fera via la console. La console affichera les instructions nécessaires pour choisir une instance. L'utilisateur importe les données stockées dans le fichier. Ensuite ses données sont parsées et les données sont extraites. Après l'utilisateur choisira le modèle à appliquer. Et enfin les résultats sont générés et sont stockés dans un fichier texte.

Le diagramme d'activité se présente ainsi :

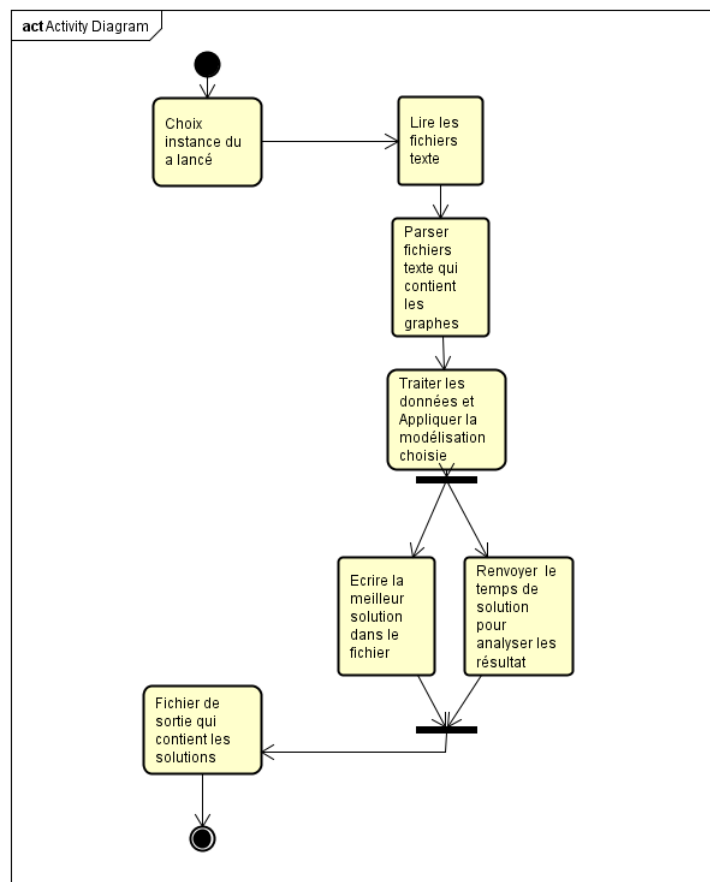


Figure 6 – Diagramme d'activité

Les descriptions détaillées des fonctionnalités sont dans l'annexe "Spécifications fonctionnelles".

4 Contraintes de développement, d'exploitation et de maintenance

Contraintes de développement

- Matériels : La solution devra fonctionner en multi-plateforme.
- Langages de programmation : Java.
- Solveur dépendra de la modélisation.
- Etudier le cours sur la programmation par contraintes.
- Base existante : La thèse de Hamed Mohamed Babou.
- Rendu des spécifications : 12 Décembre 2018.
- Rendu du projet : 04 Avril 2019.

5 Description des interfaces externes du logiciel

Nous avons prévu de faire une interface qui va interagir avec l'utilisateur. Cette dernière demandera à l'utilisateur de fournir les instances à tester sous forme de fichier qui devra respecter un format spécifique. Il aura également le choix du modèle. Un bouton pour lancer le calcul.

Voici à quoi va ressembler l'interface Homme/Machine.

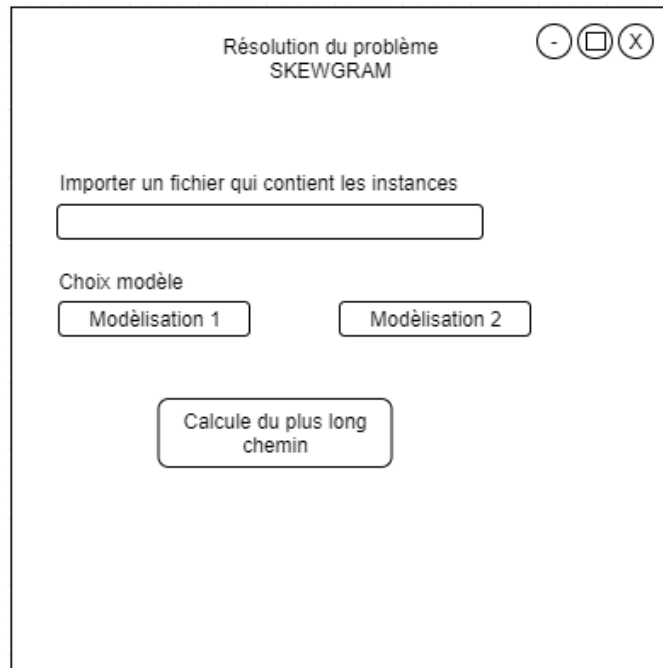


Figure 7 – Interface Homme/Machine

Également il est prévu de contrôler l'application de la console, cela se présente ainsi :

```
"C:\Program Files\Java\jdk1.8.0_74\bin\java" ...
----- Modélisation CHOCO -----
----- Veuillez choisir une instance -----
Pour choisir l'instance 10_4 choisir 1
Pour choisir l'instance 110_4 choisir 2
```

Figure 8 – Interaction avec la console

5.1 Les entrées et les sorties

Le solveur doit prendre en entrées les données récupérées dans les deux graphes D et G. Les graphes sont stockés dans deux fichiers texte, concernant ses graphes mes encadrants vont me mettre à disposition les instances à tester, il aura un parseur à implémenter qui va permettre de lire ses graphes.

Les sorties seront également stockées sous forme de fichier et seront affichées à la console, ce qui correspond au plus long chemin consistant et le temps que le solveur a mis pour calculer la solution.

3

Etat de l'art/veille

Pour répondre à cette problématique, j'ai étudié la thèse de Monsieur Hamed Mohamed Babou [2], cette thèse aborde l'alogBB, qui permet de répondre à cette problématique. L'objectif c'est de résoudre le problème avec une nouvelle approche qui est la programmation par contraintes. Ensuite j'ai étudié le cours sur la programmation par contraintes et la thèse de Monsieur Jean-Guillaume FAGES [3]. Cette thèse aborde la modélisation des graphes à l'aide de la programmation par contraintes. Dans les différentes parties qui suivent nous allons aborder les notions de base des théories des graphes, la programmation par contraintes et pour finir les différentes méthodes de modélisation.

1 Modélisation de réseaux biologiques

1.1 Réseaux homogènes VS réseaux hétérogènes

Les réseaux d'interaction protéine-protéine représentent les interactions physiques entre protéines. Les interactions entre paires de protéines sont modélisées par un graphe non orienté dans lequel les sommets sont les protéines et les arêtes représentent leurs interactions.

En distingue deux types de réseaux :

- **Des réseaux homogènes :**
 - > Même type, même type de graphe, espèces différentes (comparaison inter-espèces).
- **Des réseaux hétérogènes :**
 - > Type différent, type de graphe peut être différent (un graphe orienté vs un graphe non-orienté), même espèce (comparaison intra-espèces).

2 Théorie des Graphes

2.1 Définitions et notions de base

Dans la partie qui suit nous allons aborder des notions de bases dans la théorie des graphes certaines définitions proviennent de Wikipédia [6]

Un **graphe** est un ensemble de points nommés nœuds reliés par des arêtes. Il est généralement noté $G = (V, A)$ avec V , un ensemble de nœuds et un ensemble d'arcs $A \subseteq V^2$.

Un **chemin** est une chaîne dont tous les arcs sont orientés dans le même sens.

Un **cycle** est une suite d'arêtes consécutives (chaîne simple) dont les deux sommets extrémités sont identiques.

Une **chaîne**, dans un graphe non orienté, une chaîne reliant x à y , est définie par une suite finie d'arêtes consécutives, reliant x à y .

Une **chaîne simple** est une chaîne ne passant pas deux fois par une même arête, c'est-à-dire dont toutes les arêtes sont distinctes.

Un **graphe simple** est un graphe sans boucle et avec au plus une arête entre deux sommets quelconque.

Un **graphe non-orienté** $G = (V, E)$ est un couple formé de deux ensembles : un ensemble V (noté $V(G)$) dont les éléments sont appelés sommets et un ensemble E de paires **non-orientées** (noté $E(G)$), avec $E \subseteq V^2$, dont les éléments sont appelés arêtes.

Un **graphe orienté** $G = (V, A)$ est un couple formé de deux ensembles : un ensemble V (noté $V(G)$) dont les éléments sont appelés sommets et un ensemble E de paires **orientées** (noté $A(G)$), dont les éléments sont appelés arcs.

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

Un **graphe non-orienté** $G = (V, E)$ est dit **fortement connexe** si pour tout $u, v \in V^2$, 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.

Une **composante connexe** d'un graphe est un sous-graphe **connexe** de ce graphe.

Un **graphe planaire** est un graphe qui a la particularité de pouvoir se présenter sur un plan sans qu'aucune arête (ou arc pour un graphe orienté) n'en croise une autre. On dit qu'un graphe est planaire extérieur si on peut mettre ses sommets sur un cercle de sorte que toutes les arêtes soient à l'intérieur du cercle et ne se rencontrent qu'aux extrémités.

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

Un **graphe biparti** il existe une partition de son ensemble de sommets en deux sous-ensembles et telle que chaque sommet de est relié à chaque sommet de l'autre.

Une **forêt** est un graphe non-orienté acyclique.

Un **arbre** est un graphe non orienté, acycliques et connexes.

Définition :

[2] " Soient deux graphes non-orientés $G_1 = (V, E_1)$ et $G_2 = (V, E_2)$, une **composante connexe commune** entre G_1 et G_2 est un ensemble maximal $X \subseteq V$ tel que $G_1[X]$ et $G_2[X]$ sont connexe ".

• **Notations :** [2] Les notations suivantes sont importantes nous allons les revoir dans les algorithmes par la suite.

- On note par $i \overset{D}{\rightsquigarrow} j$ un chemin dans D partant du sommet i vers un sommet j . Quand ij est un arc de D , le chemin $i \overset{D}{\rightsquigarrow} j$ est noté *irightarrow* j .
- On note par $CCC(D^*, G, i \overset{D}{\rightsquigarrow} j)$ la composante connexe commune entre D^* et G qui contient tous les sommets du chemin $i \overset{D}{\rightsquigarrow} j$, si une telle composante existe. Si une telle composante n'existe pas, on note $CCC(D^*, G, i \overset{D}{\rightsquigarrow} j) = \emptyset$.
- On note par S_i^+ l'ensemble de sommets j dans D tel qu'il existe un chemin $i \overset{D}{\rightsquigarrow} j$.

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

Définition : [2]

" Soit $r \in V$. Le sommet r est dit pont de $i \stackrel{D}{\rightsquigarrow} j$ 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 $i \stackrel{D}{\rightsquigarrow} j$, c'est-à-dire $CCC(D^*[V - \{r\}], G[V - \{r\}], i \stackrel{D}{\rightsquigarrow} j) = \emptyset$."

Définition :

[2] " **L'ensemble couvrant d'un chemin** $i \stackrel{D}{\rightsquigarrow} j$, noté $\text{CoverSet}(D, G, i \stackrel{D}{\rightsquigarrow} j)$, est l'ensemble X vérifiant les conditions suivantes " :

1. $V(i \stackrel{D}{\rightsquigarrow} j) \subseteq X \subseteq S_i^- \cup S_j^+ \cup V(i \stackrel{D}{\rightsquigarrow} j)$;
2. $D^*[X]$ et $G[X]$ sont connexes;
3. Si r est un pont de $i \stackrel{D[X]}{\rightsquigarrow} j$ par rapport à $G[X]$ alors $X \subseteq S_r^- \cup S_r^+ \cup \{r\}$;
4. X est maximal (par rapport à l'inclusion) avec les conditions 1., 2. et 3.

Si, pour un chemin $i \stackrel{D}{\rightsquigarrow} j$, il n'y a aucun ensemble X satisfaisant les conditions 1., 2. et 3., alors par convention $\text{CoverSet}(D, G, i \stackrel{D}{\rightsquigarrow} j) = \emptyset$.

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

2.2 Notions de complexité

[W1] Déterminer la complexité d'un algorithme, c'est évaluer les ressources nécessaires à son exécution (essentiellement la quantité de mémoire requise) et le temps de calcul à prévoir. Donc on distingue deux complexités temporelle et spatiale :

- Complexité temporelle (ou en temps) : temps de calcul.
- Complexité spatiale (ou en espace) : l'espace mémoire requis par le calcul

[W1] Ces deux notions dépendent de nombreux paramètres matériels qui sortent du domaine de l'algorithmique : nous ne pouvons attribuer une valeur absolue ni à la quantité de mémoire requise ni au temps d'exécution d'un algorithme donné. En revanche, il est souvent possible d'évaluer l'ordre de grandeur de ces deux quantités de manière à identifier l'algorithme le plus efficace au sein d'un ensemble d'algorithmes résolvant le même problème.

On distingue également deux autres complexités :

- La complexité pratique est une mesure précise des complexités temporelles et spatiales pour un modèle de machine donné.
- La complexité théorique est un ordre de grandeur de ces coûts, exprimé de manière la plus indépendante possible des conditions pratique d'exécution.

2.2.1 Classe P et NP

En théorie de la complexité on distingue plusieurs classes.

Définition Classe P : [W9]

La complexité d'un algorithme est dite polynomiale si elle est $O(n^k)$, pour un certain entier k . Soit P un problème de décision. Le problème P est dans la classe P (Polynomial), si et seulement si il existe un algorithme déterministe A (c'est-à-dire une suite d'étapes élémentaires sans choix aléatoire) qui résout P en un temps polynomial en la taille des instances du problème P .

La classe P est formée des problèmes de décision qui peuvent être résolus par un algorithme polynomial.

Définition classe NP :

La classe NP contient les problèmes de décision dite non polynomiale, qui peuvent être décidés sur une machine non déterministe en temps polynomial. La classe NP est une extension de la classe P.

Les problèmes les plus difficiles dans NP sont appelés NP-complets.

Définition NP-complet :

Un problème P est dit NP-complet s'il vérifie les deux conditions suivantes :

1. $P \in \text{NP}$.
2. $P' \leq_P P$, pour tout $P' \in \text{NP}$.

Un problème P est dit **NP-difficile** si et seulement si il vérifie la condition 2.

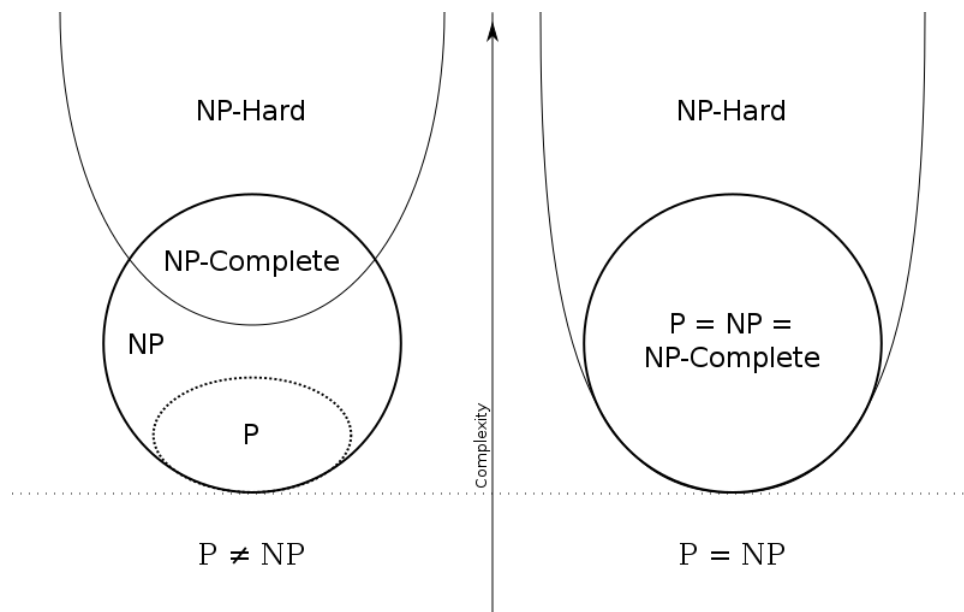


Figure 1 – Représentation des classes P, NP, NP-difficile et NP-complet

Ordre de grandeur et temps d'exécution : [W9]

La détermination de la complexité algorithmique ne permet pas d'en déduire le temps d'exécution mais seulement de comparer entre eux deux algorithmes résolvant le même problème.

La figure ci-dessous donne le temps nécessaire à l'exécution d'un algorithme en fonction de sa complexité.

	$\log n$	n	$n \log n$	n^2	n^3	2^n
10^2	7 ns	100 ns	0,7 μs	10 μs	1 ms	$4 \cdot 10^{13}$ années
10^3	10 ns	1 μs	10 μs	1 ms	1 s	10^{292} années
10^4	13 ns	10 μs	133 μs	100 ms	17 s	
10^5	17 ns	100 μs	2 ms	10 s	11,6 jours	
10^6	20 ns	1 ms	20 ms	17 mn	32 années	

Figure 2 – Temps nécessaire à l'exécution d'un algorithme en fonction de sa complexité

Qualifications usuelles des complexités :

$O(\log n)$	logarithmique
$O(n)$	linéaire
$O(n \log n)$	quasi-linéaire
$O(n^2)$	quadratique
$O(n^k) \quad (k \geq 2)$	polynomiale
$O(k^n) \quad (k > 1)$	exponentielle

Figure 3 – Qualifications usuelles des complexités

2.3 Algorithme étudié

Dans un premier temps j'ai étudié la thèse de Monsieur Hamed Mohamed Babou, dans cette thèse j'ai découvert un ensemble d'algorithmes qui traite notre problème. Il m'a permis d'approfondir et de bien comprendre ce problème. Dans ce qui suit nous allons présenter l'ensemble de ses algorithmes.

2.3.1 Algorithme GETCOVERSET

Le premier algorithme est l'algorithme "GETCOVERSET", il permet de calculer l'ensemble couvrant d'un chemin donné en un temps de $O(n^2 \log(n) + n \cdot m \cdot \log^2(n))$, avec $n = |V|$ et $m = |A(G)| + |E(G)|$.

Cet algorithme prend en entrées deux graphes : un graphe orienté $D = (V, A(D))$ et un graphe $G = (V, E(G))$ non orienté et un chemin $i \rightsquigarrow^D j$. Avec $A(D)$, l'ensemble des arcs est $E(G)$, l'ensemble des arêtes.

DAG : (Direct Acyclic graph) est un graphe orienté sans cycle.

L'algorithme se présente ainsi : [2]

Algorithme 1 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:  $S := \text{CCC}(D^*[S], G[S], i \rightsquigarrow^{D[S]} j)$ ;  $STOP := \text{faux}$ ;
3: tantque  $((STOP = \text{faux}) \text{ et } (S \neq \emptyset))$  faire
4:    $S_{tmp} := S$ ; /* 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:      $S_{tmp} := S_{tmp} \cap (\{r\} \cup S_r^- \cup S_r^+)$ ;
7:   fin pour
8:   si  $(S = S_{tmp})$  alors
9:      $STOP := \text{vrai}$ ;
10:  sinon
11:     $S := S_{tmp}$ ;  $S := \text{CCC}(D^*[S], G[S], i \rightsquigarrow^{D[S]} j)$ ;
12:  finsi
13: fin tantque
14: return  $S$ 

```

Figure 4 – L'algorithme GETCOVERSET($D, G, i \rightsquigarrow^D j$)

Pour bien comprendre le fonctionnement du GETCOVERSET nous allons prendre l'exemple qui suit :

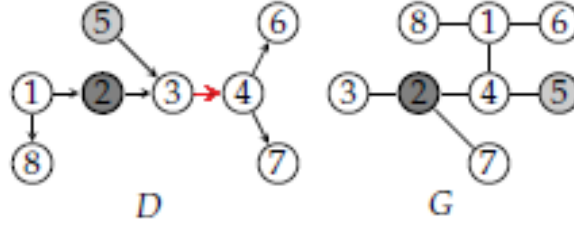


Figure 5 – Calcul de CoverSet($D, G, 3 \overset{D}{\rightsquigarrow} 4$)

On initialise dans un premier temps l'ensemble S à $S = S_3^- \cup S_4^+ \cup V(3 \overset{D}{\rightsquigarrow} 4) = \{1, 2, 5\} \cup \{6, 7\} \cup \{3, 4\} = \{1, 2, 3, 4, 5, 6, 7\}$

S_3^- correspond à l'ensemble de sommet j dans D tel qu'il existe un chemin $4 \overset{D}{\rightsquigarrow} 3$.

S_4^+ correspond à l'ensemble de sommet j dans D tel qu'il existe un chemin $3 \overset{D}{\rightsquigarrow} 4$.

Ensuite on affecte le résultat de S à la composante connexe $CCC(D^*[S], G[S], 3 \overset{D[S]}{\rightsquigarrow} 4)$ et on initialise STOP = faux.

À la ligne 5, on regarde s'il existe un pont de $3 \overset{D}{\rightsquigarrow} 4$ par rapport à $G[S]$ dans cet exemple nous avons bien un pont qui est le sommet 2.

À la ligne 6, on retire de S tous les sommets qui ne peuvent pas appartenir à un chemin dans D passant par le sommet 2 (dans cet exemple, on retire le sommet 5). Donc $S = \{1, 2, 3, 4, 6, 7\}$.

La prochaine exécution de la boucle "tant que" ne va pas modifier l'ensemble S , donc $S = \{1, 2, 3, 4, 6, 7\}$.

Donc à la fin de l'algorithme GETCOVERSET, nous obtenons $S = \{1, 2, 3, 4, 6, 7\}$ l'ensemble **couvrant** de $3 \overset{D}{\rightsquigarrow} 4$.

Pour calculer $CCC(D^*, G, i \overset{D}{\rightsquigarrow} j)$ il faudra utilisé l'algorithme **GENPARTREFINEMENT**, [2]. Cet algorithme calcule les composantes connexes communes entre $D^* = (V, E(D^*))$ et $G = (V, E(G))$ en un temps de $O(n \log n + m \log 2n)$, avec $n = |V|$ et $m = |E(D^*)| + |E(G)|$.

2.3.2 Complexité de ONE-TO-ONE SKEWGRAM

La complexité du problème **ONE-TO-ONE SKEWGRAM** dépend de la contrainte topologique sur les deux graphes D et G que le problème prend en entrées.

Le problème One-to-One SkewGraM est **NP-complet** même si D^* est planaire extérieur et G est un arbre de diamètre 4.

[2] Le problème One-to-One SkewGraM se résout en un temps polynomial quant au moins l'une des conditions suivantes est satisfaite :

- a) D^* est un arbre.
- b) G est un chemin ou cycle élémentaire.
- c) G est une bi-étoile.
- d) G est une étoile.

2.3.3 Algorithmme ALGOH(D, G, xy) - heuristique

L'algorithme que nous allons voir est l'heuristique ALGOH. Il permet de calculer un chemin (D, G) -consistant passant par un arc xy . Cet algo construit progressivement un chemin consistant de manière progressive, par contre il n'exploite pas toutes les solutions possibles.

Le principe de l'algorithme AlgoH :

[2] "L'algoH initialise une variable cc (chemin courant) à un chemin de longueur 1 contenant l'arc xy ($cc := i \xrightarrow{D} j$). Le chemin cc est noté $s \rightsquigarrow^D t$ où s est le premier sommet du chemin (dit source) et t est son dernier sommet (dit terminal)."

L'algorithme se présente ainsi : [2]

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 := 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 */
24:       S := msc ; STOP := vrai ;
25:     sinon
26:       cc :=  $p_{max}$  ; /* choix de l'extension la plus prometteuse */
27:       s :=  $s_{max}$  ; t :=  $t_{max}$ 
28:     finsi
29:   finsi
30: fin tantque
31: return S

```

Figure 6 – L'heuristique ALGOH(D, G, xy)

Voici un diagramme illustrant l'heuristique ALGOH : [2]

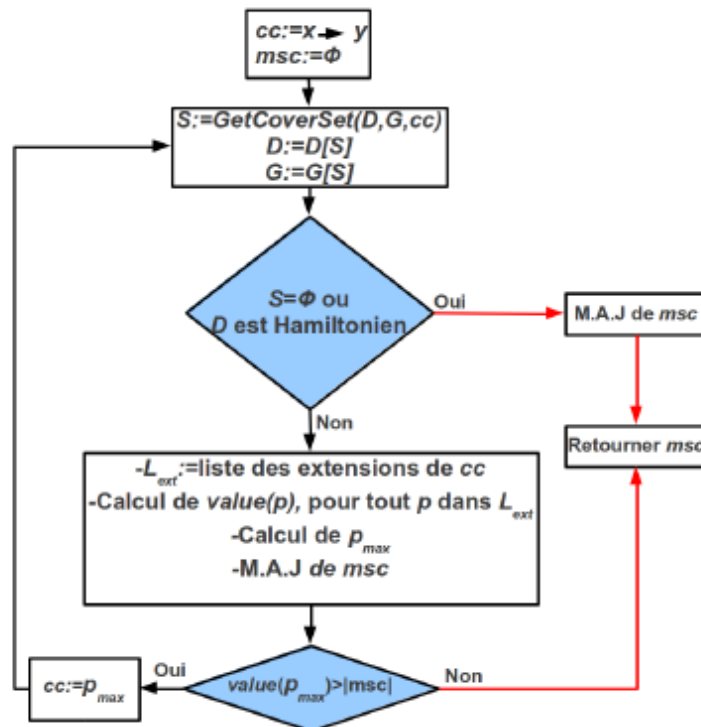


Figure 7 – Diagramme illustrant l'heuristique ALGOH(D, G, xy)

Pour expliquer le fonctionnement de cet algorithme je me suis basé sur la thèse de Monsieur Hafedh Mohamed Babou. [2]

On initialise le chemin courant $cc = x \rightarrow y$ et la meilleure solution msc est égale à l'ensemble vide. On applique l'algorithme du GETCOVERSET, pour trouver l'ensemble couvrant du chemin cc . Après cela on distingue les cas suivants :

1. Le premier cas c'est quand l'ensemble $V(D) = \emptyset$ dans ce cas aucun chemin consistant n'est trouvé.
2. Le cas où le graphe D est Hamiltonien, dans ce cas on trouve la meilleure solution courante (msc) et donc le chemin (D, G) -consistant correspondant est $D[msc]$.
3. Si l'ensemble $V(D) \neq \emptyset$ et que le graphe D n'est pas Hamiltonien. Dans ce cas on calcule la liste des chemins dans $V(D)$ cela correspond aux L_{ext} (lignes 12 à 16) dans l'algo, nous obtenons une liste chemin obtenus en prolongeant à l'aide la fonction "EXTEND" le chemin courant (cc) par un seul sommet à chaque fois que ça soit un prédécesseur de s ou un successeur de t .

La ligne 17 permet de calculer l'ensemble DEC des sous-graphes induits dans D par les chemins de la liste L_{ext} .

La ligne 18 on calcule l'ensemble des graphes HEC sont inclus dans DEC ce sont des chemins consistant passant par xy et ils sont utilisés pour mettre à jour la meilleure solution courante msc (ligne 19 et 20).

Il faudra choisir une extension dans L_{ext} , pour cela on utilise la fonction $value$ défini comme suit : pour $p \in L_{ext}$, $value(p)$ est la longueur du plus long chemin dans le sous-graphe $D[CovertSet(D, G, p)]$.

On choisit l'extension de telle que $value(p_{max}) = \max\{value(p) : p \in L_{ext}\}$, Ici on distingue deux cas :

3.1 Si $|msc| \geq value(p_{max})$. Dans ce cas il n'existe aucun chemin (D, G) -consistant passant par xy .

3.2 Sinon dans ce cas $cc = s_{max}$ et $s = p_{max}$, $t = t_{max}$

2.3.4 Exemple de l'algoH

Prenant l'exemple suivante :

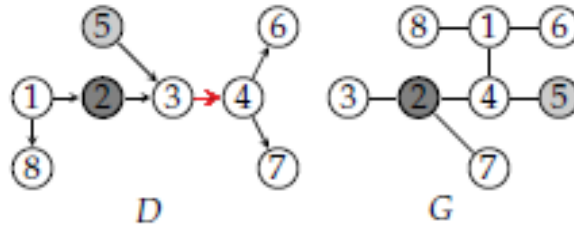


Figure 8 – Application de l'algoH sur cet exemple

Nous allons déroulé l'algoH sur les deux graphes ci-dessus.

A la ligne 5 les variables suivantes sont initialisées comme suit :

$msc = \emptyset$; $cc := 3 \xrightarrow{D} 4$; $s = 3$; $t = 4$; STOP = faux.

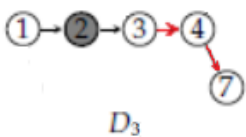
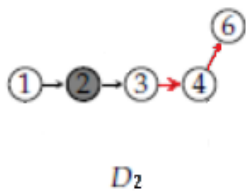
Après avoir appliqué le GETCOVERSET nous obtenons : $S = \{1; 2; 3; 4; 6; 7\}$

Comme $S \neq \emptyset$ et que le graphe D n'est pas Hamiltonien donc on passe à la ligne 12, on initialise la liste des chemin dans D qui sont obtenus en prolongeant le chemin cc , dans D par un seul sommet à la fois comme suit :

$L_{ext} = \{ 2 \rightarrow 3 \rightarrow 4; 3 \rightarrow 4 \rightarrow 6; 3 \rightarrow 4 \rightarrow 7 \}$

A la ligne 17 et 18 on initialise DEG et HEC :

DEC = {



}

Et, $HEC = \{ D_1 \text{ et } D_2 \}$ il contient tous les graphes Hamiltonien.

A la ligne 19, $h_{max} = 5$ donc on peut choisir soit D_1 soit D_2 donc $msc = (1, 2, 3, 4, 6)$ et $|msc| = 5$.

A la ligne 21 on calcule le "value" de chaque élément de la L_{ext} et on prend la valeur maximale ici $value(p_{max}) = 6$ et donc $p_{max} = 2 \rightarrow 3 \rightarrow 4$:

- $value(2 \rightarrow 3 \rightarrow 4) = 6$
- $value(3 \rightarrow 4 \rightarrow 6) = 5$
- $value(3 \rightarrow 4 \rightarrow 7) = 5$

A la ligne 25, ici dans notre cas : $|msc| \leq value(p_{max})$

Donc le chemin courant $cc := 2 \rightarrow 3 \rightarrow 4$ et $s = 2$; $t = 4$ et on retourne $S := msc = \{1,2,3,4,6\}$

2.3.5 Complexité de l'algorithme ALGOH

[2] La complexité de l'algorithme algoH dépend de plusieurs paramètres telle que le degré maximum du graphe D , la longueur du plus long chemin etc...

Si on note Δ le degré maximum du graphe D c'est-à-dire $\Delta = \max_u (\Delta_u^+ + \Delta_u^-)$ avec $u \in V$ et Δ_u^+ le degré sortant de u et Δ_u^- le degré entrant de u .

On note L la longueur du plus long chemin (D, G) -consistant passant par xy . La complexité de l'algoH est de $O(\Delta L(n^2 \log n + n m \log^2 n))$, avec $n = |V|$ et $m = |A(D)| + |E(G)|$.

2.3.6 Algorithme exact ALGOBB

L'algorithme **ALGOBB** permet de résoudre le problème One-to-One SkewGraM, contrairement à l'heuristique AlgoH, il permet d'explorer tous les cas pour trouver le plus long chemin (D, G) -consistant. Il est basé sur la méthode de séparation et évaluation (branch-and-bound).

La méthode de séparation et évaluation "branch-and-bound" est une manière de résoudre un problème NP-difficile. Le principe de cette méthode de construire un arbre qui pour traiter tout les solutions réalisables et d'éliminer les solutions qui ne mènent pas à la solution. De cette manière nous exploitant toutes les solutions possibles.

L'algorithme ALGOBB prend en entrées un DAG $D = (V, A)$, un graphe non-orienté $G = (V, E)$ et un arc xy de D . En sortie on obtient le plus long chemin (D, G) -consistant passant par xy .

On construit un arbre TS des sous-solutions, on associe à la racine l'arc xy à inclure dans la solution. Pour chaque sommet s est associé à un chemin, noté $p(s)$, prolongeant l'arc xy dans sommet à la fois (un successeur du sommet y ou bien un prédécesseur du sommet x). À la fin de la construction de l'arbre TS , ses feuilles sont associées aux chemins (D, G) -consistants passant par xy . Donc le plus long chemin (D, G) -consistant passant par xy est le plus long chemin $i \xrightarrow{D} j$ associé à une feuille. Nous allons voir cela à travers un exemple par la suite.

L'algorithme AlgoBB se présente ainsi : [2]

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 9 – Algorithme AlgoBB

Dans ce qui suit nous allons expliquer le fonctionnement de l'algoBB, pour cela je me suis appuyé sur l'explication présenter par Monsieur Hamed Mohamed Babou dans sa thèse [2].

Séparation [2]

"Dans un premier temps nous allons affecter à la racine de l'arbre à construire (TS) l'arc ou bien un chemin d'arc imposé et qui doit faire partie de la solution. Pour tout sommet s , avec $p(s) = v_1 \xrightarrow{D} v_m$ qui est prédécesseur ou respectivement successeur dans D , on ajoute dans l'arbre TS un fils de s associé au chemin $v_1.p(s)$ (resp. $p(s).v_k$)".

[2] "Pour tout $s \in V(TS)$, on rappelle que $value(p(s))$ est la longueur du plus long chemin dans le graphe $D[CoverSet(D, G, p(s))]$. Nous utilisons la fonction notée $BBvalue$ pour évaluer les sommets de TS . La fonction **BBvalue** est définie comme suit" [2] :

- "Si s n'est pas encore exploré, alors **BBvalue(s) = value(p(s))**".
- "Si s a été déjà exploré à un moment donné de la construction de TS , alors on distingue deux cas".
 - "Le chemin $p(s)$ est (D, G) -consistant. Dans ce cas $BBvalue(s)$ est égale à la longueur du chemin $p(s)$ " [2].
 - "Le chemin $p(s)$ n'est pas (D, G) -consistant. Dans ce cas **BBvalue(s) = 0**" [2].

En utilisant la fonction $BBvalue$, on définit les opérations évaluation (bounding) et élagage (pruning) de la méthode branch-and-bound comme suit :

Évaluation (Règle 1) [2]

" 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) [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 $BBvalue(s) \leq BBvalue(s_{max})$. Cette suppression est appliquée récursivement sur les sommets (sauf s_{max}) qui deviennent feuilles après la suppression de leurs fils ".

2.3.7 Explication de l'algoBB

Pour bien comprendre l'algoBB nous allons reprendre l'exemple de la Figure 8.
 On commence par définir la racine de l'arbre TS qui est l'arc de départ dans notre cas $3 \rightarrow 4$.

A la ligne 2, tant qu'on peut explorer un sommet de cet arbre on continue à explorer jusqu'à ne plus avoir de sommet à explorer. Pour chaque sommet $s \in TS$, on commence par ajouter des sommet à notre arbre en rajoutant à chaque fois un prédécesseur du sommet 3 de notre graphe D ou bien un successeur du sommet 4. Dans notre cas nous obtenons quatre sommets dans notre arbre : $(5 \rightarrow 3 \rightarrow 4; 2 \rightarrow 3 \rightarrow 4; 3 \rightarrow 4 \rightarrow 6; 3 \rightarrow 4 \rightarrow 7)$, ensuite on commence par calculer pour chaque sommet $s_i \in V(TS)$ la longueur du chemin $BBvalue(s_i)$, Dans notre $BBvalue(s_1) = BBvalue(s_2) = BBvalue(s_3) = BBvalue(s_4) = 3$, dans ce cas nous choisissons un sommet arbitrairement. Ensuite on sépare ce sommet et on applique la règle 2.

Après application nous obtenons l'arbre suivant :

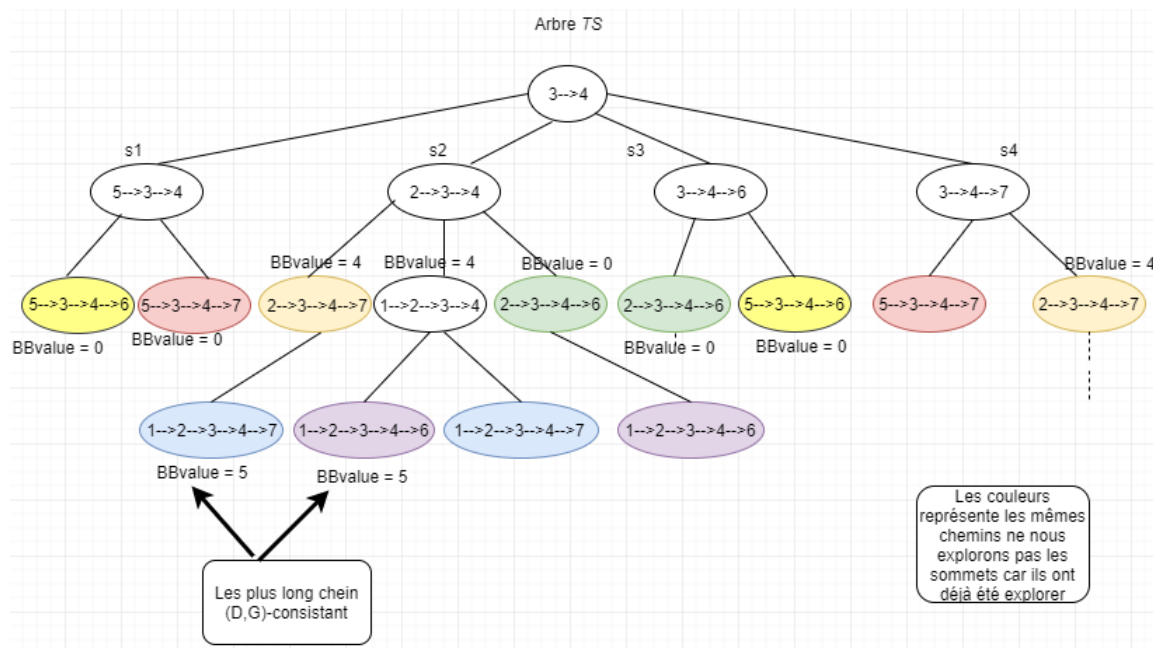


Figure 10 – Application AlgoBB

2.4 Programmation Par Contraintes (PPC)

La deuxième étape dans la littérature est d'étudier le cours sur la Programmation par contraintes. Ce qui va me permettre de modéliser le problème. Pour cela j'ai lu quelques cours sur internet, dans les liens sont dans la "Webographie" et également mon tuteur a mis à ma disposition un cours [7]. Dans ce qui va suivre nous allons présenter la PPC et pour finir comment modéliser un graphe avec la programmation par contraintes. Pour cela je me suis appuyé sur la thèse de Monsieur Jean-Guillaume FAGES [3].

2.4.1 Présentation de la PPC

La Programmation Par Contraintes est une technique de programmation qui permet de modéliser et résoudre efficacement des problèmes de satisfaction de contraintes (**CSP : Constraint Satisfaction Problem**).

CSP (Problème de Satisfaction de Contraintes) est un problème modélisé sous la forme d'un ensemble de contraintes posés sur des variables, chacune de ces variables prenant ses valeurs dans un domaine.

La programmation par Contraintes permet à la fois de **modéliser le problème** et de le **résoudre**. La programmation par Contraintes, combine entre deux choses : la puissance de calcul

des ordinateurs pour explorer un grand nombre de cas et l'intelligence humaine qui permet d'éliminer un grand nombre de cas dès le début. La PPC présente de nombreux avantages [3] :

- Très flexible.
- Modélisation facile.
- Réutilisable.

2.4.2 Domaines d'application de la PPC

La programmation par contraintes permet de résoudre n'importe quel type de problème combinatoire. Elle a été utilisée pour une très grande variété d'applications réelles. Voici quelques domaines d'applications de la PPC :

- Problèmes d'ordonnancement.
- Problèmes de planification, planifier le trafic aérien.
- Problèmes d'optimisation par exemple le problème de 3D-bin packing, optimiser le rangement d'objet de toute les formes de sorte à optimiser le maximum de place. etc...

2.4.3 Comment résoudre un CSP? et Quelques notions

Dans ce qui suit je me suis appuyé sur le cours de la programmation par contraintes mis à ma disposition par mon encadrant [7].

Comment résoudre un CSP?

[7] Pour résoudre un **Problème de Satisfaction de Contraintes** ou **CSP** est défini par un triplet $(X, D; C)$ tel que :

- $X = \{X_1, X_2, \dots, X_n\}$ ensemble des variables (les inconnues) du problème.
- D est le domaine de définition des variables définies précédemment. Il est noté $D(X_i)$ ou bien $\text{dom}(X_i)$.
- $C = \{C_1, C_2, \dots, C_k\}$ correspond à l'ensemble des contraintes, chaque contrainte C_j est une relation entre deux ou plusieurs variables X , permet de restreindre les valeurs que peut prendre les variables.

Donc on conclue que pour résoudre un **Problème de Satisfaction de Contraintes** il faut trouver une valeur pour chacune des variables qui respecte les contraintes prédéfinies.

Une variable peut prendre ses valeurs dans un **domaine fini ou non**. Les domaines finis permettent de modéliser les problèmes combinatoires discrets. Il existe plusieurs types de domaines finis :

- Les **entiers consécutifs** : $D(X) = [1 \dots 20]$
- Les **ensembles énumérés de constantes** :
 - $D(X_1) = \{1, 2, 8\}$
 - $D(X_2) = \{\text{vert, bleu, jaune}\}$

Il existe plusieurs types de **contraintes** :

- Une contrainte **restreinte** lorsqu'on restreint les valeurs que peuvent prendre des variables.
- Une contrainte **relationnelle** si la valeur d'une variable dépend d'autres variables. Par exemple on peut pas déterminer la valeur d'une variable que si on détermine les valeurs des autres variables.
- Une contrainte **déclarative** permet de définir les relations à trouver entre les variables.

Une contrainte peut être **définie** :

- " En **intension**, c'est-à-dire à l'aide de propriétés mathématiques, par exemple :
 $X \leq Y$ ou $A \text{ et } B \Rightarrow \text{non}(C)$ " etc... [7]
- " En **extension**, en énumérant les tuples de valeurs appartenant à la relation, par exemple :
 $X \in [0..3], y \in [0..3], X < Y$ devient en extension $(X=0; Y=1)$ ou $(X=0; Y=2)$ ou $(X=0; Y=3)$ ou $(X=1; Y=2)$ ou $(X=1; Y=3)$ ou $(X=2; Y=3)$ ou encore $(X,Y) \in \{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$ ". [7]

Une contrainte est dite **globale** si elle fait intervenir plusieurs variables.

Une contrainte est **booléenne**, si elle contient que des variables booléennes (true ou false).

Affectation c'est l'instanciation des variables par des valeurs de leurs domaines. On note par $A = (X_1, V_1), (X_2, V_2), \dots, (X_r, V_r)$ l'affectation qui instancie la variable X_1 par la valeur V_1 , la variable X_2 par la valeur V_2 , ..., et la variable X_r par la valeur V_r . [7]

Une affectation est **totale** si les variables du problème sont toutes instanciées c'est-à-dire que pour chaque variable est instancié par une valeur de son domaine.

Une affectation est dite **partielle** quand une partie des variables du problème sont instanciées.

Une affectation (partielle ou totale) **viole** une contrainte, s'il ne respecte pas l'une des contraintes prédéfinies.

Une affectation (totale ou partielle) est **consistante**, si toutes les contraintes sont respectées.

Une affectation est **inconsistante** si elle ne respecte pas l'une ou plusieurs contraintes.

2.5 Principes de la PPC et problématique de la modélisation

Dans cette partie nous allons aborder l'utilisation de graphes en programmation par contraintes. Pour cela je vais m'appuyer sur la thèse de Monsieur Jean-Guillaume FAGES [3]. Je vais également reprendre certains exemples de cette thèse que je vais détailler et bien expliquer.

2.5.1 Graphe de contraintes

Le problème de Satisfaction de Contraintes $CSP(X, D, C)$ est associé à un graphe de contraintes $G = (X, C)$ avec X l'ensemble des variables et C l'ensemble des contraintes :

- Les sommets du graphe G représentent les variables.
- Les arêtes de G représentent les contraintes qui lient les sommets (les variables).

Exemple :

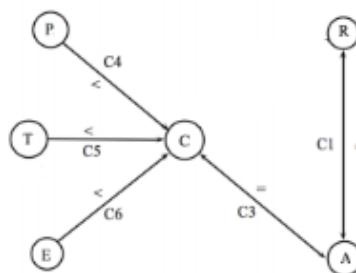


Figure 11 – Exemple de graphe de contrainte

2.5.2 Principes de la PPC

La programmation par contraintes s'articule autour de quatre axes :

- Le réseau de contraintes.
- Les algorithmes de filtrage.
- Mécanisme de propagation.
- Mécanisme de recherche de solutions, c'est-à-dire les méthodes pour parcourir l'espace de recherche.

2.5.3 Algorithme de filtrage

[W10]

Un algorithme de **filtrage**, ou **réduction de domaines**, permet de supprimer des valeurs des domaines des variables de la contrainte pour lesquelles il n'est pas possible de satisfaire la contrainte.

• Par exemple, pour la contrainte $(X < Y)$ avec leurs domaines, $D(X) = [10, 20]$, $D(Y) = [0, 15]$, un algorithme de filtrage associé à cette contrainte va permettre de supprimer les valeurs de 15 à 20 de $D(X)$ et les valeurs de 0 à 10 de $D(Y)$ car pour que la contrainte soit respectée.

- Consistances locales (filtrage) :

Le principe de la consistance locale repose sur la vérification qu'aucune variable ne viole les contraintes liées à cette variable, on ignorant certaines contraintes de ce fait il est impossible d'avoir un filtrage parfait, puisqu'on ignore certaines contraintes.

Quelques méthodes d'importance de l'algorithme de filtrage :

- Consistance d'arc (AC)

Une des méthodes les plus importantes d'un algorithme de filtrage est la **consistance d'arc**. On dit qu'une contrainte réalise la consistance d'arc s'il supprime toutes les valeurs des variables impliquées dans la contrainte qui ne sont pas consistantes avec la contrainte.

Par exemple, pour la contrainte $X+3 = Y$ avec les domaines $D(X) = \{1, 3, 4, 5\}$ et $D(Y) = \{4, 5, 8\}$, un algorithme de filtrage établissant la consistance d'arc modifiera les domaines pour obtenir $D(X) = \{1, 5\}$ et $D(Y) = \{4, 8\}$.

- Consistance d'hyperarc (HAC)

La consistance d'hyperarc, est la généralisation de la consistance d'arc pour les contraintes non binaires. Une contrainte est HAC si et seulement si chaque valeur de chaque variable appartient à une solution de la contrainte.

Par exemple, la contrainte **Alldiff** implique que les variables sur lesquelles elle est définie prennent des valeurs deux à deux différentes

- Consistance de Noeud (NC)

Un CSP (X, D, C) est **consistant de noeud** si pour toute variable X_i de X , et pour toute valeur v de $D(X_i)$, l'affectation partielle (X_i, v) satisfait toutes les contraintes unaires de C . Par exemple, Si nous avons une seule contrainte $X_1 > 2$ et $D(X_1) = \{1, 2, 3, 4, 5, 6\}$ pour que le CSP soit consistant de noeud il faut enlever du domaine les valeurs 1 et 2 qui violent la contrainte.

- Consistance de chemin (PC)

Un CSP vérifie la consistance de chemin, si et seulement si, pour tout couple de variables (X, Y) et pour toute troisième variable Z , si (x, y) est une instantiation consistante de X et de Y , alors il existe z dans D_z tel que (x, z) et (z, y) soient consistants.

- k-consistance

Elle consiste à considérer k variables, et de tester tous les k -uplets de valeurs possibles afin de tester s'ils ne violent pas les contraintes. Plus k est grand, plus le filtrage sera efficace.

Variables : $X = \{x_1, x_2, \dots, x_n\}$ avec $n = 5$

Contraintes : $\forall i, j \in [1, n] \ x_1 + x_3 = x_2$ avec $i \neq j$ et $x_i \neq x_j$

Domaines initiaux :

$$D(x_1) = \{1\}$$

$$D(x_2) = \{1, 3, 5\}$$

$$D(x_3) = \{2, 4\}$$

$$D(x_4) = \{2, 4\}$$

$$D(x_5) = \{2, 4, 5\}$$

Ensuite on applique l'algorithme de filtrage et on obtient :

Domaines après filtrage :

$$D(x_1) = \{1\}$$

$$D(x_2) = \{3, 5\}$$

$$D(x_3) = \{2, 4\}$$

$$D(x_4) = \{2, 4\}$$

$$D(x_5) = \{2, 4, 5\}$$

On prenant une valeur de chaque domaine, nous obtenons une solution.

2.5.4 Mécanisme de propagation

Quand l'algorithme de filtrage associé à une contrainte modifie le domaine d'une variable, il faut revoir les conséquences de cette modification pour les autres contraintes impliquent cette variable. On répète ce mécanisme de propagation jusqu'à ce que plus aucune modification n'apparaisse. Comme les domaines sont finis, ce mécanisme finit par prendre fin.

2.5.5 Mécanisme de recherche de solutions

Le mécanisme de recherche de solutions a pour but de trouver une solution optimale. Pour cela plusieurs moyens permettront au solveur de trouver la solution :

- Les choix des variables et des valeurs.
- Les méthodes de décomposition, Si on est en face d'un très gros problème, il est nécessaire de le décomposer et de résoudre les différentes parties de façon indépendante.
- Les améliorations itératives, cela consiste à trouver une solution optimale puis essayer de les améliorer à l'aide de techniques d'améliorations locales.

2.6 Modélisation et Utilisation de graphes en programmation par contraintes

Dans cette partie nous allons traiter la problématique de la modélisation. Ensuite nous allons revoir le concept de contraintes globales qui est un point important en PPC, car ces contraintes contiennent beaucoup d'informations.

Pour résoudre un problème à l'aide d'un solveur, nous devons définir les contraintes ainsi que les méthodes de résolution et les stratégies de choix des variables et de valeurs.

La première chose à faire lors de la modélisation d'un problème est l'identification de sous-problème qui vont correspondre aux contraintes. Pour étudier ses différentes modélisations je me suis appuyer sur la thèse de Monsieur Jean-Guillaume FAGES [3]

2.6.1 Contraintes globales

Les contraintes globales encapsulent un ensemble de contraintes. Cette notion est née avec l'utilisation de graphes au coeur des algorithmes de filtrage.

- Contrainte globale **all-different**

$\text{all-different}(X_1, \dots, X_n)$

- Remplace $\frac{n}{2}$ contraintes binaires.
- Très souvent utilisée en pratique.
- On peut la rendre arc-consistant.

Algorithme :

1. On construit un graphe biparti « variables-valeurs ».
2. On trouve un couplage maximum (si sa taille est inférieure à n , alors pas de solution).
3. On établit les arêtes qui n'appartiennent pas
 - à aucun chemin ou circuit alternant,
 - et au couplage trouvé.
4. On supprime les valeurs correspondantes à ces arêtes

- Contrainte globale **gcc** [3]

$\text{gcc}(x_1, \dots, x_n, l_1, \dots, l_k, u_1, \dots, u_k)$

• Cette contrainte de cardinalité globale est une généralisation de all-different : le nombre de fois chaque valeur v_j peut être prise doit être dans $[l_j, u_j]$ (pour all-different, $l_j = 0, u_j = 1, \forall j$).

2.7 Modélisation d'un graphe par des variables

Comme expliquer dans la thèse Jean-Guillaume FAGES, la plupart des problèmes consistent à trouver un graphe G^* tel que ce graphe soit compris entre deux autres graphes que nous allons noter \underline{G} et \bar{G} plus mathématiquement cela veut dire $\underline{G} \subseteq G^* \subseteq \bar{G}$. Nous allons nous baser sur cet exemple pour expliquer les différentes modélisations qui vont suivre. Cette exemple a été pris de la thèse de Monsieur Jean-Guillaume FAGES [3]

2.7.1 Modèle booléen

Peu importe la nature d'un graphe on peut le représenter sous forme de variables booléennes. Pour chaque sommet i du graphe on pose b_i la variable booléenne qui représente le sommet i . On définit également $b_{i,j}$ la variable booléenne qui représente l'arête $(i,j) \in E$. Pour chaque arête présente dans le graphe son domaine de définitions vaudra (true et false) pour laisser le soin au solveur de choisir la variable ou pas. Si l'arête n'existe pas on met false sinon on met true.

- **Exemple :** [3]

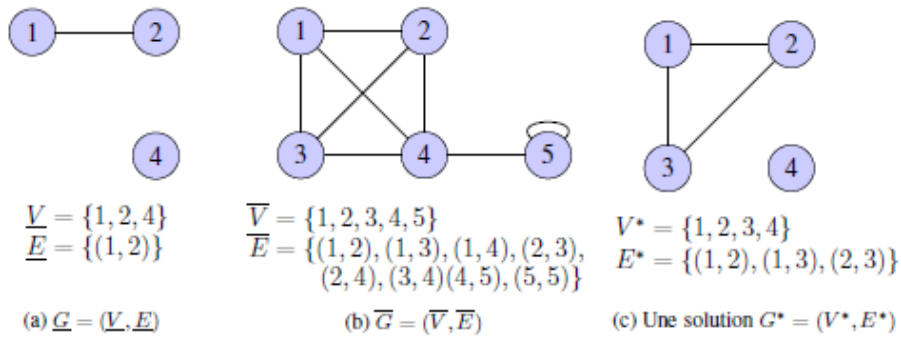


Figure 12 – Exemple de graphe G^* à rechercher, tel que $\underline{G} \subseteq G^* \subseteq \overline{G}$

Considérent l'exemple de la Figure 12 :

Le modèle suivant correspond à son modèle booléen :

Domaines de définitions des sommets :

$$D(b_1) = \{\text{true}\}$$

$$D(b_2) = \{\text{true}\}$$

$$D(b_3) = \{\text{true}, \text{false}\}$$

$$D(b_4) = \{\text{true}\}$$

$$D(x_5) = \{\text{true}, \text{false}\}$$

Domaines de définitions des arêtes :

$D(b_{1,2}) = \{\text{true}\}$, cette variable est égale à true car il existe dans les trois graphes \underline{G} , G^* et \overline{G}

$$\forall (i, j) \in \overline{E} - \{(1, 2)\} D(b_{i,j}) = \{\text{true}, \text{false}\}$$

Contraintes :

$$\forall (i, j) \in \overline{E} \ x_{i,j} \Rightarrow x_i \wedge x_j$$

Dans les domaines de définitions on retrouve les valeurs que chaque sommet peut prendre (true ou false) ils correspondent à la présence ou non de ses sommets dans les graphes : \underline{G} , G^* , \overline{G} . Il y a également une initialisation des arêtes du graphe \overline{G} , la seule arête qui prend la valeur "true" c'est l'arête (1,2) car il est présente dans trois graphes. Il y a une seule contrainte : variable $b_{i,j}$ implique les variables représentant les noeuds i et j (b_i et b_j).

2.7.2 Modèle ensembliste

Un graphe est un ensemble de noeuds et un ensemble d'arêtes, il peut être modélisé avec deux variables ensemblistes.

Une variable ensembliste S représente un ensemble de valeurs. Son domaine de définition est défini par un intervalle d'ensembles $D(S) = [\underline{S}, \overline{S}]$. " Une valeur S^* est un ensemble satisfaisant la relation $\underline{S} \subseteq S^* \subseteq \overline{S}$ " [3].

" On note par S une variable ensembliste qui désigne les noeuds appartenant à la solution et un ensemble de variables ensemblistes S^v telle que pour chaque noeud i , la variable ensembliste S_i^v indique l'ensemble des voisins de i dans le graphe solution". [3] La contrainte **INVERSESET** [3] assure la symétrie de la relation de voisinage et la cohérence de la solution.

Exemple de modèle ensembliste : [3]

Nous allons reprendre l'exemple précédent Figure 12 :

Domaines de définitions :

$$D(S^n) = [\underline{V}, \overline{V}]$$

$$D(S_1^v) = [\{2\}, \{2,3,4\}]$$

$$D(S_2^v) = [\{1\}, \{1,3,4\}]$$

$$D(S_3^v) = [\{\}, \{1,2,4\}]$$

$$D(S_4^v) = [\{\}, \{1,2,3,5\}]$$

$$D(S_5^v) = [\{\}, \{4,5\}]$$

Contraintes :

$$\text{INVERSESET}(S^v, S^v)$$

$$\forall (i, j) \in \overline{E}, j \in S_v^i \Rightarrow i \in S^n \wedge j \in S^n$$

Solution :

$$D(S^n) = S^*$$

$$D(S_1^v) = \{2,3\}$$

$$D(S_2^v) = \{1,3\}$$

$$D(S_3^v) = \{1,2\}$$

$$D(S_4^v) = \{\}$$

$$D(S_5^v) = \{\}$$

Ce modèle ensembliste offre deux avantages :

- Permet d'utiliser une grande collection de contraintes définies sur des variables ensemblistes déjà implémentées dans les solveurs.
- Permet une meilleure structuration des données. Etant donné un noeud, l'ensemble des voisins potentiels et l'ensemble des voisins obligatoires de i peuvent chacun être retrouvé en temps constant. Cela améliore la complexité temporelle des algorithmes de filtrage et des heuristiques de branchement par rapport à l'approche booléenne.

L'inconvénient majeur est que cette représentation possède une redondance d'information, une arête (i, j) présente dans une solution implique que $i \in S_j^v$ et $j \in S_i^v$. Si l'évènement $i \in S_j^v$ a déjà été propagé ; alors l'évènement $j \in S_i^v$ doit être ignoré.

2.7.3 Modèle basé sur des variables entières (successeurs)

On peut modéliser un graphe en associant à chaque noeud i avec une variable entière $succ_i$ qui représente son unique successeur. Si $succ_i = j$ cela signifie que l'arc $i \rightarrow j$ appartient au graphe. $succ_i = i$ signifie que le sommet i n'appartient pas à la solution.

Exemple de modèle basé sur des variables entières : [3]

Reprenant l'exemple précédent de la **Figure 7** :

Domaines :

$$D(succ_1) = \{-1,1,2,3,4,6\}$$

$$D(succ_2) = \{-1,1,2,3,4,6\}$$

$$D(succ_3) = \{-1,1,2,3,4,6\}$$

$$D(succ_4) = \{-1,1,2,3,4,5,6\}$$

$$D(succ_5) = \{1,4,5,6\}$$

Contraintes :

$$succ_1 = 2 \vee succ_2 = 1$$

Solution :

$$succ_1 = 3 \vee succ_2 = 1 \vee succ_3 = 2 \vee succ_4 = 6 \vee succ_5 = -1$$

Essayons de comprendre cette modélisation, les différents successeurs peuvent prendre les valeurs $\{-1, 1, 2, 3, 4, 5, 6\}$, pour tous sommet i appartenant à l'ensemble des sommets, $succ_i = -1$ si le sommet i n'appartient pas à la solution.

$succ_i = 6$ si le sommet i n'est pas relié avec aucun autre sommet.

$succ_i = j$ s'il y a une arête qui relie le sommet i et j

L'inconvénient de cette modélisation c'est qu'il n'est pas appropriée dans le cas général, elle convient que à certaines contraintes, telles que CIRCUIT et SUBCIRCUIT [3]. Cette modélisation ne convient pas à tous les cas par exemple le cas ou il aurait plus d'arêtes que de noeud, et aussi le cas ou il aurait trop de successeurs pour un noeud i , voir ce qui suit [3] :

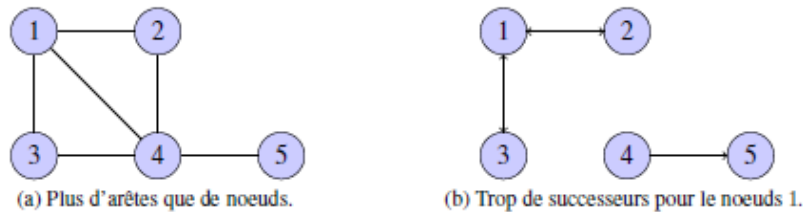


Figure 13 – Quelques exemples de graphes ne pouvant pas être correctement modélisés par la représentation successeurs.

2.8 Les contraintes sur les graphes

Dans cette partie nous allons traiter les contraintes orientées par exemple : **TREE**, **ANTI-ARBO**, **ARBO** et **PATH**, beaucoup de problèmes combinatoires impliquent de trouver un chemin dans un graphe orienté comme dans notre cas trouver le plus long chemin (D,G)-consistant. Pour cela je vais m'appuyer sur la thèse suivante [3].

2.8.1 Les contraintes TREE et PROPER-TREE

Définitions :

[w3] Les contraintes **tree** et **proper-tree** sont noté respectivement sous la forme **tree**(NTREE,VER) et **proper-tree**(NPROP,VER), avec NTREE et NPROP des variables entières et VER est la collection des n sommets $VER[1] \dots VER[n]$ du graphe à partitionner. A chaque sommet $v_i = VER[i]$ est associé un ensemble d'attributs définis comme suit :

- L est un entier compris entre 1 et n qui peut être interprété comme le nom du sommet v_i
- S est une variable entière définie sur l'intervalle $[1, n]$. Elle représente, dans une solution de la contrainte tree ou proper-tree, l'unique successeur du sommet v_i dans une couverture. Si i appartient au domaine de la variable $VER[i].S$, alors on dira que v_i est une racine potentielle

[w3] La contrainte **TREE**(G, N) et la contrainte **PROPER-TREE** partitionnent les sommets d'un graphe orienté en un ensemble d'arbres disjointes. Ces contraintes sont composées d'un

ensemble de composantes connexes, sans circuit tel que chaque sommet possède un unique successeur, qu'il existe un chemin élémentaire de chaque sommet vers la racine. Comme dis précédemment cette contrainte interdit la présence de circuits de taille strictement supérieure à un, mais les boucles sont autorisées puisqu'elles sont utilisées à des fins de modélisation.

La contrainte **proper-tree** spécifie que le graphe orienté G qui lui est associé doit être une forêt composée de NPROP arbres propres.

Exemple illustrant la contrainte $TREE(G,N)$ [3]

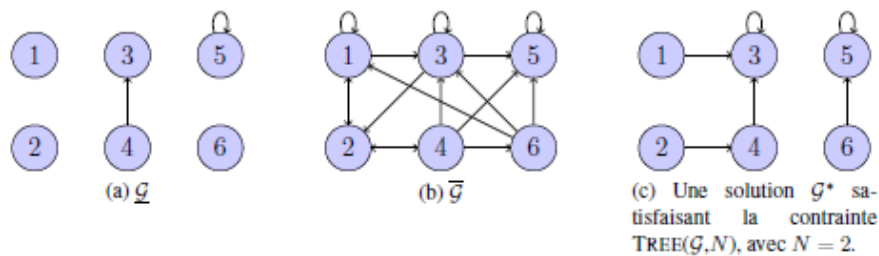


Figure 14 – Illustration de la contrainte $TREE(G,N)$

Définitions :

Un graphe orienté $G = (V, E)$ satisfait la contrainte d'arbre **tree(NTREE, VER)** si et seulement si :

1. Pour chaque $i \in [1, n]$ on a $VER[i].L = i$.
2. Le graphe G est composé de NTREE composantes connexes.
3. Chaque composante connexe de G ne contient pas de circuits impliquant plus d'un seul sommet.

2.8.2 Les contraintes ANTIARBO

[3] La contrainte ANTIARBO se note **ANTIARBO(G,R)** avec G un graphe orienté et R un noeud. Cette contrainte force une variable graphe orientée G à former une anti-arborescence enracinée au noeud. C'est une variante simplifiée de TREE dans laquelle la cardinalité de la partition est égale à un et le noeud racine, indiqué par une constante, n'a pas d'arc sortant (il n'a donc pas de boucle).

La contrainte ANTIARBO a deux avantages majeurs :

1. Il se concentre sur les propriétés structurelles liées à la notion d'(anti)-arborescence.
2. Permet d'éviter les défauts liés à la modélisation de la contrainte TREE.

2.8.3 Les contraintes ARBO

[3] La contrainte ARBO se note **ARBO(G,R)** avec G un graphe orienté et R un noeud. Cette contrainte force une variable graphe orientée G à former une anti-arborescence enracinée au noeud R . C'est une variante de la contrainte ANTIARBO dans laquelle l'orientation des arcs est inversée, les arcs sont orientés de la racine vers tous noeuds du graphe. Donc la racine a plusieurs arcs sortants mais les feuilles n'ont pas.

Il y a une relation de passage entre la contrainte ANTIARBO et ARBO, cette relation de passage n'engendre aucune perte de filtrage :

$$\text{ARBO}(G,R) \Leftrightarrow \text{ANTIARBO}(G^{-1},R)$$

3 Etude Solveurs

Pour résoudre un problème avec la programmation par contraintes nous aurons besoin d'un solveur. La contrainte qui existe est que tous les solveurs ne supportent pas toutes les modélisations. Par exemple nous ne pouvons pas résoudre un modèle ensembliste avec le solveur CHOCO. Donc le choix de notre solveur dépend de la modélisation que nous allons proposer par la suite. Par conséquent nous allons réaliser une étude sur différents solveurs dans ce qui suit.

3.1 Solveur CHOCO

Dans ce qui suit je me suis basé sur le lien suivant : [W6]

- Définition [W6]

CHOCO est un solveur de contraintes en Java. Choco est lancé en 1999 par *François La-burthe* et *Narendra Jussien*. C'est une librairie de modélisation et de résolution de problèmes mathématiques basés sur la programmation par contraintes (PPC). CHOCO est l'un des outils open source les plus performants dans son domaine, il est utilisé depuis plus de 15 ans par de nombreuses entreprises et universités. Écrit en Java, s'intègre facilement pour réaliser des web-services d'aide à la décision. La librairie est développée en collaboration entre "Cosling" qui est une entreprise fondée en 2014 par "Jean-Guillaume Fages" et "l'Ecole des Mines de Nantes" qui la distribue sous licence BSD.

Cette technologie traite efficacement les problèmes de planification d'ordonnancement, de placement d'objet, etc. On trouve de nombreux projets qui sont traités avec cette librairie par exemple : l'affectation de conducteurs à des lignes de bus, la génération automatique de résumé vidéo, la génération automatique de tests, l'optimisation de placement de machines virtuelles sur des plateformes d'hébergement etc.

Qui sont ses utilisateurs ?

De nombreux utilisateurs utilisent cette librairie cela va des étudiants, les laboratoires de recherche aux industriels.

CHOCO est l'une des librairies rares codées en Java, elle présente des spécificités intéressantes : des contraintes globales et locales, propagation des contraintes, des résolutions multi-threadées, un code facilement modifiable.

Les variables et les contraintes incluent dans CHOCO

CHOCO inclut des types divers de variables :

- 1 . Des variables booléennes.
- 2 . Des variables entières.
- 3 . Ensemble et réel.

CHOCO inclut des types divers de contraintes :

- 1 . Des contraintes alldifferent.
- 2 . Des contraintes count.
- 3 . Des contraintes nvalues. etc.

Il offre une recherche configurable (recherche personnalisée, recherche à base d'activité, grande recherche du voisinage, etc), il inclue des explications de conflit (conflict-based back jumping, dynamic backtracking, path repair, etc.)

Architecture logicielle de Programmation par Contraintes

Dans cette section nous allons voir la hiérarchie des classes de CHOCO ainsi que quelques services génériques implémentés par l'héritage permettent de partager des services entre deux implémentations différentes d'un même concept : une des manières de contrôler la propagation de CHOCO consiste à créer des sousclasses de variables et contraintes, qui redéfiniront leurs propres fonctions de gestion de la propagation. Quelques-unes parmi ces classes sont des classes abstraites (des noeuds internes de l'arbre d'héritage, sans instances)

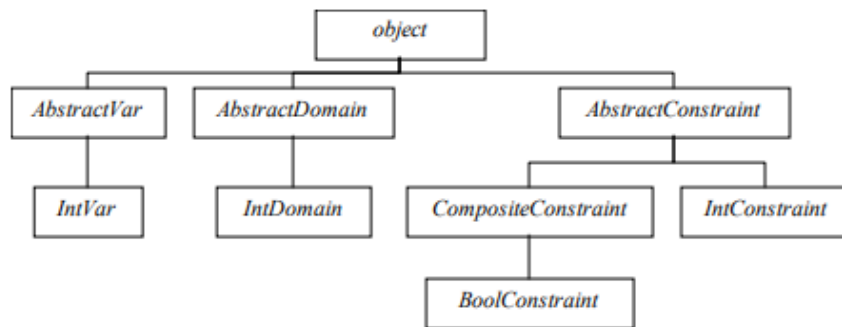


Figure 15 – Hiérarchie de classes de CHOCO

- La classe **AbstractVar** est la racine des classes qui représentent les variables de domaines. Elle implémente des liens vers toutes les contraintes.
- La classe **AbstractDomain** est la racine de toutes les classes implémentant des structures de données pour représenter un domaine de valeurs. Elle implémente une méthode pour tester si une valeur est présente ou non dans le domaine.
- La classe **AbstractConstraint** est la racine de toutes les classes implémentant des contraintes. Tous ses héritiers implémentent :
 - > Une référence qui permet de retrouver la contrainte parmi toutes les contraintes liées à une de ses variables.
 - > Tester satisfaisabilité, qui vérifie la satisfaisabilité, il est appelé lorsque toutes les variables sont initialisées.
 - > Un service d'accès indexé à chacune des variables.
- La classe **CompositeConstraint** est la racine de toutes les classes implémentant des contraintes faites de plusieurs sous-contraintes.
- La classe **BoolConstraint** est une sous-classe de **CompositeConstraint** implémentant les combinaisons Booléennes.
- La classe **IntVar** contient toutes les variables de domaine fini et implémente les fonctions suivantes :
 - > Une référence qui permet de retrouver la contrainte parmi toutes les contraintes liées à une de ses variables.
 - > Fonction **RemoveVal(x :IntVar, a :integer, i :integer)**, cette fonction retire la valeur v du domaine de la variable x et crée un événement REMOVAL généré par la i ème contrainte de x .
 - > Un service d'accès indexé à chacune des variables.
 - > Fonction **Instantiate(x :IntVar, a :integer, i :integer)** instancie la variable x à la valeur a et crée un nouvel événement INSTANTIATE généré par la i ème contrainte de x .
 - > Fonction **UpdateInf(x :IntVar, a :integer, i :integer)** augmente la borne inférieure de x à a , et crée un événement INCINF généré par la i ème contrainte de x .
 - > Fonction **UpdateSup(x :IntVar, a :integer, i :integer)** diminue la borne supérieure de

- x à a, et crée un événement **DECSUP** généré par la ième contrainte de x.
- La classe *IntDomain* est la racine de toutes les structures de données stockant des représentation d'un domaine fini. Elle implémente les services suivants :
 - > Compter le nombre de valeurs stockées.
 - > Supprimer une valeur.
 - > Calculer, mise à jour des bornes inférieures et supérieures du domaine.
 - > Itérer sur toutes les valeurs continues dans le domaine.
- La classe *IntConstraint* est la racine de toutes les classes représentant une contrainte portant exclusivement sur des variables à domaine fini. Tous ses descendants *XXConstraint* doivent implémenter une réaction à chacun des quatre types d'événement, par les méthodes suivantes :
 - > La fonction *AwakeOnInf(c :XXConstraint, i :integer)* pour la propagation d'une contrainte c suite à un événement INCINF de la ième variable de c.
 - > La fonction *AwakeOnSup(c :XXConstraint, i :integer)* pour la propagation d'une contrainte c suite à un événement DECSUP de la ième variable de c.
 - > La fonction *AwakeOnInst(c :XXConstraint, i :integer)* pour la propagation d'une contrainte c suite à un événement INSTANTIATE de la ième variable de c.
 - > *AwakeOnRem(c :XXConstraint, i :integer, a :integer)* pour la propagation d'une contrainte c suite à un événement REMOVAL(X,a) si x est la ième variable de c.

3.2 ILOG Solver ou CP Optimizer

- Définition [W7]

CP Optimizer connu aussi sous le nom de **ILOG Solver** il est créé en 1994 et permet de modéliser les problèmes et les résoudre. Il est plus complexe à utiliser comparé au solveur CHOCO. Pour développer une application l'utilisateur doit maîtriser le langage C++. Il permet d'écrire de nouvelles contraintes, explorer l'arborecence de recherche. Il faut disposer d'une licence pour pouvoir utiliser ce solveur. CP Optimizer est disponible via les interfaces suivantes :

- OPL
- C ++ : interface native
- Java : habillage du moteur C ++
- .NET : habillage du moteur C ++

Et sur les plateformes suivantes :

- Windows, Debian, Solaris, AIX et Darwin (Mac OS X)
- 64 et 32 bits

- Liste des contraintes disponible dans CP Optimizer

- Des contraintes de type *allDifferent* : contrainte des variables dans un tableau dvar à prendre chacune des valeurs différentes
- Des contraintes *allMinDistance* : contrainte des variables dans un tableau dvar à prendre chacune des valeurs différentes entre elles d'au moins un écart donné
- Des contraintes *inverse* : prend deux tableaux de variables entières qui doivent être indexées par un entier et être unidimensionnels
- Des contraintes *pack* : représente certaines contraintes de conditionnement unidimensionnelles mais puissantes.
- Des contraintes *lex* : indique que le premier tableau de variables est inférieur ou égal au second tableau de variables dans l'ordre alphabétique.
- Des contraintes *arithmétiques* : égalité, >, <, etc.

4

Analyse et conception

Dans ce chapitre nous allons proposer une modélisation de notre problème. Dans un premier temps nous allons étudier la proche de la modélisation mathématique. Ensuite nous allons détailler et expliquer clairement cette modélisation. Ce qui suit reprend le travail fait par Monsieur Hafedh MOHAMED BABOU. Il m'a été fourni par mon encadrant. Mon but est de corriger et expliquer cette modélisation [8].

1 Modélisation mathématique

Pour résoudre un problème à l'aide d'un solveur, nous devons définir les contraintes ainsi que les méthodes de résolution et les stratégies de choix des variables et de valeurs.

La première étape à faire lors de la modélisation d'un problème est l'identification de sous problèmes qui vont correspondre aux contraintes.

On considère un graphe orienté $D = (V, A)$ et un graphe non-orienté $G = (V, E)$ avec V l'ensemble des sommets, A l'ensemble des arcs et E l'ensemble des arêtes.

La modélisation que nous allons présenter se divise en deux grandes parties, la première partie va nous permettre de trouver le plus long chemin dans premier temps. Ensuite la deuxième partie consiste à vérifier la connexité.

1.1 Première étape (Recherche du plus long chemin)

La première étape est de chercher le plus long chemin dans le graphe orienté. Pour cela nous allons ajouter deux sommets à notre graphe. Un sommet source nommé " s " et un sommet terminal " t ". Le sommet s est connecté par des arcs sortants allant vers tous les sommets du graphe. Tous les sommets du graphe sont connectés par des arcs partant de tous les sommets vers le sommet terminal t.

Donc nous obtenons de nouveaux ensembles :

$$V' = V \cup \{ s \} \cup \{ t \} \text{ avec } V = \{1, 2, \dots, n\}.$$

$$A' = A \cup \{ (s, i) \} \cup \{ (i, t) \} \text{ avec } i \in V$$

$$D' = (V', A')$$

Variables :

Nous allons définir une variable booléenne. La variable x_{ij} représente l'arc entre le sommet i et j .
 $x_{ij} = 1$ si l'arc $(i, j) \in A'$ est pris dans le chemin à construire.

Nous allons définir une autre variable booléenne $\forall i \in V$ z_i représente le sommet i du graphe.

La variable $z_i = 1$ si le sommet i est pris dans le chemin. Cette variable va nous permettre de lier la partie recherche du plus long chemin et la partie connexité.

Comme nous cherchons le plus long chemin donc nous allons chercher à maximiser le nombre d'arcs pris dans la solution.

Donc notre fonction objectif est

$$\text{Max } \sum_{(i,j) \in A} x_{ij}$$

Contraintes :

Pour avoir un chemin il est indispensable de poser des contraintes :

1. $\sum_{(s,j) \in A'} x_{sj} = 1$, cette contrainte spécifie qu'il faut avoir un seul arc qui sort du sommet source "s".
2. $\sum_{(i,k) \in A'} x_{ik} = \sum_{(k,j) \in A'} x_{kj}$ avec $\forall k \in V$ cette contrainte permet de construire un chemin, pour tout sommet du graphe le nombre d'arc sortant sont égaux au nombre d'arc entrant dans k excepté le sommet source "s" et le sommet terminal "t".
3. $z_i = \sum_{(i,j) \in A'} x_{ij} \forall i \in V$, cette contrainte permet de dire que si un sommet i appartient au chemin recherché c'est-à-dire $z_i = 1$ donc la somme des arcs sortant de i est égale à 1, d'une autre manière si le sommet i appartient au chemin recherché il existe forcément un sommet $j \in V'$ formant un arc avec le sommet i tel que cet arc appartient au chemin.

1.2 Deuxième étape (connexité)

Maintenant nous avons fixé le chemin, il faudra maintenant vérifier la connexité dans le graphe non-orienté G . Pour que le chemin soit connexe, il suffit qu'il induit un arbre dans G . On s'est inspiré de la modélisation d'un arbre couvrant de poids minimal à l'aide d'une formulation linéaire, dite formulation de Martin.

1)- Formulation de Martin

L'objectif de la formulation de Martin est de trouver un arbre couvrant de poids minimal et qui passe par tous les sommets.

Variables :

Nous allons définir une variable booléenne y_{ij} qui représente l'arête entre le sommet i et j .

$\forall (i,j) \in E$ $y_{ij} = 1$, si l'arête est prise dans l'arbre couvrant.

$\forall (i,j) \in E \quad \forall k \in V - \{i,j\} \quad b_{ij}^k = 1$, si l'arête (i,j) est prise dans l'arbre couvrant et le sommet k est du côté de j .

La variable $b_{ji}^k = 1$, si l'arête (i,j) est prise dans l'arbre couvrant et le sommet k est du côté de i .

Plus généralement les deux précédentes variables spécifient que si l'arête (i,j) est prise dans l'arbre couvrant alors pour tout autre sommet k du graphe soit il est connecté à i soit à j mais pas les deux, cas sinon on aurait un cycle.

L'objectif de la formulation de Martin est de sélectionner le moins d'arête possible.

Donc
$$\text{Min } \sum_{(i,j) \in E} w_{ij} y_{ij} \quad \text{avec } w_{ij} \text{ le poids de l'arête } (i,j)$$

Contraintes :

1. $\sum_{(i,j) \in E} y_{ij} = |V| - 1 \quad |V| = n$ cette contrainte dit qu'il faut sélectionner exactement $n-1$ arêtes. C'est une propriété nécessaire des arbres. En d'autres termes, je veux exactement $n-1$ arêtes et avec la fonction objective, il va essayer de s'arranger pour sélectionner les $n-1$ arêtes telles que la somme en termes de poids et la plus petite possible.
2. $b_{ij}^k + b_{ji}^k = y_{ij} \quad \forall (i,j) \in E \quad \forall k \in V - \{i,j\}$ cette contrainte permet de dire si je ne sélectionne pas l'arête (i,j) donc $y_{ij} = 0$ alors forcément la somme de tous les b_{ij}^k et b_{ji}^k seront égales à zéro. Si l'arête (i,j) est sélectionnée donc $y_{ij} = 1$ pour tout autre sommet k du graphe soit il est relié à j ($b_{ij}^k = 1$) soit il est relié à i ($b_{ji}^k = 1$), mais pas les deux en même temps sinon nous aurons un cycle.
3. $\sum_{(i,j) \in E - \{(i,j)\}} b_{ik}^j + y_{ij} = 1 \quad \forall (i,j) \in E$ cette contrainte permet de dire si l'arête (i,j) n'est pas sélectionnée ($y_{ij} = 0$) donc la somme des $b_{ik}^j = 1$ cela signifie qu'il faut exactement un b_{ik}^j qui soit égale à 1. D'une autre façon si je ne sélectionne pas l'arête (i,j) il faut quand même un chemin passant par le sommet k qui relie i et j . Cette contrainte assure la connexité. Sinon si j'ai sélectionné l'arête (i,j) il ne peut pas exister de sommet k qui serait relié à i sélectionner et du côté de j car si c'est le cas cela veut dire qu'il y a un cycle.
Nous allons adapter la formulation de Martin à notre cas. Nous on ne veut pas forcément un arbre couvrant entre tous les sommets mais juste entre les sommets qui ont été sélectionnés dans la partie recherche du plus long chemin. Dans notre cas nous n'avons pas de fonction objective on veut juste vérifier s'il existe une solution.

2)- Adaptation du modèle de Martin à notre problème

Maintenant que nous avons expliqué la formulation de Martin, nous allons adapter celle-ci pour mettre les contraintes qui vont assurer que les sommets du chemin sélectionné dans la première partie du modèle forment un arbre (et par conséquent un sous-graphe connexe) dans G .

Contraintes connexité :

1. $z_i \geq y_{ij} \quad \forall (i,j) \in E$.
 2. $z_j \geq y_{ij} \quad \forall (i,j) \in E$.
- ces deux premières contraintes spécifient que si nous avons sélectionné le sommet i et j ($z_i = z_j = 1$) alors y_{ij} doit être inférieure à 1 et donc valoir 0 ou 1. En d'autres termes si j'ai sélectionné i et j je peux sélectionner ou pas l'arête (i,j) . Par contre z_i ou z_j est égale à 0 alors forcément y_{ij} est égale à 0. D'une façon plus claire on a le droit de sélectionner que des arêtes entre les sommets sélectionnés.
3. $\sum_{(i,j) \in E} y_{ij} = \sum_{i \in V} z_i - 1$ cette contrainte spécifie qu'il faut sélectionner exactement le nombre de sommet sélectionner moins une arête.
 4. $b_{ij}^k + b_{ji}^k = y_{ij} * z_k \quad \forall (i,j) \in E \text{ et } \forall k \in V - \{i,j\}$ cette contrainte spécifie que :
 - Si l'arête (i,j) est sélectionnée et le sommet k est sélectionné alors forcément il faut que k soit du côté de i soit du côté de j .
 - Si on ne sélectionne pas le sommet k ou bien l'arête (i,j) alors forcément la somme des b_{ij}^k et b_{ji}^k vaut zéro.
 5. $\sum_{(i,k) \in E - \{(i,j)\}} b_{ik}^j + y_{ij} = z_i * z_j \quad \forall (i,j) \in E$ cette contrainte spécifie que si j'ai sélectionné les sommets i et j donc j'ai le droit éventuellement ou non de prendre l'arête (i,j) c'est ce que nous avons vu dans les contraintes 1 et 2. Le cas où on ne sélectionne pas l'arête (i,j) , cette contrainte permet de dire qu'il faut trouver un sommet k qui relie le sommet i et le sommet j .
Si les sommets i et j sont sélectionnés et le sommet k n'est pas sélectionné alors forcément $b_{ik}^j = 0$ du coup le solveur n'a pas le droit de trouver un chemin entre i et j qui passe par k .

5

Mise en œuvre

Dans cette partie nous allons parler des librairies utilisées, le choix des versions, les différents risques rencontrés pendant l'implémentation, les choix techniques et l'évaluation des performances.

1 Librairie CHOCO

Le solveur que j'ai choisi d'utiliser est le solveur Choco que j'ai introduit dans "*Etude Solveurs*". Concernant le choix de la version utilisé est la version **4.0.4**, car les versions les plus récentes ne sont pas très stable. Pour certaines versions, ne contiennent pas certains opérateurs que j'utilise dans mes contraintes. Au fur et à mesure du développement je me suis rendu compte de ce problème.

J'ai mis en place un manuel d'installation de Choco, en utilisant les dépendances Maven. Ce manuel d'installation a été joint à ce rapport.

2 Framework JUNIT

Les différentes fonctionnalités implémenter sont tester avec le framework **JUNIT** version **4.12**. Les tests unitaires sont implémenter selon une stratégie définit dans le fichier de test joint à ce rapport.

On peut lancer tous les tests unitaires on allons dans le projet dans le terminal ensuite on saisie "*mvn test*" comme suit :

```

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 sec
Running org.polytech.solution.SolutionTest
== Set up class
- setup
= teardown
- setup
= teardown
== Tear down class
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 sec

Results :

Tests run: 15, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:29 min
[INFO] Finished at: 2019-03-23T22:27:36+01:00
[INFO] -----

```

Figure 1 – Lancement de tous les tests unitaires

3 Intégration continue (GitLab-CI)

L'intégration continue est mise en place en utilisant **GitLab-CI**, pour le processus de build et l'automatisation des tests. À chaque modification du code source, le processus de build est lancé automatiquement. Cela permet de vérifier si les modifications ne produisent pas de régression dans l'application développée. Comme le montre l'image ci-dessous :

MoradSan > PRD_Project_D15 > Pipelines

All 12

Pending 0

Running 0


Finished 12

Branches

Tags


Run Pipeline


passed


#52715810 by 

Y master -> da6ac035


Can't find HEAD commit for this branch



 00:01:13


 3 days ago


passed


#52654642 by 

Y master -> 3fafb9e6


Can't find HEAD commit for this branch



 00:01:11


 3 days ago


passed


#52616251 by 

Y master -> f1c649d1


Can't find HEAD commit for this branch




 00:01:08


 4 days ago


passed


#52615333 by 

Y master -> e530d0b8


 Update .gitlab-ci.yml




 00:01:07


 4 days ago


failed

#52614902 by 

Y master -> 3e58a95e

 Add .gitlab-ci.yml



 00:01:09


 4 days ago

Figure 2 – Intégration continue GitLab-CI

Cette partie est bien détaillé dans le cahier de tests.

4 Versioning du code (GitLab)

Pendant l'ensemble du processus de développement un versioning du code a été mis en place avec **GitLab**. Ceci m'a permis durant le développement d'avoir la possibilités de revenir en

arrière.

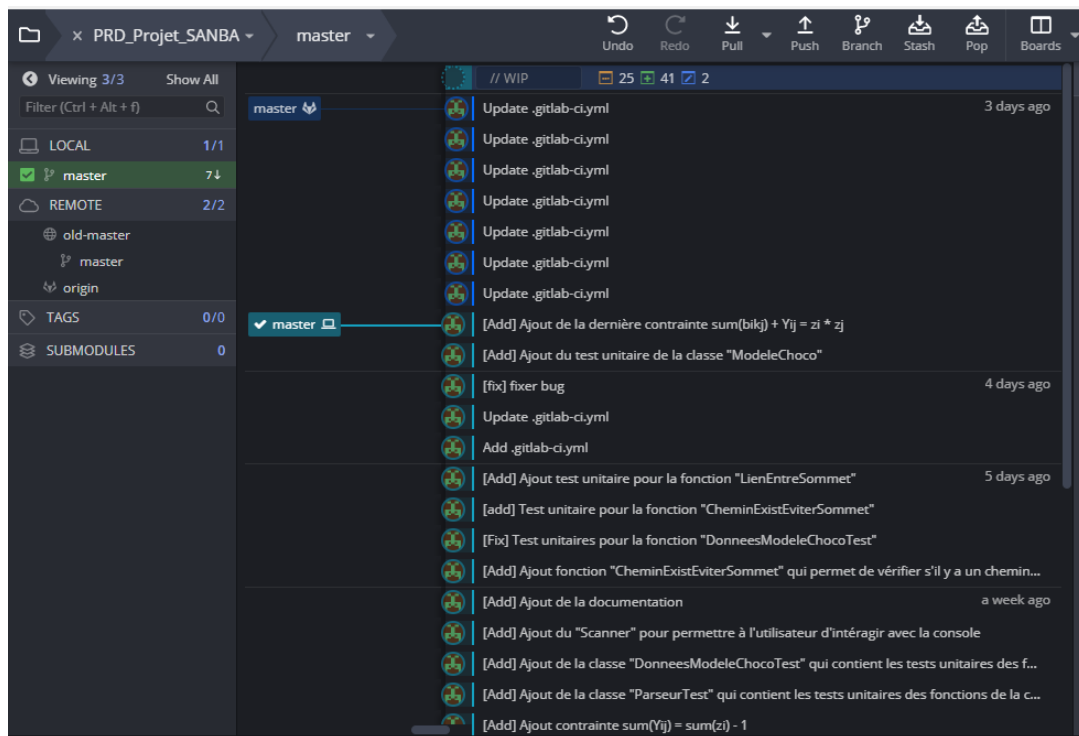


Figure 3 – Versioning du code GitKraken

5 Revue du code : plugin CheckStyle

Pour la revue du code j'ai utilisé *CheckStyle*, c'est un plugin dans **IntelliJ-IDEA**. Ce dernier permet d'assurer un niveau bien défini de qualité du code fourni. Un code lisible et convenablement commenté. Le plugin CheckStyle vérifie essentiellement la forme du code fourni. Il se présente ainsi :

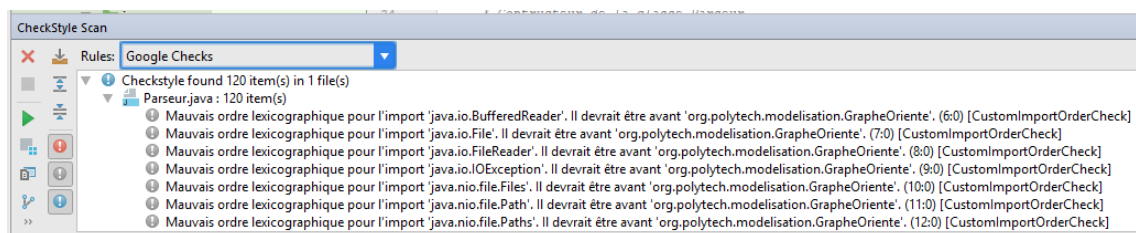


Figure 4 – Revue du code avec checkStyle

6 Documentation

Une documentation du code est faite pour faciliter la compréhension du code. Une documentation de l'ensemble du code est généré selon la norme Doxygen.

The screenshot shows the Doxygen documentation for the class `DonneesModeleChocoTest`. The sidebar on the left lists 'All Classes' and 'Packages'. The main content area has tabs for 'OVERVIEW', 'PACKAGE', 'CLASS' (selected), 'TREE', 'DEPRECATED', 'INDEX', and 'HELP'. Below these are links for 'PREV CLASS', 'NEXT CLASS', 'FRAMES', and 'NO FRAMES'. A 'SUMMARY' section shows the class hierarchy: `org.polytech.modelisation` → `java.lang.Object` → `org.polytech.modelisation.DonneesModeleChocoTest`. The class is described as `public class DonneesModeleChocoTest extends java.lang.Object`. A note states: 'Created by Morad on 07/03/2019. Classe DonneesModeleChocoTest cette permet de tester les fonctions de la classe DonneesModeleChoco'. Below this is a 'Constructor Summary' section with a 'Constructors' tab showing the constructor `DonneesModeleChocoTest()`. A 'Method Summary' section follows, with tabs for 'All Methods', 'Instance Methods', and 'Concrete Methods'. A table lists methods with columns 'Modifier and Type' and 'Method and Description'. The table shows a method `void connexionArcXij()`.

Figure 5 – Exemple documentation Doxygen

7 Choix techniques

7.1 Choix solveurs

Au premier semestre une étude a été réalisée sur les solveurs qui peuvent être utilisés. Mon choix est le solveurs **Choco**. C'est une librairie de modélisation et de résolution de problèmes mathématiques basés sur la programmation par contraintes (PPC). CHOCO est l'un des outils open source les plus performants dans son domaine, il est utilisé depuis plus de 15 ans par de nombreuses entreprises et universités. Contrairement au solveur **ILOG Solver** qui n'est pas un solveur open source est demande une license.

7.2 Choix langages de programmation

Lors de l'étude choix des solveurs, j'ai effectué une étude sur les langages de programmation que nous pouvons utiliser pour chaque solveurs, voir partie **Choix solveurs**. Mon choix s'est porté sur le langage **JAVA**.

8 Identification et gestion des risques

Une analyse des risques est faite pour identifier les entités comportant une influence négative sur le déroulement du projet et sur l'atteinte des objectifs. Cette étude m'a permis d'identifier les différents types de risques et leurs degrés d'influences. Cela m'a permis de bien gérer mon projet et d'éviter tout retard sur le projet et de mieux maîtriser les aléas.

Démarche de gestion du risque

Voici la démarche de gestion du risque mise en place :

- Identification des risques.
- Evaluer l'impact des risques sur le délai et la qualité.

- Mettre en place les actions pour réduire les risques.
- Suivre les actions et contrôler l'état des risques

8.0.1 risques humains

Il y a un risque humain par exemple (maladie), ce qui va apporter atteinte au déroulement du projet. Ce risque pourra entraîner un retard sur le projet où bien la finalisation de celui-ci.

8.0.2 risques sur les délais

Ce risque est liée à l'estimation initiale faite auparavant, les tâches prédéfinies, certaines tâches peuvent prendre plus de temps que prévu. Dans le cas ou une tâche prend plus de temps que prévu une réestimation est faite pour connaître l'influence de ce risque sur l'objectif du projet. Pour remédier à ce problème il faudra essayer de passer moins de temps sur d'autres tâche pour respecter les délais sans impacter ces dernières tâches.

8.0.3 risques techniques

Les risques techniques sont par exemple l'utilisation de la librairie **Choco** que je n'ai pas utilisée auparavant. Pour éviter ce risque j'ai intégré une tâche à mon planning qui consiste à prendre en main cette librairie, découvrir les différentes fonctionnalités à travers des exemples. Concernant le risque lié au langage de programmation j'ai déjà travaillé avec le langage **JAVA** ce qui ne nécessite pas d'effectuer une formation dessus.

9 Fichier de sortie

A l'exécution de l'application les résultats obtenus sont stockés dans un fichier texte et affiché à la console. Pour chaque instance, un fichier "**Resultat.txt**" est créé dans le même dossier où se trouve l'instance.

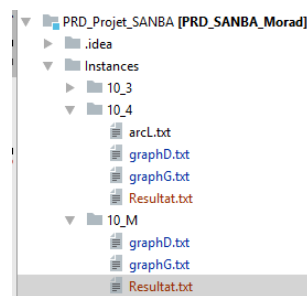


Figure 6 – Fichier résultat

Ce fichier contient le plus long chemin consistant trouvé il se présente ainsi :

```

===== Informations instances =====
Nombre de sommet de l'instance = 10
Nom de l'instance du graphe non orienté est : instances/10_4/graphG.txt
Nom de l'instance du graphe orienté est : instances/10_4/graphD.txt
=====
***** Plus long chemin (D,G)-consistant *****
=====
0---->5
5---->3
3---->7
7---->1
1---->11
=====
***** Statistiques de résolution : temps d'exécution, objectif, nombre de solution trouvé etc ... *****
=====
Building time = 1.5470315
Number solution find = 2
MAXIMIZE objectif = 5

```

Figure 7 – Exemple résultat stocker dans le fichier texte

L'affichage dans la console se présente ainsi :

```

===== Informations instances =====
Nombre de sommet de l'instance = 10
Nom de l'instance du graphe non orienté est : instances/10_4/graphG.txt
Nom de l'instance du graphe orienté est : instances/10_4/graphD.txt
=====
***** Plus long chemin (D,G)-consistant *****
=====
0---->5
5---->3
3---->7
7---->1
1---->11
=====
***** Statistiques de résolution : temps d'exécution, objectif, nombre de solution trouvé etc ... *****
=====
** Choco 4.0.4 (2017-04) : Constraint Programming Solver, Copyleft (c) 2010-2017
- Model[ModeleChoco] features:
  Variables : 3162
  Constraints : 3076
  Building time : 3,688s
  User-defined search strategy : yes
  Complementary search strategy : no
- Complete search - 2 solution(s) found.
  Model[ModeleChoco]
  Solutions: 2
  MAXIMIZE objectif = 5,
  Building time : 2,151s
  Resolution time : 1,539s

```

Figure 8 – Exemple affichage console

10 Comparaison des résultats

Le but est de proposer une modélisation qui permet de résoudre le problème en un temps court. La modélisation mise en place nous permet de résoudre le problème en un temps court pour les instances qui contiennent moins de 110 sommets. Concernant les instances de plus de 110 sommets cela prend plus de temps pour trouver la solution optimale. Donc nous pouvons conclure que la modélisation qui a été mise est efficace sur de petites instances. Le matériel utilisé peut influencer le temps de résolutions, dans mon je travaille avec une machine qui a les propriétés suivantes :

Système	
Processeur :	Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz 1.80 GHz
Mémoire installée (RAM) :	6,00 Go (5,89 Go utilisable)
Type du système :	Système d'exploitation 64 bits, processeur x64

Dans un premier temps j'ai étudié le code de l'algoBB de l'année dernière, que j'ai modifié un peu pour spécifier les instances à tester et mettre un "timeout" pour arrêter l'exécution. Ensuite pour pouvoir comparer la modélisation et l'algoBB j'ai utilisé les mêmes instances. J'ai également lancé le calcul pendant une heure et demie. Après cela j'ai constaté que pour les petites instances l'exécution est plutôt rapide pour les deux. Voici un récapitulatif des différents résultats obtenus :

Dans le tableau qui suit on trouve les temps de résolution des instances :

Résultats						
Instances	Modélisation Pas d'arc imposé		Modélisation Arc imposé		AlgoBB Arc imposé	
	Temps de résolution (secondes)	Nombre de sommets choisi dans la solution	Temps de résolution (secondes)	Nombre de sommets choisi dans la solution	Temps de résolution (secondes)	Nombre de sommets choisi dans la solution
Instance 10_1	0,1478351	2	0.11762	2	0,0012462	2
Instance 10_2	0,1461302	2	0.1265	2	0,0045114	2
Instance 10_3	0,1425288	3	0.1263	3	0,0039733	3
Instance 10_4	0,76335853	0	0.345	0	0,0021877	0
Instance 10_5	0,6038372	3	0.313	3	0,005779	3
Instance 30_1	292,5486	7	53,893	7	0,0467443	5
Instance 30_2	229,59007	7	60,362	7	0,1494419	7
Instance 30_3	71,94015	7	52.56	7		
Instance 30_4	168,43814	7	76.128	7	0,537659	7
Instance 30_5	131,346466	14	128.79	14	0,805506	11
Instance 60_1	4176,67	12	2188.32	12	0,6688119	15
Instance 60_2	6 883,146	12	3 521,12	12	3,144079	14
Instance 60_3	6 884,646	16	3 540,32	16	3,078002	17
Instance 60_4			4676,512	23	3,19333	18
Instance 60_5					3.9982	19
Instance 110_1					4,8049722	25
Instance 110_2					6,265789	35

Figure 9 – Comparaison résultats AlgoBB et la modélisation

Dans le tableau ci-dessus on a trois comparaisons dans un premier temps j'ai testé la modélisation implémentée sans imposé de passer par un arc, on constate que pour les grandes instances le temps d'exécution est long car à chaque fois qu'une solution est trouvée il essaye de trouver une meilleure. Ensuite pour mieux comparer la modélisation et l'AlgoBB, j'ai imposé de passer par des arcs, car l'algoBB impose cela, de cette sorte nous pouvons mieux comparer les deux méthodes, voir la deuxième colonne du tableau (*Modelisation arc imposé*) là on peut constater que

le temps de résolution à considérablement diminué. Par contre l'aloBB reste le plus performant concernant le temps de résolution même pour les grandes instances.

6

Bilan et Conclusion

Dans ce chapitre nous allons faire le point sur ce qui a été fait pendant ce semestre, ce qui reste à faire et définir les prochains lots.

1 Bilan du semestre 9

Pour bien gérer mon projet j'ai utilisé un outil en ligne "Trello" cela m'a permis de définir les tâches à faire et de définir des dates limites pour chaque tâche. Chaque tâche est validée auprès de mon encadrant.

1. Les tâches réalisées pendant le semestre 9 :

- Comprendre le sujet et étudier sa faisabilité.
- Étude du problème biologique.
- Étude de la thèse de "Hafedh Mohamed Babou" et la PPC et la thèse de "Jean-Guillaume FAGES".
- Étudier le "GetCoverSet" + application sur des exemples.
- Étudier l'heuristique "ALGOH" + application sur des exemples.
- Étudier l'algo exacte "ALGOBB" + application sur des exemples.
- Étudier la PPC + appliquer sur des exemples simples.
- Rédaction du cahier des spécification + validation.
- Rédaction Etat de l'art.
- Etude du modèle ensembliste, booléen, etc.
- Etude sur différents solveurs (CHOCO, CP Optimizer).
- Proposer un Modèle, diagramme d'activité, diagramme des séquences, définir l'architecture système.
- Planification du S9 et S10.
- Lister les fonctionnalités et découpage en tâches.
- Etudier une modélisation mathématique + Proposer une modélisation.
- Rédaction du rapport.

Les tâches en cours :

- Préparer la soutenance du semestre 9.

La tâche validée modélisation prend un peu de retard, cela est due aux faites que mon projet est orienté plus Recherche que Développement.

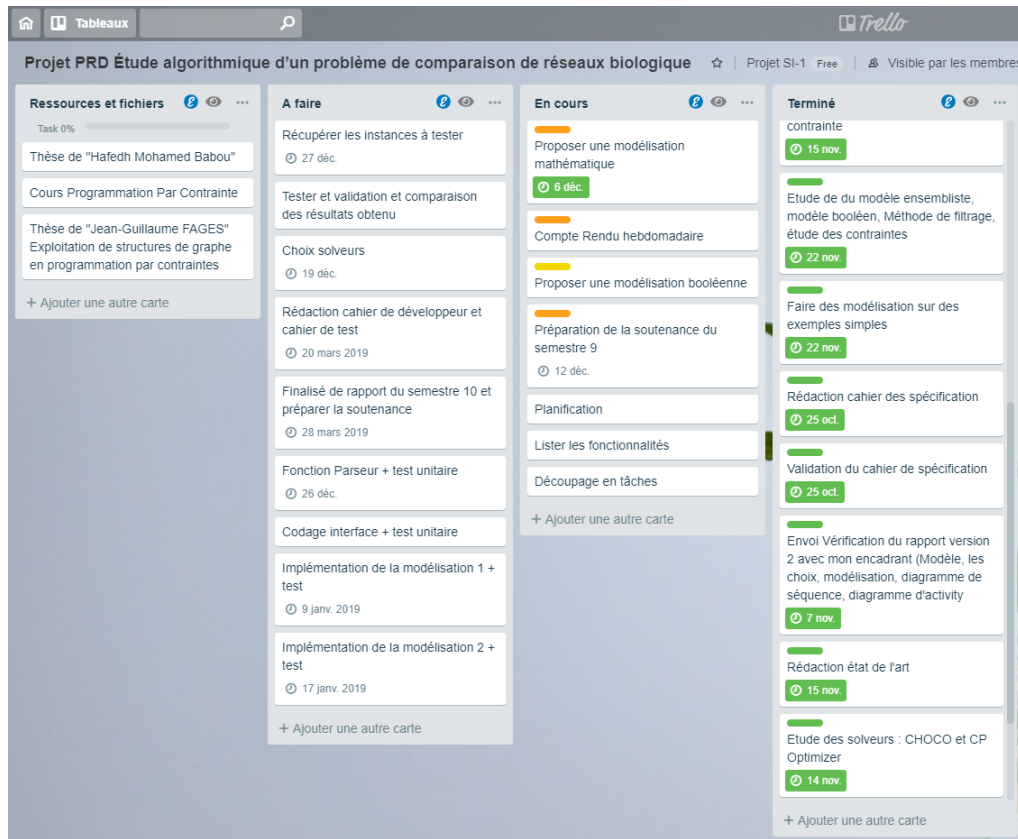


Figure 1 – les tâches du semestre 9 et du S10

2 Conclusion

Pour conclure au bout de ce premier semestre, le projet se déroule bien, j'ai pu bien comprendre le sujet et j'ai approfondi les recherches, j'ai eu quelques difficultés à faire la tâche de modélisation. La prochaine étape est de valider la modélisation avant de passer au codage.

3 Planning et tâche du semestre 10

Le semestre 10 est consacré au développement, nous allons lister par la suite les tâches prévues. Nous allons développer plus dans la partie "Plannification", le temps que chaque tâche va prendre, les entrées et les sorties.

1. Les tâches et fonctionnalités du semestre 10 :

- Proposer une modélisation.
- Valider la modélisation.
- Choix du solveurs (cela dépend de la modélisation validée)
- Récupérer les instances auprès de mon encadrant.
- Rédaction rapport sur la modélisation et validation.
- Manuel d'installation choco.
- Prise en main de la librairie Choco.
- Codage parseur graphe orienté et parseur graphe non orienté.
- Tester classes parseur.
- Implémentation de la classe graphe et la classe **GrapheNonOriente** et classe **GrapheOriente** + test.

- Générer les données du graphes orienté et du graphe non orienté + test.
- Implémentation de la classe **DonneesModeleChoco** + test.
- Validation des données générées.
- Modélisation logicielle (Diagramme de classe) + validation.
- Codage de la partie plus long chemin.
- Codage de la contrainte $\sum_{(s,j) \in A'} x_{sj} = 1$ + test.
- Codage de la contrainte $\sum_{(i,k) \in A'} x_{ik} = \sum_{(k,j) \in A'} x_{kj}$ avec $\forall k \in V$ + test.
- Codage de la contrainte $z_i = \sum_{(i,j) \in A'} x_{ij} \forall i \in V$ + test.
- Codage de la fonction objective : **Max** $\sum_{(i,j) \in A} x_{ij}$
- Validation de la partie plus long chemin + test sur des petites et des grandes instances.
- Implémentation de la partie 2 (connexité)
- Codage de la contrainte $z_i \geq y_{ij} \quad \forall (i,j) \in E$ et de la contrainte $z_j \geq y_{ij} \quad \forall (i,j) \in E$ + test.
- Codage de la contrainte $\sum_{(i,j) \in E} y_{ij} = \sum_{i \in V} z_i - 1$ + test.
- Codage de la contrainte $b_{ij}^k + b_{ji}^k = y_{ij} * z_k \quad \forall (i,j) \in E$ et $\forall k \in V - \{i,j\}$ + test.
- Codage de la contrainte $\sum_{(i,k) \in E - \{(i,j)\}} b_{ik}^j + y_{ij} = z_i * z_j \quad \forall (i,j) \in E$ + test.
- Validation de la partie 2connexité + test sur des petites et grandes instances.
- Stockage des résultats dans un fichier texte.
- Intégration continue.
- Cahier de test.
- Cahier de développeur.
- Cahier de recette.
- Manuel d'installation.
- Cahier d'utilisation.
- Implémentation d'une interface.
- Documentation et génération de la JavaDoc.
- Rédaction du rapport + préparé soutenance semestre 10.

4 Gestion des tâches

Comme au premier semestre un **Trello** est mis en place pour la gestion des tâches en cours, les tâches à faire et les tâches faites. Il se présente ainsi :

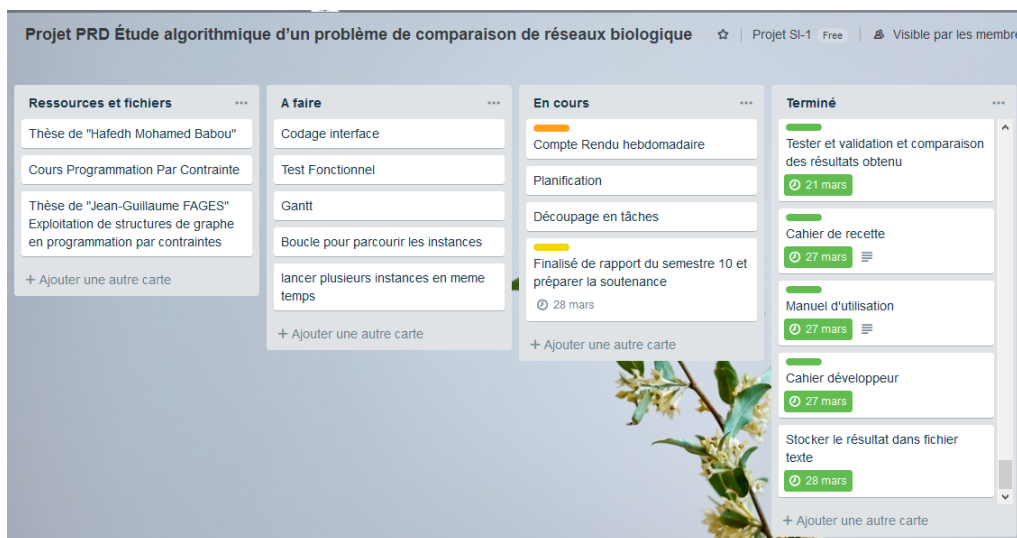


Figure 2 – les tâches du semestre 10

5 Bilan sur la qualité

Le code et les différents documents rendus vont permettre d'assurer la qualité du projet.

- La modélisation

J'ai mis en place une modélisation facilement modulable, qui permet d'ajouter d'autre modélisation sans changer totalement la structure du projet. Les différentes données récupérées sont séparées du modèle implémenter cela va permettre leurs réutilisations si on le souhaite.

- Structuration du code

Le code qui a été mis en place et bien structuré, découpage en package et en classes. Le code est bien commenté pour faciliter la compréhension de celui-ci. Le plugin **CheckStyle** utilisé assure la qualité du code fourni, permet d'avoir un code bien lisible et convenablement commenté.

- Documentation

Comme mentionné auparavant le code est bien commenté, également une documentation respectant la norme **Doxygen** a été générée pour comprendre les fonctions implémentées.

- Documents fournis

Un ensemble de documents a été joint à ce rapport pour faciliter la reprise de celui-ci. Parmi ses documents :

- *Le rapport sur la modélisation* qui contient les explications de la modélisation implémentée.
- *Un manuel d'installation Choco*, pour faciliter l'installation et de passer moins de temps sur cette tâche dans les projets avenir.
- *Un cahier de test* qui contient l'ensemble des tests effectués et leurs résultats.
- *Un cahier de recette* pour contient les différents résultats obtenus des instances.
- *Un cahier développeur*, qui contient les différents diagrammes (diagramme de classes, diagramme d'activité, user case) et leurs explications.
- *Un manuel d'utilisation*, qui contient les étapes à suivre pour utiliser l'application qui a été mise en place.

6 Bilan auto-critique

Dans la première partie de ce projet de réalisation et de développement faite au premier semestre, j'ai pu toucher légèrement aux bases de la recherche opérationnelle à travers les différents documents lus et analysés. J'ai bien respecté les délais de mes différentes tâches planifiées tout au début de la réalisation de ce projet. J'ai rencontré des difficultés quant à la modélisation de mon projet. Cependant, j'ai réussi à mettre en place une modélisation fonctionnelle pendant le second semestre. En effet, les résultats obtenus à la fin de la phase de développement affichent bien les bons résultats.

Par contre, au début de la phase de programmation j'ai passé plus de temps sur des tâches que prévu, ce qui a entraîné un petit retard à cause de l'utilisation d'une librairie que j'ai jamais utilisée auparavant. Cependant, ce retard a été rattrapé juste après la maîtrise de cette bibliothèque.

J'ai réussi à comparer les différents résultats de la modélisation et l'algoBB. J'ai ajouté un petit créneau dans mon temps personnelle afin d'effectuer cette tâche. L'analyse des résultats est expliquée dans la partie (Comparaison des résultats).

Enfin, tout au long de la réalisation de ce projet de fin d'études, j'ai pu approfondir mes connaissances et bien maîtriser le langage Java que j'ai utilisé pendant toute la phase de développement. J'ai aussi appris à avoir un peu de recul quant à la modélisation du problème, et aussi à avoir une vision globale de celui-ci. En outre, j'ai appris à utiliser la librairie CHOCO, que j'ai jamais employé auparavant.

Annexes

A

Plannification

Ce projet se déroule sur 2 jours par semaine à temps plein. Pour la gestion du projet j'ai utilisé le modèle en "cascade". Dans cette nous allons présenter le planning, les découpages en tâches, une synthèse des tâches réalisées.

1 Diagramme de Gantt prévisionnel

La planification de ce projet se présente ainsi :

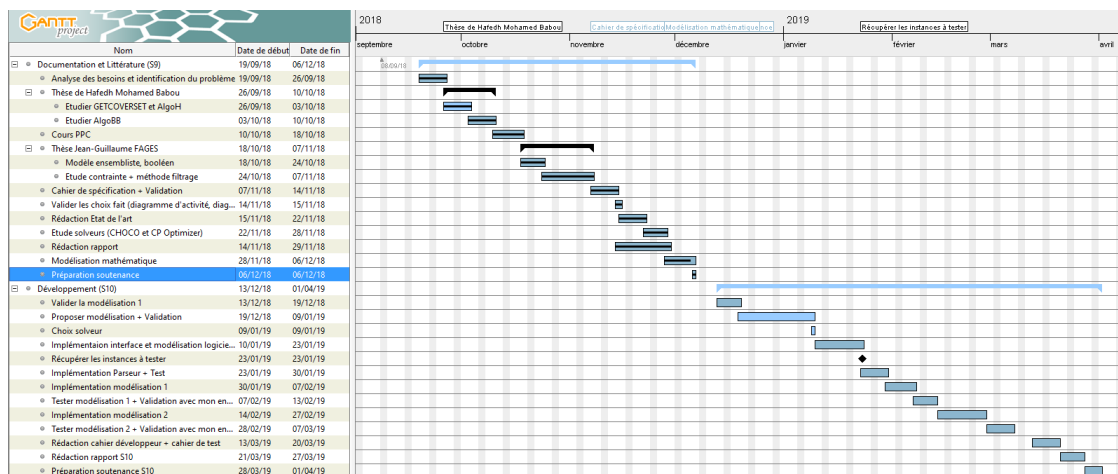



Figure 1 – Diagramme de Gantt prévisionnelle

2 Découpage en tâches au semestre 9

- Les tâches se présente comme suit :



Nom	Date de début	Date de fin
☐ • Documentation et Littérature (S9)	19/09/18	06/12/18
• Analyse des besoins et identification du problème	19/09/18	26/09/18
☐ • Thèse de Hamed Mohamed Babou	26/09/18	10/10/18
• Etudier GETCOVERSET et AlgoH	26/09/18	03/10/18
• Etudier AlgoBB	03/10/18	10/10/18
• Cours PPC	10/10/18	18/10/18
☐ • Thèse Jean-Guillaume FAGES	18/10/18	07/11/18
• Modèle ensembliste, booléen	18/10/18	24/10/18
• Etude contrainte + méthode filtrage	24/10/18	07/11/18
• Cahier de spécification + Validation	07/11/18	14/11/18
• Valider les choix fait (diagramme d'activité, diag...	14/11/18	15/11/18
• Rédaction Etat de l'art	15/11/18	22/11/18
• Etude solveurs (CHOCO et CP Optimizer)	22/11/18	28/11/18
• Rédaction rapport	14/11/18	29/11/18
• Modélisation mathématique	28/11/18	06/12/18
★ Préparation soutenance	06/12/18	06/12/18
☐ • Développement (S10)	13/12/18	01/04/19
• Valider la modélisation 1	13/12/18	19/12/18
• Proposer modélisation + Validation	19/12/18	09/01/19
• Choix solveur	09/01/19	09/01/19
• Implémentaion interface et modélisation logicie...	10/01/19	23/01/19
• Récupérer les instances à tester	23/01/19	23/01/19
• Implémentation Parseur + Test	23/01/19	30/01/19
• Implémentation modélisation 1	30/01/19	07/02/19
• Tester modélisation 1 + Validation avec mon en...	07/02/19	13/02/19
• Implémentation modélisation 2	14/02/19	27/02/19
• Tester modélisation 2 + Validation avec mon en...	28/02/19	07/03/19
• Rédaction cahier développeur + cahier de test	13/03/19	20/03/19
• Rédaction rapport S10	21/03/19	27/03/19
• Préparation soutenance S10	28/03/19	01/04/19

Figure 2 – Découpage en tâches

- Début du projet le 19/09/2018

- Date de fin du semestre S9 le 06/12/2018

1. Analyse des besoins et identification du problème :

- Date : du 19/09/2018 au 26/09/2018.
- Description : Cette tâche est consacrée à la compréhension du sujet et l'analyse des besoins. Dans un premier temps j'ai pris rendez-vous avec mon encadrant pour discuter du projet et comprendre ses attentes. J'ai commencé à prendre connaissance des documents qui sont mis à ma disposition.

2. Etudier la thèse de Hamed Mohamed Badou :

- Date : du 26/09/2018 au 10/10/2018.
- Description : Dans cette tâche j'ai étudié deux algorithmes et les appliquer sur des

exemples :

- * Étudier le GetCoverSet.
- * Étudier l'algoH.
- * Étudier l'algoBB.

Cette tâche a donné lieu à la présentation de ses algorithmes auprès de mon tuteur.

3. Étudier la programmation par contrainte (PPC) :

- Date : du 10/10/2018 au 18/10/2018.
- Description : Cette tâche va me permettre de bien comprendre la PPC qui va me permettre le principe de la PPC et savoir modéliser un problème simple.

4. Thèse Jean-Guillaume FAGES :

- Date : du 10/10/2018 au 18/10/2018.
- Description : Cette tâche va me permet de comprendre la modélisation des graphes en PPC. Il a donné lieu à la compréhension de différents modèles (ensemblistes, booléen, etc.), étude des méthodes de filtrage.

5. Rédaction du cahier de spécification et validation

- Date : du 07/11/2018 au 14/10/2018.
- Description : Rédaction du cahier de spécification, cette tâche a donné lieu à un livrable (première version du cahier des spécification) à valider auprès de mon encadrant.

6. Validation des choix (diagramme d'activité, diagramme des séquences etc.)

- Date : du 14/11/2018 au 15/11/2018.
- Description : Cette tâche a donné lieu à un diagramme d'activité, diagramme des séquences etc. à valider auprès de mon encadrant.

7. Rédaction Etat de l'art :

- Date : du 15/11/2018 au 22/11/2018.
- Description : Rédaction de l'état de l'art qui contient un résumé de tous les documents lus auparavant. Cette tâche à donner lieu à un livrable qui sera validé avec mon encadrant.

8. Étude solveurs (CHOCO et CP Optimizer) :

- Date : du 22/11/2018 au 28/11/2018.
- Description : Cela m'a permis d'étudier une liste de solveurs, comprendre leurs différences.

9. Rédaction du rapport

- Date : du 14/11/2018 au 29/11/2018.
- Description : Rédaction du rapport qui contient la "description générale", "Analyse et conception" + "Planning" etc. Cette tâche a donné lieu à un rapport qui est envoyé à mon encadrant.

10. Modélisation mathématique :

- Date : du 28/11/2018 au 06/12/2018.
- Description : Étude d'une modélisation mathématique de notre problème. Qui a été validé auprès de mon encadrant.

11. Préparation soutenance S9 :

- Date : du 06/12/2018 au 06/12/2018.
- Description : Préparer la présentation du projet et du bilan du semestre 9.

Concernant les tâches du semestre 9, ils sont toutes accomplis (100%).

- Planification du semestre 10 consacré au développement :

1. Valider la modélisation 1 :

- Date : du 13/12/2018 au 19/12/2018.

- Description : Une première modélisation est faite et sera validé avec mon encadrant. Cette tâche est critique c'est pour cela que j'ai prévu du temps. Il va nécessiter plusieurs rendez-vous avec mon tuteur pour éviter de perdre du temps et de proposer une modélisation incorrecte.
2. Choix de solveur :
 - Date : du 09/01/2019 au 09/01/2019.
 - Description : Le choix dépend des modélisations validées précédemment cette tâche va prendre une heure.
 3. Implémentation interface et modélisation logicielle :
 - Date : du 10/01/2019 au 23/01/2019.
 - Description : Proposer une modélisation et le validé avec avant de passer au codage.
 4. Récupération des instances à tester :
 - Date : du 23/01/2019 au 23/01/2019.
 - Description : Il me faudra des instances pour le 23 janvier 2019.
 5. Implémentation parseur + test :
 - Date : du 23/01/2019 au 30/01/2019.
 - Description : implémentation d'un parseur qui va lire les graphes et le chemin qui sont stockés dans un fichier "excel" selon un format précis que j'ai décrit précédemment. Cette tâche donnera lieu à un parseur que je vais tester sur plusieurs instances.
 6. Implémentation modélisation 1 :
 - Date : du 30/01/2019 au 07/02/2019.
 - Description : Cette tâche est cruciale dans la suite du projet, il va donner lieu un premier modèle implémenté.
 7. Tester modélisation 1 + validation :
 - Date : du 07/02/2019 au 13/02/2019.
 - Description : Nous allons tester et faire valider la modélisation, cette tâche inclut les tests sur plusieurs instances et corrections des erreurs. Il donnera lieu aussi à un premier résultat donc cette tâche est très importante.
 8. Implémentation modélisation 2 :
 - Date : du 14/02/2019 au 27/02/2019.
 - Description : Cette tâche est cruciale dans la suite du projet, il va donner lieu un l'implémentation de la seconde modélisation.
 9. Tester modélisation 2 + validation :
 - Date : du 28/02/2019 au 07/03/2019.
 - Description : Nous allons tester et faire valider la seconde modélisation, il inclut également la comparaison entre la première et la deuxième modélisation.
 10. Rédaction cahier développeur + cahier de test :
 - Date : du 13/03/2019 au 20/03/2019.
 - Description : donnera lieu à un cahier de développeur et cahier de test.
 11. Rédaction rapport S10 :
 - Date : du 21/03/2019 au 27/03/2019.
 - Description : Cette tâche ce fera en parallèle avec la tâche précédente.
 12. Rédaction cahier développeur + cahier de test + Manuel d'utilisation :
 - Date : du 28/03/2019 au 01/04/2019.

- Description : Préparer la présentation du projet et du bilan du semestre 10.

La planification du semestre 10 peut être amenée à changer si l'une des tâches prévues prend du retard ou d'autres tâches vient s'ajouter.

3 Diagramme de Gantt du semestre 10

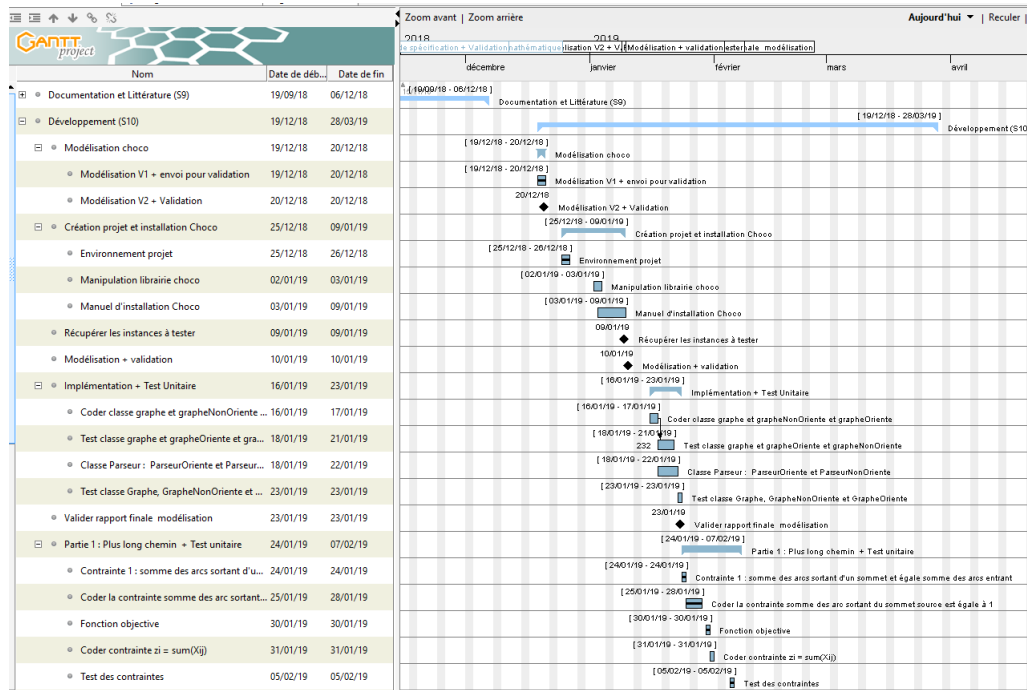


Figure 3 – Diagramme de Gantt du semestre 10

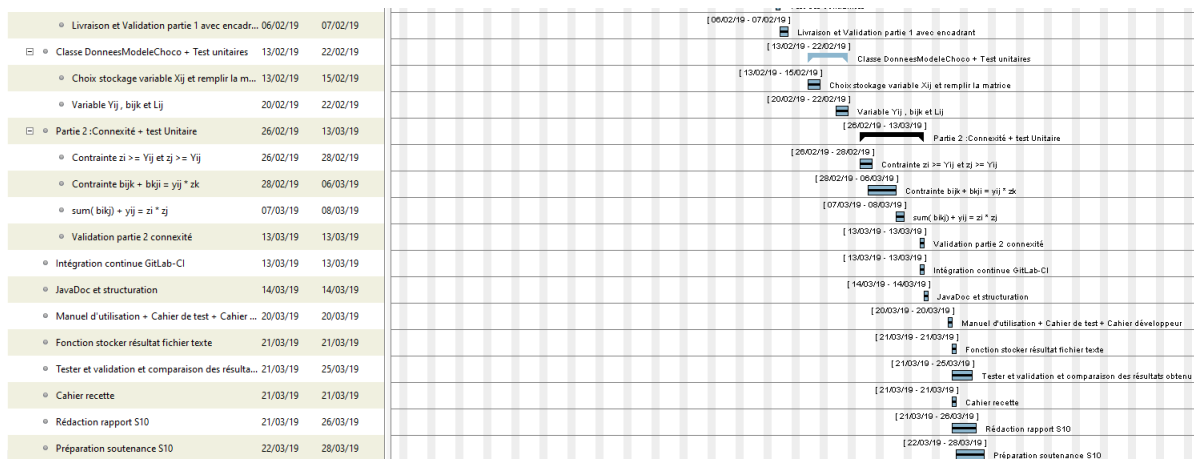


Figure 4 – Diagramme de Gantt du semestre 10

4 Tableau de synthèse du semestre 9

Les différentes tâches prévues pour le semestre 9 ce sont toutes achevées, voir le tableau de synthèse ci-dessous :

Tableau de synthèse S9	
Tâche	Taux d'achèvement
Analyse des besoins et Etude du problème	100 %
Etude du GetCoverSet, AlgoH et AlgoBB	100 %
Etude cours PPC	100 %
Exploration des structures de graphe en PPC	100 %
Etude du code existant	100 %
Validation du cahier de spécification	100 %
Etat de l'art	100 %
Etude sur les solveurs	100 %
Rédaction rapport	100 %

Figure 5 – Tableau de synthèse du S9

5 Tableau de synthèse du semestre 10

Un tableau de synthèse du semestre 10

Tableau de synthèse S10	
Tâche	Taux d'achèvement
Modélisation + Validation	100 %
Rédaction rapport sur la modélisation + validation	100 %
Génération de jeux de données	100 %
Modélisation (Diagramme de classe) + Validation	100 %
Manuel d'installation Choco	100%
Implémentation Parseur	100%
Implémentation partie 1 : plus long chemin	100%
Implémentation partie 2 : connexité	100%
Analyse résultat et test unitaires	100 %
Comparaison résultat avec AlgoBB	80%
Manuel d'installation + Manuel développeur + Manuel d'utilisation + Cahier de test	100 %
Interface	0%
JavaDoc + Génération de la documentation doxygen	100 %
Rédaction rapport	100 %
Préparation soutenance	100 %

Figure 6 – Tableau de synthèse du S10

B

Spécifications fonctionnelles

Dans cette partie nous allons expliquer les différentes fonctions qui sont implémentées. Pour résoudre le problème ONE-TO-ONE SKEWGRAM, j'ai implémenté la modélisation décrite précédemment. Chaque classe du projet possède une classe test dans laquelle sont implémentés les tests unitaires de chacune des fonctions de la classe. Nous allons présenter également la modélisation logicielle qui a été réalisée.

1 Diagramme de classe

La Diagramme de classe qui a été mis en place se présente ainsi :

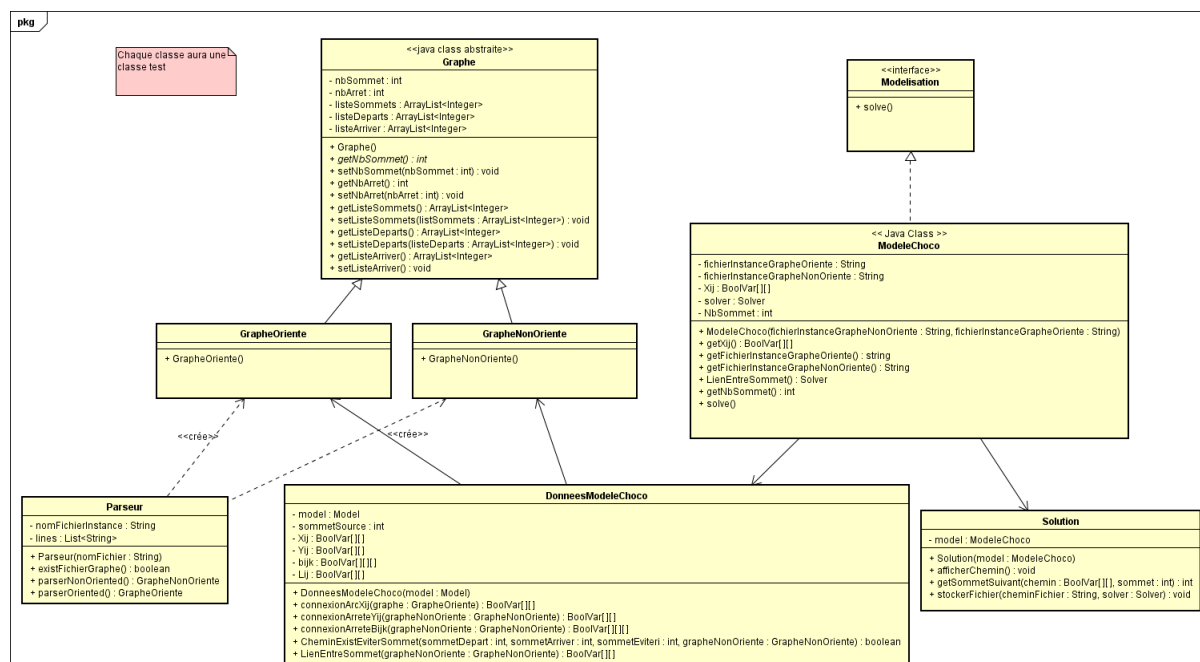


Figure 1 – Diagramme de classe

Dans ce diagramme on trouve une interface **Modelisation**, le rôle de cette dernière est si on souhaite implémenter une autre modélisation il suffit de créer une autre classe qui implémente

la méthode **solve()**. La classe **DonneesModelChoco** contient les données de modèle choco implémenté si une autre modélisation utilisé si même données nous pouvons faire appel à ces dernières directement.

La classe **Solution** contient les méthodes qui affiche correctement le chemin et la méthode qui permet de stocker dans un fichier texte également nous pouvons faire appel à ses méthodes pour afficher le chemin si plutard d'autres modélisations sont amenés à être implémenté.

Pour conclure la structure mise en place va nous permettre d'intégrer d'autres modélisations plus facilement sans refaire toute la modélisation.

2 Classe "Parseur"

La classe "Parseur" contient les différents parseurs pour le graphe orienté et le graphe non orienté. Il contient trois fonctions principale : **existFichierGraphe()**, **parserNonOriented()**, **parserOriented()** et les constructeurs. Cette classe a deux attribut **nomFichierInstance** qui représente le fichier instance et une liste de "string" qui représente les lignes des instances.

2.1 Fonction ExistFichierGraphe

- Présentation :

Cette fonction ne prend rien en paramètre et renvoie en sortie un *"boolean"*, le rôle de cette fonction est de vérifier si le fichier graphe a parsé existe ou pas. Si c'est le cas il renvoie *"true"* sinon il renvoie *"false"*. J'ai fait le choix d'implémenter cette fonction en dehors des deux parseurs car c'est le point commun entre le parseur du fichier graphe orienté et le parseur de fichier du graphe non orienté et cela va permettre d'éviter la duplication du code.

- Entrée :

- **Sortie** : Un boolean *"true"* si le fichier existe sinon *"false"*.

- **Priorité** : Primordiale.

2.2 Fonction ParserNonOriented

- Présentation :

Cette fonction représente le parseur du fichier qui contient le graphe non orientée (graphe G) il ne prend pas de paramètre, il renvoie un objet *"GrapheNonOriente"*. La fonctionnalité principale est de parser le fichier qui doit respecter le format ci-dessous :

- **Entrée** : rien

- **Sortie** : Objet *"GrapheNonOriente"*

- **Priorité** : Primordiale.

```

N:10
1-2
1-3
1-5
2-7
2-6
2-9
3-7
4-10
4-6
5-8
6-7
7-9
8-9

```

Figure 2 – Format du fichier graphe non orienté G

2.3 Fonction ParserOriented

- Présentation :

Cette fonction représente le parseur du fichier qui contient le graphe orientée (graphe D) il ne prend pas de paramètre, il renvoie un objet "**GrapheOrienté**". La fonctionnalité principale est de parser le fichier qui doit respecter le format ci-dessous :

```

N:10
P:0.2
3-10
3-7
5-3
5-6
7-1
9-7
10-6
10-2
2-4
4-8

```

Figure 3 – Format du fichier graphe orienté D

- **Entrée :**
- **Sortie :** Objet "**GrapheOrienté**"
- **Priorité :** Primordiale.

3 Classe "Graphe"

La classe "**Graphe**" est une classe abstraite, c'est la classe mère des deux classes "**GrapheNonOrienté**" et la classe "**GrapheOrienté**". Cette classe comporte les attributs suivants : "**nbSommet**" qui représente le nombre de sommet du graphe, "**NbArret**", "**ListeSommets**", "**ListeDeparts**", "**ListeArriver**". Nous pouvons ajouter d'autres attributs à cette classe si nous le souhaitons. Concernant les méthodes on trouve les constructeurs et les **getters** et **setters** de chaque attributs.

4 Classe "GrapheNonOriente"

La classe "*GrapheNonOriente*" est une classe fille, il hérite de la classe mère "*Graphe*". Dans cette classe on trouve le constructeur qui hérite du constructeur de la classe mère.

5 Classe "GrapheOriente"

La classe "*GrapheOriente*" est une classe fille, il hérite de la classe mère "*Graphe*". Dans cette classe on trouve le constructeur qui hérite du constructeur de la classe mère.

6 Classe DonneesModeleChoco

- Présentation :

La classe *DonneesModeleChoco* contient les différentes fonctions qui renvoient les données après avoir parser les fichiers graphes. Ces données sont transformées à un certain type d'objet que nous allons voir par la suite.

6.1 Fonction ConnexionArcXij

- Présentation :

Cette fonction prend en paramètre un objet "*GrapheOriente*" et renvoie en sortie une matrice de "*BoolVar*". Le rôle de cette fonction est récupérer l'objet graphe passé en paramètre, parser précédemment et de récupérer les arcs entre les sommets. Les arcs sont représentés par la variable x_{ij} .

- Entrée : *GrapheOriente*

- Sortie : *BoolVar*[][]

- Priorité : Primordiale.

6.2 Fonction connexionArreteYij

- Présentation :

Cette fonction prend en paramètre un objet "*GrapheOriente*" et renvoie en sortie une matrice de "*BoolVar*". Le rôle de cette fonction est récupérer l'objet graphe passé en paramètre, parser précédemment et de récupérer les arêtes entre les sommets. Les arêtes sont représentés par la variable y_{ij} .

- Entrée : *GrapheOriente*

- Sortie : *BoolVar*[][]

- Priorité : Primordiale.

6.3 Fonction connexionArreteBijk

- **Présentation :**

Cette fonction prend en paramètre un objet "*GrapheNonOriente*" et renvoie en sortie un "*BoolVar*". Le rôle de cette fonction est récupérer l'objet graphe passé en paramètre, parser précédemment et de récupérer les différentes valeurs de b_{ij}^k .

- **Entrée :** *GrapheNonOriente*

- **Sortie :** *BoolVar*[][][]

- **Priorité :** Primordiale.

6.4 Fonction CheminExistEviterSommet

- **Présentation :**

Cette fonction prend en paramètre un objet "*GrapheNonOriente*", un entier qui représente le sommet de départ "*sommetDepart*", un entier qui représente le sommet d'arriver "*sommetArriver*" et un autre entier qui représente le sommet à éviter "*sommetEviteri*" et renvoie en sortie un "*boolean*". Cette fonction permet de vérifier s'il y a un chemin d'un sommet de départ à un sommet d'arriver en évitant de passer par un sommet précis. Cette fonction est appelé dans la fonction "*connexionArreteBijk*" pour remplir les b_{ij}^k . La première condition et qu'il faut que l'arête (i,j) existe et il ait un chemin entre le sommet j et un sommet k différent de i à j.

- **Entrée :** *GrapheNonOriente*

- **Sortie :** *BoolVar*[][][]

- **Priorité :** Primordiale.

6.5 Fonction LienEntreSommet

- **Présentation :**

Cette fonction permet de récupérer le lien entre les sommets du graphe non orienté c'est-à-dire les arêtes. S'il y a une arête entre le sommet i et le sommet j donc on initialise la variable " L_{ij} ", cette variable n'est pas appelée par le modèle mais il est appelé par la fonction "*connexionArreteBijk*" il permet de vérifier si l'arête (i,j) existe ou pas et mettre directement les b_{ij}^k à *false* s'il n'existe pas.

- **Entrée :** *GrapheNonOriente*

- **Sortie :** *BoolVar*[][]

- **Priorité :** Primordiale.

7 Classe ModeleChoco

- **Présentation :**

Cette classe contient la modélisation Choco, il implémente l'interface *Modelisation*. Cette interface va nous permettre si plutard on souhaite implémenter une nouvelle modélisation de créer une autre classe qui contient une fonction *solve* qui résout le problème.

7.1 Fonction solve

La classe *ModelChoco* contient une fonction

solve ou le modèle est implémenter, on trouve les différentes variables, contraintes et la fonction objective.

8 Solution

- Présentation :

Cette classe contient les fonctions qui permettent de stocker le résultat dans un fichier texte, afficher dans la console les solutions trouvées et la fonction qui permet d'afficher le chemin correctement.

8.1 Fonction stockerFichier

- Présentation :

Cette fonction récupère le résultat renvoyé par le solveur et il le stocke dans un fichier texte, dans le même dossier qui contient l'instance testée.

- **Entrée :** *cheminFichier et solver*

- **Sortie :**

- **Priorité :** Primordiale.

8.2 getSommetSuivant

- Présentation :

Cette fonction permet de récupérer le sommet suivant, cela nous permettra d'afficher le chemin correctement.

- **Entrée :** *chemin de type BoolVar[][] et un sommet de type int*

- **Sortie :** le sommet suivant.

- **Priorité :** Primordiale.

8.3 afficherChemin

- Présentation :

Cette fonction permet d'afficher le chemin correctement.

- **Entrée :**

- **Sortie :**

- **Priorité :** Primordiale.

C

Spécifications non fonctionnelles

1 Contraintes de développement et conception

- Le système d'exploitation est Windows.
- Le langage de programmation : JAVA.
- L'IDE qui sera utilisé est IntelliJ IDEA 2017.

2 Contraintes de fonctionnement et d'exploitation

Dans ce projet il n'aura pas accès à l'application par un identifiant, ni sécurité. L'utilisateur accède directement à fonctionnalité principale qui est calculer le plus long chemin consistant. L'aspect performance est important dans notre cas

2.1 Performances

Le but est proposer une modélisation qui permet de résoudre le problème en un temps court.

D

Document joint à ce rapport

1 Cahier de test

Veillez trouver ci-joint à ce rapport le "cahier de test"

2 Manuel d'installation Choco

Veillez trouver ci-joint à ce rapport le "manuel d'installation de choco"

3 Cahier de recette

Veillez trouver ci-joint à ce rapport le "cahier de recette"

4 Cahier de développeur

Veillez trouver ci-joint à ce rapport le "cahier de développeur"

5 Manuel d'utilisation

Veillez trouver ci-joint à ce rapport le "manuel d'utilisation"

E

Webographie

- [W1] : Cours complexité **URL** : <https://info-llg.fr/commun-mpsi/pdf/06.complexite.pdf>
- [W2] : Notions PPC **URL** : <https://www.techno-science.net/definition/6423.html>
- [W3] : Notions PPC **URL** : <https://tel.archives-ouvertes.fr/tel-00460193/document>
- [W4] : Contraintes globales **URL** : https://wiki.bordeaux.inria.fr/realopt/uploads/Project/GT08_PPC_handout.pdf
- [W5] : Contraintes globales **URL** : <http://www.cril.univ-artois.fr/~ramon/biblio/RAM-07.pdf>
- [W6] : Solveur CHOCO **URL** : http://emergences.inria.fr/2016/newsletter_n42/L42-CHOCO
- [W7] : Solveur CHOCO **URL** : <http://www.lirmm.fr/~boureau/SOURCES/Boureau%20JNPC00%20Choco%20implementation.pdf>
- [W8] : Solveur CP Optimizer **URL** : https://www.ibm.com/support/knowledgecenter/fr/SSSA5P_12.5.0/ilog.odms.ide.help/OPL_Studio/opllangref/topics/opl_langref_constraints_types_inCP.html
- [W9] : Complexité **URL** : <https://www.enseignement.polytechnique.fr/informatique/INF423/uploads/Main/chap11-good.pdf>
- [W10] : Complexité **URL** : <https://docplayer.fr/15327086-Travail-d-etude-et-de-recherche-realisation-objet-d-un-mini-solveur-csp.html>

F

Bibliographie

- [1] : Reka Albert. « Scale-free networks in cell biology ». In : Journal of cell science 118.21 (2005), p. 4947–4957.
- [2] : Hamed Mohamed Babou. « Comparaison de réseaux biologiques ». Thèse de doct. Nantes, 2012.
- [3] : Jean-Guillaume FAGES « Exploitation de structures de graphe en programmation par contraintes ». Thèse de doct. Nantes, 2014.
- [4] : Xavier Lorca. « Contraintes de Partitionnement de Graphe ». <https://tel.archives-ouvertes.fr/tel-00484354/document>
- [5] : Livre : « Contraintes globales de partitionnement de graphe par des arbres ».
- [6] : Site : « https://fr.wikipedia.org/wiki/Graphe_planaire ».
- [7] : Cours programmation par contraintes mis à ma disposition par le tuteur : «RO06 Simulation et méthodes avancées d'optimisation Programmation par Contraintes».
- [8] : Rapport «Rapport de mission, fait par Monsieur Hamed MOHAMED BABOU, Enseignant-chercheur en collaboration avec l'équipe ROOT du laboratoire Informatique de l'Ecole Polytechnique de Tours».



Comptes rendus hebdomadaires

Compte rendu n°1 du 17/09/2018 Au 21/09/2018

Au cours de la semaine :

Lors de cette semaine, j'ai commencé par prendre connaissance des différents documents que vous m'avez envoyés. J'ai également fait quelques recherches sur le problème et sa complexité. J'ai commencé à étudier les deux algorithmes pour résoudre le problème SKEWGRAM à savoir : l'heuristique nommé également "ALGOH" et l'algorithme "ALGOBB" qui utilise la méthode branch-and-bound.

Ce qui est prévu :

- Rencontrer mes tuteurs pour avoir plus de précision sur le projets.
- Continuer à se documenter sur le projet.

Compte rendu n°2 du 24/09/2018 Au 28/09/2018

Au cours de la semaine :

- Lors de cette semaine, j'ai rencontré mon tuteur nous avons fait le point sur ce qui est demander.
- Documentation sur le sujet à travers la thèse de Hafedh Mohamed Babou.
- Dérouler le GETCOVERSET et l'algoH sur des exemples.

Ce qui est prévu :

- Prévoir un crénaux avec mon tuteur pour revoir si j'ai bien compris l'algoH.
- Réflexion sur la méthode agile à utiliser.

Au terme de cette semaine, quelques questions restent en suspens sur les algorithmes.

Compte rendu n°3 du 01/10/2018 Au 05/10/2018

Au cours de la semaine :

- Etudier les algorithmes à savoir l'alogH et l'algoBB.
- En parallèle j'ai commencé à rédiger le cahier de spécifications.

Ce qui est prévu :

- Continuer à dérouler les algorithmes sur des exemples.
- Continuer à rédiger le cahier de spécification.

Compte rendu n°4 du 08/10/2018 Au 12/10/2018Au cours de la semaine :

- Dérouler les algorithmes sur des exemples.
- Etudier la programmation par contraintes (PPC) ce qui va me permettre de modéliser et coder l'algorithme algoBB.

Ce qui est prévu :

- Continuer à comprendre la programmation par contraintes. Le but est d'écrire un modèle de PPC qui résout le problème.
- Continuer à rédiger le cahier de spécifications.

Compte rendu n°5 du 15/10/2018 Au 19/10/2018Au cours de la semaine :

- Rédaction du cahier de spécification.
- Etudier la thèse de Jean-Guillaume FAGES.

Ce qui est prévu :

- Rencontrer mon tuteur pour faire le point sur ce qui a été fait.
- Commencer à rédiger l'état de l'art qui comporte deux grandes parties : la première aborde les notions de graphes et la seconde la programmation par contraintes comment modéliser notre problème.

Compte rendu n°6 du 22/10/2018 Au 26/10/2018Au cours de la semaine :

- Rédaction de l'état de l'art (Notions de graphes, explication de l'algoBB)
- Etudier la thèse de Jean-Guillaume FAGES, le but est de comprendre la modélisation de graphe avec la PPC.
- Modélisation

Ce qui est prévu :

- Rencontrer Monsieur Jean-Yves RAMEL pour faire le point sur le cahier de spécification et l'état de l'art.

Compte rendu n°7 du 05/11/2018 Au 09/11/2018Au cours de la semaine :

- Continuer à étudier la thèse de Jean-Guillaume FAGES.
- Exercer à modéliser le problème avec la programmation par contraintes.

Ce qui est prévu :

- Envoyer la première version du cahier de spécification à valider avec le tuteur.
- Rencontrer monsieur Ramel pour faire le point sur le cahier des spécifications et l'état de l'art

Compte rendu n°8 du 12/11/2018 Au 16/11/2018Au cours de la semaine :

- Résumé la thèse de Jean-Guillaume FAGES, cette thèse aborde la façon dans on peut modéliser les graphes.

Ce qui est prévu :

- Proposer une modélisation.

Compte rendu n°9 du 19/11/2018 Au 23/11/2018

Au cours de la semaine :

- Rencontrer mon tuteur pour valider la première version du cahier de spécifications.
- Planifier le déroulement du projet.
- Ce documenter sur le solveur à utiliser.

Ce qui est prévu :

- Documentation sur les solveurs.
- Prendre rendez-vous avec mon tuteur.

Compte rendu n°10 du 26/11/2018 Au 30/11/2018

Au cours de la semaine :

- Rédaction de l'état de l'art et la partie planification.
- Je travaille également sur la modélisation du problème.
- Voir les différents solveurs qui existent et leurs avantages.

Ce qui est prévu :

- Continuer à travailler sur ses deux points.
- Proposer et valider une modélisation.

Compte rendu n°11 du 03/12/2018 Au 07/12/2018

Au cours de la semaine :

- Modélisation mathématique + validation.
- Rédaction rapport.

Compte rendu n°12 du 10/12/2018 Au 14/12/2018

Au cours de la semaine :

- Soutenance S9.

Compte rendu n°13 du 14/01/2019 Au 18/01/2019

Au cours de la semaine :

- Proposer une modélisation.
- Documentation et manipulation de la bibliothèque CHOCO

Ce qui est prévu :

- Installation de CHOCO et mise en place d'un environnement projet.
- Validation de la modélisation.

Compte rendu n°14 du 21/01/2019 Au 25/01/2019

Au cours de la semaine :

- Environnement CHOCO fonctionnelle
- Rédaction Manuel d'installation
- Validation de la modélisation avec mon encadrant.

Ce qui est prévu :

- Faire un rapport détaillé sur la modélisation

- Continuer la manipulation de la bibliothèque CHOCO
- Récupérer les instances

Compte rendu n°15 du 28/01/2019 Au 01/02/2019

Au cours de la semaine :

- J'ai modéliser mon projet (Diagramme de classe) + Validation
- Codage du parseur + test

Ce qui est prévu :

- Finir le codage du parseur et faire les tests unitaires
- Etablir le diagramme de classe

Compte rendu n°16 du 04/02/2019 Au 08/02/2019

Au cours de la semaine :

- Codage de la classe graphe et Graphe orienté et graphe non orienté

Ce qui est prévu :

- Génération de jeux de données
- Prendre rendez-vous pour valider l'avancement du projet

Compte rendu n°17 du 11/02/2019 Au 15/02/2019

Au cours de la semaine :

- Génération de jeux de données
- Codage de la classe DonneesModeleChoco (parser les instances et remplir les données)

Ce qui est prévu :

- Implémentation de la modélisation (Variable + Contraintes + Fonction objective)
- Prendre rendez-vous pour valider l'avancement du projet

Compte rendu n°18 du 25/02/2019 Au 01/03/2019

Au cours de la semaine :

- Implémentation de la partie plus long chemin (Variable, Fonction objective et contraintes)
- Prendre rendez-vous pour valider l'avancement du projet

Ce qui est prévu :

- Validation de la partie 1 de la modélisation (Plus long chemin)
- Commencer l'implémentation de la partie 2 (connexité)

Compte rendu n°19 du 04/03/2019 Au 08/03/2019

Au cours de la semaine :

- Implémentation de la partie 2 (connexité)
- Récupération des données à passer au solveur

Ce qui est prévu :

- Continuer l'implémentation de la partie connexité
- Prendre rendez-vous pour valider l'avancement du projet

Compte rendu n°20 du 11/03/2019 Au 15/03/2019Au cours de la semaine :

- Continuer le développement et l'implémentation de la modélisation
- Rédaction du manuel d'installation
- Manuel développeur

Ce qui est prévu :

- Analyse et conception
- Prendre rendez-vous pour valider l'avancement du projet

Compte rendu n°21 du 18/03/2019 Au 22/03/2019Au cours de la semaine :

- Correction des bugs liés au développement
- Validation de l'avancement du projet
- Structuration de certaines parties du code

Ce qui est prévu :

- Prendre rendez-vous pour valider l'avancement du projet
- Génération de la documentation dyoxgen
- Compléter le document de gestion projet(Sopra steria)

Compte rendu n°22 du 25/03/2019 Au 29/03/2019Au cours de la semaine :

- Préparation Soutenance
- Génération de la documentation Dyoxgen

Ce qui est prévu :

- Continuer la rédaction du rapport
- Préparation des rendus

Compte rendu n°23 du 01/04/2019 Au 05/04/2019Au cours de la semaine :

- Préparation du rapport + projet.
- Comparaison des résultats de la modélisation avec l'algoBB.

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

Morad SANBA

Encadrement : Emmanuel NÉRON et Ronan BOC-QUILLON



En collaboration avec Laboratoire
Informatique de Tours (EA630)

Objectifs

- Étudier le problème de comparaison des réseaux biologiques.
- Étudier les différents algorithmes (ALGOH et ALGOBB).
- Modélisation du problème à l'aide de la PPC et résolution avec le solveur CHOCO.



Technologie utilisé

Programmation par Contraintes (PPC)

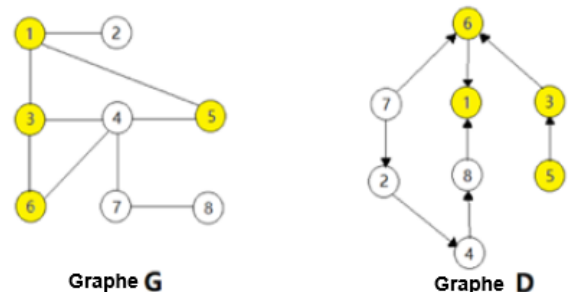
- Utilisation de la librairie Choco
- Librairie open source
- La modélisation proposée est adaptée à la PPC



Solver Choco

Résultats attendus

- La résolution du problème en trouvant le plus long chemin (D,G)-consistant.
- Affichage du temps d'exécution et la meilleur solution.



Graphe G

Graphe D

Exemple de chemin (D,G)-consistant



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

Morad SANBA

Encadrement : Emmanuel NÉRON et Ronan BOCQUILLON

Objectifs

- Étudier le problème de comparaison des réseaux biologiques.
- Étudier les différents algorithmes (ALGOH et ALGOBB).
- Modélisation du problème à l'aide de la PPC et résolution avec le solveur CHOCO.



Technologie utilisé

Programmation par Contraintes (PPC)

- Utilisation de la librairie Choco
- Librairie open source
- La modélisation proposée est adaptée à la PPC



Solver Choco

Résultats attendus

- La résolution du problème en trouvant le plus long chemin (D,G) -constant.
- Affichage du temps d'exécution et la meilleur solution.

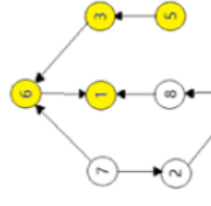
LI

En collaboration avec Laboratoire
Informatique de Tours (EA630)

Exemple de chemin (D,G) -constant



Graphe G



Graphe D

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

Résumé

Ce projet de fin d'études a permis de modéliser le problème **One-to-One SkewGraM** à l'aide de la programmation par contraintes (PPC). Cette modélisation va nous permettre de trouver le plus long chemin (D,G)-consistant passant par l'arc xy et le résoudre avec le solveur **Choco**. Ensuite nous allons comparer les résultats obtenus avec les résultats de l'**algoBB**. Ce rapport nous donne le contexte du projet, son déroulement, les diverses recherches faites, ses spécificités et son résultat final. Ainsi il donne les différents tests effectués. Au terme de ce projet j'ai pu découvrir le monde de la programmation par contraintes, découvrir la librairie Choco. J'ai pu découvrir le monde de la recherche, j'ai énormément appris à travers ce projet.

Mots-clés

AlgoBB, AlgoH, GETCOVERSET, ONE-TO-ONE SKEWGRAM, Algorithme, Réseaux biologiques, la programmation par contraintes, solveur Choco.

Abstract

This projet is about an modeled the **One-to-One SkewGraM** problem using constraint programming (PPC). This modeling will allow us to find the longest path (D,G)-consistant passing through the arc xy . This report gives us the context of the project, its progress, the various researches made, its specificities and its final result. So it gives the different tests done. At the end of this project I was able to discover the world of constraint programming, discover the Choco. I was able to discover the world of research, I learned a lot through this project.

Keywords

AlgoBB, AlgoH, GETCOVERSET, ONE-TO-ONE SKEWGRAM, Algorithms, Biological networks, constraint programming, solver Choco.

Entreprise

Laboratoire Informatique de Tours (EA630)



Tuteurs entreprise

Emmanuel NÉRON (Directeur de Polytech Tours)

Ronan BOCQUILLON (Enseignant chercheur)

Étudiant

Morad SANBA (DI5)

Tuteurs académiques

Emmanuel NÉRON

Ronan BOCQUILLON