

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

Projet Recherche & Développement
2017-2018

**Plus court chemin à contraintes de
ressources et multi-objectifs**

Tuteur académique
Yannick KERGOSIEN

Étudiant
Arthur CROCQUEVIEILLE (DI5)

2 avril 2018



Liste des intervenants

Nom	Email	Qualité
Arthur CROCQUEVIEILLE	arthur.crocquevieille@etu.univ-tours.fr	Étudiant DI5
Yannick KERGOSIEN	yannick.kergosien@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Arthur CROCQUEVIEILLE susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Yannick KERGOSIEN 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

Arthur CROCQUEVIEILLE, *Plus court chemin à contraintes de ressources et multi-objectifs*,
Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais
de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
  author={CROCQUEVIEILLE, Arthur},
  title={Plus court chemin à contraintes de ressources et multi-objectifs},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2017-2018}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
Liste des Algorithmes	vi
1 Remerciements	1
2 Introduction	2
1 Acteurs du projet	2
2 Contexte de la réalisation	3
2.1 Contexte	3
2.2 Description du problème.....	3
2.3 Existant	3
2.4 Objectifs	4
3 Contraintes	4
4 Bases méthodologiques	4
3 Description générale	5
1 Environnement du projet	5
2 Caractéristiques des utilisateurs	5
3 Fonctionnalités du système	5

4	Structure générale du système	6
5	Interface homme/machine	6
4	Etat de l'art	7
1	Introduction.....	7
2	Le problème de plus court chemin multi-objectifs de même type.....	7
3	LCDPF.....	9
3.1	Phase 1 : Preprocessing.....	9
3.1.1	Trouver les solutions initiales appartenant au front de Pareto final	9
3.1.2	Calculer des chemins partiels de n'importe quel noeud au noeud destination	10
3.2	Phase 2 : Label-Correcting.....	11
3.3	Tests et résultats.....	11
4	Le problème du plus court chemin avec contraintes de ressources	12
4.1	LP-based branch-and-bound	12
4.1.1	Tests et résultats	12
4.2	L'algorithme Pulse	12
4.2.1	Tests et résultats	13
5	Le problème du plus court chemin avec plusieurs fonctions objectifs de différents types.....	13
5.1	Le problème du plus court chemin tri-critères avec une des deux fonctions objectifs minMax	13
5.1.1	Tests et résultats	14
5.2	Un label-setting agrégé pour le problème du plus court chemin multi-objectifs.....	14
5.2.1	Tests et résultats	14
5	Analyse et conception	16
1	Plan de développement	16
2	Fonctionnalités	16
6	Mise en oeuvre	18
1	Choix et éléments d'implémentation.....	18
1.1	Outils et librairies	18
1.2	Choix techniques	18
1.3	Détails d'implémentation.....	19
1.4	Risques	19
1.5	Limites	19
2	Les algorithmes et leur implémentation	20
2.1	Approche s'inspirant du LCDPF pour résoudre le problème du plus court chemin à contrainte de ressources	20

2.1.1	Première phase : Le preprocessing.....	20
2.1.2	Seconde phase : Label Correcting	21
2.1.3	Algorithme.....	22
3	Les tests.....	24
4	Les résultats	24
4.1	Approche s’inspirant du LCDPF pour résoudre le problème du plus court chemin avec des critères de type coût et de type minMax	27
4.1.1	Première phase : Le preprocessing.....	27
4.1.2	Seconde phase : Label Correcting	28
4.1.3	Algorithme.....	28
5	Les tests.....	29
6	Les résultats	29
7	Bilan et conclusion	31
1	Bilan du projet après la partie de recherche	31
2	Etapes à venir après la partie de recherche	31
3	Bilan du projet final	31
4	Bilan personnel.....	32
	Annexes	33
A	Spécifications	34
1	Diagramme de cas d’utilisation	34
2	Diagramme de classe	35
B	Gestion de projet	36
1	Gestion du temps.....	36
2	Méthode de gestion de projet utilisée	36
3	Identification des tâches.....	38
3.1	Analyse de faisabilité.....	38
3.2	Analyse de risque.....	38
3.3	Suivi de projet.....	38
4	Git	39
5	Trello	39
	Comptes rendus hebdomadaires	40
	Bibliographie	44

Table des figures

2 Introduction

1	Logo Laboratoire d'Informatique de Tours.....	3
2	Logo Polytech Tours	3

4 Etat de l'art

1	Graphe	8
2	Graphe apres l'etape 1	10
3	Graphe apres l'etape 2.....	10

5 Analyse et conception

1	Gantt	17
---	-------------	----

6 Mise en oeuvre

1	Instances utilisées pour Howard and Kis.....	24
2	Tableau comparant le temps d'exécution en secondes sur D1	24
3	Tableau comparant le temps d'exécution en secondes sur D2	25
4	Tableau comparant le temps d'exécution en secondes sur Santos et al.....	25
5	Instances utilisées pour Lozano and Medaglia	26
6	Tableau comparant le temps d'exécution en secondes sur Santos et al.....	26
7	Tableau comparant le temps d'exécution en secondes sur Zhu and Wilhelm.	26
8	Instances utilisées et résultats en secondes pour Iori, Martello et Pretolani	30

A Spécifications

1	Diagramme de cas d'utilisation	34
---	--------------------------------------	----

2	Diagramme de classes	35
B Gestion de projet		
1	Gantt apres la mise en oeuvre	37



Liste des Algorithmes

1	LCDPF pour le problème du plus court chemin avec contraintes de ressources	23
2	IS_PROMISING	23
3	LCDPF pour le problème du plus court chemin multi-objectifs avec minMax	29

1

Remerciements

J'adresse mes remerciements aux personnes qui m'ont aidé dans la réalisation de ce projet de recherche et développement.

En premier lieu, je remercie M. Kergosien, maître de conférences à Polytech Tours, pour son expertise et ses conseils. En tant que mon encadrant, il m'a guidé dans mon travail et m'a aidé à élaborer des solutions pour résoudre certains problèmes.

Je remercie également HORVATH et KIS, « [Solving resource constrained shortest path problems with LP-based methods](#) », LOZANO et MEDAGLIA, « [On an exact method for the constrained shortest path problem](#) », PINTO et PASCOAL, « [On algorithms for the tricriteria shortest path problem with two bottleneck objective functions](#) » et IORI, MARTELLO et PRETOLANI, « [An aggregate label setting policy for the multi-objective shortest path problem](#) » pour avoir accepté de me partager leurs données afin de pouvoir comparer mon travail au leur.

Enfin, je souhaite particulièrement remercier GIRET, KERGOSIEN, NERON et SAUVANET, « [An efficient label-correcting algorithm for the multi objective shortest path problem](#) » de m'avoir donné l'opportunité de prendre la suite de leur travail sur les problèmes de plus courts chemins multi-objectifs.

2

Introduction

Le Projet de Recherche et Développement (PRD), réalisé en 5ème année, s'inscrit dans la formation dispensée à l'Ecole Polytechnique de Tours et constitue une réelle expérience en terme de conduite de projet. Le PRD se déroule du 21 Septembre 2017 au 5 Avril 2018 à raison de 2 jours par semaine à temps plein. Il donne lieu à la rédaction d'un rapport avec le cahier de spécification et de deux présentations du travail effectué lors de deux soutenances.

Le PRD est orienté vers une étude théorique et un développement algorithmique de résolution de différent problèmes du plus court chemin où la MOA était représentée par Monsieur Kergosien membre du Laboratoire d'Informatique de Tours. Le projet s'inscrit dans un cadre théorique qui fait l'objet d'une étude bibliographique en amont, pour aboutir au développement de plusieurs algorithmes novateurs.

Ce document est donc, d'abord, le cahier de spécification système du projet. Il défini les besoins, l'environnement du projet et les objectifs à réaliser. Ce rapport présente aussi les différentes tâches à effectuer et le planning prévisionnel.

1 Acteurs du projet

Les acteurs du projet sont :

- la maitrise d'oeuvre (MOE) : Arthur Crocquevieille, étudiant en 5ème année de l'Ecole d'Ingénieur Polytech Tours au sein du département informatique, dans le cadre du PRD ainsi que son encadrant Yannick Kergosien de l'équipe de recherche ROOT du Laboratoire d'Informatique de Tours. Les intérêts de recherche principaux de l'équipe ROOT incluent la conception, le développement et l'application de techniques de recherches sur des problèmes concernant : la production et la planification, le calcul de haute performance et le big data, et enfin le transport et la logistique.
- la maitrise d'ouvrage (MOA) : Yannick Kergosien, membre de l'équipe de recherche ROOT du Laboratoire d'Informatique de Tours. Ce document a été rédigé par Arthur Crocquevieille et sera validé par les différents acteurs du projet.



Figure 1 – Logo Laboratoire d'Informatique de Tours



Figure 2 – Logo Polytech Tours

2 Contexte de la réalisation

2.1 Contexte

Le problème du plus court chemin est un problème algorithmique de la théorie des graphes. Il existe de nombreuses variantes de ce problème. On suppose un graphe fini et orienté. Chaque noeud et chaque arc du graphe possède certaines informations appelées critères ou encore poids. Une des applications les plus courantes de ce problème est de trouver le plus court chemin entre deux villes. Ce problème peut paraître simple mais plus il a de critères à prendre en compte plus il devient compliqué. Le Laboratoire d'Informatique de Tours s'apprête à publier un article de recherche concernant un nouvel algorithme qu'ils ont développé. Cet algorithme s'attaque au problème du plus court chemin à plusieurs critères de même type. Il surpasse tous les algorithmes concernant ce problème. Pour aller plus loin, la MOA souhaiterait implémenter un algorithme, s'inspirant du précédent, capable de résoudre d'autres types de problèmes comme le problème du plus court chemin avec contraintes de ressources et le problème du plus court à plus fonctions objectifs de différents types. La MOA veut s'avoir si ce nouvel algorithme permet d'avoir de meilleurs résultats que les autres algorithmes de la littérature actuelle. C'est donc dans ce contexte que la MOE propose un projet afin de trouver des solutions pour développer et tester ces algorithmes novateurs.

2.2 Description du problème

2.3 Existant

Le Laboratoire d'Informatique de Tours ne possède aucun logiciel dédié au problème étudié. Il existe cependant une implémentation d'un algorithme élaboré afin de résoudre le problème du plus court chemin à plusieurs fonctions objectifs de même type. On s'inspire et adaptera cette première implémentation afin de résoudre d'autres types de problèmes. L'algorithme existant par GIRET, KERGOSIEN, NERON et SAUVANET, « [An efficient label-correcting algorithm for the multi objective shortest path problem](#) » (LCDPF) apporte une approche novatrice quand à la résolution de ce problème. Les résultats obtenus par cet algorithme sont significativement meilleurs que ceux des algorithmes de la littérature. C'est pourquoi il est important de voir si opter une même stratégie sur un autre type de problème peut donner des résultats intéressants. Il a été décidé d'implémenter nos algorithmes en C++ en utilisant l'IDE Xcode. Le LCDPF lui-même a été implémenté en C++ avec Xcode et utilise certaines bibliothèques. Nous allons utiliser les mêmes bibliothèques pour coder nos algorithmes. Il s'agit de notamment de Boost. La manipulation

de la librairie Boost n'est pas connu par la MOE alors nous comptons sa prise en main dans la période de développement.

2.4 Objectifs

Ce projet de fin d'étude a plusieurs objectifs. On peut clairement définir trois phases très distinctes sur lesquelles la MOE travaillera. Dans un premier temps, on orientera le travail sur la recherche.

- Comprendre et rechercher le problème du plus court chemin à contraintes de ressources
- Comprendre et recherche le problème du plus court chemin mutli-objectifs
- Rechercher les algorithmes de la littérature correspondant aux différents problèmes jugés pertinents

Ensuite, une fois la première étape terminée la MOE s'orientera sur le développement.

- Rechercher les algorithmes pour chaque problème
- Implémenter les algorithmes sous la forme d'un programme en console
- Tester les implémentations
- Récupérer les résultats sur des graphes définis dans la première phase

Enfin, une fois les algorithmes implémentés, testés et exécutés sur des instances on comparera les résultats afin de fournir une conclusion à la MOE. En plus de ces objectifs, un cahier de spécification, un état de l'art et un rapport seront rédigés.

3 Contraintes

Nous nous imposons un développement respectant les règles de génie logiciel afin d'assurer la réutilisabilité, la maintenance et la bonne évolution de notre programme.

4 Bases méthodologiques

Dès le commencement du projet nous avons du établir certaines normes et outils qui seront utilisés tout au long du projet. Nous avons opté pour une approche Agile concernant la gestion du projet. Cela signifie que la MOA suivra les principes suivant :

- Notre première priorité est de satisfaire le client en lui délivrant de manière continue notre travail.
- Nous accepterons les changements même tard dans le processus de développement à condition qu'ils soient au bénéfice du client.
- Chaque avancement sera délivré au client de manière régulière. Par exemple on délivrera une version du projet toutes les deux semaines.
- On privilégiera des meetings aux emails.
- On mesura le succès du projet au fonctionnement du logiciel.

Les outils suivants seront également utilisés :

- Git : logiciel de gestion de versions décentralisé
- Trello : outils de gestion de projet en ligne basé sur une organisation en planches listant des cartes
- BitBucket : outils d'hébergement Git pour projets
- Google Drive

3

Description générale

1 Environnement du projet

Le logiciel sera développé sur un MacBook Pro 2016 13-inch, un processeur 2GHz Intel Core i5 et 8GB 1867 MHz de RAM. Le projet étant complètement nouveau il ne dépend d'aucun environnement. Il existera cependant une variation d'environnement concernant l'exécution du projet. En effet, le but principale du projet est de comparer les résultats des nouveaux algorithmes avec ceux de la littérature actuelle, on essayera au maximum d'exécuter le programme sur des machines ayant des configurations les plus proches possibles de celles utilisées que lors des tests des algorithmes de la littérature actuelle.

2 Caractéristiques des utilisateurs

Le but du projet étant principalement de comparer, de développer et tester de nouveaux algorithmes pour résoudre certains types problèmes du plus court chemin, le système servira à un seul type d'utilisateur : Membre du Laboratoire d'Informatique de Tours.

- Utilisateur occasionnel
- Expérience avec les interfaces consoles
- Connaissances avancées en informatiques
- Tous les droits sur le système

3 Fonctionnalités du système

Notre système va être développé en vue d'être utilisé d'une seule manière. En effet, puisqu'il n'y a qu'un seul type d'utilisateur et qu'une seule fonctionnalité chaque utilisation du programme ce fera comme ceci :

1. Choix du type de problème à résoudre
2. Passage du fichier d'instance et du fichier de graphe
3. Résolution du problème en batch
4. Affichage de certaines variables liées à la qualité de la résolution du problème

4 Structure générale du système

La structure générale du système reste encore incertaine. En effet, puisque nous avons à disposition l'implémentation du LCDPF en C++, il est possible que nous réutilisions les fonctionnalités des classes suivante :

- Noeud : un noeud dans graphe a un identifiant unique et une position dans le graphe. Pour garder sa position dans le graphe le type location de la librairie boost est utilisé.
- Arc : un arc dans un graphe a un identifiant unique, l'identifiant du noeud source, l'identifiant du noeud cible et un vecteur de doubles contenant les valeurs des critères sur l'arc.
- Graphe : le graphe associe les noeuds aux arcs
- Instance : une instance est constituée d'un identifiant unique, l'identifiant du noeud de départ et l'identifiant du noeud d'arrivée. Elle représente un problème du type : on veut aller du noeud A au noeud B.
- Label : un label a un identifiant unique, l'identifiant d'un label parent, un noeud car un label est forcément associé à un noeud, un vecteur de doubles représentant les critères et un autre représentant les critères projetés.
- Parser : le parser est un objet qu'on utilisera afin de récupérer les informations du graphe, des noeuds et des arcs contenus dans un fichier csv.
- Writer : le writer est un objet qu'on utilisera afin d'écrire

Il s'agit ici d'une description succincte des classes, on notera que ces classes contiennent également les méthodes de manipulation de leurs attributs. Cependant, bien qu'elle accélérât grandement la partie développement, la réutilisation de ces structures ne se fera que si :

- Le projet est facilement réutilisable et compréhensif
- La prise en main des librairies n'est pas trop chronophage

Sinon, on réimplémentera le LCDPF. Cela aura pour effet de ralentir la phase de développement mais permettra d'assurer une bonne compréhension du projet par la MOE.

5 Interface homme/machine

Ce projet n'a pas pour objectif de développer une IHM. Cependant, nous avons tout de même besoin d'un moyen pour que l'utilisateur entre des données (graphe, instance, etc.) et puisse lire les données de sortie (temps d'exécution, front de Pareto, etc.). Comme expliqué précédemment, nous avons à disposition le code source de l'implémentation du LCDPF par Antoine Giret, notamment son parser et writer. Le parser permet de lire des fichiers au format CSV contenant, par exemple, le graphe ou encore les instances. Nous avons décidé d'utiliser un parser afin de permettre à l'utilisateur de rentrer les données de son problème. Avec ceci, il n'aura qu'à spécifier le chemin de ses fichiers à l'exécution du programme et les objets seront créés. Le projet d'Antoine Giret contient également un writer. Nous avons décidé d'utiliser un writer car il permet d'écrire des informations concernant la résolution d'un problème donné dans un fichier au format csv. Ceci nous permet, malgré l'absence d'interface homme/machine, de donner un retour à l'utilisateur. On utilisera également ces données dans la conclusion de ce projet.

4

Etat de l'art

1 Introduction

Le problème du plus court chemin est un problème très connu dans le domaine de l'optimisation combinatoire. L'objectif est de calculer un chemin entre deux sommets qui minimise ou maximise une certaine fonction. Un exemple classique est d'aller d'une ville à une autre le plus rapidement possible en passant par certaines routes. Les premières recherches débutèrent il y a plus de 60 ans avec BELLMAN, « [On a routing problem](#) » et DIJKSTRA, « [A note on two problems in connexion with graphs](#) ». Depuis, plusieurs variantes de ce problème ont été étudiées. Parmi ces variantes nous pouvons citer le problème du chemin le plus court dans un réseau stochastique EIGER, MIRCHANDANI et SOROUSH, « [Path preferences and optimal paths in probabilistic networks](#) », le problème du chemin le plus court en fonction du temps SHERALI, HOBEIKA et KANGWALKLAI, « [Time-dependent, label-constrained shortest path problems with applications](#) » ou encore le problème du plus court chemin avec plusieurs fonctions objectifs (Multi-Objective Short Path problem). Ce dernier problème est particulièrement intéressant. Il consiste à trouver une collection de chemin entre deux points tout en minimisant plusieurs fonctions objectifs. C'est un problème très courant dans la vie de tous les jours, notamment concernant les problématiques de transports. Par exemple, on l'utilise afin de modéliser un problème de minimisation du risque et du temps d'un trajet.

Cette partie fait l'état de l'art de trois problèmes du plus court chemin classiques. Dans un premier temps, nous présenterons le problème du plus court chemin avec plusieurs fonctions objectifs du même type. Nous décrivons les principaux algorithmes existant dans la littérature et expliquons le nouvel algorithme développé par le MOA. Le but de ce projet étant d'adapter cet algorithme à d'autres problèmes, nous introduirons, dans un second temps, deux autres problèmes du plus court chemin : le problème du plus court chemin avec contraintes de ressources et le problème du plus court chemin multi-objectifs.

2 Le problème de plus court chemin multi-objectifs de même type

Avant d'aller plus loin il est important de définir le problème du plus court chemin avec plusieurs fonctions objectifs. On définit un graphe orienté $G = (V, A)$ avec V la collection de noeuds et A la collection d'arcs. Chaque arc (i, j) a un coût qu'on appellera C_{ij}^k associé à un

critère k . On suppose que le graphe ne contient pas de cycles négatifs et que le coût associé à un arc pour un critère est forcément supérieur ou égal à 0. Les fonctions objectifs sont de même type. Le problème consiste à trouver une collection de chemins allant d'un noeud source à un noeud cible tout en minimisant une ou plusieurs fonctions objectifs.

Dans cette section, on présente les principaux travaux de recherche en rapport avec le problème du plus court chemin avec plusieurs fonctions objectifs. On notera que la plus part des travaux considèrent seulement deux fonctions objectifs. Les premières méthodes permettant de résoudre ce problème sont le label-correcting de HANSEN, « [Multiple criteria decision making theory and application](#) » et le label-setting de MARTINS, « [On a multicriteria shortest path problem](#) ». La différence principale entre ces deux méthodes est la façon avec laquelle les labels sont ordonnés. Pour le label-setting, on utilise une stratégie qui assure qu'à chaque itération un label est traité seulement s'il est non dominé jusqu'à la fin de l'algorithme. Pour le label-correcting, on peut utiliser n'importe quelle stratégie pour ordonner les labels. Par exemple, FIFO est souvent utilisée afin de simplifier la structure des données contenant la liste des labels à traiter. Chaque méthode de labelling a également deux différentes façon de choisir le prochain label à parcourir : tous les labels d'un noeud choisi sont étendus en même temps ou on gère chaque label séparément.

Il existe également une approche à deux phases introduite par MOTE, MURTHY et OLSON, « [A parametric approach to solving bicriterion shortest path problems](#) » et ULUNGU et TEGHEM, qui permet de résoudre ce problème. Cela consiste à, dans une première phase, calculer toutes les solutions supportées et, dans une seconde phase, calculer toutes les solutions non supportées. Les solutions de la première phase sont calculées en utilisant une seule fonction objectif correspondant à la somme pondérée des critères. Les solutions de la seconde phase sont calculées en utilisant une méthode pour deux fonctions objectifs qui énumère toutes les solutions non supportées. Il existe plusieurs améliorations possibles permettant d'augmenter l'efficacité de ces algorithmes. RAITH, « [Speed-up of labelling algorithms for biobjective shortest path problems](#) » a montré qu'il n'était pas toujours nécessaire de propager un label à un node cible pour montrer que ce dernier est dominé. DEMEYER, GOEDGEBEUR, AUDENAERT, PICKAVET et DEMEESTER, « [Speeding up Martins algorithm for multiple objective shortest path problems](#) » ont proposé deux améliorations basées sur les conditions d'arrêts.

Plus récemment, DUQUE, LOZANO et MEDAGLIA, « [An exact method for the biobjective shortest path problem for largescale road networks](#) » ont proposé une nouvelle méthode pour résoudre le problème du plus court chemin avec deux fonctions objectifs appelée Pulse. Elle est basée sur une méthode récursive utilisant des stratégies de "pruning" permettant d'accélérer l'exploration de graphe. L'algorithme Pulse peut également être étendu à d'autre problème comme le problème du plus court chemin avec contraintes de ressources LOZANO et MEDAGLIA, « [On an exact method for the constrained shortest path problem](#) ».

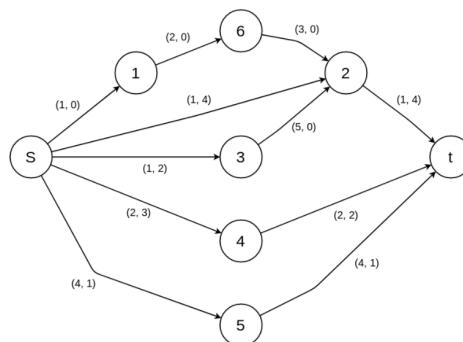


Figure 1 – Graphe

3 LCDPF

L'algorithme par GIRET, KERGOSIEN, NERON et SAUVANET, « [An efficient label-correcting algorithm for the multi objective shortest path problem](#) » est la base de ce projet. Comme expliqué plus tôt, c'est sur cet algorithme novateur que nous baserons nos recherches et notre développement de nouveaux algorithmes pour résoudre d'autres problèmes. Ainsi, nous avons consacré plus d'importance à l'explication de son fonctionnement. Il est similaire à la méthode à deux phases introduite par ULUNGU et TEGHEM, pour résoudre les algorithmes à deux fonctions objectifs. La première phase consiste à déterminer les solutions supportées. La seconde phase consiste à trouver les solutions non supportées. Durant la première phase on résout plusieurs problèmes du plus court chemin à une seule fonction objectif. Cette phase permet également de déterminer des bornes supérieures et inférieures du front de Pareto final à chaque noeud du graphe. La deuxième phase consiste à trouver le front de Pareto final. Pour cela, on utilise un label-correcting classique MARTINS, « [On a multicriteria shortest path problem](#) » auquel quelques améliorations ont été ajoutées. Dans les sections suivantes nous expliquons les différentes phases. Nous avons choisi de ne pas aller trop dans les détails pour des raisons de lisibilité du rapport. Vous pouvez consulter GIRET, KERGOSIEN, NERON et SAUVANET, « [An efficient label-correcting algorithm for the multi objective shortest path problem](#) » pour plus d'informations concernant l'algorithme.

3.1 Phase 1 : Preprocessing

Cette phase permet, dans un premier temps, de déterminer quelques solutions initiales appartenant au front de Pareto final et, dans un second temps, de calculer des chemins partiels de n'importe quel noeud au noeud destination.

3.1.1 Trouver les solutions initiales appartenant au front de Pareto final

La première étape consiste à trouver quelques solutions initiales appartenant au front de Pareto final. Pour cela, l'algorithme va résoudre un problème du plus court chemin, du noeud de destination au noeud de départ en inversant tous les arcs, qui cherche à minimiser une combinaison linéaire des critères. On résout autant de problèmes qu'il y a de critères en jouant sur la combinaison linéaire. Pour chaque résolution on obtient :

- Une solution V_s^n supportée pour le problème basée sur GEOFFRION, « [Proper efficiency and the theory of vector maximization](#) »
- Pour chaque noeud du graphe, on a une collection de valeurs V_i^n qui correspond à un chemin réalisable du noeud i à la destination. Ces valeurs ont été déterminées par l'algorithme de Dijkstra appliqué au problème du plus court chemin en minimisant la combinaison linéaire des critères.

Cette première étape permet, dans une phase antérieure, d'améliorer les performances de l'algorithme en permettant d'éliminer plus de solutions infaisables.

Prenons le graphe précédent en exemple.

L'algorithme inverse le graphe, et résout le problème du plus court chemin avec trois combinaisons linéaires différentes. D'abord, on considère que le premier critère, ensuite seulement le deuxième et enfin une combinaison linéaire des deux. Ce qui donne le graphe suivant :

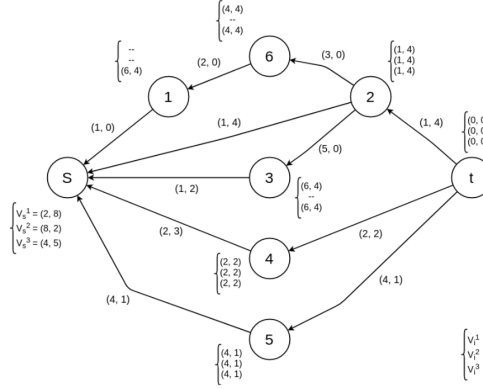


Figure 2 – Graphe apres l'etape 1

3.1.2 Calculer des chemins partiels de n'importe quel noeud au noeud destination

Cette deuxième étape est la plus compliquée et la plus importante pour la seconde phase. Elle sert à trouver les autres chemins partiels de la destination à chaque noeud prometteur. Pour cela l'algorithme résout, en utilisant l'algorithme de Dijkstra, un one-to-all mono-objectif problème du plus court chemin pour tous les labels. La différence avec l'étape précédente est qu'au lieu d'arrêter l'algorithme quand le noeud destination est atteint on s'arrête quand tous les noeuds pertinents sont atteints. Ces nouveaux chemins vont avoir deux avantages principaux sur la seconde phase :

- Mise-à-jour dynamique du front de Pareto pendant la recherche de la solution
- Accélère la recherche de la solution en empêchant la propagation de certains labels
- Supprime certains noeuds du graphe

Un noeud est prometteur, cela à une signification bien particulière qui est cruciale à cet algorithme. En effet, énumérer tous les chemins V_i^n pour tous les noeuds coûte très cher en temps et n'est pas nécessaire. Ainsi, pour réduire le temps de cette phase, seul les noeuds qui seront impliqués dans la prochaine phase doivent être pris en compte. Afin de déterminer les noeuds prometteurs, l'algorithme crée un "pire chemin" du noeud i au noeud destination qui aura pour valeur de critère les pires valeurs trouvés dans la phase 1 V_n^{worst} . Nous propageons le label seulement si ce pire chemin n'est pas dominé par le front de Pareto actuel. Cette seconde phase va également supprimer certains noeuds du graphe car on peut être sur qu'ils ne seront jamais parcouru. Cette seconde phase donne un graphe avec tous les V_i^s déterminés et, dans certains cas, un graphe réduit.

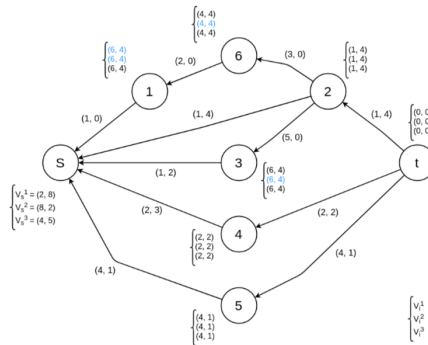


Figure 3 – Graphe apres l'etape 2

3.2 Phase 2 : Label-Correcting

Cette phase est basée sur un algorithme de label-correcting introduit par MARTINS, « [On a multicriteria shortest path problem](#) ». Le but étant de trouver toutes les solutions non dominées du noeud source au noeud destination. On utilise les résultats de l'étape précédente afin de diminuer le nombre de propagation et d'ainsi augmenter la rapidité de l'algorithme.

Pour conclure il est possible de résumer le LCDPF en cinq étapes principales :

- L'initialisation : c'est la phase 1. Elle permet de déterminer les solutions supportées, de les ajouter au front de Pareto, de déterminer des chemins faisables d'un noeud i au noeud cible, de supprimer les noeuds inutiles du graphe, d'initialiser la file d'exploration et la liste des labels associée à chaque noeud.
- Sélection du prochain label à traiter : on utilise pour cela les valeurs des labels calculées dans la première phase.
- Création et évaluation de nouveaux labels : pour un noeud i et un label l , un nouveau label est créé pour chaque successeur de i . Cependant, seulement les labels prometteur doivent être considérés.
- Ajouter des nouveaux labels à la liste de labels associée à un noeud et la mettre à jour : un label prometteur va être ajouté à la liste des labels d'un noeud s'il n'est pas dominé par d'autres labels de cette liste. Si ce noeud est le noeud destination alors on ajoutera ce label au front de Pareto car il représente un chemin non dominé vers le noeud final. Ce label peut également dominer d'autres labels dans le front de Pareto courant, il est alors important de supprimer ces derniers du front. Si ce noeud n'est pas le noeud final, alors le label est ajouté à la file d'exploration. Après cet ajout, on supprime les labels dominés de la file d'exploration.
- Mettre à jour le front de Pareto : cette étape concerne la mise à jour dynamique et rapide du front de Pareto pendant la recherche. Cela permet de rapidement créer un nouveau chemin d'un noeud source à un noeud destination. Ce nouveau chemin est ajouté au front de Pareto et les chemins dominés par ce dernier en sont supprimés. Cela permet d'accélérer la convergence du front de Pareto actuel vers le front finale pendant la recherche.

3.3 Tests et résultats

Les tests ont été effectués sur une machine sous linux avec un Intel Xeon 2.67GHz, 8 core et 8GB de RAM. Les instances du 9ème challenge DIMACS ont été utilisées. Les résultats de la résolution du plus court chemin multi-objectifs résolu par cet algorithme, ont été comparés aux meilleurs algorithmes de la littérature. Dans un premier temps le LCDPF est comparé à une variation de NAMOA* proposé par MACHUCA et MANDOW, « [Lower bound sets for biobjective shortest path problems](#) », appelée KDLS, sur des instances de New York. Les résultats sont spectaculaires. Par exemple, le LCDPF peut être jusqu'à 1200 fois plus rapide restant en dessous de la barre des 5 secondes. Ensuite, ils ont également comparé leur algorithme avec blSET par RATH, « [Speed-up of labelling algorithms for biobjective shortest path problems](#) » et Pulse par DUQUE, LOZANO et MEDAGLIA, « [An exact method for the biobjective shortest path problem for largescale road networks](#) ». Les comparaisons ont été effectuées sur des instances de NY, BAY et FLA utilisées pour le 9ème DIMACS challenge. Les tests montrent que le biSET prend en moyenne plus de temps que les autres algorithmes testés. En ce qui concerne l'algorithme Pulse et le LCDPF, le Pulse se montre particulièrement efficace sur de petites instances. Le LCDPF est en moyenne moins efficace que le Pulse sur ce type d'instances. En revanche, le LCDPF est bien plus efficace que les deux autres algorithmes sur des instances moyennes et grandes. En effet, il est en moyenne 20 à 450 plus rapide que l'algorithme Pulse et 120 à 850 fois plus rapide que le biSET.

Enfin, les auteurs comparent le LCDPF au Ratio-Labeling BSP et au SLSET par SEDENO-NODA et RAITH, « *A Dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem* ». Les graphes NY, BAY et FLA du 9ème DIMACS ont été utilisés. Les résultats montrent que LCDPF se montre particulièrement efficace comparé aux autres sur des instances moyennes. Cependant, sur de grandes instances le LCDPF demande trop de ressource car le nombre de labels explose. Le problème n'est alors pas résolu dans le temps imposé de 10 minutes à cause d'un manque de puissance de la machine.

4 Le problème du plus court chemin avec contraintes de ressources

On définit le problème suivant : $G = (V, A)$ avec V la collection de noeuds et A la collection d'arcs. Chaque arc (i, j) a un coût qu'on appellera C_{ij}^k associé à un critère k et un vecteur de poids spécifiant la valeur d'une collection de ressources de capacités finies. Le but est de trouver le plus court chemin qui n'utilise pas plus de ressources que disponibles. Ce genre de problème est commun dans des domaines comme la planification de vols dans les aéroports, la répartition des membres d'une équipe, etc.

4.1 LP-based branch-and-bound

HORVATH et KIS, « *Solving resource constrained shortest path problems with LP-based methods* » ont décidé d'utiliser une approche appelée branch-and-cut se basant sur l'algorithme de Garcia GARCIA, *Resource constrained shortest paths and extensions*. Cependant, ils ont apporté quelques améliorations comparé aux algorithmes de la littérature. Dans HORVATH et KIS, « *Solving resource constrained shortest path problems with LP-based methods* » ils expliquent qu'ils ont créé une nouvelle méthode des plans sécants, de nouvelles heuristiques et une nouvelle stratégie d'ajustement de variables. Cette approche novatrice leur a permis de réduire de manière drastique le temps d'exécution de l'algorithme de Garcia sur un graphe orienté, sans cycles et sans critères négatifs.

4.1.1 Tests et résultats

Les tests ont été effectués sur un ordinateur portable Intel Core i7-4710MQ, 2.5 GHz CPU sous Windows 7. Ils ont implémenté leur programme en C++ en utilisant Xpress en tant que framework pour l'approche branch-and-cut et la librairie LEMON pour gérer les graphes. Ils ont comparé leur algorithme aux meilleurs algorithmes de la littérature, c'est-à-dire : PUGLIESE et GUERRIERO, « *A reference point approach for the resource constrained shortest path problems* » et LOZANO et MEDAGLIA, « *On an exact method for the constrained shortest path problem* ». Pour cela ils ont utilisé 3 différentes instances bien connues de la littérature dans : DUMITRESCU et BOLAND, « *Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem* » et SANTOS, COUTINHO-RODRIGUES et CURRENT, « *An improved solution algorithm for the constrained shortest path problem* ». Leurs expériences montrent que leur phase de preprocessing joue un rôle clé pour réduire le temps de calcul comparé aux autres algorithmes.

4.2 L'algorithme Pulse

Les algorithmes de type Pulse sont bien connus de la littérature concernant les problèmes du plus court chemin. Le principe est simple, il s'agit de propager une pulsation d'un noeud de départ à

un noeud d'arriver. En se propageant de noeud à noeud la pulsation construit un chemin partiel. Une fois arrivée au noeud destination le pulsation a trouvé un chemin faisable, une solution. Le but est d'énumérer toutes solutions afin de trouver les solutions optimales.

La force d'un tel algorithme repose sur la stratégie de pruning. C'est-à-dire, on cherche à propager le pulse sur le moins de noeud possible. Cet algorithme novateur LOZANO et MEDAGLIA, « [On an exact method for the constrained shortest path problem](#) » met l'accent sur deux stratégies de pruning originales. D'abord elle va empêcher la propagation de pulse sur certain noeud très tôt dans l'exploration du graphe. Cela est fait dès qu'il est possible de savoir qu'un Pulse lancé sur un noeud n'atteindra jamais la destination finale. L'agressivité de cette stratégie permet de stopper la propagation de beaucoup de noeuds et donc d'accélérer l'exploration du graphe. Ensuite, ils ont ajouté une deuxième stratégie qui va permettre de stopper la propagation des pulses. Elle consiste à borner les fonctions objectifs afin de pouvoir stopper la propagation du pulse dès qu'elle dépasse cette borne.

4.2.1 Tests et résultats

Les tests ont été implémenté sur un ordinateur Intel Mobile Core 2 Dual @ 2.4 Ghz CPU P8600 avec 512 MB de RAM tournant sous Windows XP. Ils ont utilisé le langage de programmation Java et Eclipse SDK.

Ils ont testé leur algorithme sur de nombreux graphes allant de 40.000 noeuds jusqu'à 800.000. Les résultats qu'ils ont obtenu sont probants. Leur implémentation résout le CSP en moins de 0.3 seconde en moyenne. Ils ont comparé leur algorithme aux meilleurs algorithmes de la littérature actuelle. Leur implémentation bat SANTOS, COUTINHO-RODRIGUES et CURRENT, « [An improved solution algorithm for the constrained shortest path problem](#) » sur de nombreuses instances connues de la littérature, ils ont obtenu un facteur de 60 sur la vitesse d'exécution. Ils ont également de meilleures performances que label-setting DUMITRESCU et BOLAND, « [Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem](#) » atteignant un facteur de 900 pour la vitesse.

5 Le problème du plus court chemin avec plusieurs fonctions objectifs de différents types

Ce problème est similaire au premier mise-à-part que les fonctions objectifs peuvent être de différents types. Prenons l'exemple suivant : nous avons un réseau de routes, chaque arc est associé à une mesure de la qualité de la route, la densité du trafic et le temps de trajet. Résoudre ce problème consiste à trouver un chemin qui maximise la qualité des routes et de minimiser le trafic et le temps de trajet. On a donc plusieurs fonctions objectifs de types différents.

5.1 Le problème du plus court chemin tri-critères avec une des deux fonctions objectifs minMax

Cet article de recherche PINTO et PASCOAL, « [On algorithms for the tricriteria shortest path problem with two bottleneck objective functions](#) » traite de la résolution des problèmes du plus court chemin avec deux minMax/maxMin fonctions objectifs et une fonction additive. Cela concerne un graphe sans cycles, orienté et sans critères négatifs. Il introduit une méthode améliorée, inspirée de LIMA PINTO, BORNSTEIN et MACULAN, « [The tricriterion shortest path problem with at least two bottleneck objective functions](#) » et LIMA PINTO, BORNSTEIN et MACULAN FILHO, « [UM PROBLEMA DE CAMINHO TRI-OBJETIVO](#) » pour calculer les plus courts chemins

dans un sous-graphe, obtenu en restreignant la collection d'arc en fonction des deux valeurs des deux fonctions minMax/maxMin dans le but de trouver la collection complète minimal des solutions du front de Pareto. Il prend également en compte les valeurs des fonctions objectifs des chemins trouvés afin de réduire le sous-graphe et ainsi le nombre de problème du plus court chemin à résoudre en utilisant un procédure de label.

5.1.1 Tests et résultats

Les tests ont été effectués sur un Intel Core 2 Duo 2.0 GHz avec 3GB de RAM. Leur algorithme a été comparé à LIMA PINTO, BORNSTEIN et MACULAN, « [The tricriterion shortest path problem with at least two bottleneck objective functions](#) » et LIMA PINTO, BORNSTEIN et MACULAN FILHO, « [UM PROBLEMA DE CAMINHO TRI-OBJETIVO](#) » sur des graphes de 2000, 7000, 12000, 15000, 20000 et 30000 noeuds. On remarque que leur algorithme qu'ils appellent MMS-B met en moyenne moins d'une seconde pour trouver toutes les solutions du front de Pareto sur des petits graphes (de 2000 à 7000 noeuds) là où les autres algorithmes comparés ici mettent en moyenne entre 0 et 30 secondes. Pour les plus gros graphes, le MMS-B met en moyenne moins de 20 secondes là où les autres algorithmes peuvent mettre jusqu'à plus d'une minute. On constate que cette nouvelle stratégie permet d'améliorer considérablement le temps de résolution du problème du plus court chemin avec deux minMax fonctions. On notera cependant une variable importante à prendre en compte. Les graphes utilisés pour les tests ont été générés aléatoirement. Il sera donc difficile d'obtenir les mêmes graphes afin de se comparer aux résultats de PINTO et PASCOAL, « [On algorithms for the tricriteria shortest path problem with two bottleneck objective functions](#) ». La MOE essaiera au maximum d'utiliser les mêmes graphes.

5.2 Un label-setting agrégé pour le problème du plus court chemin multi-objectifs

Cet article IORI, MARTELLO et PRETOLANI, « [An aggregate label setting policy for the multi-objective shortest path problem](#) » traite du problème du plus court chemin avec plusieurs fonctions objectifs. Ils exposent un algorithme novateur afin de trouver la collection complète et maximale des chemins allant dans d'un noeud de départ à n'importe quel autre noeud d'arrivée avec des fonctions objectifs incluant minMax/maxMin et additives. Pour cela, ils s'inspirent d'un algorithme de label setting de Dijkstra classique. Une importante amélioration des performances de cet algorithme peut être obtenue en remplaçant l'ordre lexicographique des labels par un ordre agrégé. C'est-à-dire :

- On ajoute à chaque label d'un noeud une information supplémentaire correspondant à une combinaison linéaire des valeurs des fonctions objectives d'un noeud de départ jusqu'à ce noeud.
- Ensuite, pendant l'exploration du graphe, on choisit, à chaque itération, un label provisoire où cette combinaison linéaire est minimale.

Ils constatent également que l'ordre dans lequel les comparaisons de labels sont effectuées affecte les performances de l'algorithme.

5.2.1 Tests et résultats

Plusieurs versions de l'algorithme ont été développées et testées. Ils ont été codés en C++ et compilés avec gcc 4.3.2. Les tests ont été effectués sur un Pentium IV PC avec 3GHz et 2GB de RAM sous Linux. Ils ont testé leur algorithme en utilisant des graphes aléatoirement créés présentés dans GANDIBLEUX, BEUGNIES et RANDRIAMASY, « [Martins algorithm revisited for multi-objective shortest](#) »

path problems with a MaxMin cost function ». Ils ont fait varier plusieurs paramètres notamment le nombre de fonctions objectifs additives et le nombre de fonction objectifs min/max. La comparaison de leur algorithme s'est faite contre leur propre implémentation du label setting de Dijkstra ordonnant les labels de manière lexicographique. On constate qu'ordonner les labels de leur manière permet de réduire drastiquement le temps d'exécution. Par exemple, sur de grosses instances on atteint un facteur d'amélioration de 2.78.

5

Analyse et conception

1 Plan de développement

Le plan de développement de ce PRD est expliqué dans le Gantt ci-dessous. Cependant, certaines étapes nécessitent certaines clarifications. La MOE a divisé le projet en 2 grandes étapes. D'abord, une étape de recherche sera effectuée. Cette étape est nécessaire afin d'établir le cahier de spécification et l'état de l'art. Elle renforce la compréhension du projet par la MOE et permet à la MOA et la MOE de se mettre d'accord sur l'étape à suivre. Ensuite, la dernière étape sera une étape de développement. Cette étape sera divisée en deux cycles de développement. En effet, la MOE implémentera deux algorithmes, un cycle est donc nécessaire pour chacun d'entre eux. Un cycle répète les étapes suivantes sur un algorithme jusqu'à ce que la MOA et la MOE soit d'accord sur ses performances.

- Adaptation du LCDPF
- Implémentation en C++
- Tests et résultats

Chaque fin d'étape est marquée par un rapport et une soutenance. La soutenance sera donnée par la MOE aux encadrant de l'école Polytech Tours. La MOA est bien entendu conviée.

2 Fonctionnalités

- Cycle 1 : Problème du plus court chemin à contraintes de ressources
 - Conception de l'algorithme à l'aide de Yannick Kergosien, membre du Laboratoire d'Informatique de Tours.
 - Implémentation en C++
 - Comparaison des résultats avec : HORVATH et KIS, « [Solving resource constrained shortest path problems with LP-based methods](#) » et LOZANO et MEDAGLIA, « [On an exact method for the constrained shortest path problem](#) ».
- Cycle 2 : Problème du plus court chemin multi-critères
 - Conception de l'algorithme à l'aide de Yannick Kergosien, membre du Laboratoire d'Informatique de Tours.
 - Implémentation en C++

- Comparaison des résultats avec : PINTO et PASCOAL, « On algorithms for the tricriteria shortest path problem with two bottleneck objective functions » et IORI, MARTELLO et PRETOLANI, « An aggregate label setting policy for the multi-objective shortest path problem ».

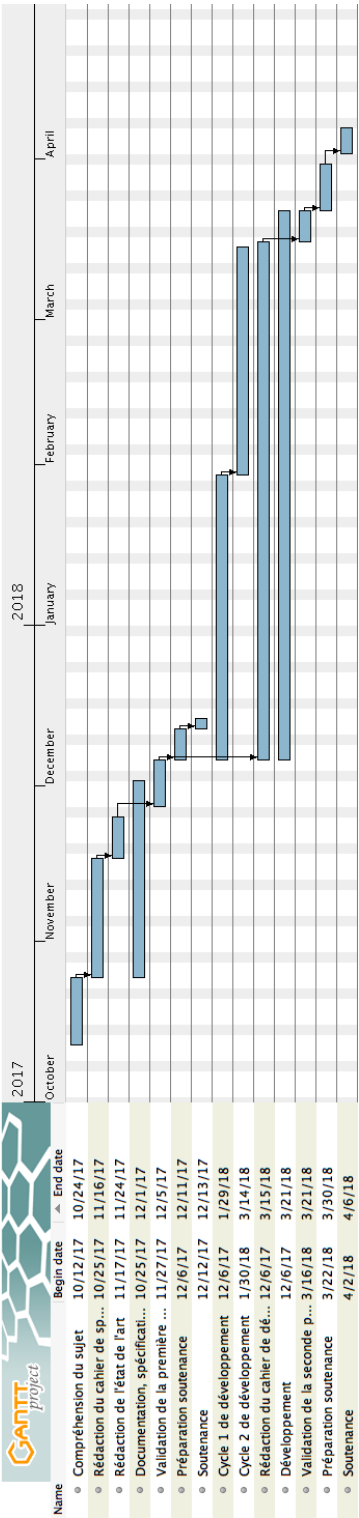


Figure 1 – Gantt

6

Mise en oeuvre

1 Choix et éléments d'implémentation

1.1 Outils et librairies

Plusieurs outils et librairies ont été utilisées tout au long de ce projet. Dans un premier temps, les différentes adaptations du LCDPF ont été développées sous Xcode en C++. Xcode est un IDE (Integrated Development Environment), un environnement de développement. Il comprend un éditeur de texte, un compilateur et un débogueur ainsi que plusieurs fonctionnalités comme un affichage hiérarchique des fichiers, un éditeur graphique (interface builder) ou encore un logiciel d'exportation du programme créé. Xcode permet de développer en C, C++, Obj-C, Java, AppleScript, Python, Ruby, Rez et Swift. Pour implémenter les adaptations du LCDPF la MOE a utilisé deux librairies C++ :

- Boost : est un ensemble de bibliothèques bâties sur le standard du C++. Chaque bibliothèque répond à un besoin précis d'une manière homogène et réutilisable. Pour ce projet, la MOE s'est servie des bibliothèques de manipulation de graphes.
- Osmium : est une librairie C++ rapide et flexible pour travailler avec les données d'OpenStreetMap. Le LCDPF s'en sert afin de pouvoir stocker la longitude et latitude d'un noeud dans un graphe représentant un environnement réel.

Dans un second temps, l'IDE PyCharm a été utilisé. En effet, pour des besoins de manipulations de contenus de fichier, expliqués dans la partie suivant, la MOE a utilisé Python avec PyCharm.

1.2 Choix techniques

L'implémentation des algorithmes s'inspirant du LCDPF a entraîné un choix technique. En effet, le but de ce projet était de comparer les performances de nos algorithmes à d'autres algorithmes de la littérature et pour cela nous avons besoin de récupérer des graphes et des instances utilisés par différents papiers de recherche. Il s'est avéré que les différents graphes et instances récupérés étaient stockés dans des fichiers mais jamais de la bonne façon. C'est-à-dire, que le parser implémenté par Antoine Giret n'était pas capable de lire les fichiers correctement. Ainsi, pour remédier à ce problème nous avons dû choisir entre développer un nouveau parser pour

chaque format différent et l'ajouter à notre projet ou créer plusieurs scripts afin de transformer un certain format vers celui utilisé par le parser déjà implémenté. Il a donc été choisi de créer plusieurs script Python afin de transformer tous les formats différents vers :

- Un fichier csv contenant l'instance du problème : l'identifiant du noeud de départ, l'identifiant du noeud cible et les valeurs maximales pour chaque critère de ressource (s'il y en a).
- Un fichier csv contenant les noeuds du problème : l'identifiant du noeud ainsi que sa longitude et sa latitude. Les informations de longitude et latitude ont dû être générées aléatoirement car elles n'étaient jamais renseignés. Cela n'influence pas la résolution du problème.
- Un fichier csv contenant les arcs du problème : l'identifiant du noeud de départ, l'identifiant du noeud cible et les valeurs de chaque critère.

Il a été aussi question de choisir si la MOE allait réutiliser l'implémentation du LCDPF par Antoine Giret ou tout re développer du fait de la prise en main de Boost. Après un certain temps à lire la documentation de la librairie, la tester et comprendre l'implémentation déjà existante, il a été choisi de garder l'implémentation d'Antoine Giret car la prise en main de Boost s'est avérée être plus facile qu'anticipée.

1.3 Détails d'implémentation

1.4 Risques

Ce projet comportait deux risques principaux. Le premier risque, qui était aussi une des premières tâches du projet, concernait la reprise de l'implémentation du LCDPF par Antoine Giret. En effet, cette implémentation en C++ utilisait la librairie boost qui n'avait été utilisée auparavant par la MOE. Ainsi, nous n'étions pas certain que la MOE puisse reprendre cette implémentation. Heureusement, tout s'est bien passé mais en cas d'échec nous avions prévu que la MOE implémente elle-même le LCDPF avec le langage de son choix. Cela aurait eu pour conséquence de raccourcir grandement les deux phases de développement et les performances de l'algorithme auraient donc pu être impacté. Le deuxième risque concernait les comparaisons avec les algorithmes de la littérature actuelle. En effet, afin de pouvoir comparer notre implémentation avec les algorithmes de la littérature actuelle il était nécessaire de récupérer les données utilisées par les autres papiers de recherche. Il y avait une possibilité que ces chercheurs n'aient pas gardé leurs données et que nous puissions donc pas se comparer à eux. Heureusement nous avons pu récupérer suffisamment de données. En cas d'échec nous avions prévu de créer nos propres données afin d'avoir des résultats. Nous aurions pu ensuite essayer de récupérer le code des autres algorithmes afin de les implémenter et les tester localement avec les données créées.

1.5 Limites

Il y a certaines limites à ce projet. En effet, il n'est pas facile de comparer les performances de deux algorithmes. Il y a de nombreux paramètres qui peuvent influencer les performances de l'exécution d'un algorithme. Par exemple, un algorithme sera plus adapté pour certains graphes, aura de meilleures performances dépendant de la machine sur laquelle il est exécuté etc. Dans ce projet, nous avons voulu réduire au maximum le nombre de paramètres influençant les performances de nos algorithmes comparés à ceux de la littérature. Cependant, il n'a pas été possible d'exécuter nos algorithmes et les algorithmes de la littérature sur un même environnement.

Cela limite quelque peu ce projet. En effet, à cause de la différence d'environnement il n'est pas possible de tirer des conclusions sur les performances des algorithmes à moins que celles-ci soient vraiment différentes.

De plus, il n'a pas été possible de récupérer les solutions (chemins) trouvées par les algorithmes, nous n'avons donc pas pu s'assurer que tous les algorithmes obtenaient les mêmes solutions aux mêmes problèmes. Cependant, nous avons pu comparer le nombre de solutions trouvées par les algorithmes afin de limiter ce problème.

2 Les algorithmes et leur implémentation

La mise en oeuvre de ce projet s'est faite en plusieurs phases. En effet, chaque problème du plus court chemin a été traité séparément. On rappelle que chaque algorithme implémenté s'inspire et s'aide de l'implémentation du GIRET, KERGOSIEN, NERON et SAUVANET, « [An efficient label-correcting algorithm for the multi objective shortest path problem](#) » par Antoine Giret.

2.1 Approche s'inspirant du LCDPF pour résoudre le problème du plus court chemin à contrainte de ressources

Comme précisé dans le gantt à la figure 1 (Annexe B), le premier algorithme implémenté est celui qui va résoudre le problème du plus court chemin avec contraintes de ressources. La différence avec le LCDPF classique est qu'au lieu de traiter seulement des critères de coût, nous traitons également des critères de ressources.

Pour pouvoir s'attaquer au problème du plus court chemin avec contraintes de ressources il a fallu changer quelques informations comparé au problème concerné par le LCDPF classique. En effet, nous allons rajouter sur chaque arc des informations de consommation de ressource. Un arc contient maintenant a critères de coût et b critères de ressources. De plus, il a fallu changer la définition d'une instance du problème. Auparavant, une instance était simplement constituée d'un noeud de départ et d'un noeud d'arrivée, il a fallu rajouter b limites à ne pas dépasser pour toutes ressources considérées dans le problème.

Les parties suivantes décrivent un algorithme qui s'inspire du LCDPF permettant de résoudre le problème du plus court chemin à contraintes de ressources. Cette algorithme a été élaboré par la MOE avec l'aide de Yannick Kergosien. L'algorithme, tout comme le LCDPF, est séparé en deux phases. Dans un premier temps, une phase de préprocessing qui va permettre d'obtenir des informations sur les graphes afin de réduire le nombre de labels à explorer (faire converger le front de Pareto plus rapidement) pendant la seconde phase. Cette dernière correspond à la résolution du problème avec un label correcting classique en s'appuyant sur les informations récupérées pendant la première phase.

2.1.1 Première phase : Le préprocessing

On rappelle que la phase de préprocessing consiste à trouver des bornes inférieures et supérieures pour les différents critères et également de calculer des chemins partiels de n'importe quel noeud au noeud de destination. Celle-ci résout plusieurs problèmes du plus court chemin mono-objectif (un pour chaque critère), obtenu par une combinaison linéaire de chaque critère, en utilisant l'algorithme de Dijkstra sur le graphe inversé. On renvoie à l'article GIRET, KERGOSIEN, NERON et SAUVANET, « [An efficient label-correcting algorithm for the multi objective shortest path problem](#) » pour plus d'information concernant la première phase. La phase de préprocessing pour la résolution du problème du plus court chemin avec contraintes de ressources est très

similaire à celle du LCDPF. On peut remarquer qu'un critère de ressource peut être manipulé de la même manière qu'un critère de coût pendant la première phase. En effet, le calcul d'un coût ou d'une ressource d'un noeud à un autre se fait de manière additive.

Dans un second temps, comme pour le LCDPF classique, on va procéder à un one-to-all mono-objectif sur le graphe inversé pour chaque critère pour tous les noeuds intéressant. Cependant, il est nécessaire de changer la notion de label intéressant. On rappelle qu'un label intéressant est un label qui n'est pas dominé par un pire chemin fictif créé en utilisant les bornes supérieures obtenu à la première étape de la phase de preprocessing. Si ce label est dominé par le pire chemin alors il n'est pas nécessaire de l'explorer. Cette notion change dans le cadre du problème du plus court chemin à contrainte de ressource. En effet, la borne supérieure d'une ressource n'est pas suffisamment pertinente pour pouvoir l'utiliser dans le pire chemin fictif. En plus du fait que tous les critères de coût ne doivent pas dépasser leur borne supérieure, s'ajoute le fait que tous les critères de ressources ne doivent pas dépasser leur limite respective spécifié dans l'instance du problème. On utilise donc la limite d'une ressource pour la valeur du critère correspondant dans le pire chemin fictif.

Après cette première phase, tout comme pour le LCDPF classique, on obtient :

- Une solution supportée V_s^n qui correspond au chemin optimal entre le noeud s et t qui minimise une combinaison linéaire des critères.
- Pour chaque noeud, une collection V_i^n correspondant à un chemin réalisable du noeud i à la destination
- Les bornes supérieures des critères de coûts
- Les bornes inférieures pour chaque critère

Ces informations vont permettre, comme pour le LCDPF classique, d'accélérer la seconde phase de l'algorithme. En effet, on sera capable dans un premier temps de mettre à jour le front de Pareto pendant la recherche. On utilise pour cela, la collection de chemin réalisable de n'importe quel noeud au noeud destination t . Si un chemin partiel P_i depuis un noeud s à i est trouvé alors une collection de chemin du noeud s à t peut être déduite tel qu'on ajoute les valeurs des critères du chemin de s à i avec ceux de i à t . Cela nous donne un chemin réalisable, on peut donc mettre à jour le front de Pareto.

Dans un second temps, on peut empêcher la propagation de certains labels. On utilisant les bornes inférieures déterminées par la phase de preprocessing on peut déterminer un chemin idéal fictif d'un noeud i à t . Ainsi, si un chemin partiel de s à i est trouvé alors il est possible de déterminer le meilleur chemin possible pour aller de i à t . Ce chemin peut ne pas être réalisable mais on ne peut pas trouver de meilleurs chemins. Il est trouvé en ajoutant les bornes inférieures trouvées pendant la phase précédente à chaque critère. Ainsi, si ce chemin est déjà dominé sur le front de Pareto, on peut arrêter l'exploration du label lié. En effet, il ne sera pas possible de trouver un meilleur chemin et donc on sera toujours dominé sur le front de Pareto. Cette technique est fréquemment utilisée dans la littérature actuelle, elle a été introduite par TUNG et CHEW, « [A multicriteria Pareto-optimal path algorithm](#) ».

2.1.2 Seconde phase : Label Correcting

Cette phase permet, comme pour le LCDPF classique, de trouver toutes les solutions non dominées d'un noeud source s au noeud destination t en utilisant un label correcting introduit par MARTINS, « [On a multicriteria shortest path problem](#) ». Le principe est le même que pour le LCDPF classique, seul la notion de label dominé sur un noeud et sur le front de Pareto change. En effet, il n'est pas question de minimiser les ressources utilisées, il est simplement question de minimiser les critères de coût et de ne pas dépasser les limites de chaque ressource. Ainsi, un label est considéré dominé (pas intéressant à explorer) si un ou plusieurs des critères de

ressource dépassent leur limite ou s'il existe un label sur le noeud (ou dans le front de Pareto) qui a tous ses critères de coût inférieurs à ceux du label en question.

L'algorithme est détaillé dans la partie suivante.

2.1.3 Algorithme

Dans un premier temps, on définit les notations suivantes valable pour tous les algorithmes de ce projet :

- l_u : un label qui représente un chemin partiel défini par la paire (i, U) où
 - i est le noeud associé au label qui correspond à un chemin partiel de s à i .
 - U représente la collection de critère associé au label. u_k représente la valeur du critère k dans la collection U . On rappelle que pour k critères on a les a premiers sont des critères de coûts et que les b suivants sont des critères de ressource. On a $a + b = k$.
- $l_v < l_u$ signifie que le label l_v est strictement dominé par le label l_u au sens de Pareto.
- l_0 est le label initial.
- LN_i est la collection de label sur le noeud i .
- Q est la queue d'exploration
- c_{ij}^l correspond au critère l sur l'arc (i, j) , on rappelle que pour k critères, les a premiers sont des critères de coûts et que les b suivants sont des critères de ressource. On a $a + b = k$.
- v_{ik}^n la valeur du critère k sur le chemin supporté créé dans la phase de preprocessing.
- MAX_k la limite pour la ressource k

Le front de Pareto (les solutions non dominées) finale est donné par LN_t .

L'algorithme peut-être expliqué à travers plusieurs étapes principales :

- **L'initialisation** : a pour but de déterminer les solutions supportées V_s^n et les ajouter au front de Pareto, de déterminer tous les chemins faisables de i à t (V_t^n), et initialiser la queue d'exploration ainsi que la liste des labels associés à chaque noeud.
- **La sélection du prochain label à traiter** : à chaque itération de l'algorithme on va vouloir traiter un nouveau label. Pour cela on fait appel à la fonction GET_NEXT_LABEL . Cette fonction va choisir un label l_u dans Q à traiter. Plusieurs stratégies ont été étudié par les auteurs de GIRET, KERGOSIEN, NERON et SAUVANET, « [An efficient label-correcting algorithm for the multi objective shortest path problem](#) » et la plus intéressante (celle que la MOE a également adopté) est la suivante : on va choisir le label le plus prometteur concernant les critères de coûts. On ne tiens pas compte des ressources.
- **Créer les labels** : depuis un noeud i et un label l_u , un nouveau label $l_{u'}$ est créé pour chaque successeur de i . On construit un nouveau label en utilisant les valeurs des critères de l'arc reliant i et un successeur. Dans notre cas, on a k critères, les x premiers sont des critères de coûts et les y suivants sont les ressources. Pour construire le nouveau label on ajoute, pour chaque critère, la valeur sur le label à celle sur l'arc.
- **Evaluer les labels** : une fois le label créé on veut s'assurer qu'il est prometteur et qu'il est important de l'explorer. Pour cela on fait appel à la fonction $IS_PROMISING$. Cette fonction va créer un chemin fictif idéal de s à t en utilisant $l_{u'}$, en effet $l_{u'}$ représente un chemin de s à un successeur de i appelé j et on peut donc estimer un chemin idéal en utilisant les données de la phase de preprocessing de s à j et de j à t . Ce chemin est en théorie le meilleur possible (il n'est souvent pas réalisable) ainsi, la fonction va vérifier si le label créé associé à ce chemin fictif est dominé sur le front de Pareto et s'il l'est il n'est pas intéressant de considérer $l_{u'}$ car n'importe quel label découlant de celui-ci sera dominé sur le front de Pareto.
- **Ajouter un label sur un noeud** : chaque label $l_{u'}$ est ajouté à LN_j associé au noeud j si et seulement si il n'est pas dominé sur ce noeud et qu'aucun de ses critères de ressource ne

Algorithme 1 : LCDPF pour le problème du plus court chemin avec contraintes de ressources

```

Data :  $G = (V, A), s, t$ 
Result :  $LN_t$  l'ensemble des labels non dominés représentant un chemin de  $s$  à  $t$ 
RUN_FIRST_PHASE()
/* Détermine  $V_s^n, V_i^n$  pour tout  $i$ , les bornes supérieures et les bornes inférieures de chaque critère */
 $l_0 = (s, \{0, \dots, 0\})$ 
 $LN_i \leftarrow \emptyset \forall i \in V, i \neq t$ 
 $LN_s \leftarrow \{l_0\}$ 
 $LN_t \leftarrow LN_t \cup (t, V_s^n) \forall n \in \{1, \dots, K\}$ 
 $Q \leftarrow \{l_0\}$ 
while  $Q \neq \emptyset$  do
   $l_u(i, U) \leftarrow \text{GET\_NEXT\_LABEL}(Q)$ 
   $Q \leftarrow Q \setminus S$ 
  for all  $(i, j) \in A$  do
    /* Création du nouveau label */
     $U' \leftarrow \{u_1 + c_{ij}^1, \dots, u_k + c_{ij}^k\}$ 
     $l_{u'} \leftarrow (j, U')$ 
    if  $\text{IS\_PROMISING}(l_{u'})$  then
      /* Si le label créé n'est pas dominé sur le noeud et qu'aucun des critères de ressource
      ne dépasse sa limite */
      if  $\neg \exists u'_i \in [u'_a, \dots, u'_k] \subset U' / u'_i > \text{MAX}_i \wedge \neg \exists l_v \in LN_j / l_v < l_{u'}$  then
         $LN_j \leftarrow LN_j \cup l_{u'}$ 
        if  $j \neq t$  then
          /* On ajoute le label créé dans la queue d'exploration et on purge les labels
          dominés par  $l_{u'}$  sur le noeud  $j$  */
           $Q \leftarrow Q \cup l_{u'}$ 
           $S \leftarrow \text{PURGE}(LN_j, l_{u'})$ 
           $Q \leftarrow Q \setminus S$ 
          for all  $n$  tuples in  $W' \subseteq W$  do
            /* Mise-à-jour dynamique du front de Pareto */
             $U'' \leftarrow \{u''_1 + v_{j1}^n, \dots, u''_k + v_{jk}^n\}$ 
             $l_{u''} \leftarrow (t, U'')$ 
            if  $\neg \exists u''_i \in [u''_a, \dots, u''_k] \subset U'' / u''_i > \text{MAX}_i \wedge \neg \exists l_v \in LN_t / l_v < l_{u''}$  then
               $LN_t \leftarrow LN_t \cup l_{u''}$ 
               $\text{PURGE}(LN_t, l_{u''})$ 
  return  $LN_t$ 

```

Algorithme 2 : IS_PROMISING

```

Data :  $l_u = (i, U)$ 
Result : True si le label est prometteur, False sinon
 $U' \leftarrow \{u_1 + v_{i1}^{n_1}, \dots, u_k + v_{ik}^{n_k}\}$ 
 $l'_u \leftarrow (t, U')$ 
/* Si le nouveau label n'est pas dominé sur le front de Pareto et qu'aucun de ces critères de
ressource ne dépasse sa limite */
if  $\neg \exists u'_i \in [u'_a, \dots, u'_k] \subset U' / u'_i > \text{MAX}_i \wedge \neg \exists l_v \in LN_t / l_v < l_{u'}$  then
  return True
return False

```

dépasse leur limite. S'il est ajouté alors on s'assure d'enlever les labels déjà dans LN_j qui sont dominés par $l_{u'}$ (fonction PURGE).

- **Mettre à jour le front de Pareto dynamiquement :** cette étape (lignes x à y) est une étape cruciale pour accélérer la recherche. En effet, l'algorithme crée un nouveau chemin du noeud s à t en utilisant l'_u et V_i^n . Ce nouveau chemin correspond à un chemin faisable de s à t . Ce chemin est alors ajouté au front de Pareto, s'il n'est pas dominé et qu'aucun des critères de ressource n'est dépassé, afin de faire converger le front de Pareto courant vers le front de Pareto final plus facilement. Une fois le label ajouté on appelle la fonction PURGE afin d'enlever du front de Pareto les labels dominés par le nouveau label ajouté.
- **Notion de dominance :** on dit qu'un label l_u est dominé par l_v si tous les critères de coûts de l_v sont meilleurs que ceux de l_u .

3 Les tests

Après l'algorithme élaboré par la MOE il a fallu s'assurer que celui-ci donnait les résultats escomptés. Pour cela, la MOE a dans un premier temps testé l'algorithme à la main sur des petits graphes. Cette méthode est cependant limitée car il se peut que certains comportements non désirés apparaissent sur des grands graphes dû aux nombres importants de labels créés et à l'importance de la phase de preprocessing qui augmente avec la taille d'un graphe. Ainsi, afin de tester plus précisément notre algorithme, nous avons récupéré des graphes pour lesquels les résultats sont connus pour des instances données, récupéré l'implémentation en Java de citelozano2013exact et comparé les résultats obtenus. Nous avons obtenus la même solution optimale à chaque fois. Nous n'avons cependant pas pu comparer le chemin associé à cette solution. En effet, il n'est pas possible d'effectuer un backtracking afin de retrouver le chemin d'une solution mais nous avons jugé cette comparaison suffisante.

4 Les résultats

Le but de ce projet était de s'inspirer du LCDPF afin de pour résoudre d'autre problème du plus court chemin et de comparé nos performances avec la littérature actuelle. Pour cela, il a été décidé, au tout début du projet, de comparer notre algorithme avec l'algorithme de ces deux papiers : HORVATH et KIS, « [Solving resource constrained shortest path problems with LP-based methods](#) » et LOZANO et MEDAGLIA, « [On an exact method for the constrained shortest path problem](#) ».

La MOE a dans un premier temps comparé son implémentation à HORVATH et KIS, « [Solving resource constrained shortest path problems with LP-based methods](#) » sur les instances suivantes :

Instance	Set	Res	Noeuds min	Noeuds max	Arcs min	Arcs max
D1	Dumitrescu and Boland	1	10 002	135 002	29 900	404 850
D2	Dumitrescu and Boland	1	625	40 000	2400	159 200
S	Santos et al.	1	10 000	40 000	15 000	800 000

Figure 1 – Instances utilisées pour Howard and Kis

Ces instances ont été récupéré directement par la MOE en les demandant directement aux auteurs de HORVATH et KIS, « [Solving resource constrained shortest path problems with LP-based methods](#) ». Afin de comparer les performances de ces deux algorithmes on compare leur temps d'exécution pour résoudre les mêmes instances.

Identifiant	Horvath and Kis	Adaptation LCDPF
D1-L-1	0.5	0.130
D1-L-2	64.57	0.6
D1-L-3	65.82	0.9
D1-L-4	721.17	1.930
D1-M-1	2.05	0.33
D1-M-2	47.7	1.44
D1-M-3	101.02	2.850
D1-M-4	403.86	5.280

Figure 2 – Tableau comparant le temps d'exécution en secondes sur D1

Il est important de mentionner qu'il n'a pas été possible de comparer directement les résultats obtenus par ces deux algorithmes. Il n'a donc pas été possible de vérifier si on obtenait les

mêmes résultats que l'autre algorithme. Cependant, pour s'assurer d'une résolution cohérente, on a vérifié que chaque solution obtenue ne dépasser aucune des limites de ressources spécifiées dans l'instance.

Les résultats sont très encourageant pour ces instances. On remarque une grande différence entre le temps d'exécution de l'adaptation du LCDPF et l'algorithme de Horvath et Kis. Cette différence est très significative, au maximum on atteint un facteur d'amélioration d'environ 380. Ici encore,

Identifiant	Horvath et Kis	Adaptation LCDPF
D2-L-1	0.01	0.01
D2-L-2	0.25	0.05
D2-L-3	0.75	0.1
D2-L-4	3.17	0.272
D2-L-5	9.13	0.432
D2-L-6	32.27	0.552
D2-L-7	599.98	0.755
D2-M-1	0.04	0.012
D2-M-2	0.34	0.06
D2-M-3	3.01	0.135
D2-M-4	11.91	0.457
D2-M-5	19.56	0.667
D2-M-6	43.14	0.88
D2-M-7	105.07	1.195

Figure 3 – Tableau comparant le temps d'exécution en secondes sur D2

on peut lire une différence impressionnante entre les deux algorithmes. L'algorithme développé par la MOE est bien meilleur. Bien entendu, il faut tenir en compte certain facteur comme par exemple la machine sur lesquels les tests ont été faits ou encore la qualité de la solution avant de se prononcer, mais avec un facteur d'amélioration aussi important il est quand même raisonnable d'affirmer que nous sommes meilleurs sur les instances de Dumitrescu et Boland.

Noeuds	Arcs	Horvath et Kis	Adaptation LCDPF
10 000	75 000	0.07	0.154
10 000	100 000	0.1	0.186
10 000	200 000	0.15	0.279
20 000	100 000	0.1	0.229
20 000	200 000	0.20	0.38
20 000	300 000	0.33	0.437
20 000	400 000	0.34	0.640
40 000	60 000	0.11	0.279
40 000	100 000	0.19	0.390
40 000	200 000	0.3	0.515
40 000	400 000	0.42	0.690
40 000	600 000	0.67	1.049
40 000	800 000	0.77	1.101

Figure 4 – Tableau comparant le temps d'exécution en secondes sur Santos et al.

En ce qui concerne les instances de Santos et al., il n'est pas évident de se prononcer. En effet, si on étudie les chiffres précisément on remarque que Horvath et Kis sont légèrement meilleurs à

chaque fois. Cependant, toutes les instances sont globalement résolues en moins d'une seconde, il est donc difficile de se prononcer sur les performances d'un algorithme à cette échelle.

MOE a ensuite comparé ses résultats avec LOZANO et MEDAGLIA, « *On an exact method for the constrained shortest path problem* » sur les instances suivantes :

Instance	Set	Ressources	Noeuds min	Noeuds max	Arcs min	Arcs max
rscp	Zhu and Wilhelm	1 ou 10	100	500	959	48868
S	Santos et al.	1	10 000	40 000	15 000	800 000

Figure 5 – Instances utilisées pour Lozano and Medaglia

Nous avons obtenu les résultats suivants. Ici encore, il est très difficile de se prononcer. On peut

Noeuds	Arcs	Lozano and Medaglia	Adaptation LCDPF
10 000	15 000	0.01	0.05
10 000	25 000	0.02	0.065
10 000	50 000	0.02	0.093
10 000	75 000	0.04	0.154
10 000	100 000	0.06	0.186
10 000	200 000	0.08	0.279
20 000	30 000	0.02	0.119
20 000	50 000	0.03	0.167
20 000	100 000	0.06	0.229
20 000	200 000	0.12	0.38
20 000	300 000	0.19	0.437
20 000	400 000	0.18	0.640
40 000	60 000	0.05	0.279
40 000	100 000	0.08	0.390
40 000	200 000	0.12	0.515
40 000	400 000	0.21	0.690
40 000	600 000	0.31	1.049
40 000	800 000	0.43	1.101

Figure 6 – Tableau comparant le temps d'exécution en secondes sur Santos et al.

dans un premier temps affirmer que notre algorithme est bon comparé à celui de Lozano et Medaglia mais dire lequel des deux est meilleur n'est pas possible. Bien entendu, si on regarde précisément les résultats on voit que Lozano et Medaglia sont meilleurs mais encore une fois, à une échelle si petite, il n'est pas pertinent de dire lequel des deux algorithmes est le meilleur.

Instance	Ressources	Noeuds	Arcs	Lozano and Medaglia	Adaptation LCDPF
rcsp4	1	100	959	0.00069	0.0046
rcsp12	1	200	1971	0.00072	0.0062
rcsp20	1	500	4978	0.00119	0.0128
rcsp8	10	100	999	0.00418	0.0454
rcsp16	10	200	1960	0.00459	0.0354
rcsp24	10	500	4868	0.01361	0.1755

Figure 7 – Tableau comparant le temps d'exécution en secondes sur Zhu and Wilhelm.

On remarque la même chose que pour les tests précédents en ce qui concerne les instances de Zhu et Wilhelm. Bien que la différence entre les deux algorithmes paraisse importante, nous

sommes encore sous la barre de la seconde. Il n'est donc pas possible de tirer de conclusion. On peut simplement dire que notre algorithme est aussi bon que celui de Lozano et Medaglia.

4.1 Approche s'inspirant du LCDPF pour résoudre le problème du plus court chemin avec des critères de type coût et de type minMax

Comme précisé dans le gaant figure x, le second algorithme implémenté est celui qui va résoudre le problème du plus court chemin avec des critères de type coût et une fonction objectif de type minMax, c'est-à-dire qu'on va chercher à minimiser le coût maximum dans le chemin. On rappelle que d'après Hansen, il est possible de considérer également une fonction objectif de manière équivalente maximisant le coût minimum du chemin, maxMin. La différence avec le LCDPF classique est qu'au lieu de traiter seulement un seul type de fonction objectif, nous en traitons deux. Une fonction qui va avoir pour objectif de minimiser le coût global d'un chemin concernant a critères, et une fonction qui va avoir pour objectif de minimiser le coût maximal d'un chemin concernant b autres critères.

Pour pouvoir s'attaquer au problème du plus court chemin avec a fonctions de coût et b fonction mixMax, il a fallu changer quelques informations comparé au problème concerné par le LCDPF classique. En effet, nous allons rajouter sur chaque arc des informations. Un arc contient maintenant a critères de coût et b critères qui seront utilisés par la fonction minMax.

Les parties suivantes décrivent un algorithme qui s'inspire du LCDPF permettant de résoudre le problème du plus court chemin multi-objectif avec b fonctions minMax.. Cet algorithme a été élaboré par la MOE avec l'aide de Yannick Kergosien.

L'algorithme, tout comme le LCDPF, est séparé en deux phases. Dans un premier temps, une phase de préprocessing qui va permettre d'obtenir des informations sur les graphes afin de réduire le nombre de labels à explorer (faire converger le front de Pareto plus rapidement) pendant la seconde phase. Cette dernière correspond à la résolution du problème avec un label correcting classique en s'appuyant sur les informations récupérées pendant la première phase.

4.1.1 Première phase : Le préprocessing

Cette phase de préprocessing est très similaire à celle présentée pour le premier algorithme. En effet, elle a pour but de trouver les bornes inférieurs et supérieurs pour les différents critères et de calculer des chemins partiels de n'importe quel noeud au noeud de destination t . Elle fonctionne de la même manière que la phase de pré processing de l'algorithme précédent. Cependant, parce qu'on a différents types de fonctions objectifs, certaines modifications quant à la résolution de plusieurs problème du plus court chemin mono-objectif de s à t et de n'importe quel noeud i à t sur le graphe inversé.

Dans un premier temps, il n'est pas possible de considérer les critères de type minMax comme des critères de coût. Ainsi, l'algorithme de Dijkstra utilisé pour résoudre les problèmes du plus court chemin mono-objectif sur le graphe inversé a dû être légèrement modifié. En effet, jusqu'ici l'algorithme de Dijkstra calculé la valeur du critère sur un noeud en ajoutant la valeur du critère sur le noeud précédent et celle de l'arc. Cependant, cela n'est pas applicable au minMax. Nous allons à la place calculer la valeur du critère sur un noeud en prenant le maximum entre la valeur de ce critère le noeud précédent et l'arc.

Ensuite, la notion de label intéressant a dû être changé. On rappelle qu'un label est considéré intéressant s'il est meilleur qu'un pire chemin fictif créé pendant la résolution des problèmes du plus court chemin mono-objectif en récupérant les pires valeurs trouvées pour chaque critère.

La pire valeur d'un critère minMax est la valeur maximum trouvée pour ce critère, toutes résolutions confondues.

Après cette phase, on obtient :

- Une solution supportée V_n qui correspond au chemin optimal entre le noeud s et t qui minimise une combinaison linéaire des critères.
- Pour chaque noeud, une collection V_n correspondant à un chemin réalisable de noeud i à t .
- Les bornes de chaque critère.

De la même manière que pour l'algorithme précédent, ces informations vont permettre d'accélérer la seconde phase de l'algorithme. En effet, on pourra ainsi mettre à jour dynamiquement le front de Pareto courant afin qu'il converge vers le front final plus rapidement et empêcher la propagation de certains labels pas intéressants.

4.1.2 Seconde phase : Label Correcting

Cette phase permet, de la même manière que l'algorithme précédent, de trouver toutes les solutions non dominées d'un noeud source s au noeud destination t en utilisant un label correcting classique. L'algorithme est détaillé dans la partie suivante.

4.1.3 Algorithme

Pour cet algorithme on reprendra les définitions faite précédemment. On modifiera cependant les éléments suivant :

- c_{ij}^l correspond au critère l sur l'arc (i, j) , on rappelle que pour K critères, les a premiers sont des critères de coûts et que les b suivants sont des critères minMax. On a $a + b = K$.

Cet algorithme est très similaire à celui décrit précédemment.

L'algorithme peut être expliqué à travers plusieurs étapes :

- **L'initialisation et la sélection du prochain label à traiter** restent identiques à l'algorithme précédent.
- **Créer les labels** : depuis un noeud i et un label l_u , un nouveau label l'_u est créé pour chaque successeur de i . On construit un nouveau label en utilisant les valeurs des critères de l'arc reliant i et un successeur. Dans notre cas, on a k critères, les a premiers sont des critères de coûts et les b suivants sont les critères minMax. Afin de construire le label l'_u , chaque critère de coût est égal à la valeur de ce critère sur l_u ajoutée à celle sur l'arc. Pour chaque critères minMax, on prend le maximum entre la valeur sur l_u et celle de l'arc.
- **Evaluer les labels** : une fois le label créé on veut s'assurer qu'il est prometteur et qu'il est nécessaire de l'explorer. Pour cela, on appelle la fonction IS_PROMISING. Comme pour l'algorithme précédent, cette fonction va créer un chemin fictif idéal de s à t en utilisant l'_u et V_i^n . Ce chemin est en théorie le meilleur possible depuis le label l'_u . Ainsi, si ce meilleur chemin est dominé sur le front de Pareto, il n'est pas intéressant de considérer l'_u .
- **Ajouter les labels sur le noeud et la mise-à-jour dynamique du front de Pareto** fonctionnent de la même manière que pour l'algorithme précédent.
- **Notion de dominance** : on dit qu'un label l_u est dominé par l_v si tous les critères de l_v sont meilleurs que ceux de l_u . C'est-à-dire que les valeurs des critères de coût et minMax de l_v soient plus petits que ceux de l_u .

Algorithme 3 : LCDPF pour le problème du plus court chemin multi-objectifs avec minMax

Data : $G = (V, A), s, t$
Result : LN_t l'ensemble des labels non dominés représentant un chemin de s à t
RUN_FIRST_PHASE()
 /* Détermine V_s^n, V_i^n pour tout i , les bornes supérieurs et les bornes inférieurs de chaque critère */
 $l_0 = (s, \{0, \dots, 0\})$
 $LN_j \leftarrow \emptyset \forall i \in V, i \neq t$
 $LN_s \leftarrow \{l_0\}$
 $LN_t \leftarrow LN_t \cup (t, V_s^n) \forall n \in \{1, \dots, K\}$
 $Q \leftarrow \{l_0\}$
while $Q \neq \emptyset$ **do**
 $l_u(i, U) \leftarrow \text{GET_NEXT_LABEL}(Q)$
 $Q \leftarrow Q \setminus S$
 for all $(i, j) \in A$ **do**
 /* Création du nouveau label */
 $U' \leftarrow \{u_l + c_{ij}^l \mid \forall l \in [1, a]\}$
 $U' \leftarrow U' \cup \{\max(u_l, c_{ij}^l) \mid \forall l \in [a, K]\}$
 $l_{u'} \leftarrow (j, U')$
 if $\text{IS_PROMISING}(l_{u'})$ **then**
 /* Si le label créé n'est pas dominé sur le noeud */
 if $\neg \exists l_v \in LN_j / l_v < l_{u'}$ **then**
 $LN_j \leftarrow LN_j \cup l_{u'}$
 if $j \neq t$ **then**
 /* On ajoute le label créé dans la queue d'exploration et on purge les labels dominés par $l_{u'}$ sur le noeud j */
 $Q \leftarrow Q \cup l_{u'}$
 $S \leftarrow \text{PURGE}(LN_j, l_{u'})$
 $Q \leftarrow Q \setminus S$
 for all n **tuples in** $W' \subseteq W$ **do**
 /* Mise-à-jour dynamique du front de Pareto */
 $U'' \leftarrow \{u_1'' + v_{j1}^n, \dots, u_k'' + v_{jk}^n\}$
 $l_{u''} \leftarrow (t, U'')$
 if $\neg \exists l_v \in LN_t / l_v < l_{u''}$ **then**
 $LN_t \leftarrow LN_t \cup l_{u''}$
 $\text{PURGE}(LN_t, l_{u''})$
 return LN_t

5 Les tests

Plusieurs tests ont été effectués sur cet algorithme afin de prouver et vérifier son fonctionnement. Dans un premier temps, lors de son élaboration, la MOE, avec l'aide de Yannick Kergosien, s'est assurée que chaque étape aller faire converger le front de Pareto courant de l'algorithme vers le front de Pareto final pour ainsi trouver toutes les solutions non dominées. Dans un second temps, l'algorithme a été testé à la main sur des petites instances afin de vérifier son fonctionnement. Enfin, on a exécuté l'algorithme sur des plus grosses instances dont on avait une approximation de la taille de front de Pareto mais les résultats n'étaient pas concluant avec plus de 1 critère de chaque type.

6 Les résultats

Au début de ce projet, nous avons décidé de prendre IORI, MARTELLO et PRETOLANI, « [An aggregate label setting policy for the multi-objective shortest path problem](#) » et PINTO et PASCOAL, « [On algorithms for the tricriteria shortest path problem with two bottleneck objective functions](#) » afin de comparer nos résultats. Il n'a pas été possible de récupérer les instances utilisées par PINTO et PASCOAL, « [On algorithms for the tricriteria shortest path problem with two bottleneck objective functions](#) » car les auteurs n'avaient pas gardé les instances qu'ils avaient utilisé.

Cependant, la MOE a récupéré les graphes et instances utilisées par IORI, MARTELLO et PRETOLANI, « [An aggregate label setting policy for the multi-objective shortest path problem](#) ».

De nombreux graphes ont été générés par les auteurs. Ils ont joués sur certains paramètres afin d'obtenir des graphes différents :

- Le nombre de noeuds $n \in [50, 100, 200]$
- La densité $d \in [5\%, 10\%, 20\%]$
- Les types de coût C correspondant à un intervalle pour la valeur des critères de coût.
- Le nombre de fonctions de coût s
- Le nombre de fonctions minMax b

Il n'est pas intéressant pour nous de lancer notre algorithme sur tous les graphes créés, ainsi nous avons décidé de n'utiliser qu'un sous-ensemble des graphes créé par IORI, MARTELLO et PRETOLANI, « [An aggregate label setting policy for the multi-objective shortest path problem](#) ». Nous avons fixé C .

n	d	s	b	Lori, Martello et Pretolani	Adapation LCDPF
50	20	1	1	< 2	0.0003
100	20	1	1	< 2	0.01
200	20	1	1	< 2	0.041

Figure 8 – Instances utilisées et résultats en secondes pour Iori, Martello et Pretolani

Après avoir lancé l'algorithme sur des graphe et des instances différentes, on arrive rapidement à une conclusion. On peut voir que l'algorithme fonctionne correctement quand il y a un seul critère de chaque type. Nous sommes aussi bon que IORI, MARTELLO et PRETOLANI, « [An aggregate label setting policy for the multi-objective shortest path problem](#) ». Cependant, une fois qu'on passe à plusieurs critères de chaque type, le nombre de labels créés augmente de manière exponentielle et l'algorithme prend bien plus longtemps à résoudre le problème. Il est difficile de trouver une raison à cela. Cela peut venir de plusieurs choses. Il est possible que l'implémentation de la MOE ne soit pas correcte (bien que testée à plusieurs reprise mais que sur des petits graphes). En effet, il est suspecté que la phase de preprocessing ne se fasse pas correctement du à l'incapacité par la MOE de modifier l'implémentation du Dijkstra mono-objectif implémenté par la librairie Boost Malheureusement, par manque de temps, la MOE n'a pas pu explorer plus de pistes afin de résoudre ce problème.

7

Bilan et conclusion

1 Bilan du projet après la partie de recherche

La MOE a terminé la première partie de ce projet. Nous rappelons que cette partie consistait à spécifier la problématique du PRD et de rechercher certains aspects du problème du plus court chemin. Elle a donné lieu à la rédaction d'un cahier de spécification et d'un état de l'art. Le cahier de spécification présente le contexte de la réalisation du projet et une description générale. L'état de l'art fait l'état des algorithmes existants solvant plusieurs problèmes : le problème du plus court chemin avec contraintes de ressources et le problème du plus court chemin multi-objectifs. Enfin, un rendez-vous a été pris avec la MOA afin de débiter la partie développement. Comme expliqué plus tôt, la partie développement commence avec une phase de recherche pendant laquelle la MOE et la MOA travailleront ensemble afin d'élaborer des algorithmes s'inspirant de GIRET, KERGOSIEN, NERON et SAUVANET, « [An efficient label-correcting algorithm for the multi objective shortest path problem](#) ».

2 Etapes à venir après la partie de recherche

La prochaine étape pour la MOE est la présentation du travail effectué jusqu'à présent lors d'une soutenance. La prochaine étape dans l'avancement de ce projet est la modification de GIRET, KERGOSIEN, NERON et SAUVANET, « [An efficient label-correcting algorithm for the multi objective shortest path problem](#) » afin de pouvoir résoudre le problème du plus court chemin à contraintes de ressources. Un rendez-vous est prévu avec la MOA afin de réfléchir sur les premières modifications à apporter.

3 Bilan du projet final

Après plus de 7 mois de travail, ce projet touche à sa fin. Cette partie détaille l'état du projet, son futur et ce qu'il a apporté à la MOE. On rappelle que l'objectif de ce projet de recherche et développement était de s'inspirer du LCDPF, algorithme développé par la MOA pour résoudre le problème du plus court chemin multi-objectif avec fonction de coût, afin de résoudre deux autres problèmes du plus court chemin. Cet objectif a été partiellement atteint. En effet, la

MOE a dans un premier temps implémenté un premier algorithme qui résout le problème du plus court chemin multi-objectif avec contraintes de ressources. Ses performances comparées à HORVATH et KIS, « [Solving resource constrained shortest path problems with LP-based methods](#) » et LOZANO et MEDAGLIA, « [On an exact method for the constrained shortest path problem](#) » sont très encourageantes. Dans un second temps, la MOE a implémenté un second algorithme pour résoudre le problème du plus court chemin multi-objectif avec fonctions minMax. Ses performances comparées à IORI, MARTELLO et PRETOLANI, « [An aggregate label setting policy for the multi-objective shortest path problem](#) » ne sont pas aussi bonnes car la MOE n'a pas été en mesure de modifier suffisamment l'implémentation de Dijkstra par Boost pour effectuer correctement la phase de preprocessing.

Certaines améliorations plus complexes n'ont pas été appliquées aux algorithmes de la MOE par manque de temps. On peut alors imaginer relancer ce projet dans un futur proche afin d'améliorer les implémentations existantes. En effet, cela serait facilement possible car la MOE s'est assurée que la qualité du code et la documentation était suffisante pour une reprise éventuelle du projet.

Enfin, la MOE a identifié des pistes d'améliorations à partir de l'état actuelle du projet. Dans un premier temps, il serait très intéressant de récupérer le code utilisé par les auteurs des papiers auxquels nous nous sommes comparés et exécuter leur algorithme sur notre machine. Cela résoudrait le problème de l'environnement différent et permettrait d'obtenir des résultats plus précis. Enfin, le problème concernant l'implémentation de l'algorithme pour résoudre le problème de plus court chemin multi-objectif avec minMax a été clairement identifié par la MOE. Nous utilisons l'implémentation de Dijkstra pour effectuer le preprocessing. Celle-ci n'est pas adaptée pour résoudre les mono-critères avec une fonction objectif minMax. Il faut changer la façon dont cette implémentation examine un arc.

4 Bilan personnel

Ce projet fût, d'un point de vue personnel, particulièrement intéressant et enrichissant. En effet, le but étant d'essayer de rivaliser avec les meilleurs algorithmes de la littérature actuelle, il était très motivant de travailler à améliorer les performances de mes algorithmes. De plus, réfléchir avec mon encadrant sur les méthodes à utiliser pour résoudre les problèmes était très motivant. Ensuite, j'ai beaucoup appris au sujet de la résolution de problème du plus court chemin. En effet, bien que cela avait été introduit dans le cadre de ma formation, ce PRD m'a permis d'améliorer mes connaissances des algorithmes existants dans la littérature. De plus,

Annexes

A

Spécifications

1 Diagramme de cas d'utilisation

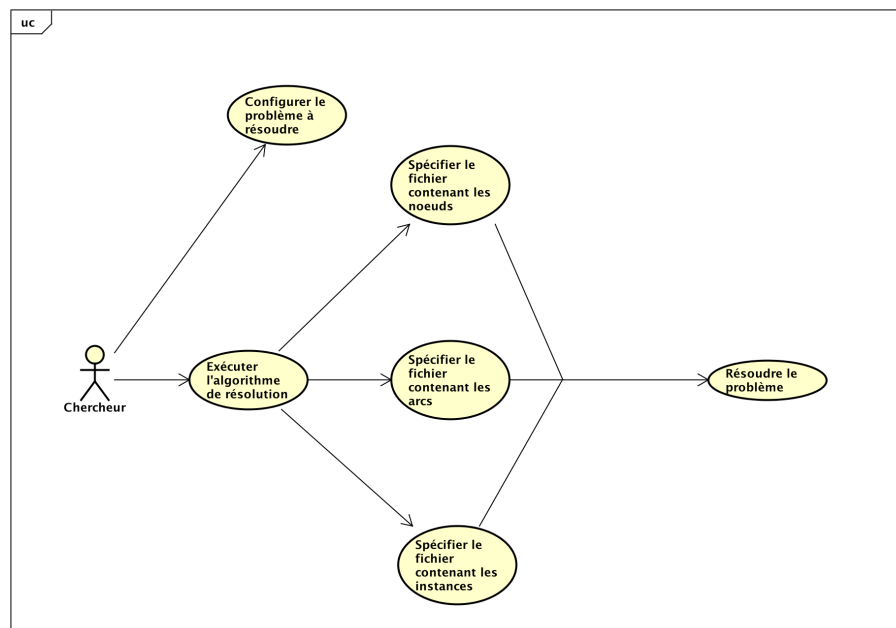


Figure 1 – Diagramme de cas d'utilisation

Le but de ce projet étant de comparer les performances d'algorithmes, les cas d'utilisation sont limités et les types d'utilisateurs sont limités. Dans un premier temps, il est possible de configurer le problème à résoudre. C'est-à-dire, qu'il est possible de changer certains paramètres (nombre de critères de chaque type, nombre de problème du plus court chemin mono-critère à résoudre etc.) afin de cibler un problème du plus court chemin particulier. Ensuite, en lançant le programme il sera nécessaire de fournir les fichiers appropriés pour le problème. Enfin, la résolution du problème donnera lieu à un affichage des performances.

2 Diagramme de classe

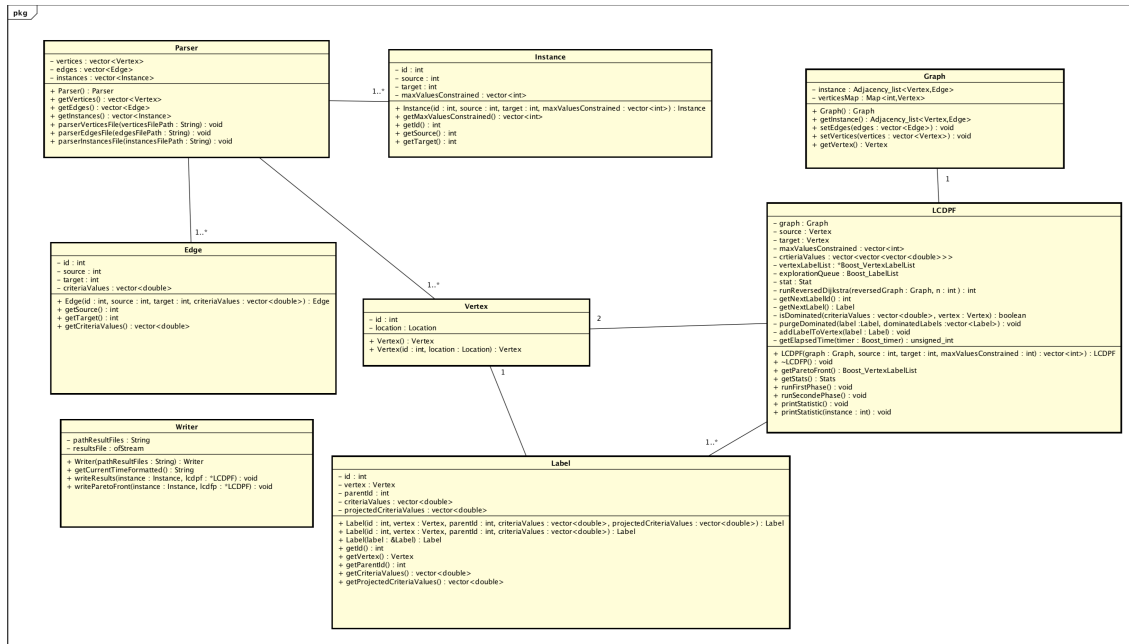


Figure 2 – Diagramme de classes

Les classes ci-dessous correspondent à l'implémentation finale du projet. La MOE a conservé la plus part des éléments précédemment implémentés par Antoine Giret à l'exception de :

- La classe instance a été modifié pour permettre de récupérer les valeurs maximales des critères de ressource.
- Le parser et le writer ont été modifiés pour permettre de configurer leur comportement en fonction du problème du plus court chemin à résoudre.
- La classe LCDPF, classe principale de l'implémentation, a été modifié afin de pouvoir contenir les valeurs maximales des critères de ressource. Certaines fonctions comme isDominate, purgeDominate, LCDPF et runSecodePhase ont également été modifiées.

B

Gestion de projet

1 Gestion du temps

Il est intéressant de comparer le diagramme de Gaant qui avait été construit au début du projet en prévision des tâches à venir et celui qui relate réellement ce qu'il s'est passé. En effet, si on compare le premier Gantt 1 (Chapitre 5) avec celui-ci 1 il est possible d'en tirer les conclusions suivantes :

- Le cycle de développement 1 a été plus long que prévu. En effet, la rédaction du cahier de spécification et la prise en main de l'existant a pris plus de temps que prévu.
- Les cycles suivant ont donc dû être raccourci.
- La récupération des graphes et instances pour comparer les algorithmes a été faite assez tardivement et aurait dû avoir été fait dès le commencement des cycles de développement et non à la fin une fois l'algorithme implémenté.

2 Méthode de gestion de projet utilisée

Pour gérer ce projet, la MOE a mis en place une méthode Agile et Scrum car elle correspondait le plus au projet. En effet, une méthode agile se définit comme une manière de développer un logiciel :

- De manière itérative et incrémentale
- S'appuyant sur une documentation minimale mais pratiquant une collaboration poussée entre la MOE et la MOA
- Avec l'objectif de répondre, dans un court délai, aux besoins du client qui peuvent changer
- Laissant une grande flexibilité au développeur

Parce que ce projet est un projet de recherche, la méthode agile s'est imposée comme étant la mieux adaptée. En effet, le développement devait se faire de manière itérative. Il fallait dans un premier temps développer un algorithme résolvant le problème, puis l'améliorer pour avoir les meilleures performances possibles. Il était plus facile et utile de privilégier les rendez-vous avec la MOA que d'écrire une documentation extensive les algorithmes étant élaborés de manière collaborative entre la MOE et la MOA. De plus, le délai était assez court et les besoins clients pouvaient changer. En effet, le client aurait très bien pu réorienter le projet si l'adaptation du LCDPF se trouvait infructueuse.

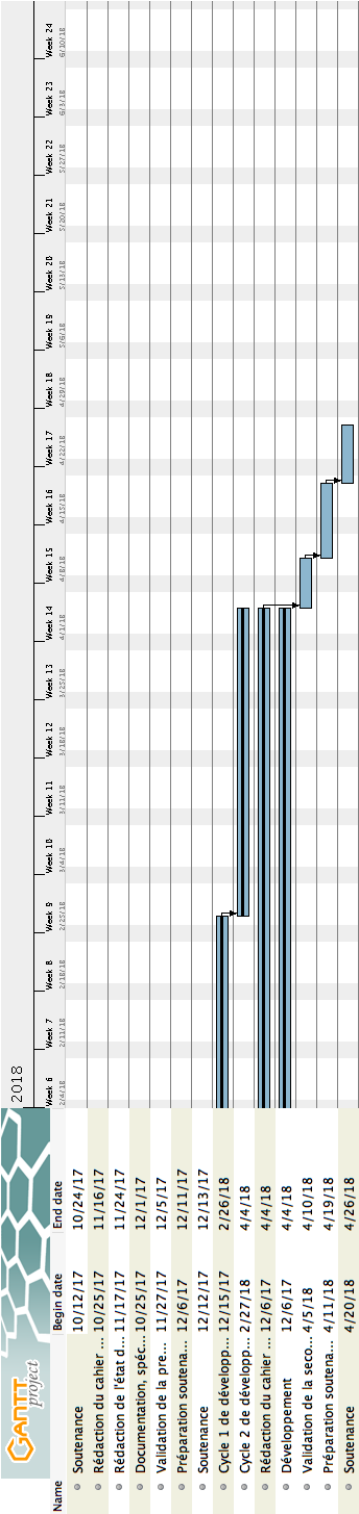


Figure 1 – Gantt apres la mise en oeuvre

3 Identification des tâches

3.1 Analyse de faisabilité

Encore une fois, le projet étant un projet de recherche, la contrainte de temps était particulière. En effet, nous ne pouvions pas vraiment prévoir comment l'algorithme allait performer pour chaque problème. Pour s'assurer de respecter la contrainte de temps, le projet a été divisé en deux phases et chaque phase a été donnée une contrainte de temps. Dans un premier temps, la MOA et la MOE ont concentré leurs efforts sur l'adaptation du LCDPF pour qu'il puisse résoudre le problème du plus court chemin avec contraintes de ressource. Cette phase comprenait l'élaboration de l'algorithme, l'implémentation, les tests et les comparaisons avec la littérature. Nous avions prévu 8 semaines pour cette phase. Elle a pris en réalité 10 semaines car la prise en main de l'existant a pris plus de temps que prévu. Dans un second temps, la MOA et la MOE se sont concentrés sur l'adaptation du LCDPF pour résoudre le problème du plus court chemin multi-objectif, notamment avec fonctions additives et minMax. Cette phase s'est déroulée comme la précédente. 6 semaines étaient prévues mais en réalité cette phase a duré 5 semaines.

Le client n'a pas modifié les objectifs du projet. Cependant, il était question au début du projet de pousser un peu plus l'optimisation de l'algorithme si on avait le temps. Malheureusement, la phase de rédaction du cahier de spécifications ayant pris plus de temps que prévu nous n'avons pas pu essayer des modifications de l'algorithme plus complexes.

3.2 Analyse de risque

Ce projet comportait deux risques principaux. Le premier risque, qui était aussi une des premières tâches du projet, concernait la reprise de l'implémentation du LCDPF par Antoine Giret. En effet, cette implémentation en C++ utilisait la librairie boost qui n'avait été utilisée auparavant par la MOE. Ainsi, nous n'étions pas certains que la MOE puisse reprendre cette implémentation. Heureusement, tout s'est bien passé mais en cas d'échec nous avions prévu que la MOE implémente elle-même le LCDPF avec le langage de son choix. Cela aurait eu pour conséquence de raccourcir grandement les deux phases de développement et les performances de l'algorithme auraient donc pu être impactées. Le deuxième risque concernait les comparaisons avec les algorithmes de la littérature actuelle. En effet, afin de pouvoir comparer notre implémentation avec les algorithmes de la littérature actuelle il était nécessaire de récupérer les données utilisées par les autres papiers de recherche. Il y avait une possibilité que ces chercheurs n'aient pas gardé leurs données et que nous puissions donc pas nous comparer à eux. Heureusement nous avons pu récupérer suffisamment de données. En cas d'échec nous avions prévu de créer nos propres données afin d'avoir des résultats. Nous aurions pu ensuite essayer de récupérer le code des autres algorithmes afin de les implémenter et les tester localement avec les données créées.

3.3 Suivi de projet

Le suivi de l'avancée du projet s'est fait de plusieurs manières. D'une part, une réunion avec la MOA était faite à chaque étape importante atteinte par la MOE. Par exemple, pour le problème du plus court chemin avec contraintes de ressources des rendez-vous étaient pris après chacune des étapes suivantes :

- Elaboration de l'algorithme

- Implémentation en c++
- Tests
- Comparaison avec papier n°1
- Comparaison avec papier n°2

De plus, le laboratoire informatique de Tours était dans le même bâtiment que la MOE, des réunions spontanés ont eu lieu en cas de problèmes : difficultés de lecture des données récupérées, problème sur l'algorithme etc.

Enfin, afin que l'encadrant du PRD puisse avoir un rapport hebdomadaire, un document partagé sur Google Drive a été créé. Ce document contient le résumé de chacune des séances de PRD.

4 Git

L'utilisation de Git dans le cadre de ce projet a été très bénéfique. En effet, le versioning géré avec Git a permis d'essayer d'optimiser les performances des algorithmes sans perdre trop de temps. Après la première implémentation d'un algorithme, la MOE essayait d'optimiser le code afin de maximiser les performances. Pour cela, chaque modification était suivie de tests de performance puis, si la modification ne s'avérait pas pertinente celle-ci était supprimée à l'aide d'un retour à la version précédente via Git.

De plus, avec Bitbucket, qui est un service web d'hébergement et de gestion de développement logiciel utilisant Git, le projet a pu être sauvegardé sur un serveur distant afin d'éviter toutes pertes potentielles dû à une panne matériel. Ce service fait également office de cahier de développeur, car on peut y retrouver toutes les modifications que la MOE a apporté avec un commentaire approprié.

5 Trello

Trello a été utilisé afin de d'organiser les tâches de la MOE et d'avoir une vue globale du projet à tout instant donné. Par exemple, il a été très utile quand il a fallu décider si la MOE continuait d'améliorer l'algorithme pour résoudre le problème du plus court chemin à contraintes de ressources ou commencer à développer le second algorithme. La quantité de tâches restantes par rapport à la limite de temps imparti était importante, alors il a été décidé de commencer le développement du second algorithme et considérer le premier terminé.



Comptes rendus hebdomadaires

Compte rendu n°1 du 11-10-2017

Premier rendez-vous avec la MOA, représentée par Monsieur Kergosien, membre du laboratoire informatique de Tours. Nous avons défini le projet, la MOE a pu poser des questions quant à la rédaction du rapport de PRD. La MOA a également fourni des points d'entrées documentaires à la MOE concernant les algorithmes existant de la littérature actuelle ainsi qu'un papier de recherche sur le LCDPF.

Compte rendu n°2 du 12-10-2017

La MOE a entamé l'étude du LCDPF et a mis en place des outils de gestion de projet.

Compte rendu n°3 du 18-10-2017

Prise de note sur le LCDPF. La rédaction du rapport a été entamée : introduction, contexte, enjeux, etc. J'ai également recherché plus d'algorithmes concernant le problème du plus court chemin multi-objectifs.

Compte rendu n°4 du 19-10-2017

L'étude du LCDPF est terminée, j'ai débuté l'étude des algorithmes concernant le problème du plus court chemin à contraintes de ressources. J'ai également pris rendez-vous avec Monsieur Kergosien pour le 25-10-2017

Compte rendu n°5 du 25-10-2017

Réunion avec Monsieur Kergosien : explication de certains détails sur le LCDPF, clarification quant à la lecture des tableaux de résultats et l'écriture de l'état de l'art

Compte rendu n°6 du 26-10-2017

Rédaction du cahier de spécification en suivant les consignes données pendant la réunion avec Monsieur Soukhal

Compte rendu n°7 du 8-11-2017

Rédaction de l'état de l'art, ajout de la bibliographie et étude des papiers de recherche concernant le problème du plus court chemin multi-objectifs et à contraintes de ressources

Compte rendu n°8 du 9-11-2017

Rédaction de l'état de l'art, ajout de la structure et des fonctionnalités du système dans le cahier de spécification

Compte rendu n°9 du 16-11-2017

Réunion avec Monsieur Kergosien afin de discuter de l'implémentation en C++ du LCDPF, lui faire valider la structure de mon rapport et de discuter des prochaines semaines de projet à venir

Compte rendu n°10 du 23-11-2017

Correction du cahier des spécifications sous les conseils de Monsieur Soukhal

Compte rendu n°11 du 29-11-2017

Compréhension du code d'Antoine Giret (LCDPF), changement du diagramme de classe, rédaction des parties tests et résultats de l'état de l'art

Compte rendu n°12 du 6-12-2017

Finalisation de la bibliographie, de l'état de l'art, du cahier des spécifications et prise de rendez-vous avec Monsieur Kergosien pour qu'il valide mon rapport.

Compte rendu n°13 du 7-12-2017

Rendez-vous avec Monsieur Kergosien, le rapport est validé. Finalisation de l'état de l'art et préparation de la soutenance.

Compte rendu n°14 du 13-12-2017

Installation de toutes les dépendances du projet d'Antoine Giret, mise-en-place du git et configuration du projet sous xCode.

Compte rendu n°15 du 20-12-2017

Exécution du programme d'Antoine Giret avec plusieurs instances données (Paris, etc.). Planification des modifications à effectuer sur le LCDPF via Trello.

Compte rendu n°16 du 10-1-2018

Compréhension en détails de toutes les fonctionnalités et structures de données. Implémentation de la première phase du preprocessing. Modification du parser.

Compte rendu n°17 du 17-1-2018

J'ai fini l'implémentation des deux phases de preprocessing pour le LCDPF à contraintes de ressources.

Compte rendu n°18 du 18-1-2018

J'ai testé les deux phases de preprocessing à la main sur des petits graphes. J'ai débuté l'implémentation de la phase de label correcting en modifiant dans un premier temps isDominated et purgeDominated.

Compte rendu n°19 du 24-1-2018

Réorganisation du code afin d'éviter d'avoir du code redondant. Ajout de documentation et création d'un script python pour modifier les instances fournies par Antoine Giret pour le LCDPF à contraintes de ressources.

Compte rendu n°20 du 25-1-2018

Réunion avec l'encadrant pour décider de comment tester l'algorithme pour contraintes de ressources plus en détails. J'ai débuté les tests avec des instances récupérées en ligne. Il a été nécessaire de changer le format des instances/graphes afin que notre parser puisse les utiliser.

Compte rendu n°21 du 31-1-2018

J'ai testé l'algorithme à la main. J'ai rencontré quelques problèmes liés au parser. En effet, un noeud a une longitude et latitude dans notre programme mais pas sur les instances récupérées.

Compte rendu n°22 du 7-2-2018**Compte rendu n°23 du 8-2-2018**

Fin des tests sur le LCDPF à contraintes de ressources, on commence à regarder pour le minMax/maxMin.

Compte rendu n°24 du 14-2-2018

J'ai réfléchi sur papier au fonctionnement du LCDPF avec minMax. J'ai également regardé les algorithmes existant dans la littérature afin de bien comprendre le fonctionnement du minMax.

Compte rendu n°25 du 21-2-2018

Rendez-vous qualité logiciel, début de l'implémentation du minMax.

Compte rendu n°26 du 22-2-2018

Rendez-vous avec encadrant pour réfléchir ensemble au problème minMax.

Compte rendu n°27 du 28-2-2018**Compte rendu n°28 du 1-3-2018**

J'ai testé l'adaptation du LCDPF pour le minMax à la main et sur de petites instances.

Compte rendu n°29 du 7-3-2018

J'ai récupéré et transformé les instances/graphes des autres papiers pour tester avec nos adaptations du LCDPF.

Compte rendu n°30 du 14-3-2018

J'ai obtenu les résultats de nos algorithmes sur les instances/graphes des autres papiers. Les résultats sont prometteurs pour les contraintes de ressources mais moins pour le minMax. Il est possible que la phase de preprocessing ne fasse pas bien pour le minMax.

Compte rendu n°31 du 15-3-2018

J'ai essayé de modifier la fonction `examine edge de boost` afin de changer le comportement de Dijkstra pour `minMax` mais la documentation n'est pas suffisant pour que je puisse changer le comportement par défaut de Boost.

Compte rendu n°32 du 21-3-2018

Ecriture du rapport et préparation de la soutenance qualité.

Compte rendu n°33 du 22-3-2018

Ecriture du rapport et préparation de la soutenance qualité.

Compte rendu n°34 du 28-3-2018

Ecriture du rapport et préparation de la soutenance PRD.

Compte rendu n°35 du 29-3-2018

Ecriture du rapport et préparation de la soutenance PRD.

Bibliographie

- BELLMAN, Richard. « On a routing problem ». In : *Quarterly of applied mathematics* 16.1 (1958), p. 87–90.
- DEMEYER, Sofie, Jan GOEDGEBEUR, Pieter AUDENAERT, Mario PICKAVET et Piet DEMEESTER. « Speeding up Martins algorithm for multiple objective shortest path problems ». In : *4OR* 11.4 (2013), p. 323–348.
- DIJKSTRA, Edsger W. « A note on two problems in connexion with graphs ». In : *Numerische mathematik* 1.1 (1959), p. 269–271.
- DUMITRESCU, Irina et Natashia BOLAND. « Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem ». In : *Networks* 42.3 (2003), p. 135–153.
- DUQUE, Daniel, Leonardo LOZANO et Andres L MEDAGLIA. « An exact method for the biobjective shortest path problem for largescale road networks ». In : *European Journal of Operational Research* 242.3 (2015), p. 788–797.
- EIGER, Amir, Pitu B MIRCHANDANI et Hossein SOROUSH. « Path preferences and optimal paths in probabilistic networks ». In : *Transportation Science* 19.1 (1985), p. 75–84.
- GANDIBLEUX, Xavier, Frederic BEUGNIES et Sabine RANDRIAMASY. « Martins algorithm revisited for multi-objective shortest path problems with a MaxMin cost function ». In : *4OR A Quarterly Journal of Operations Research* 4.1 (2006), p. 47–59.
- GARCIA, Renan. *Resource constrained shortest paths and extensions*. Georgia Institute of Technology, 2009.
- GEOFFRION, Arthur M. « Proper efficiency and the theory of vector maximization ». In : *Journal of Mathematical Analysis and Applications* 22.3 (1968), p. 618–630.
- GIRET, A, Y KERGOSIEN, E NERON et G SAUVANET. « An efficient label-correcting algorithm for the multi objective shortest path problem ». In : (2017).
- HANSEN. « Multiple criteria decision making theory and application ». In : *Lecture Notes in Economics and Mathematical Systems* 177 (), p. 109–127.
- HORVATH, Marko et Tamas Kis. « Solving resource constrained shortest path problems with LP-based methods ». In : *Computers and Operations Research* 73 (2016), p. 150–164.
- IORI, Manuel, Silvano MARTELLO et Daniele PRETOLANI. « An aggregate label setting policy for the multi-objective shortest path problem ». In : *European Journal of Operational Research* 207.3 (2010), p. 1489–1496.
- LIMA PINTO, Leizer de, Claudio Thomas BORNSTEIN et Nelson MACULAN FILHO. « UM PROBLEMA DE CAMINHO TRI-OBJETIVO ». In : ().

- LIMA PINTO, Leizer de, Claudio Thomas BORNSTEIN et Nelson MACULAN. « The tricriterion shortest path problem with at least two bottleneck objective functions ». In : *European Journal of Operational Research* 198.2 (2009), p. 387–391.
- LOZANO, Leonardo et Andres L MEDAGLIA. « On an exact method for the constrained shortest path problem ». In : *Computers and Operations Research* 40.1 (2013), p. 378–384.
- MACHUCA, Enrique et Lawrence MANDOW. « Lower bound sets for biobjective shortest path problems ». In : *Journal of Global Optimization* 64.1 (2016), p. 63–77.
- MARTINS, Ernesto Queiros Vieira. « On a multicriteria shortest path problem ». In : *European Journal of Operational Research* 16.2 (1984), p. 236–245.
- MOTE, John, Ishwar MURTHY et David L OLSON. « A parametric approach to solving bicriterion shortest path problems ». In : *European Journal of Operational Research* 53.1 (1991), p. 81–92.
- PINTO, Leizer Lima et Marta MB PASCOAL. « On algorithms for the tricriteria shortest path problem with two bottleneck objective functions ». In : *Computers and Operations Research* 37.10 (2010), p. 1774–1779.
- PUGLIESE, Luigi Di Puglia et Francesca GUERRIERO. « A reference point approach for the resource constrained shortest path problems ». In : *Transportation Science* 47.2 (2013), p. 247–265.
- RAITH, Andrea. « Speed-up of labelling algorithms for biobjective shortest path problems ». In : *Proceedings of the 45th annual conference of the ORSNZ. Auckland, New Zealand. 2010*, p. 313–322.
- SANTOS, Luis, Joao COUTINHO-RODRIGUES et John R CURRENT. « An improved solution algorithm for the constrained shortest path problem ». In : *Transportation Research Part B : Methodological* 41.7 (2007), p. 756–771.
- SEDENO-NODA, Antonio et Andrea RAITH. « A Dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem ». In : *Computers & Operations Research* 57 (2015), p. 83–94.
- SHERALI, Hanif D, Antoine G HOBEIKA et Sasikul KANGWALKLAI. « Time-dependent, label-constrained shortest path problems with applications ». In : *Transportation Science* 37.3 (2003), p. 278–293.
- TUNG, Chi Tung et Kim Lin CHEW. « A multicriteria Pareto-optimal path algorithm ». In : *European Journal of Operational Research* 62.2 (1992), p. 203–209.
- ULUNGU, E et J TEGHEM. In : *Foundations of Computing and Decision Sciences* (1995).

Plus court chemin à contraintes de ressources et multi-objectifs

Arthur CROCQUEVIEILLE

Encadrement : Yannick KERGOSIEN

Objectifs

Le laboratoire d'Informatique de Tours veut implémenter et tester de nouveaux algorithmes s'inspirant du LCDPF afin de résoudre :

- Le problème du plus court chemin avec contraintes de ressources
- Le problème du plus court chemin multi-objectifs avec minMax

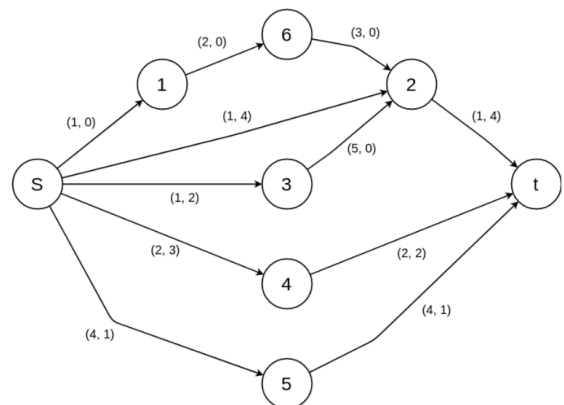


Laboratoire d'Informatique
EA 6300

Logo Laboratoire d'Informatique de Tours

Mise en œuvre

La MOE implémentera deux algorithmes, un pour chaque problème du plus court chemin. Cette implémentation se fera en C++ et donnera lieu à des tests sur des graphes et des instances connus de la littérature. On regardera surtout le temps d'exécution des algorithmes.



Graphe représentant un problème du plus court chemin

Résultats attendus

La comparaison du temps des exécutions des algorithmes s'inspirant du LCPDF aux meilleurs algorithmes de la littérature nous permettra donc conclure quant à leur efficacité.

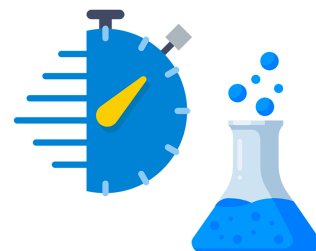


Illustration de comparaison

Plus court chemin à contraintes de ressources et multi-objectifs

Arthur CROCQUEVIELLE

Encadrement : Yannick KERGOSIEN

Objectifs

Le laboratoire d'Informatique de Tours veut implémenter et tester de nouveaux algorithmes s'inspirant du LCDPF afin de résoudre :

- Le problème du plus court chemin avec contraintes de ressources
- Le problème du plus court chemin multi-objectifs avec minMax

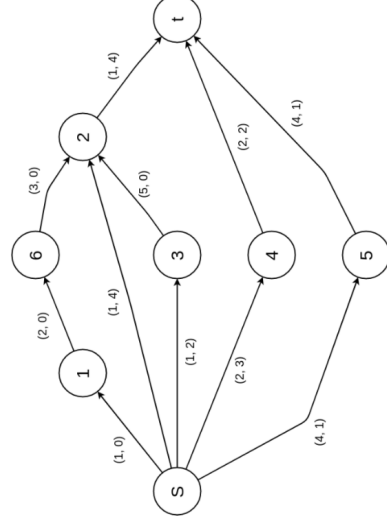


Laboratoire d'Informatique
EA 6300

Logo Laboratoire d'Informatique de Tours

Mise en œuvre

La MOE implémentera deux algorithmes, un pour chaque problème du plus court chemin. Cette implémentation se fera en C++ et donnera lieu à des tests sur des graphes et des instances connus de la littérature. On regardera surtout le temps d'exécution des algorithmes.



Graphe représentant un problème du plus court chemin

Résultats attendus

La comparaison du temps des exécutions des algorithmes s'inspirant du LCDPF aux meilleurs algorithmes de la littérature nous permettra donc de conclure quant à leur efficacité.



Illustration de comparaison

Plus court chemin à contraintes de ressources et multi-objectifs

Résumé

Ce projet de recherche et développement réalisé en 5ème année s'inscrit dans la formation dispensée à l'Ecole Polytechnique de Tours et constitue une réelle expérience en matière de conduite de projet. Ce projet s'inscrit dans la continuité d'un papier de recherche appelé "An efficient label-correcting algorithm for the multi objective shortest path problem" par Giret A. , Nero E. , Kergosien Y. et Sauvanet G. L'objectif est d'appliquer une stratégie similaire à d'autres problèmes du plus court chemin : le problème du plus court chemin à contraintes de ressources et le problème du plus court chemin multi-objectifs, afin de comparer les résultats avec les algorithmes de la littérature.

Mots-clés

Plus court chemin, Plus court chemin à contraintes de ressources, Plus court chemin multi-objectifs, LCDPF, Laboratoire d'informatique de Tours, Polytech Tours

Abstract

This research and development project, given during the fifth year of the graduate school of engineering of Tours, is a real experience of conduct of project. This project is an extension to a research paper called "An efficient label-correcting algorithm for the multi objective shortest path problem" published by Giret A. , Neron E. , Kergosien Y. et Sauvanet G. Our goal is to implement new algorithms, inspired by that paper, to solve problems like constrained shortest path problems and multi-objective shortest path problems. We want to compare our results with the state of the art algorithms existing in the literature.

Keywords

Shortest path problem, constrained shortest path problem, multi-objective shortest path problem, Polytech Tours, LCDPF, Laboratory of Computer Science of Tours