

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

[polytech.univ-tours.fr](http://polytech.univ-tours.fr)

**Projet Recherche & Développement**

**2018-2019**

# **Rapport de projet de recherche et développement : Détection et reconnaissance avec des graphes**

**Tuteur académique**  
**Darwiche MOSTAFA**

**Étudiant**  
**Xavier PELTIER (DI5)**

4 avril 2019



# Liste des intervenants

Nom	Email	Qualité
Xavier PELTIER	<a href="mailto:xavier.peltier@etu.univ-tours.fr">xavier.peltier@etu.univ-tours.fr</a>	Étudiant DI5
Darwiche MOSTAFA	<a href="mailto:mostafa.darwiche@univ-tours.fr">mostafa.darwiche@univ-tours.fr</a>	Tuteur académique, Département Informatique



# Avertissement

Ce document a été rédigé par Xavier Peltier susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Darwiche Mostafa susnommé le tuteur académique.

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 du tuteur académique 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

Xavier Peltier, *Rapport de projet de recherche et développement : Détection et reconnaissance avec des graphes*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
  author={Peltier, Xavier},
  title={Rapport de projet de recherche et développement : Détection et reconnaissance
    avec des graphes},
  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
<b>1 Introduction</b>	<b>1</b>
1 Acteurs, enjeux et contexte .....	1
2 Objectifs et hypothèses.....	1
3 Bases méthodologiques .....	2
<b>2 Description générale</b>	<b>3</b>
1 Spécifications non fonctionnelles.....	3
1.1 Contraintes de développement .....	3
1.2 Contraintes de fonctionnement.....	3
1.3 Documentation et support .....	3
1.4 Tests.....	3
1.5 Contrôlabilité.....	4
1.6 Sécurité .....	4
1.7 Performances.....	4
2 Description des utilisateurs .....	4
<b>3 Etat de l'art/veille</b>	<b>6</b>
1 Les graphes .....	6

2	Le Graph Matching.....	6
3	Graph Edit Distance.....	7
4	Formulation PLNE du problème GED.....	8
4.1	Formulation F2.....	8
4.1.1	Données.....	8
4.1.2	Variables.....	8
4.1.3	Fonction objectif.....	9
4.1.4	Contraintes.....	9
4.2	Formulation F3.....	9
4.2.1	Données.....	9
4.2.2	Variables.....	9
4.2.3	Fonction objectif.....	9
4.2.4	Contraintes.....	10
5	Les différents algorithmes de résolution.....	10
5.1	BeamSearch.....	10
5.2	SBPBeam.....	11
5.3	IPFP.....	11
5.4	GNCCP.....	11
5.5	Graph Neural Networks.....	11
6	L'algorithme Local Branching.....	11
6.1	Définitions.....	12
6.2	Déroulement de l'algorithme.....	12
6.3	Les conditions d'arrêt.....	12
<b>4</b>	<b>Analyse et conception.....</b>	<b>13</b>
1	Description des interfaces externes du logiciel.....	13
1.1	Interfaces homme/machine.....	13
1.1.1	Entrées.....	13
1.1.2	Sorties.....	13
1.2	Interfaces logiciel/logiciel.....	13
2	Structure générale.....	14
2.1	Le package graph.....	14
2.2	Le package matrix.....	14
2.3	Le package ged.....	15
2.4	Le package locbra.....	15
2.5	Le package utils.....	15
3	Spécifications fonctionnelles.....	16
3.1	Le package graph.....	16
3.1.1	La classe GraphNode.....	16

3.1.2	La classe GraphEdge.....	18
3.1.3	La classe Graph.....	19
3.1.4	La classe GraphParser.....	20
3.2	Le package matrix.....	21
3.2.1	La classe AbstractMatrix (abstraite).....	21
3.2.2	La classe IloIntMatrix.....	24
3.3	Le package ged.....	24
3.3.1	La classe GEDProblem .....	24
3.3.2	La classe GEDSolution.....	25
3.3.3	L'interface IGEDSolver .....	26
3.3.4	La classe GEDIloNodeMatches.....	27
3.3.5	La classe NeighborhoodConstraintsManager .....	29
3.3.6	La classe UpperBoundConstraintsManager.....	29
3.4	Le package locbra .....	30
3.4.1	L'interface ILocBraAlgorithm.....	30
3.4.2	La classe LocBraAlgorithm.....	30
3.5	Le package utils .....	31
3.5.1	L'interface ILogger.....	31
3.5.2	La classe Log.....	32
4	Planification.....	32
4.1	Découpage des tâches.....	33
4.1.1	Compréhension du sujet et étude de la faisabilité.....	33
4.1.2	Etude du problème GED .....	34
4.1.3	Compréhension de la formulation F2 du problème GED .....	34
4.1.4	Compréhension de la formulation F3 du problème GED .....	34
4.1.5	Modélisation du projet .....	34
4.1.6	Rédaction du rapport de milieu de projet.....	34
4.1.7	Préparation pour la soutenance de milieu de projet .....	34
4.1.8	Réalisation des tests unitaires .....	35
4.1.9	Réalisation de la classe Graph.....	35
4.1.10	Réalisation d'un parseur de graphes en fichiers XML.....	35
4.1.11	Intégration de CPLEX.....	35
4.1.12	Implémentation d'un solveur de la formulation F2 du problème GED.....	35
4.1.13	Implémentation de l'algorithme LocBra .....	35
4.1.14	Implémentation de l'interface utilisateur.....	36
4.1.15	Réalisation de benchmarks.....	36
4.1.16	Développement de l'algorithme multithread .....	36
4.1.17	Rédaction du rapport final .....	36
4.1.18	Préparation pour la soutenance finale .....	36

<b>5</b>	<b>Mise en oeuvre</b>	<b>37</b>
1	Technologies utilisées .....	37
2	Implémentation .....	37
2.1	Les briques de base .....	37
2.1.1	Matrice .....	37
2.1.2	Graphe .....	37
2.2	Implémentation des formulations F2 et F3 avec IloCplex.....	38
2.3	Implémentation de l'algorithme Local Branching non parallélisé .....	38
2.4	Implémentation de l'algorithme Local Branching parallélisé .....	38
2.5	Tests.....	39
2.5.1	Tests unitaires .....	39
2.5.2	Tests fonctionnels .....	39
<b>6</b>	<b>Bilan et conclusion</b>	<b>40</b>
1	Point sur l'avancement .....	40
2	Bilan auto-critique sur la gestion du projet .....	40
2.1	Points positifs.....	40
2.2	Points négatifs.....	40
3	Conclusion .....	40



# Table des figures

## 2 Description générale

1	Tableau des caracteristiques des utilisateurs .....	4
2	Diagramme des cas d'utilisation .....	5

## 3 Etat de l'art/veille

1	Graph .....	6
2	SubgraphMatching .....	7
3	GraphEditDistance .....	7
4	BeamSearchAlgorithm.....	10
5	NeuralNetwork .....	11

## 4 Analyse et conception

1	Package graph .....	14
2	Package matrix.....	15
3	Package ged.....	16
4	Package ged.f2 .....	17
5	Package ged.f3 .....	17
6	Package locbra .....	18
7	Package utils .....	18
8	Le modèle en cascade .....	33
9	Diagramme de Gantt .....	33

## 5 Mise en oeuvre

1	LocBraMultithread .....	38
---	-------------------------	----

# 1

## Introduction

Je tiens tout d'abord à remercier toutes les personnes qui m'ont aidé lors de ce projet, ainsi que pour la rédaction du rapport :

Merci à mon tuteur Mr Darwiche Mostafa qui m'a aidé à comprendre et analyser le sujet.

Merci à Mr Jean-Yves RAMEL pour ses conseils sur la rédaction de ce rapport.

Ce rapport contient les spécifications du projet de recherche et de développement "Détection et reconnaissance avec des graphes".

J'ai réalisé ce projet dans le cadre de ma 5ème année d'étude à Polytech Tours, sous la tutelle de Mr Darwiche Mustafa, auteur de l'algorithme LocBra.

## 1 Acteurs, enjeux et contexte

Un graphe est un modèle mathématique abstrait permettant de décrire un réseau de points (aussi appelés noeuds ou sommets) par des liens (aussi appelés arêtes). Ce modèle générique permet de modéliser beaucoup de problèmes concrets, et donc tout algorithme lié aux graphes a le potentiel d'être utile dans une multitude d'applications.

Le problème GED (Graph Edit Distance) est un problème dont la résolution nous donne une valeur de similarité entre deux graphes différents. Cela permet entre autres de mesurer la correspondance entre deux graphes différents.

Appliqué à des problèmes concrets, l'exploitation de ce résultat permet par exemple de reconnaître des formes dans une image (en caractérisant ces formes par des graphes), ou le type de celle-ci.

C'est donc très utilisé dans les domaines en relation avec des données visuelles, tels que la reconnaissance de texte manuscrit ou d'empreintes digitales. C'est notamment utilisé en médecine, afin de reconnaître automatiquement différentes maladies.

## 2 Objectifs et hypothèses

Le but de ce projet de recherche et développement est tout d'abord d'implémenter les algorithmes LocBra et VPLS afin de les étudier sur différents problèmes.

Ce projet comparera les différents algorithmes existants aux algorithmes LocBra et VPLS, et tentera de tirer des conclusions sur les valeurs optimales des paramètres à envoyer à l'algorithme, en fonction du problème donné en entrée.

Aussi, il m'est demandé de trouver une méthode d'initialisation de ces algorithmes qui soit plus efficace que celle utilisée actuellement. En effet, la méthode actuelle n'est autre qu'une itération complète de l'algorithme, ce qui n'est pas forcément le choix le plus optimisé. De plus, les 3 minutes requises par une itération rendent la manipulation de l'algorithme difficile, car il faut attendre 3 minutes pour avoir le premier résultat.

Enfin, une application java permettant à un utilisateur de tester l'algorithme LocBra sur des graphes sera développée.

On estime qu'il est possible d'approcher une solution correcte à partir d'un algorithme simple. Celui-ci n'a pas pour but de trouver une bonne solution, mais seulement d'aiguiller au mieux l'algorithme en trouvant une position de départ facilement améliorable.

On estime aussi que plus le score renvoyé par la position de départ est élevé, plus celle-ci est intéressante.

### 3 Bases méthodologiques

Afin de m'aider à réaliser ce projet, j'utilise les outils suivants :

1. Astah UML : Un programme permettant de créer différents diagrammes UML et proposant une version gratuite.
2. Trello : Un outil gratuit permettant de gérer les différentes tâches d'un projet en proposant une vue simple et intuitive.

Ce projet se déroulera en quatre étapes :

1. Compréhension et analyse du problème
2. Conception de la solution
3. Réalisation d'un bilan intermédiaire
4. Réalisation de la solution
5. Réalisation d'un bilan final

# 2

## Description générale

### 1 Spécifications non fonctionnelles

#### 1.1 Contraintes de développement

- Langage de programmation : Java / Kotlin
- Moteur de production : Gradle
- Gestionnaire de version : Git / Github
- IDE : IntelliJ
- Bibliothèques : Cplex

#### 1.2 Contraintes de fonctionnement

- Le programme doit pouvoir être lancé depuis un poste de travail de type PC.
- Le programme tournera sur la Java Virtual Machine, l'installation de la JVM est donc requise pour pouvoir l'utiliser.
- Le programme a dépend de la bibliothèque Cplex, une installation de Cplex est donc requise.

#### 1.3 Documentation et support

- Le code source sera clairement et commenté.
- Le programme doit être simple à mettre en place.
- Un manuel d'utilisation comportant des exemples d'utilisations sera disponible.

#### 1.4 Tests

- Les classes de base (matrice, graphe..) bénéficieront de tests unitaires.
- Les algorithmes implémentés seront testés par des tests fonctionnels.

## 1.5 Contrôlabilité

Le status de l'algorithme sera régulièrement affiché sur le terminal, et un fichier de log pourra être spécifié au lancement afin de sauvegarder les traces de l'exécutions du programme.

## 1.6 Sécurité

L'application est destinée à des utilisateurs avertis, et celle ci ne communique par réseau. Il n'y a pas de contrainte de sécurité.

## 1.7 Performances

La performance de l'application finale n'est pas un élément majeur, et ne sera pris en compte qu'à la toute fin du projet. Cependant il n'y aura pas de grosse faute de code impactant la performance, le programme sera donc aussi performant que ce dont on attend d'une application java.

# 2 Description des utilisateurs

	Connaissance de l'informatique	Expérience de l'application	Type de l'utilisateur	Droits d'accès
Réponse	Peu	Non	Général et régulier.	Tout droit
Commentaire	L'utilisateur doit savoir entrer des commandes dans un terminal	Le logiciel propose un interface très simple.	Le logiciel est principalement destiné aux personnes connaissant le problème GED, mais peut être utilisé par tout le monde	Les utilisateurs peuvent faire ce qu'ils veulent.

**Figure 1** – Tableau des caracteristiques des utilisateurs

Le programme n'a pas pour but d'être utilisé tel quel en production, dans des application réelles. Cependant, il pourra être intéressant pour des chercheurs voulant approfondir ou se renseigner sur le sujet, de manipuler l'exécutable rendu avec différents paramètres, afin de comprendre les algorithmes implémentés, ou de les tester sur différents problèmes et avec différents paramètres. L'utilisateur aura donc un droit d'accès et de manipulation total du programme.

Il devra être capable d'effectuer des commandes basiques sur Windows, pour lancer le programme, et pour le manipuler depuis un terminal.

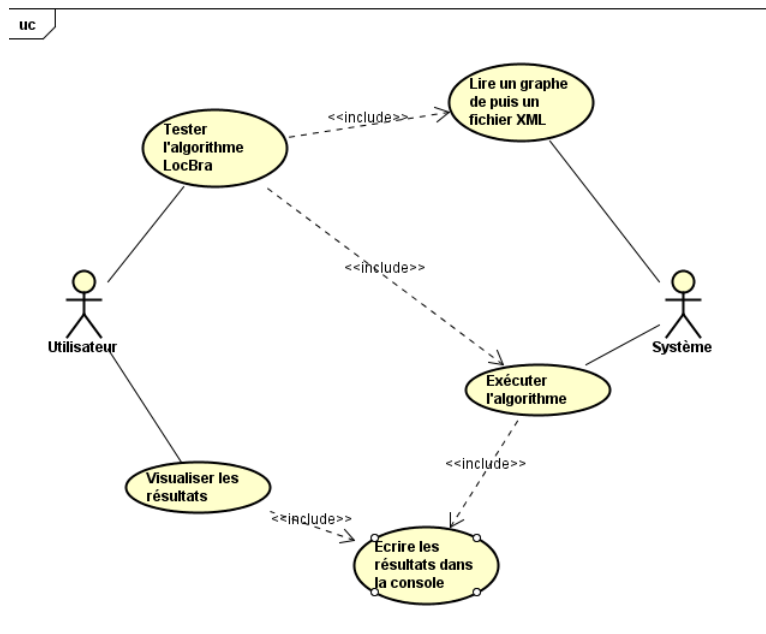


Figure 2 – Diagramme des cas d'utilisation

# 3

## Etat de l'art/veille

Au cours de ce chapitre, nous allons tout d'abord nous intéresser au problème de graph matching, puis nous allons nous concentrer sur le problème GED et ses algorithmes de résolution. Enfin nous allons détailler l'algorithme LocBra sur lequel je vais travailler plus tard.

### 1 Les graphes

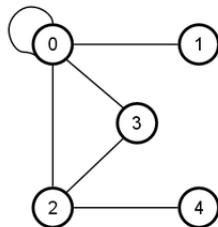


Figure 1 – Graph

Un graphe  $G = (V, E)$  est une structure mathématique composée de noeuds (aussi appelés sommets ou points) liés deux à deux par des arêtes.

Un graphe peut être orienté (ses arêtes que l'on nomme arcs lient deux noeud dans un seul sens) ou non orienté.

Bien que le modèle ait été déjà utilisé auparavant, le mot "graph" pour le désigner a été introduit en 1878 par un mathématicien anglais : James Joseph Sylvester.

Ce modèle générique permet de modéliser beaucoup de problèmes concrets, et donc il en va de même pour les algorithmes appliqués aux graphes.

### 2 Le Graph Matching

Le graph matching est le fait de trouver des similarités entre des graphes.

Ce problème est très étudié car c'est un problème générique qui s'applique à de nombreux problèmes concrets tels que la reconnaissance d'images qui est un sujet de recherche qui a pris beaucoup d'importance dernièrement.

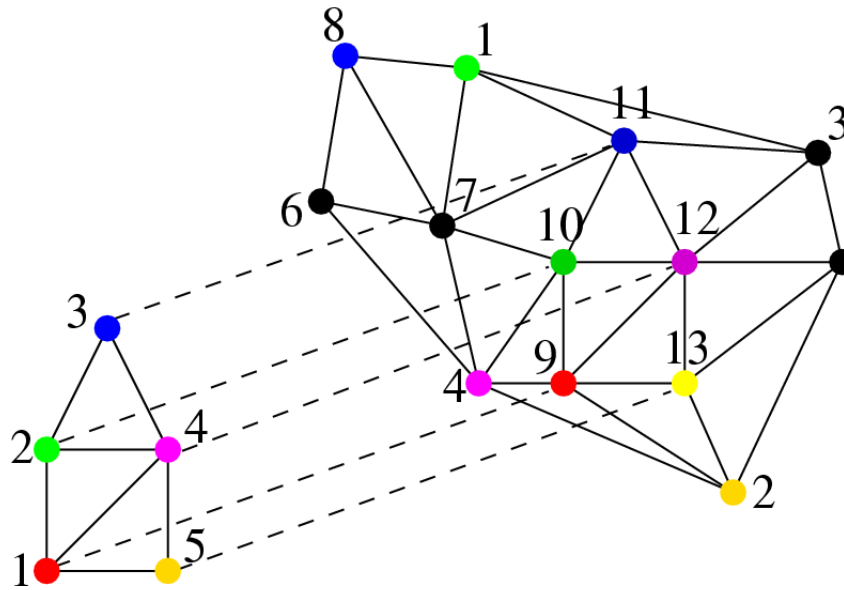


Figure 2 – SubgraphMatching

Il existe deux branches de ce problème :

1. Exact graph matching (graph isomorphism problem) : On tente de savoir si deux graphes (ou parties d'un graphe) sont isomorphes.
2. Inexact graph matching : On tente de savoir si deux graphes (ou parties d'un graphe) sont similaires.

La seconde branche est plus générale et est plus intéressante pour la résolution de problèmes concrets tels que la reconnaissance d'images, pour la suite, nous n'allons nous intéresser que problème d'inexact graph matching.

La résolution de ce problème peut se faire de différentes manières, mais la méthode la plus courante est de calculer une mesure de similarité entre deux graphes afin de trouver la meilleure paire, et l'une des mesures inventées est la Graph edit distance.

### 3 Graph Edit Distance

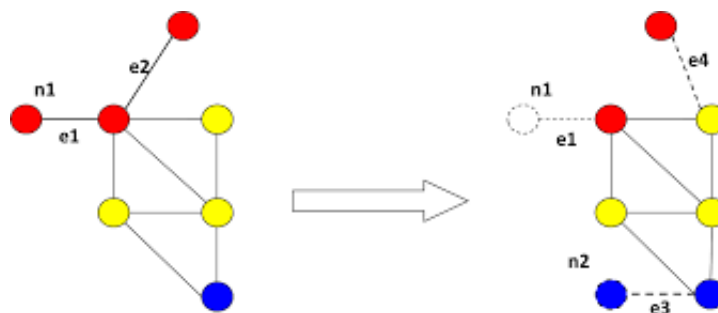


Figure 3 – GraphEditDistance

La GED (Graph Edit Distance) est une valeur correspondant à la similarité entre deux graphes. L'idée de base de l'Edit Distance est de calculer le coût de la transformation d'un objet A en objet B.

Pour cela, on définit différentes opérations élémentaires de transformation de graphes, et on leur donne un coût. Ensuite, on cherche à trouver la succession d'opérations élémentaires permettant



de transformer le graphe A en graphe B, et dont la somme des coûts est la plus faible.

Ce calcul peut être mis en relation avec d'autres types de calcul de distance utilisant la même approche :

1. La String Edit Distance (SED) : Une mesure de la similarité entre deux chaînes de caractères
2. La Levenshtein distance : Une mesure de la similarité entre deux séquences.

Aussi, on peut voir la Graph Edit Distance comme une généralisation de la Tree Edit Distance qui mesure la similarité entre deux arbres.

Les opérations élémentaires de transformation de graphe sont les suivantes :

1. Insertion d'un noeud : On ajoute un noeud au graphe
2. Suppression d'un noeud : On supprime un noeud du graphe
3. Substitution d'un noeud : On change un noeud du graphe.
4. Insertion d'une arête : On ajoute une arête au graphe
5. Suppression d'une arête : On supprime une arête du graphe
6. Substitution d'une arête : On change une arête du graphe.

Les coûts de ces opérations peuvent varier en fonction du problème et de l'algorithme de résolution utilisé, mais en règle général, on attribue un coût élevé aux opérations d'insertion et de suppression, et un coût plus faible aux opérations de substitution.

## 4 Formulation PLNE du problème GED

Afin de pouvoir résoudre le problème à l'aide de solveurs classiques, on cherche à le modéliser par un set de données, un set de variables, un set de contraintes et une fonction objectif.

Il existe plusieurs formulations mathématiques possibles pour ce problème, mais la formulation F2 (proposée par Lerouge et al.) reste à ce jour la plus simple, car elle est une version compacte de la formulation F1 (proposé par les mêmes auteurs) sur laquelle elle est basée, la fonction objectif a été reformulée de façon à réduire le nombre de variables et de contraintes du problème.

### 4.1 Formulation F2

#### 4.1.1 Données

Deux graphes  $G = (V, E)$  et  $G' = (V', E')$

Pour toute paire  $(i, k)$  dans  $(V, V')$ ,  $c_v(i, k)$  le coût de la substitution du noeud  $i$  par le noeud  $k$ .

Pour tout noeud  $i$  dans  $V$ ,  $c_v(i, e)$  le coût de la suppression du noeud.

Pour tout noeud  $k$  dans  $V'$ ,  $c_v(e, k)$  le coût de l'addition du noeud.

Pour toute paire  $(i, j)$  de  $(E, E')$ ,  $c_e(i, j)$  le coût de la substitution de l'arête  $i$  par l'arête.

Pour toute arête  $i$  dans  $E$ ,  $c_e(i, e)$  le coût de la suppression de l'arête.

Pour toute arête  $k$  dans  $E'$ ,  $c_e(e, k)$  le coût de l'addition de l'arête.

#### 4.1.2 Variables

Pour toute paire  $(i, k)$  dans  $(V, V')$ ,  $x_{i,k} = 1$  lorsque les noeuds  $i$  et  $j$  sont matchés, 0 sinon.

Pour toute paire  $(ij, kl)$  dans  $(E, E')$ ,  $y_{ij,kl} = 1$  lorsque les arêtes  $ij$  et  $kl$  sont matchées, 0 sinon.

### 4.1.3 Fonction objectif

La fonction objectif à minimiser est donc la suivante :

$$\min x, y \sum_{i \in V} \sum_{j \in V'} (c_v(i, k) - c_v(i, e) - c_v(e, k)) * x_{i, k} + \sum_{(i, j) \in E} \sum_{(k, l) \in E'} (c_e(ij, kl) - c_e(ij, e) - c_e(e, kl)) * y_{ij, kl} + \gamma \quad (1)$$

$$\text{avec } \gamma = \sum_{i \in V} (c_v(i, e)) + \sum_{k \in V'} (c_v(e, k)) + \sum_{(i, j) \in E} (c_e(ij, e)) + \sum_{(k, l) \in E'} (c_e(e, kl))$$

### 4.1.4 Contraintes

Pour tout noeud i de V,

$$\sum_{k \in V'} x_{i, k} \leq 1 \quad (2)$$

Pour tout noeud k de V',

$$\sum_{i \in V} x_{i, k} \leq 1 \quad (3)$$

Pour toute paire (k, ij) de (V', E),

$$\sum_{(kl) \in E'} y_{ij, kl} \leq x_{i, k} + x_{j, k} \quad (4)$$

## 4.2 Formulation F3

Lors de la réalisation de ce projet, je vais implémenter et utiliser la formulation F3, qui est une nouvelle formulation inspirée par F2 et qui en est très similaire.

Cette formulation est plus intéressante dans le sens où le set de contraintes est indépendant du nombre d'arêtes dans le graphe.

### 4.2.1 Données

Les mêmes données que pour la formulation F2

### 4.2.2 Variables

Pour toute paire (i, k) dans (V, V'),  $x_{i, k} = 1$  lorsque les noeuds i et j sont matchés, 0 sinon.

Pour toute paire (ij, kl) dans (E, E' ∪ E'),  $y_{ij, kl} = 1$  lorsque les arêtes ij et kl sont matchées, 0 sinon.

### 4.2.3 Fonction objectif

La fonction objectif est la même que pour la formulation F2, en incluant le changement dans les variables :

$$\min x, y \sum_{i \in V} \sum_{j \in V'} (c_v(i, k) - c_v(i, e) - c_v(e, k)) * x_{i, k} + \sum_{(i, j) \in E} \sum_{(k, l) \in E' \cup E'} (c_e(ij, kl) - c_e(ij, e) - c_e(e, kl)) * y_{ij, kl} + \gamma \quad (5)$$

#### 4.2.4 Contraintes

Seule la troisième contrainte est modifiée :

Pour toute paire (i, k) de (V, V'),

$$\sum_{(ij) \in E} \sum_{(kl) \in E'} y_{ij,kl} \leq d_{i,k} * x_{i,k} \quad (6)$$

Avec  $d_{i,k} = \min(\text{degree}(i), \text{degree}(k))$

Avec pour tout noeud i,  $\text{degree}(i)$  = Le nombre d'arêtes incidentes à i

## 5 Les différents algorithmes de résolution

Il existe différents algorithmes plus ou moins performants permettant de résoudre les problèmes GED et GEDenA

### 5.1 BeamSearch

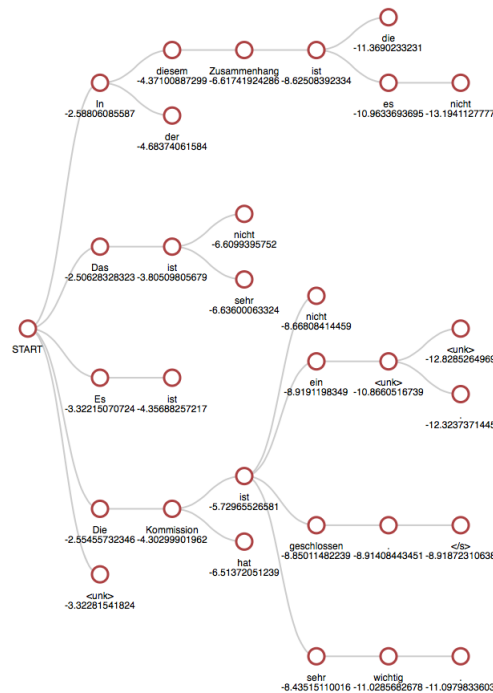


Figure 4 – BeamSearchAlgorithm

L'algorithme BeamSearch parcourt l'arbre des possibilités de successions d'opérations élémentaires permettant, en partant du graphe A, d'obtenir le graphe B. On ne prend à chaque fois qu'un nombre définis d'enfants dans l'arbre des possibilités (d'où le nom de l'algorithme, qui ne "voit" qu'un faisceau de l'arbre) Enfin, on choisit la feuille ayant le plus petit coût pour résoudre le problème GED.

## 5.2 SBPBeam

Cet algorithme est une combinaison des algorithmes BPGM et BeamSearch. On associe tout d'abord les points du graphe de départ à ceux du graphe d'arrivée grâce à BPGM. Puis on affine le résultat à l'aide de BeamSearch.

## 5.3 IPFP

Cet algorithme décrit le problème comme un cas particulier d'un QAP. On approxime la fonction objectif par son développement de Taylor au premier ordre, et autour d'une solution initiale  $x_0$ . On dérive ensuite cette fonction en  $x_0$  pour avoir la pente et donc le changement à apporter aux variables pour la prochaine itération. (Un peu comme la rétropropagation d'un gradient)

## 5.4 GNCCP

Cet algorithme reprend le principe de IPFP, en utilisant une version modifiée de la fonction objectif (en ajoutant un paramètre qui décroît au fil du temps) de sorte à la rendre totalement convexe ou concave.

## 5.5 Graph Neural Networks

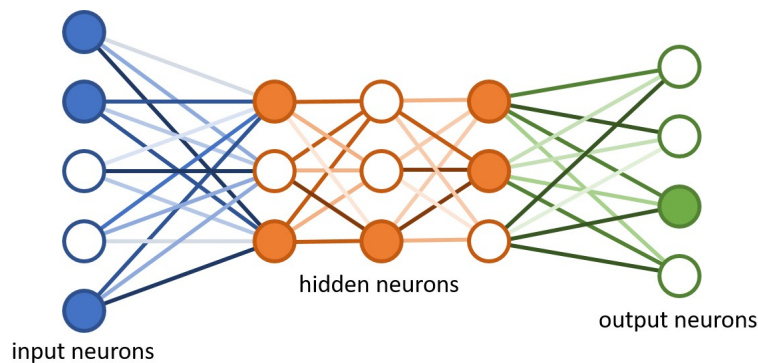


Figure 5 – *NeuralNetwork*

Des recherches récentes se sont penchées sur l'utilisation de réseaux neuronnaires pour l'évaluation de la Graph Edit Distance.

Les réseaux utilisés pour ce problème sont majoritairement des réseaux convolutionnels, et ceux-ci tentent d'"apprendre" une fonction prenant deux graphes en entrée, et calculant la distance entre ceux-ci.

Ce sujet est très récent et nous n'allons pas l'étudier au cours de ce projet.

## 6 L'algorithme Local Branching

L'algorithme Local Branching est un algorithme élaboré par Mr Mostafa Darwiche, Mr Romain Raveaux, Mr Donatello Conte, Mr Vincent T'Kindt à l'Université de Tours.

Cet algorithme heuristique tente de trouver la meilleure solution en réduisant le problème par un problème de programmation linéaire en nombres entiers (PLNE) dont l'espace des solutions est restreint et mis à jour au cours de l'algorithme.

## 6.1 Définitions

L'algorithme LocBra (Local Branching) introduit les notions suivantes :

1. Neighborhood definition : On choisit les voisins proche de la solution initiale, en prenant en compte la distance de Hamming.

$$N(x^p, k) \quad (7)$$

2. Intensification : On utilise CPLEX afin de rechercher de meilleures solutions dans la zone définie précédemment.
3. Complementary intensification : Si CPLEX n'arrive pas à trouver de solution réalisable après un temps de calcul donné, on considère que le voisinage est trop large, et on le réduit

$$N(x^p, k) - > N(x^p, k/2) \quad (8)$$

4. Diversification : Lorsque l'on n'arrive pas à obtenir une solution améliorée après la Complementary intensification, on ajoute une contrainte pour que le solveur cherche des solutions à un endroit différent.

## 6.2 Déroulement de l'algorithme

L'algorithme se déroule de la façon suivante :

1. Tout d'abord, une solution initiale correcte est calculée.
2. Puis, tant qu'aucune des conditions d'arrêt n'est respectée :
3. On recherche une solution à l'aide d'un solveur, et de la dernière solution trouvée
4. Si on a trouvé une autre solution : On tente de l'améliorer avec une intensification
5. Si on retombe sur la même solution, on fait une diversification
6. Si on trouve une solution incorrecte, on fait une diversification
7. Sinon, on diversifie ou on intensifie, en fonction du contexte

## 6.3 Les conditions d'arrêt

Il y a trois conditions d'arrêt :

1. Le temps d'exécution dépasse le temps limite passé en entrée dans le programme.
2. Le nombre total de diversifications effectuées dépasse un seuil passé en entrée dans le programme : On considère que l'algorithme a cherché à un nombre d'endroit assez conséquent pour s'arrêter.
3. Le nombre de diversifications consécutives dépasse un seuil passé en entrée dans le programme : On considère que l'algorithme ne trouve absolument rien (il ne cesse de changer de point de recherche)

# 4

## Analyse et conception

### 1 Description des interfaces externes du logiciel

#### 1.1 Interfaces homme/machine

##### 1.1.1 Entrées

L'utilisateur pourra interagir avec le logiciel par le biais d'une interface en ligne de commande. Il aura la possibilité de choisir les données d'entrées :

1. Les fichiers XML contenant les graphes à comparer
2. Le fichier dans lequel l'algorithme écrira le résultat

L'utilisateur pourra aussi modifier les meta-paramètres de l'algorithme LocBra.

##### 1.1.2 Sorties

L'utilisateur aura en sortie :

1. Les logs de l'exécution de l'algorithme choisi
2. Le résultat de l'algorithme : La valeur de GED calculée et la solution trouvée, si une solution a été trouvée

Si un fichier de sortie est spécifié en paramètre d'entrée, les résultats seront écrits dans celui-ci. Sinon, ils seront écrits dans la sortie standard.

#### 1.2 Interfaces logiciel/logiciel

Le logiciel devra utiliser un solveur de problèmes linéaires en nombres entiers : Le solveur CPLEX a été choisi pour cela. Une interface contenant les fonctions de manipulation d'un problème utilisées par l'algorithme LocBra.

Le sous-problème défini par l'algorithme LocBra est créé par le constructeur de la classe implémentant cet interface. Ensuite, les quatre fonctions utilisées par l'algorithme LocBra sont directement implémentées.

L'utilisateur aura en sortie le résultat de l'algorithme ainsi que les informations intéressantes sur l'exécution de celui-ci.

## 2 Structure générale

Le projet sera composé de 5 packages

### 2.1 Le package graph

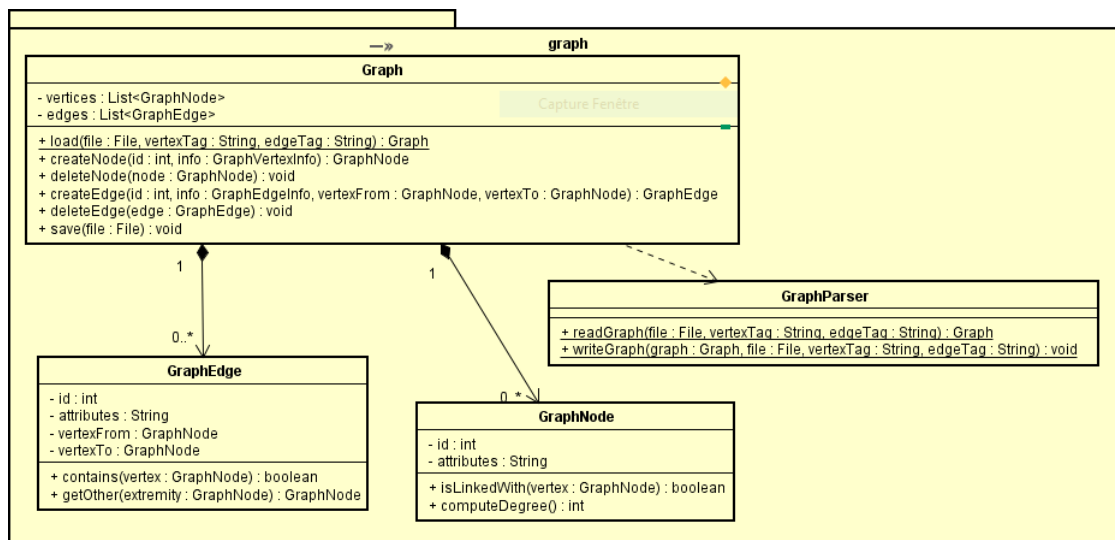


Figure 1 – Package graph

Ce package sera composé de :

- La classe GraphNode représentant un noeud de graphe.
- La classe GraphEdge représentant une arête de graphe.
- Le singleton GraphParser, contenant les méthodes permettant de lire et d'écrire un graphe depuis un fichier gxl.
- La class Graph représentant un graphe.

### 2.2 Le package matrix

Ce package sera composé de :

- La classe AbstractMatrix contenant une implementation générique de matrice.
- La classe IntMatrix implémentant AbstractMatrix pour le type Int.
- La classe DoubleMatrix implémentant AbstractMatrix pour le type Double.
- La classe IloIntMatrix implémentant AbstractMatrix pour le type IloIntVar.

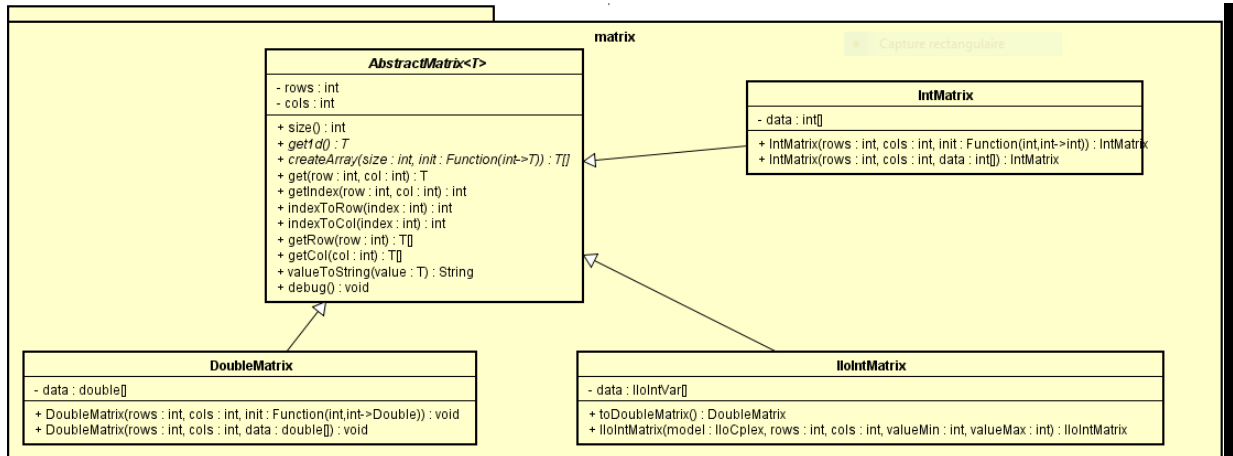


Figure 2 – Package matrix

### 2.3 Le package ged

Ce package sera composé de :

- L'interface GEDSolver, un solver de GED générique.
- La class GEDProblem représentant un problème GED.
- La classe GEDSolution représentant une solution d'un problème GED.
- La classe GEDIloNodeMatches représentant les matches de noeuds d'un problème GED, avec des valeurs Cplex.
- La classe NeighborhoodConstraintsManager, permettant la manipulation des contraintes de voisinage.
- La classe UpperBoundConstraintsManager, permettant la manipulation des contraintes d'upper bound.
- Le package f2 proposant une implémentation d'un GED solver avec la formulation f2.
- Le package f3 proposant une implémentation d'un GED solver avec la formulation f3.

### 2.4 Le package locbra

Ce package sera composé de :

- L'interface ILocBraAlgorithm, une algorithmme LocBra générique
- La classe LocBraAlgorithm, proposant une implémentation avec un seul thread.
- La classe MultithreadedLocBraAlgorithm, proposant une implémentation avec plusieurs threads.

### 2.5 Le package utils

Ce package sera composé de :

- L'interface ILogger, un logger générique
- La classe Log comportant des fonctions utiles pour déboguer le programme.



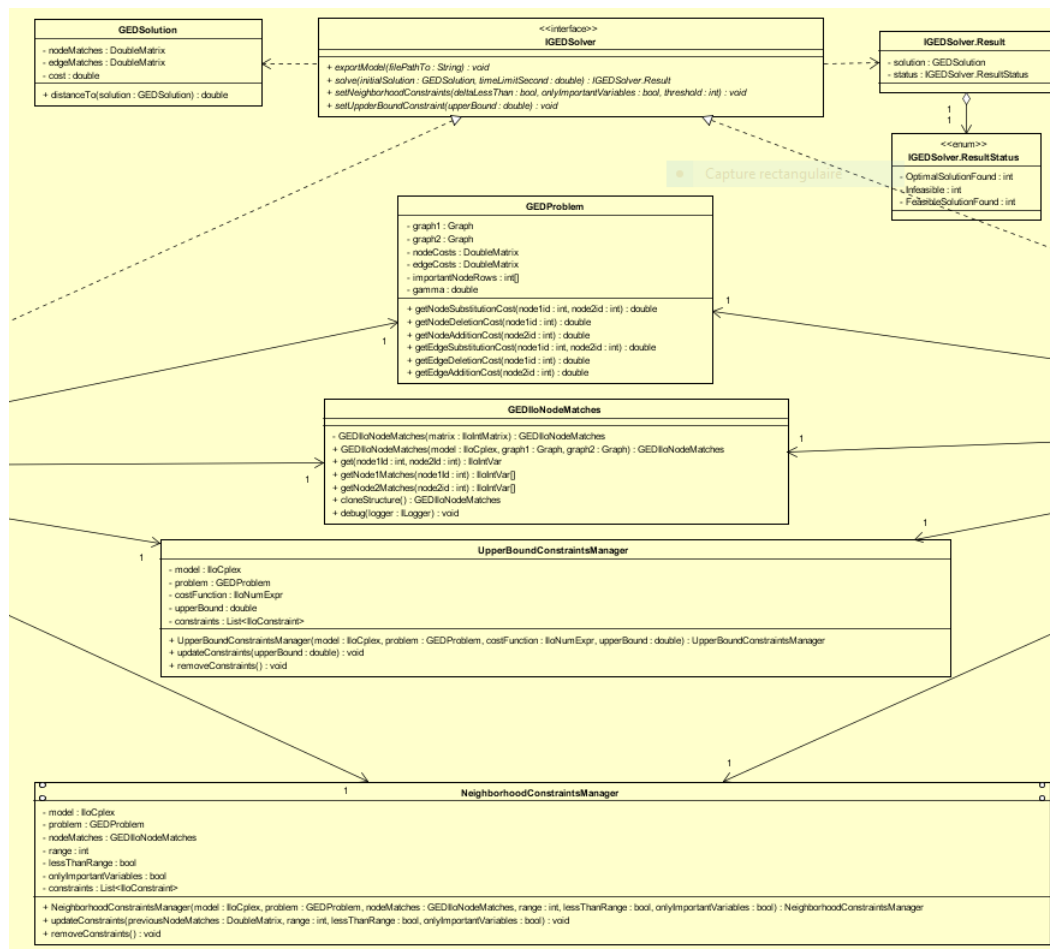


Figure 3 – Package ged

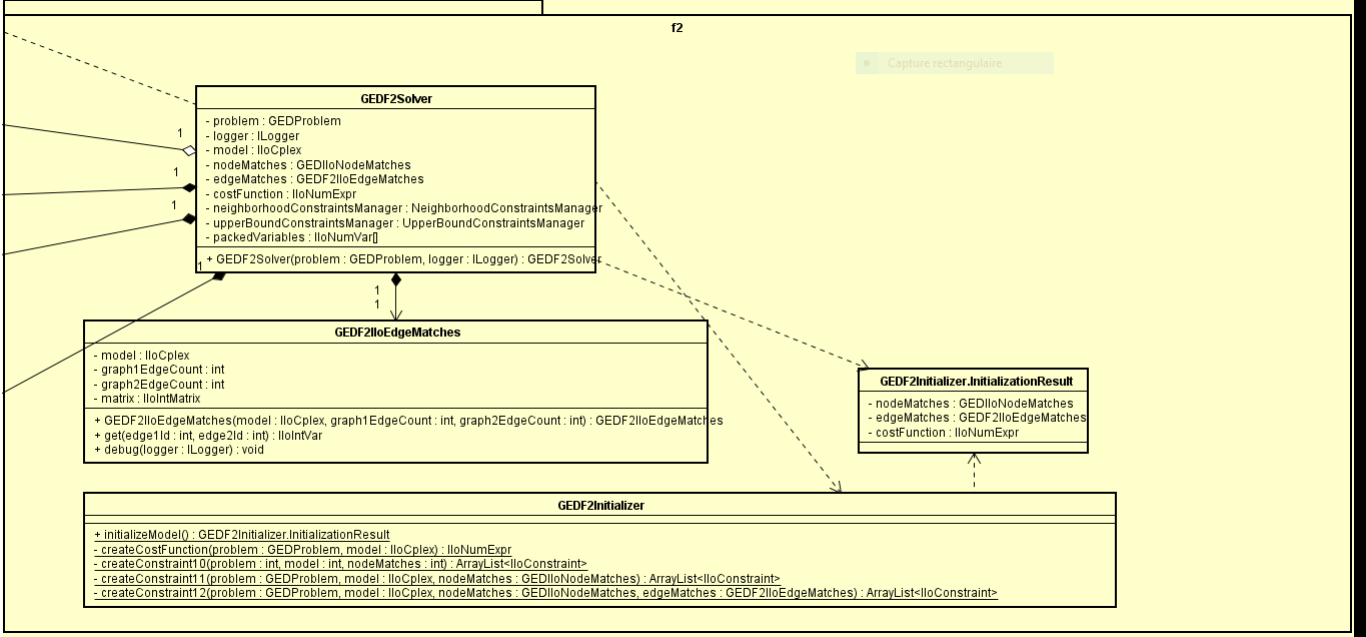
### 3 Spécifications fonctionnelles

#### 3.1 Le package graph

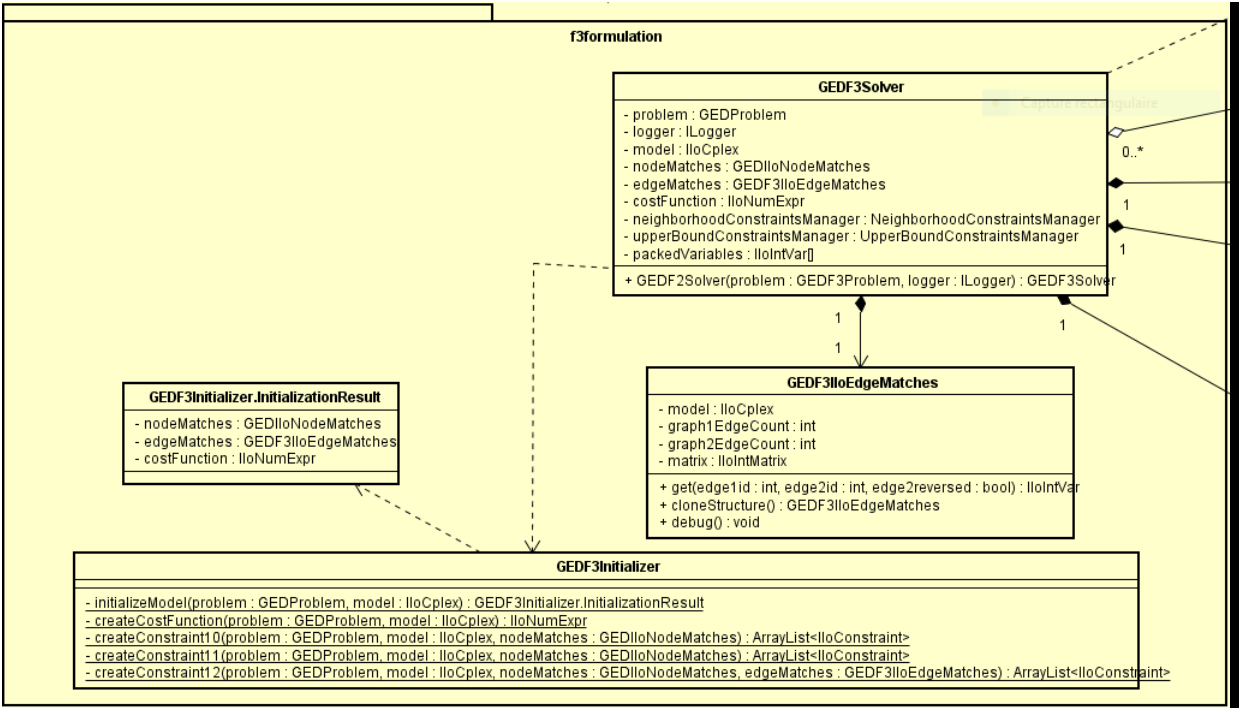
Le package graph offre un objet permettant de manipuler simplement un graph non orienté : Graph. Celui-ci est composé d'une liste de GraphNode pour les noeuds, et d'une liste de GraphEdge pour les arrêtes.

##### 3.1.1 La classe GraphNode

Nom de fonction	isLinkedWith
Description	Cette fonction détermine si le GraphNode est lié à un autre GraphNode par le biais d'un GraphEdge.
Pré-conditions	-
Post-conditions	-
Entrées	GraphNode node : Le noeud à comparer
Sortie	bool : Vrai si les deux noeuds sont liés, faux sinon



**Figure 4** – *Package ged.f2*



**Figure 5** – *Package ged.f3*

Nom de fonction	hasSameAttributes
Description	Cette fonction détermine si le GraphNode possède les mêmes attributs (et les mêmes valeurs pour ces attributs) qu'un autre GraphNode
Pré-conditions	-
Post-conditions	-
Entrées	GraphNode node : Le noeud à compare
Sortie	bool : Vrai si les deux noeuds possèdent les mêmes attributs, faux sinon

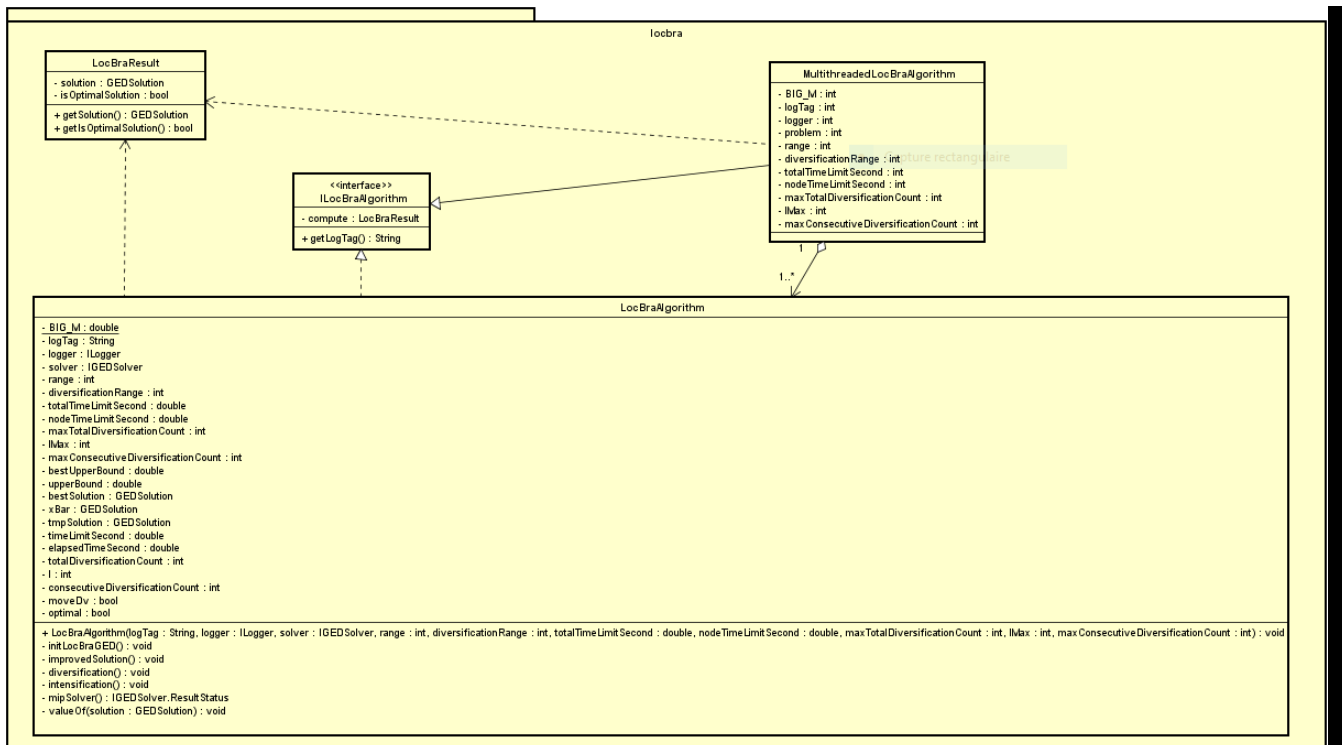


Figure 6 – Package locbra

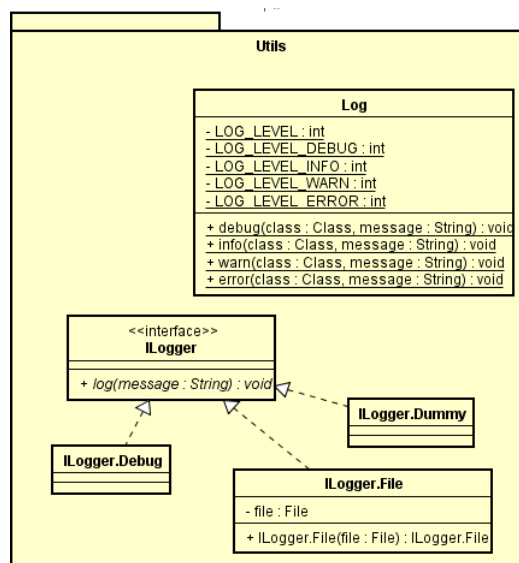


Figure 7 – Package utils

### 3.1.2 La classe GraphEdge

Une arête est représentée par son id dans le graphe, par les deux noeuds qu'elle relie et par des attributs qui varient en fonction du problème.

Nom de fonction	contains
Description	Cette fonction détermine si le GraphEdge est liée à un noeud.
Pré-conditions	-
Post-conditions	-
Entrées	GraphNode node : Le noeud à regarder.
Sortie	bool : Vrai si l'arrête est liée au noeud, faux sinon.

Nom de fonction	getOther
Description	Cette fonction retourne l'autre extrémité de l'arrête si le noeud passé en entrée est lié à celle-ci.
Pré-conditions	-
Post-conditions	Le noeud retourné est l'un des noeuds de l'arête
Entrées	GraphNode node : Le noeud à regarder.
Sortie	GraphNode : Si le noeud est lié à l'arrête, retourne l'autre extrémité de l'arrête, sinon retourne null.

### 3.1.3 La classe Graph

Un objet Graph est composé d'une liste d'objets GraphEdge représentant les arrêtes, et d'une liste d'objets GraphNode représentant les noeuds du graphe. Cette classe à été créée dans l'idée de prévenir au maximum les possibilités d'erreurs : L'on ne peut ajouter directement un objet GraphNode/GraphEdge, mais il faudra passer par une méthode de Graph qui crée l'objet et modifie le reste du graphe pour que celui-ci reste cohérent.

Nom de fonction	createNode
Description	Cette fonction crée un nouveau noeud, l'ajoute au graphe et renvoie l'objet GraphNode créé.
Pré-conditions	Aucun noeud avec le même id n'a été créé avant
Post-conditions	Le graphe contient le noeud.
Entrées	int id : L'id du noeud à créer. String attrs : Les attributs du noeud à créer.
Sortie	GraphNode : Le noeud crée.

Nom de fonction	createEdge
Description	Cette fonction crée une nouvelle arête, l'ajoute au graphe et renvoie l'objet GraphEdge créé.
Pré-conditions	node0 et node1 sont deux noeuds du graphe
Post-conditions	Le graphe contient l'arête
Entrées	int id : L'id de l'arrête à créer. String attrs : Les attributs du noeud à créer. GraphNode node0 : L'une des extrémités de l'arrête GraphNode node1 : L'autre extrémité de l'arrête
Sortie	GraphEdge : L'arrête créée.

Nom de fonction	deleteNode
Description	Cette fonction tente de supprimer un noeud du graphe. Et retourne vrai si le noeud existe et a été supprimé, faux sinon.
Pré-conditions	Le noeud est un noeud du graphe.
Post-conditions	Le graphe ne contient plus le noeud.
Entrées	GraphNode node : Le noeud à supprimer.
Sortie	bool : Vrai si le noeud existe et a été supprimé, faux sinon.

Nom de fonction	deleteEdge
Description	Cette fonction tente de supprimer une arrête du graphe. Et retourne vrai si l'arrête existe et a été supprimée, faux sinon.
Pré-conditions	L'arête est une arête du graphe.
Post-conditions	Le graphe ne contient plus l'arête
Entrées	GraphEdge edge : L'arrête à supprimer.
Sortie	bool : Vrai si l'arrête existe et a été supprimée, faux sinon.

### 3.1.4 La classe GraphParser

La classe GraphParser permet de lire et écrire des graphes en fichiers XML. Pour cela, elle propose deux simples méthodes statiques

Nom de fonction	readGraph
Description	Cette fonction lie un fichier GXL et construit le graph représenté. Les noms des noeuds/arrêtes dans le fichier GXL peuvent varier, ils ont donc été mis en paramètre de la fonction.
Pré-conditions	Le fichier est un fichier gxl valide.
Post-conditions	-
Entrées	File file : Le fichier à lire.
Sortie	Graph : Le graph créé

Nom de fonction	writeGraph
Description	Cette fonction lie un fichier GXL et construit le graph représenté. Les noms des noeuds/arrêtes dans le fichier GXL peuvent varier, ils ont donc été mis en paramètre de la fonction.
Pré-conditions	Le fichier est un fichier valide.
Post-conditions	-
Entrées	Graph graph : Le graphe à écrire. File file : Le fichier à lire.
Sortie	-

## 3.2 Le package matrix

### 3.2.1 La classe AbstractMatrix (abstraite)

Cette classe représente une matrice dont les données sont stockées dans un tableau 1d ordonné en ligne.

Nom de fonction	ged1d (abstraite)
Description	Retourne la valeur indexée en une dimension.
Pré-conditions	Le paramètre index est positif, et inférieur ou égal à la taille de la matrice.
Post-conditions	-
Entrées	int index : L'indice de la valeur
Sortie	La valeur

Nom de fonction	createArray (abstraite)
Description	Crée un nouveau tableau du type de la matrice.
Pré-conditions	Le paramètre size est positif
Post-conditions	-
Entrées	int size : La taille du tableau (int) -> T : init : Une fonction pour initialiser le tableau
Sortie	Le tableau créé

Nom de fonction	get
Description	Retourne la valeur spécifiée par sa ligne et sa colonne.
Pré-conditions	Le paramètre row est positif et inférieur au nombre de lignes de la matrice Le paramètre col est positif et inférieur au nombre de colonnes de la matrice
Post-conditions	-
Entrées	int row : La ligne int col : La colonne
Sortie	La valeur

Nom de fonction	getIndex
Description	Retourne l'indice 1d à partir de la ligne et de la colonne.
Pré-conditions	Le paramètre row est positif et inférieur au nombre de lignes de la matrice. Le paramètre col est positif et inférieur au nombre de colonnes de la matrice.
Post-conditions	Le résultat est positif, et inférieur à la taille de la matrice.
Entrées	int row : La ligne int col : La colonne
Sortie	L'indice 1d

Nom de fonction	indexToRow
Description	Retourne la ligne à partir de l'indice 1d.
Pré-conditions	Le paramètre index est positif et inférieur à la taille de la matrice
Post-conditions	Le résultat est positif, et inférieur au nombre de lignes de la matrice.
Entrées	int index : L'indice 1d
Sortie	La ligne

Nom de fonction	indexToCol
Description	Retourne la colonne à partir de l'indice 1d.
Pré-conditions	Le paramètre index est positif et inférieur a la taille de la matrice
Post-conditions	Le résultat est positif, et inférieur au nombre de colonnes de la matrice.
Entrées	int index : L'indice 1d
Sortie	int : L'indice de la colonne

Nom de fonction	getRow
Description	Retourne une ligne de la matrice.
Pré-conditions	Le paramètre row est positif et inférieur au nombre de lignes de la matrice.
Post-conditions	-
Entrées	int row : L'indice de la ligne.
Sortie	T[] : La ligne

Nom de fonction	getCol
Description	Retourne une colonne de la matrice.
Pré-conditions	Le paramètre col est positif et inférieur au nombre de colonnes de la matrice.
Post-conditions	-
Entrées	int col : L'indice de la colonne.
Sortie	T[] : La colonne

Nom de fonction	valueToString
Description	Retourne une string lisible représentant la valeur.
Pré-conditions	-
Post-conditions	-
Entrées	T value : La valeur.
Sortie	String : La string lisible.

Nom de fonction	debug
Description	Affiche la matrice dans le logger.
Pré-conditions	-
Post-conditions	-
Entrées	String tag : Un tag à afficher ILogger logger : Le logger
Sortie	-



## 3.2.2 La classe IloIntMatrix

Nom de fonction	toDoubleMatrix
Description	Transforme cette matrice en matrice de doubles.
Pré-conditions	-
Post-conditions	La matrice créée est de la même dimension, et contient les mêmes valeurs.
Entrées	-
Sortie	DoubleMatrix : La matrice créée.

## 3.3 Le package ged

## 3.3.1 La classe GEDProblem

Nom de fonction	getNodeSubstitutionCost
Description	Retourne le cout de substitution d'un noeud par un autre.
Pré-conditions	node1Id est un id valide du graphe 1 node2Id est un id valide du graphe 2
Post-conditions	-
Entrées	int node1Id : L'id du noeud à substituer int node2Id : L'id du noeud par lequel on substitue le premier
Sortie	double : Le cout de substitution du noeud 1 par le noeud 2.

Nom de fonction	getNodeDeletionCost
Description	Retourne le cout de suppression d'un noeud.
Pré-conditions	node1Id est un id valide du graphe 1
Post-conditions	-
Entrées	int node1Id : L'id du noeud à supprimer
Sortie	double : Le cout de suppression du noeud.

Nom de fonction	getNodeAdditionCost
Description	Retourne le cout d'ajout d'un noeud.
Pré-conditions	node2Id est un id valide du graphe 2
Post-conditions	-
Entrées	int node2Id : L'id du noeud à ajouter.
Sortie	double : Le cout de suppression du noeud.

Nom de fonction	getEdgeSubstitutionCost
Description	Retourne le cout de substitution d'une arête par une autre.
Pré-conditions	edge1id est un id valide du graphe 1 edge2id est un id valide du graphe 2
Post-conditions	-
Entrées	int edge1id : L'id de l'arête à substituer int edge2id : L'id de l'arête par lequel on substitue la première
Sortie	double : Le cout de substitution de l'arête 1 par l'arête 2.

Nom de fonction	getEdgeDeletionCost
Description	Retourne le cout de suppression d'une arête.
Pré-conditions	edge1id est un id valide du graphe 1
Post-conditions	-
Entrées	int edge1id : L'id de l'arête à supprimer
Sortie	double : Le cout de suppression de l'arête.

Nom de fonction	getEdgeAdditionCost
Description	Retourne le cout d'ajout d'une arête.
Pré-conditions	edge2id est un id valide du graphe 2
Post-conditions	-
Entrées	int edge2id : L'id de l'arête à ajouter.
Sortie	double : Le cout de suppression de l'arête.

### 3.3.2 La classe GEDSolution

Nom de fonction	distanceTo
Description	Retourne la distance de hamming entre les deux solutions.
Pré-conditions	-
Post-conditions	Le résultat est positif
Entrées	GEDSolution solution : La solution à comparer.
Sortie	double : La distance de hamming entre les deux solutions.

## 3.3.3 L'interface IGEDSolver

Nom de fonction	exportModel
Description	Exporte le modele cplex vers un fichier.
Pré-conditions	Le chemin est valide.
Post-conditions	-
Entrées	String filePathTo : Le chemin du fichier vers lequel exporter le modèle.
Sortie	-

Nom de fonction	solve
Description	Résoud le problème en partant d'une solution initiale.
Pré-conditions	-
Post-conditions	-
Entrées	GEDSolution initialSolution : La solution initiale double timeLimitSecond : Le temps limite de la fonction
Sortie	IGEDSolver.Result : Le résultat, contenant la solution et son cout.

Nom de fonction	setNeighborhoodConstraints
Description	Met à jour les contraintes de voisinage de l'algorithme Local Branching.
Pré-conditions	-
Post-conditions	-
Entrées	bool deltaLessThan : Si vrai, le voisinage est toutes les solutions dont la distance est inférieure à threshold par rapport à la solution initiale bool onlyImportantVariables : Si vrai, on ne regarde que les variables importantes définies par l'algorithmes LocBra int threshold : La range de voisinage
Sortie	-

Nom de fonction	setUpperBoundConstraint
Description	Met à jour les contraintes d'upper bound de l'algorithme Local Branching
Pré-conditions	-
Post-conditions	-
Entrées	Double upperBound : L'upper bound.
Sortie	-

### 3.3.4 La classe GEDiloNodeMatches

Nom de fonction	get
Description	Retourne la valeur de matching d'un noeud du graphe 1 avec un noeud du graphe 2.
Pré-conditions	node1Id est un id valide du graphe 1 node2Id est un id valide du graphe 2
Post-conditions	-
Entrées	int node1Id : L'id du noeud du graphe 1 int node2Id : L'id du noeud du graphe 2
Sortie	IloIntVar : La valeur de matching du noeud du graphe 1 avec le noeud du graphe 2.

Nom de fonction	getNode1Matches
Description	Retourne la liste des valeurs de matching d'un noeud du graphe 1 avec ceux du graphe 2.
Pré-conditions	node1Id est un id valide du graphe 1
Post-conditions	-
Entrées	int node1Id : L'id du noeud du graphe 1
Sortie	IloIntVar[] : Les valeurs de matching d'un noeud du graphe 1 avec ceux du graphe 2.

Nom de fonction	getNode2Matches
Description	Retourne la liste des valeurs de matching d'un noeud du graphe 2 avec ceux du graphe 1.
Pré-conditions	node1Id est un id valide du graphe 2
Post-conditions	-
Entrées	int node1Id : L'id du noeud du graphe 2
Sortie	IloIntVar[] : Les valeurs de matching d'un noeud du graphe 2 avec ceux du graphe 1.

Nom de fonction	cloneStructure
Description	Retourne un GEDIloNodeMatches avec le même nombre de lignes et de colonnes que celui-ci, mais sans copier les valeurs.
Pré-conditions	-
Post-conditions	-
Entrées	-
Sortie	GEDIloNodeMatches : L'objet cloné.

Nom de fonction	debug
Description	Affiche cet objet dans le logger.
Pré-conditions	-
Post-conditions	-
Entrées	ILogger logger : Le logger
Sortie	-

## 3.3.5 La classe NeighborhoodConstraintsManager

Nom de fonction	setConstraints
Description	Met à jour les contraintes de voisinage de l'algorithme Local Branching.
Pré-conditions	threshold est positif.
Post-conditions	Le model contient les contraintes de voisinage spécifiées.
Entrées	bool deltaLessThan : Si vrai, le voisinage est toutes les solutions dont la distance est inférieure à threshold par rapport à la solution initiale bool onlyImportantVariables : Si vrai, on ne regarde que les variables importantes définies par l'algorithme LocBra int threshold : La range de voisinage
Sortie	-

Nom de fonction	removeConstraints
Description	Enlève les contraintes du problème.
Pré-conditions	-
Post-conditions	Le model ne contient plus de contrainte de voisinage.
Entrées	-
Sortie	-

## 3.3.6 La classe UpperBoundConstraintsManager

Nom de fonction	setConstraints
Description	Met à jour les contraintes d'upper bound de l'algorithme Local Branching
Pré-conditions	-
Post-conditions	Le modèle contient les contraintes d'upper bound spécifiées.
Entrées	Double upperBound : L'upper bound.
Sortie	-

Nom de fonction	removeConstraints
Description	Enlève les contraintes du problème.
Pré-conditions	-
Post-conditions	Le model ne contient plus d'upper bound.
Entrées	-
Sortie	-

### 3.4 Le package locbra

#### 3.4.1 L'interface ILocBraAlgorithm

Nom de fonction	getLogTag
Description	Retourne le tag de cette instance.
Pré-conditions	-
Post-conditions	-
Entrées	-
Sortie	String : Le tag de cette instance

Nom de fonction	compute
Description	Applique l'algorithme à l'instance donnée.
Pré-conditions	-
Post-conditions	-
Entrées	-
Sortie	LocBraResult : Le résultat de l'algorithme.

#### 3.4.2 La classe LocBraAlgorithm

Nom de fonction	initLocBraGED
Description	Applique l'initialisation de l'algorithme telle que décrite dans la publication.
Pré-conditions	-
Post-conditions	-
Entrées	-
Sortie	LocBraResult : Le résultat si une solution optimale a été trouvée, null sinon

Nom de fonction	improvedSolution
Description	Applique la fonction improvedSolution de l'algorithme, telle que décrite dans la publication.
Pré-conditions	-
Post-conditions	-
Entrées	-
Sortie	-

Nom de fonction	diversification
Description	Applique la fonction diversification de l'algorithme, telle que décrite dans la publication.
Pré-conditions	-
Post-conditions	-
Entrées	-
Sortie	-

Nom de fonction	intensification
Description	Applique la fonction intensification de l'algorithme, telle que décrite dans la publication.
Pré-conditions	-
Post-conditions	-
Entrées	-
Sortie	-

### 3.5 Le package utils

#### 3.5.1 L'interface ILogger

Nom de fonction	log
Description	Affiche un message.
Pré-conditions	-
Post-conditions	-
Entrées	String message : Le message à afficher.
Sortie	-



## 3.5.2 La classe Log

Nom de fonction	info
Description	Affiche un message avec le niveau de log info.
Pré-conditions	-
Post-conditions	-
Entrées	Class c : La classe appelant la fonction String message : Le message à afficher.
Sortie	-

Nom de fonction	debug
Description	Affiche un message avec le niveau de log debug.
Pré-conditions	-
Post-conditions	-
Entrées	Class c : La classe appelant la fonction String message : Le message à afficher.
Sortie	-

Nom de fonction	warn
Description	Affiche un message avec le niveau de log warn.
Pré-conditions	-
Post-conditions	-
Entrées	Class c : La classe appelant la fonction String message : Le message à afficher.
Sortie	-

Nom de fonction	error
Description	Affiche un message avec le niveau de log error.
Pré-conditions	-
Post-conditions	-
Entrées	Class c : La classe appelant la fonction String message : Le message à afficher.
Sortie	-

## 4 Planification

Ce projet sera réalisé en suivant le modèle en Cascade :

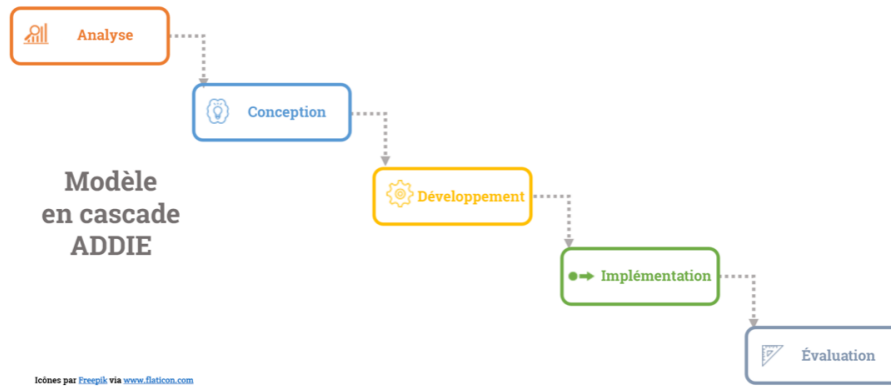


Figure 8 – Le modèle en cascade

L'analyse du sujet ainsi que la conception de la solution (comprennant les spécifications) se font pendant le premier semestre, et le développement et l'implémentation du livrable se fera pendant le second semestre.

#### 4.1 Découpage des tâches

Le rapport sera écrit en même temps que la réalisation des tâches.

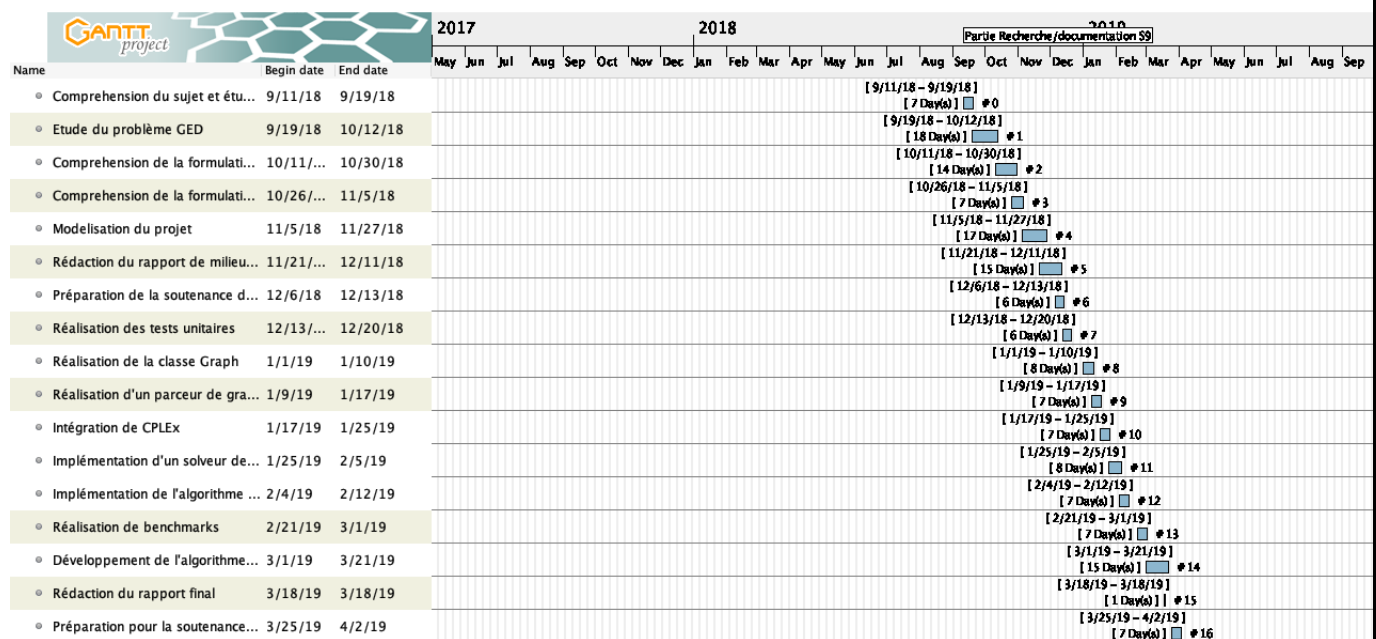


Figure 9 – Diagramme de Gantt

##### 4.1.1 Compréhension du sujet et étude de la faisabilité

Comprendre le sujet et l'objectif du projet.

— Date de début : Le 11 Septembre 2018

- Date de fin : Le 18 Septembre 2018
- Durée : 7 jours

#### 4.1.2 Etude du problème GED

Recherches générales sur le problème GED et ses méthodes de résolution

- Date de début : Le 19 Septembre 2018
- Date de fin : Le 10 Octobre 2018
- Durée : 18 jours

#### 4.1.3 Compréhension de la formulation F2 du problème GED

Comprendre le modèle mathématique, et reproduire le modèle sur CPLEX pour le tester.

- Date de début : Le 11 Octobre 2018
- Date de fin : Le 25 Octobre 2018
- Durée : 14 jours

#### 4.1.4 Compréhension de la formulation F3 du problème GED

Comprendre le modèle mathématique. Et commencer à penser à son implémentation.

- Date de début : Le 26 Octobre 2018
- Date de fin : Le 02 Novembre 2018
- Durée : 7 jours

#### 4.1.5 Modélisation du projet

Modéliser la structure de l'application Java.

- Date de début : Le 03 Novembre 2018
- Date de fin : Le 20 Novembre 2018
- Durée : 17 jours

#### 4.1.6 Rédaction du rapport de milieu de projet

Finir la rédaction du rapport (qui doit tout de même être commencé).

- Date de début : Le 21 Novembre 2018
- Date de fin : Le 6 Décembre 2018
- Durée : 15 jours

#### 4.1.7 Préparation pour la soutenance de milieu de projet

Se préparer pour la soutenance.

- Date de début : Le 6 Décembre 2018
- Date de fin : Le 12 Décembre 2018
- Durée : 6 jours

#### 4.1.8 Réalisation des tests unitaires

Créer des tests pour la classe Graph et le parseur de XML.

- Date de début : Le 13 Décembre 2018
- Date de fin : Le 31 Décembre 2018
- Durée : 6 jours

#### 4.1.9 Réalisation de la classe Graph

Créer la classe Graph permettant de gérer simplement un graphe.

- Date de début : Le 01 Janvier 2018
- Date de fin : Le 09 Janvier 2018
- Durée : 8 jours

#### 4.1.10 Réalisation d'un parseur de graphes en fichiers XML

Créer le parseur de graphes permettant de lire et d'écrire un graphes depuis/dans un fichier XML.

- Date de début : Le 09 Janvier 2019
- Date de fin : Le 16 Janvier 2019
- Durée : 7 jours

#### 4.1.11 Intégration de CPLEX

Intégration de la librairie CPLEX dans le projet.

- Date de début : Le 17 Janvier 2019
- Date de fin : Le 24 Janvier 2019
- Durée : 7 jours

#### 4.1.12 Implémentation d'un solveur de la formulation F2 du problème GED

Comprendre le sujet et l'objectif du projet.

- Date de début : Le 25 Janvier 2019
- Date de fin : Le 02 Février 2019
- Durée : 8 jours

#### 4.1.13 Implémentation de l'algorithme LocBra

- Date de début : Le 03 Février 2019
- Date de fin : Le 10 Février 2019
- Durée : 7 jours

#### 4.1.14 Implémentation de l'interface utilisateur

- Date de début : Le 11 Février 2019
- Date de fin : Le 20 Février 2019
- Durée : 9 jours

#### 4.1.15 Réalisation de benchmarks

Créer des benchmarks pour comparer les exécutions de l'algorithmes sur différentes données.

- Date de début : Le 21 Février 2019
- Date de fin : Le 28 Février 2019
- Durée : 8 jours

#### 4.1.16 Développement de l'algorithme multithread

Développer l'algorithme LocBra en version multi thread

- Date de début : Le 01 Mars 2019
- Date de fin : Le 15 Mars 2019
- Durée : 14 jours

#### 4.1.17 Rédaction du rapport final

Finir la rédaction du rapport (qui doit tout de même être commencé).

- Date de début : Le 16 Mars 2019
- Date de fin : Le 23 Mars 2019
- Durée : 7 jours

#### 4.1.18 Préparation pour la soutenance finale

Se préparer pour la soutenance.

- Date de début : Le 24 Mars 2019
- Date de fin : Le 01 Avril 2019
- Durée : 7 jours

# 5

## Mise en oeuvre

### 1 Technologies utilisées

- IDE :IntelliJ
- Langage : Kotlin/Java
- Moteur de production Gradle : Le moteur de production le plus populaire avec Maven (pour Java)
- Gestionnaire de versions : Git (github)
- Cplex : La librairie de solveur utilisée
- IloCplex : Une librairie servant d'interface entre Cplex et Java

### 2 Implémentation

#### 2.1 Les briques de base

##### 2.1.1 Matrice

Les matrices ont été implémentées à partir de l'interface générique `AbstractMatrix`, celui ci impose la règle suivante :

Les matrices implémentées doivent stocker leurs données dans un tableau 1D, ordonné en ligne.

Cela permet deux choses :

- Il est simple d'itérer sur toutes les valeurs en une boucle
- Il est simple de récupérer toutes les valeurs dans un tableau, ce qui est notamment utile pour initialiser les matrices contenant des valeurs Cplex.

##### 2.1.2 Graphe

Une classe `Graph` contenant toutes les méthodes de manipulation de graphe est implémentée.

Cette implémentation proposée stocke les attributs des noeuds et des arêtes sous forme de chaîne de caractères.

Les méthodes qui permettent de lire et d'écrire un graphe dans un fichier ont été implémentées dans un fichier à part, afin d'alléger le code de la classe Graph.

## 2.2 Implémentation des formulations F2 et F3 avec IloCplex

Les solveurs des problèmes GED formulés avec la formulation F2 et F3 sont implémentés dans le package GED.

Ces solveurs auront pour but d'offrir un interface indépendant de Cplex, en suivant l'interface IGEDSolver.

Cette étape a été plus longue que prévue : Il n'est pas simple de déboguer un modèle Cplex. Cette étape était aussi celle pour laquelle j'ai eu le plus besoin de l'aide de mon tuteur.

Nous avons pu valider l'implémentation avec deux méthodes :

- Nous avons comparé les modèles .lp générés par nos deux implémentations
- Nous avons comparé les résultats donnés par nos implémentations sur des instances données

## 2.3 Implémentation de l'algorithme Local Branching non parallélisé

L'algorithme Local Branching non parallélisé a été implémenté en suivant l'interface ILocBraAlgorithm.

Celui-ci a été implémenté en suivant à la lettre la description de l'algorithme proposée par "The Graph Edit Distance Problem treated by the Local Branching Heuristic".

## 2.4 Implémentation de l'algorithme Local Branching parallélisé

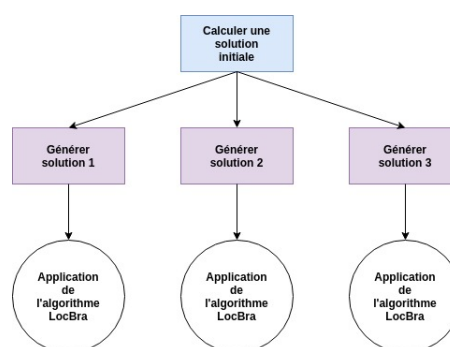


Figure 1 – LocBraMultithread

L'algorithme parallélisé a été implémenté en suivant l'interface ILocBraAlgorithm.

Celui-ci agit de la façon suivante :

- Une solution initiale valide est calculée grâce à une iteration de Cplex
- Cette solution est ensuite modifiée afin de produire plusieurs solutions valides
- L'algorithme Local Branching est appliqué à chacune de ces solutions initiales.
- La meilleure solution est retenue

Afin de produire plusieurs solutions valides à partir d'une solution, des mutations sont appliquées à la solution d'origine :

Une mutation consiste à intervertir deux noeuds dans la solution, en prenant deux graphes A et B contenant respectivement les noeuds a1, a2 et b1, b2. Si a1 était attribué à b1 et a2 à b2, la solution mutée attribuera a1 à b2 et a2 à b1.

## 2.5 Tests

Les tests sont implémentés dans un module séparé du module principal.

Cela permet d'alléger le code du module principal, et de spécifier des dépendances uniquement pour le module test, telles que les dépendances vers la librairie de test JUnit.

### 2.5.1 Tests unitaires

Les briques sont testées par des tests unitaires.

### 2.5.2 Tests fonctionnels

Les implémentations des formulations F2 et F3, ainsi que les implémentations des différentes versions de l'algorithme Local Branching sont testées par des tests fonctionnels :

J'ai reçu au début du projet des instances de tests comportant les graphes à comparer, ainsi que les résultats obtenus, je n'ai donc qu'à comparer mes résultats avec ceux-ci.



# 6

## Bilan et conclusion

### 1 Point sur l'avancement

Le programme permettant de tester les algorithmes est terminé. Et une implémentation simple de l'algorithme parallélisé est incluse.

Cependant je n'ai pas eu le temps de tester d'autres implémentations de cet algorithme, ou d'autres méthodes d'initialisation de celui-ci.

### 2 Bilan auto-critique sur la gestion du projet

#### 2.1 Points positifs

Je pense avoir su me renseigner sur le problème, ce qui m'a permis de bien appréhender celui-ci. Je suis satisfait de la structure de projet que j'ai réalisée.

#### 2.2 Points négatifs

Un gros point négatif a été que j'ai passé du temps à ré-implémenter des classes au lieu d'importer une librairie, notamment pour la gestion des matrices et des graphes.

Aussi, j'ai mal évalué le temps de réalisation de deux tâches de mon projet, ce qui m'a forcé à décaler le planning général.

### 3 Conclusion

Ce projet m'a permis d'améliorer mes compétences dans la gestion d'un projet à long terme, notamment dans la gestion de rendus professionnels.

De plus, ce projet m'a permis d'améliorer ma capacité de présentation orale.

Ce projet m'a aussi permis d'avancer sur un projet personnel de génération de musique par un réseau neuronal. Je n'arrivais pas à créer un algorithme qui calcule la différence entre deux musiques. j'utilise désormais une version modifiée de l'algorithme String Edit Distance de Landau, Myers et Schmidt.

# Rapport de projet de recherche et développement : Détection et reconnaissance avec des graphes

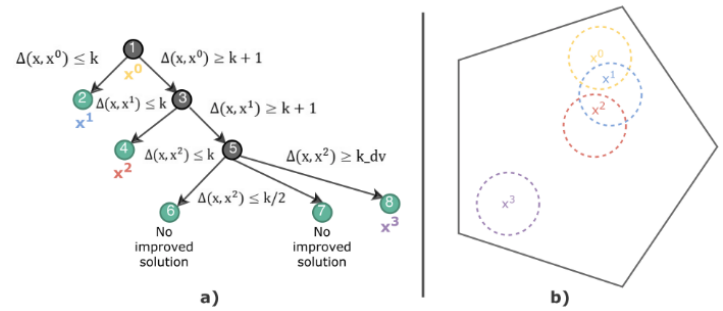
Xavier Peltier

Encadrement : Darwiche Mostafa

## Objectifs

Ce projet porte sur le thème du graph matching, plus particulièrement sur le calcul de la Graph Edit Distance.

Ce projet a pour but de réaliser une version parallélisée de l'algorithme de calcul de Graph Edit Distance : Local Branching.



Mode de recherche de l'algorithme Local Branching

## Mise en oeuvre

Premièrement, l'algorithme Local Branching sera réimplémenté en Kotlin.

Ensuite, une version parallélisée de l'algorithme sera réalisée, en suivant le fonctionnement décrit par ce schéma.

Enfin, des tests de performance seront effectués afin de comparer les algorithmes parallélisés et non parallélisés.

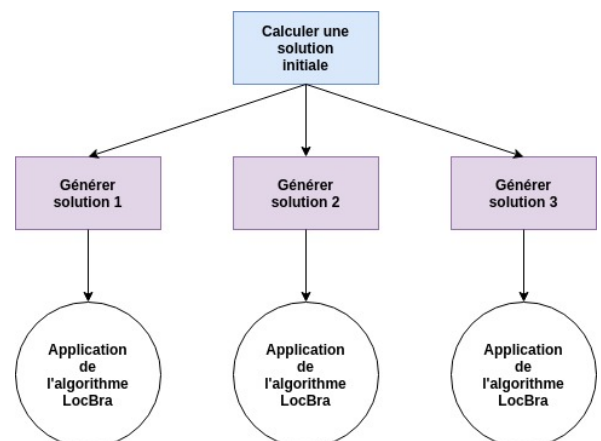
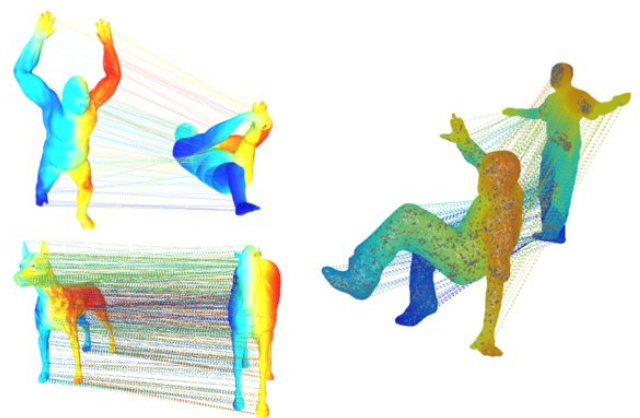


Schéma de la parallélisation de l'algorithme

## Résultats attendus

L'algorithme parallélisé devrait être plus performant, notamment au niveau de son initialisation.



Applications du graph matching.

# Rapport de projet de recherche et développement : Détection et reconnaissance avec des graphes

Xavier Peltier

Encadrement : Darwiche Mostafa

## Objectifs

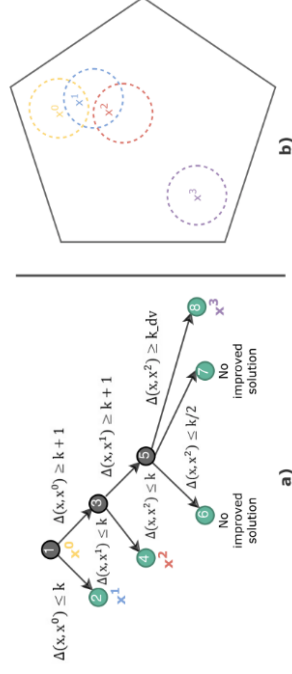
Ce projet porte sur le thème du graph matching, plus particulièrement sur le calcul de la Graph Edit Distance. Ce projet a pour but de réaliser une version parallélisée de l'algorithme de calcul de Graph Edit Distance : Local Branching.

## Mise en oeuvre

Premièrement, l'algorithme Local Branching sera réimplémenté en Kotlin. Ensuite, une version parallélisée de l'algorithme sera réalisée, en suivant le fonctionnement décrits par ce schéma. Enfin, des tests de performance seront effectués afin de comparée les algorithmes parallélisés et non parallélisés.

## Résultats attendus

L'algorithme parallélisé devrait être plus performant, notamment au niveau de son initialisation.



Mode de recherche de l'algorithme Local Branching

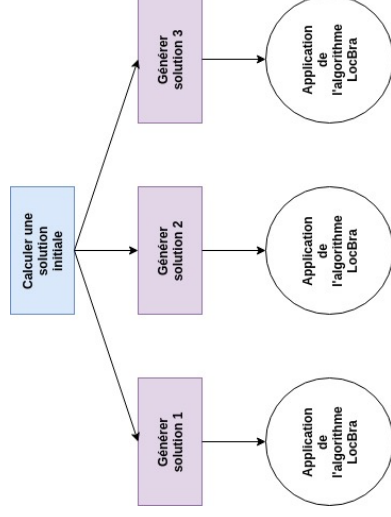
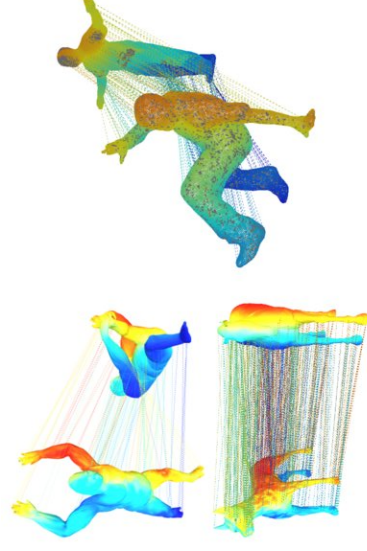


Schéma de la parallélisation de l'algorithme



Applications du graph matching.

# Rapport de projet de recherche et développement : Détection et reconnaissance avec des graphes

## Résumé

Rapport de PRD

## Mots-clés

PLNE, Graph, LocBra, GED

## Abstract

PRD project report

## Keywords

PLNE, Graph, LocBra, GED