École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Rapport de projet de fin d'études

Utilisation de Landmarks pour le calcul de chemins multicritères

Encadrant
Emmanuel NÉRON
emmanuel.neron@univ-tours.fr

Étudiant
Raphaël ROGER
raphael.roger@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2013 - 2014

Version du 4 mai 2014

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens à remercier particulièrement Monsieur Emmanuel Néron pour le temps qu'il a pu m'accorder malgré ses contraintes administratives, les réunions que l'on a pu avoir ainsi que ses conseils et son écoute face à mes propositions. Je le remercie également pour son soutien, ses encouragements et sa bonne humeur lors de nos entretiens.

Je remercie également Monsieur Gaël Sauvanet pour le temps qu'il m'a accordé malgré ses contraintes professionnelles, les nombreux e-mails que l'on a pu échanger et ses conseils avisés sur ses propres travaux. Je le remercie également pour son aide sur la bibliothèque Boost, ses conseils et mises en garde ainsi que le ton cordial de nos échanges.

Je remercie aussi mes amis de promotion avec qui nous avons pu créer un système d'entraide et de soutien durant l'intégralité de ce projet.

Enfin, je remercie Monsieur Vincent T'Kindt pour ses indications quant à certaines métriques de calcul ainsi que les enseignants de Polytech Tours en général.

Introduction

Dans le cadre de ma formation d'ingénieur au sein du département Informatique de l'Ecole Polytechnique de l'Université de Tours, un projet de fin d'étude est effectué lors de la 5^{e} année. Ce projet s'est déroulé du 26 septembre 2013 au 18 avril 2014 au sein de l'école.

Ce document a pour objectif de résumer le travail effectué durant le projet, les analyses et les interprétations sur le sujet « Utilisation de Landmarks pour le calcul de chemins multicritères ».

Ce projet s'inscrit dans une problématique de théorie des graphes : la recherche de chemins multicritères.
La théorie des graphes est utilisée dans de nombreux domaines tels que la modélisation de problèmes ou dans le cadre de problèmes liés à la mobilité notamment. Dans le cas de recherches monocritères, de nombreux algorithmes existent déjà et sont très performants. Mais dans un contexte où les déplacements sont dépendants de plusieurs contraintes (sécurité, distance, rapidité, intéret, etc), il est nécessaire de définir de nouvelles procédures de calcul.

Ce projet de fin d'étude reprend une partie des travaux de la thèse de Gaël Sauvanet et a pour but d'approfondir certaines parties.

Contexte de la réalisation

1.1 Objectifs du projet

L'objectif principal de ce projet est de reprendre les travaux présentés dans la thèse « Recherche de chemins multiobjectifs pour la conception et la réalisation d'une centrale de mobilité destinée aux cyclistes. », présentée par Gaël SAUVANET [1] et dirigée par Emmanuel NÉRON et Hervé BAPTISTE. Dans cette thèse sont présentés des algorithmes et des solutions pour résoudre le problème de recherche de chemins multicritères ainsi que des pistes pour accélérer ces calculs.

Gaël Sauvanet propose de nombreuses améliorations aux algorithmes présentés et des solutions permettant d'approximer plus rapidement le front de Pareto, dans le but de réduire les temps de calcul. L'objectif étant de comprendre ses travaux pour pouvoir ensuite les implémenter en utilisant la librairie Boost.

Le second objectif est d'approfondir l'étude menée dans la thèse sur l'utilisation des Landmarks (points d'intéret), dans le calcul de solutions approchées et son utilisation dans l'approximation du front de Pareto. Cette première étude décrite dans la thèse a été réalisée dans un cadre figé sans être réellement approfondi. De nombreuses pistes sont à développer et de nouvelles méthodes peuvent être proposées.
Les différents logiciels à développer sont des logiciels scientifiques sans interface graphique particulière.

Présentation du problème

2.1 principe général

Le problème de recherche du plus court chemin est un problème classique de la théorie des graphes. Il existe pléthore d'algorithmes performants et de nombreuses variantes s'adaptent à tous types de problèmes.

Seulement, dans un contexte multiobjectif, il n'existe pas qu'une seule solution mais un ensemble de solutions de compromis, appelé front de Pareto.

Dans la thèse étudiée, Gaël Sauvanet propose des méthodes permettant de calculer des itinéraires adaptés aux cyclistes à l'échelle d'une agglomération. Le calcul de ce genre d'itinéraire prend en compte plusieurs objectifs comme la distance, l'insécurité, l'effort et l'attrait du paysage par exemple. La détermination du front de Pareto pour le problème de plus court chemin multiobjectif est un problème à complexité exponentielle. Pour rendre les temps d'exécution corrects, il est nécessaire de mettre en place des procédures de calcul rapides.

Deux approches sont abordées dans la thèse[1] pour résoudre ce problème : l'approche *a posteriori* et l'approche *a priori*.
La première consiste à calculer l'ensemble des solutions de compromis. Pour cela, on utilise une méthode classique améliorée ainsi que des prétraitements. Les améliorations proposées sont basées sur des recherches mono-objectif qui ont pour but de fixer des bornes inférieures et supérieures sur les coûts des chemins du noeud courant au noeud de destination.
La seconde approche consiste à prendre en compte les préférences de l'utilisateur pour se concentrer sur le calcul d'une solution de meilleur compromis. La méthode utilisée permet d'orienter la recherche des chemins, de façon à privilégier les sous-chemins les plus prometteurs du point de vue des préférences de l'utilisateur.

Ce projet de fin d'étude s'intéressera uniquement à l'approche *a posteriori* ainsi qu'aux méthodes proposées par Gaël Sauvanet.

2.2 Représentation des données

Au cours de ce projet, nous nous limiterons à l'étude de chemins bicritères. Les recherches à mettre en place concerneront des graphes présentés de la manière suivante :

- Un carrefour est symbolisé par un noeud. Ces noeuds peuvent être identifiés par une lettre ou un chiffre. Dans notre cas, nous les indicerons par des nombres à partir de 0.
- Une route liant deux noeuds est symbolisée par un arc pondéré d'une distance et d'une insécurité.

Afin de permettre l'utilisation de ces graphes au travers d'un programme informatique, les noeuds et les arcs d'un graphe seront modélisés dans deux fichiers séparés au format CSV. En effet, ce format est conçu pour stocker des données tabulaires où chaque ligne représente une entité d'un tableau. Dans le fichier des noeuds, chaque ligne contient donc l'indice du noeud suivi de sa longitude et de sa latitude. Le fichier des

arcs contient quant à lui l'indice du noeud initial, celui du noeud terminal, la distance et l'insécurité de l'arc. Ci-dessous, l'exemple d'un graphe et ses fichiers :

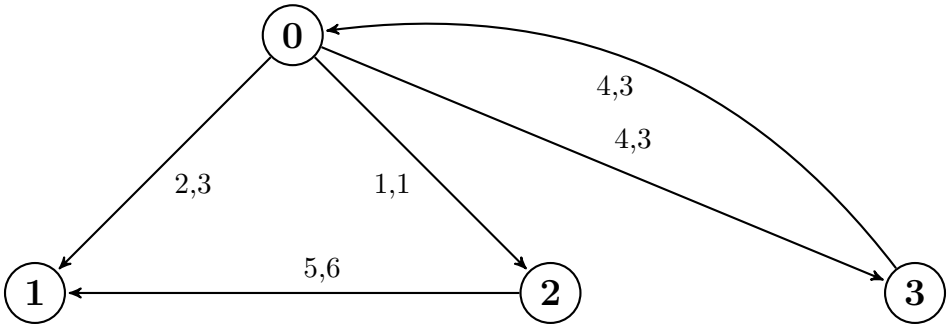

FIGURE 2.1 – Modélisation d'un graphe routier

Si nous respectons les consignes de formatage énoncées précédemment, nous obtenons le fichier de noeuds suivant :

0	$long_0$	lat_0
1	$long_1$	lat_1
2	$long_2$	lat_2
3	$long_3$	lat_3

Et le fichier des arcs :

0	1	2	3
0	2	1	1
0	3	4	3
2	1	5	6
3	0	4	3

Pour faciliter la compréhension de ce rapport et des algorithmes présentés par la suite, on utilisera le formalisme suivant.

Description d'un graphe G :

G :	le graphe
V :	l'ensemble des nœuds
A :	l'ensemble des arcs
K :	l'ensemble des objectifs
$n = \|V\|$:	le nombre de nœuds
$m = \|A\|$:	le nombre d'arcs
$q = \|K\|$:	le nombre d'objectifs
s :	le nœud de départ
t :	le nœud de destination
(i, j) :	l'arc reliant le nœud i au nœud j
c_{ij} :	le vecteur de coûts associé à l'arc (i, j)
c_{ij}^k :	le coût selon l'objectif k associé à l'arc (i, j)

Données relatives à une étiquette l :

l :	l'étiquette courante
P_l :	sous-chemin associé à l'étiquette l
d_l :	le vecteur de coûts associé à l'étiquette l
d_l^k :	le coût selon l'objectif k associé à l'étiquette l
nœud(l) :	le nœud de l'étiquette l
$l \triangleright$nœud(l) :	l'étiquette l associée au nœud(l)

Données relatives à un noeud v :

$étiquettes(v)$:	l'ensemble des étiquettes permanentes du nœud v
LB_v^k :	la borne inférieure de l'objectif k pour relier v à t
UB_v^k :	la borne supérieure de l'objectif k pour relier v à t

Les graphes étudiés sont au format CSV et séparés en deux fichiers distincts, l'un pour les noeuds et l'autre pour les arcs.

L'ensemble des tests ont été réalisés sur les graphes suivants :

- Paris : 29086 nœuds et 64538 arcs
- Berlin : 59673 nœuds et 145839 arcs
- San Francisco : 174975 nœuds et 435960 arcs

Description générale

3.1 Environnement du projet

Ce projet ne dépend pas d'un logiciel existant ou d'autres projets réalisés en parallèle. Le logiciel a été réalisé en C++ sous visual studio, avec la librairie Boost. D'un point de vue matériel, l'environnement de développement a été installé sur un ordinateur portable ordi-centre et tous les tests ont été effectués sur une seule machine pour que les performances des différents programmes puissent être correctement évaluées. D'un point de vue développement, il n'y avait aucun existant et donc toutes les structures et implémentations étaient à développer.

3.2 Caractéristiques des utilisateurs

Le logiciel à développer étant scientifique, l'utiliseur visé est un scientifique ou un chercheur. Il n'y a pas d'interface graphique et les interactions avec l'utilisateur sont inexistantes.

3.3 Contraintes de développement

Il n'y a pas de contraintes de dévelopemment liées au matériel. Cependant, la puissance de la machine pourra influencer les performances et améliorer les résultats. Une différence de performance pourra se faire sentir notamment sur de grands graphes.

Les seules contraintes de développement sont liées à la librairie imposée : Boost. Le langage de programmation sélectionné est le C++ et l'environnement de développement est Visual Studio 2012.

La librairie Boost

4.1 Présentation globale

Boost est un grand ensemble de bibliothèques logicielles libres écrites en C++. L'objectif principal de cet ensemble est de remplacer à terme la bibliothèque standard du C++. L'écriture de modules ou de fonctionnalités au sein de cette bibliothèque est soumise à un comité de lecture. La plupart des fondateurs de Boost sont aussi membres du comité du standard C++. D'ailleurs, de nombreuses parties de Boost ont été ajoutées aux standards C++.

La licence logicielle Boost autorise l'intégration de son code dans des logiciels libres comme propriétaires. Boost est une sorte de bac à sable pour les normes C++1x et de nombreuses fonctionnalités sont déjà implémentées dans la norme C++11.

FIGURE 4.1 – logo de Boost C++ libraries

Boost est composé d'un important nombre de sous-librairies et chacune répond à un besoin précis tout en restant générique, permettant son utilisation avec une grande flexibilité dans la plupart des cas.

On retrouve notamment les éléments suivants (plus d'une centaine de librairies) :
- liste et pile
- géometrie
- pointeurs intelligents
- pointeurs de fonction
- gestion simplifiée des threads
- hash
- statistique
- conversion
- ...

Au cours de ce projet, de nombreuses autres librairies que Boost.Graph ont été utilisées. Pour optimiser et aussi faciliter certaines implémentations, les modules suivants ont été installés :
- Pile de fibonacci

- Multi index container et ses sous-librairies
- Thread
- Timer
- Shared ptr

4.2 Présentation de la partie Graph

La partie graphe de Boost est aussi vaste que complexe. Il existe au sein de Boost plusieurs structures de graphes qui ne sont pas adaptées à tous les problèmes, toutes les typologies ou encore à toutes les tailles de graphe. La documentation n'est pas facile à prendre en main et les informations ne sont pas faciles à trouver. La compréhension de cette librairie a été longue mais payante au vu des résultats finaux.

Cette librairie se décompose en deux parties intéressantes pour ce projet : graph_traits et adjacency_list qui définissent la structure générale d'un graphe et une seconde partie qui définit les algorithmes principaux comme Dijkstra ou Bellman-Ford.

Les structures de graphe sont génériques et c'est au développeur de créer sa propre structure en se basant sur les types proposés par la bibliothèque. Il est important de réaliser de nombreux tests et différentes implémentations pour se rendre compte de la puissance de Boost mais aussi des structures plus gourmandes en temps et en espace mémoire. De plus, avoir une implémentation de Dijkstra par Boost permet de vérifier facilement les résultats caclulés par notre propre implémentation de Dijkstra.

Calcul de chemin bi-critère

5.1 Algorithme de Label-Setting hors amélioration

Ce projet s'articule principalement autour de l'algorithme de Label-Setting, de ses améliorations et des approximations de front de Pareto qui permettent une meilleure convergence. À la différence de l'algorithme de Dijkstra ou des algorithmes monocritère en général, le Label-Setting ne calcule pas la meilleure solution mais un ensemble de solutions de compromis. En effet, lors de la présentation des structures de nos graphes, on a souligné l'existence de deux paramètres pour la caractérisation d'un arc : la distance et l'insécurité. L'objectif de cet algorithme est alors de prendre en considération ces deux paramètres et de calculer des chemins optimisés que l'on caractérisera de « non-dominés ». En effet, au cours de cette méthode, on travaillera avec des étiquettes (*labels* en anglais) qui nous permettront de stocker la distance et l'insécurité cumulées d'un point de départ à un noeud donné sous la forme *(distance,insécurité)*. Ainsi nous pouvons obtenir plusieurs chemins caractérisés par les couples de valeurs suivants :

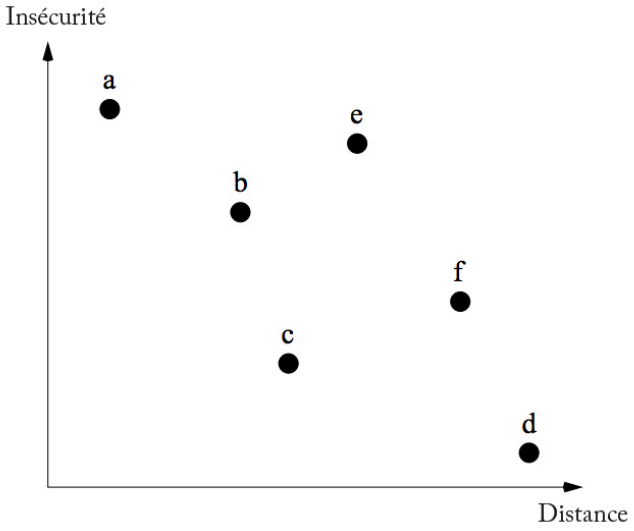

FIGURE 5.1 – Représentation des solutions bicritères

Sur le schéma ci-dessus, nous pouvons remarquer que certaines solutions sont plus intéressantes que d'autres. Par exemple, la solution e est moins bonne que la solution b pour chacun des deux critères mais il nous est impossible, à partir des deux critères, de savoir qui de a ou de b est meilleure. Dans le premier cas, on dira que e est dominée par b. Pour résumer, une étiquette (α, β) domine donc l'étiquette (γ, δ) si et seulement si on a : $\alpha \leq \gamma$ et $\beta \leq \delta$. Par la suite on notera $(\alpha, \beta) \leq (\gamma, \delta)$. Ce critère de dominance est une dominance faible, on dit alors que le noeud est faiblement Pareto-dominé. Ce critère supprime également les chemins de valeurs équivalentes et n'impacte pas l'apect du front de Pareto.

Le principe de l'algorithme de Label-Setting (*« arrangement d'étiquettes »*) consiste en une queue d'exploration (contenant des étiquettes) de laquelle on en extrait la plus faible (selon la distance). À l'initialisation, seule l'étiquette $(0, 0)$ concernant le point de départ est présente. L'étiquette la plus faible est supprimée

de la queue d'exploration et on commence alors l'étude des successeurs du noeud correspondant à cette étiquette. Pour chaque successeur, l'objectif est de calculer une nouvelle étiquette à partir de celle extraite et des informations concernant l'arc entre les deux noeuds. Si cette nouvelle étiquette n'est pas dominée pour son noeud, elle est ajoutée à la fois dans la queue d'exploration et dans la liste des étiquettes du noeud correspondant (les étiquettes désormais dominées pour ce noeud sont alors totalement supprimées). Une nouvelle étiquette est ensuite extraite de la queue d'exploration et ce processus est réitéré jusqu'à l'absence d'étiquette dans la queue. À la fin de l'algorithme, nous obtenons donc pour tous les noeuds du graphe, une liste d'étiquettes non-dominées contenant chacune la distance et l'insécurité cumulées entre le point de départ donné et ce noeud. Chaque étiquette représente donc un chemin possible entre ces deux points. Pour appuyer l'explication de cet algorithme, prenons l'exemple du graphe suivant avec le noeud 0 comme point de départ.

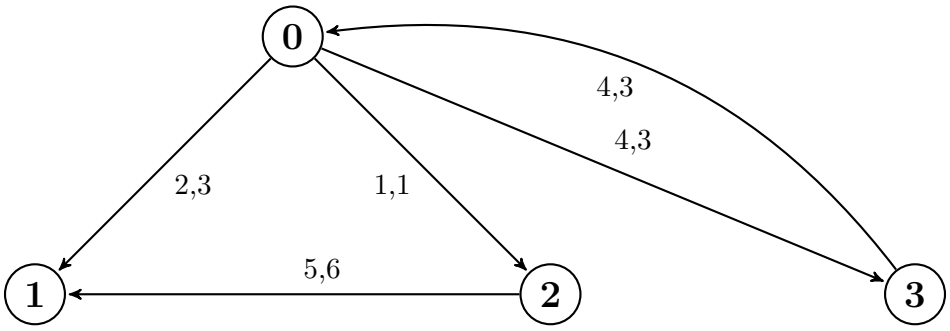

FIGURE 5.2 – Label-Setting : exemple

Tout d'abord, nous avons à l'initialisation $(0, 0)$, l'étiquette du point de départ dans la queue d'exploration et dans la liste des étiquettes de 0 (les autres noeuds ont leur liste vide). Etant la seule, cette étiquette est bien entendu la plus faible donc supprimée de la queue d'exploration. Nous analysons ensuite les arcs sortant du noeud correspondant. Nous obtenons 1, 2 et 3 d'étiquettes respectives $(2, 3)$, $(1, 1)$ et $(4, 3)$. Dans le cas où ces étiquettes ne sont pas dominées par une autre du même noeud, elles sont ajoutées à la fois à la liste du noeud adéquat et à la queue d'exploration. Cette dernière devient alors : $(2, 3) \triangleright 1$, $(1, 1) \triangleright 2$, $(4, 3) \triangleright 3$. Et pour chaque noeud, nous avons les listes suivantes :

Noeud 0 : $(0, 0)$
Noeud 1 : $(2, 3)$
Noeud 2 : $(1, 1)$
Noeud 3 : $(4, 3)$

Ensuite, l'étiquette minimale de la queue d'exploration est de nouveau extraite. Il s'agit de $(1, 1)$ appartenant au noeud 2. Si l'on calcule l'étiquette de son seul successeur (1) de la manière suivante : $(1, 1) + (5, 6) = (6, 7)$, on remarque que cette dernière est dominée par $(2, 3)$ déjà présente dans la liste du noeud 1. L'étiquette $(6, 7)$ n'est alors pas ajoutée.

On précise que la queue d'exploration contient désormais : $(2, 3) \triangleright 1$, $(4, 3) \triangleright 3$. L'étiquette minimale $(2, 3) \triangleright 1$ est extraite. Le noeud 1 correspondant à cette étiquette n'a pas d'arc sortant. On réitère alors en supprimant la dernière étiquette : $(4, 3)$ correspondant au noeud 3. Son seul successeur (0) possédant déjà l'étiquette $(0, 0)$, il n'est pas utile d'aller plus loin, car non seulement cette étiquette ne pourra être dominée et il s'agit de notre point initial.

La queue d'exploration est désormais vide, l'algorithme s'arrête. Notre liste d'étiquettes de chaque noeud (qui dans cet exemple n'a pas évoluée depuis la première extraction) nous informe des chemins optimaux (non-dominés) liant notre point de départ au noeud indiqué.

L'algorithme est donc le suivant :

Algorithm 1 Algorithme de *Label-Setting*

1: $étiquettes(i) = \phi, i = \{1, ..., n\} \backslash \{s\}$
2: $étiquettes(s) = (0, 0)$
3: $Q \Leftarrow \{(0, 0) \rhd s\}$
4: **while** $Q \neq \phi$ **do**
5: $(l_i \rhd i) \Leftarrow Q.Min()$
6: $Q \Leftarrow Q \backslash (l_i \rhd i)$
7: **for all** $(i, j) \in A$ **do** // Pour chaque arc sortant du noeud i
8: $l_j \Leftarrow l_i + c_{ij}$
9: **if** $\nexists\, l \in étiquettes(j)$ such as $l \prec_p l_j$ **then**
10: $étiquettes(j) \Leftarrow étiquettes(j) \cup l_j$ // On ajoute l'étiquette l_j
11: $Q \Leftarrow Q \cup (l_j \rhd j)$
12: **for all** $l \in étiquettes(j)$ such as $l_j \prec_p l$ **do**
13: // On supprime les étiquettes dominées
14: $étiquettes(j) \Leftarrow étiquettes(j) \backslash l$
15: **if** $(l \rhd j) \in Q$ **then**
16: $Q \Leftarrow Q \backslash (l \rhd j)$
17: **end if**
18: **end for**
19: **end if**
20: **end for**
21: **end while**

5.2 Algorithme de Label-Setting amélioré (LSAP)

L'algorithme de Label-Setting, dans sa version basique, est très long d'exécution. Ceci s'explique par son mode d'exploration et sa condition d'arrêt.

Cette version s'arrête lorsqu'il n'y a plus aucune étiquette dans la queue d'exploration et en prenant en compte la stratégie d'exploration actuelle il est possible d'étudier tous les arcs du graphe. Pour palier à cela, comme on le ferait dans une PSE classique, on va instaurer un système de bornes inférieures et de bornes supérieures. Pour cela on implémentera une variante de Dijkstra et on modifiera sensiblement l'algorithme de base.

5.3 Utilisation de Dijkstra pour le calcul des bornes

Comme dit précédemment, une variante de Dijkstra sera implémentée pour calculer les bornes inférieures (LB) et les bornes supérieures(UB) de chaque noeud.
L'inconvénient de Dijkstra est qu'il donne la distance minimale d'un noeud à tous les autres du système, tandis que nous voulons la distance de tous les noeuds à notre noeud terminal. Nous utiliserons donc les arcs entrants des noeuds dans l'exploration et le noeud de départ sera en fait le noeud terminal. Ainsi nous aurons les distances minimales de chaque noeud au noeud terminal pour chaque critère. Il faudra réaliser autant d'instances de cette version de Dijkstra qu'il existe de critères. La borne UB sera calculée en parallèle d'une borne LB d'un autre critère car on conservera le système d'étiquette pour avoir une information de la valeur cumulée de tous les critères.

Soit l'algorithme :

Algorithm 2 Algorithme de calcul des bornes LB_i^1 et UB_i^2

```
1: Q ⇐ {(0 ▷ s)}
2: LB_t^1 ⇐ 0 and LB_i^1 = +∞, i = {1, ..., n}\{t}
3: UB_t^2 ⇐ 0 and UB_i^2 = +∞, i = {1, ..., n}\{t}
4: while Q ≠ φ do
5:     (c ▷ i) ⇐ Q.Min()
6:     Q ⇐ Q\(c ▷ i)
7:     for all (i, j) ∈ A do // Pour chaque arc entrant au noeud i
8:         // Si critère 1 amélioré ou critère 1 identique mais critère 2 amélioré
9:         if (LB_i^1 + c_ji^1 < LB_j^1) ou (LB_i^1 + c_ji^1 = LB_j^1 et UB_i^2 + c_ji^2 < UB_j^2) then
10:            LB_j^1 ⇐ LB_i^1 + c_ji^1
11:            UB_j^2 ⇐ UB_i^2 + c_ji^2
12:            if j ∈ Q then
13:                Q.majPriorite(LB_j^1 ▷ j)
14:            else
15:                Q ⇐ Q ∪ (LB_j^1 ▷ j)
16:            end if
17:        end if
18:    end for
19: end while
```

Cet algorithme va nous permettre d'établir trois nouvelles propriétés pour éliminer des étiquettes en cours d'exploration ou pour enrichir le front de Pareto.

5.3.1 Elimination d'étiquettes en temps constant

Cette première règle permet d'éliminer des étiquettes en temps constant en comparant les bornes du noeud actuel aux bornes du noeud de départ en complétant les chemins partiels. Soit la règle : Si $d_l^1 + LB_i^1 > UB_s^1$ ou $d_l^2 + LB_i^2 > UB_s^2$, alors l'étiquette courante peut-être écartée.

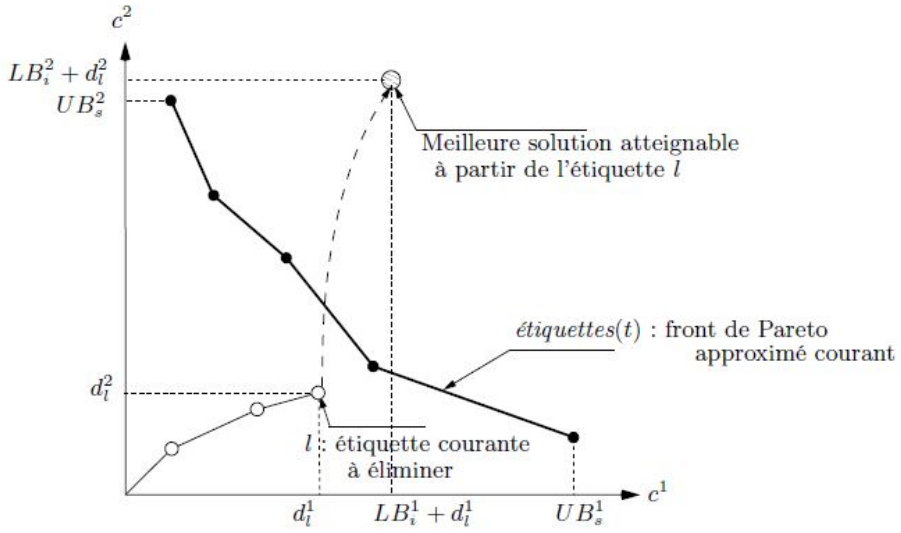

FIGURE 5.3 – Illustration de la propriété d'élimination en temps constant

5.3.2 Elimination d'étiquettes en temps variable

Cette règle d'élimination permet de supprimer des étiquettes en cours d'exploration en complétent un chemin partiel par les bornes inférieures. Ainsi, si ce chemin est dominé par le front de Pareto, il n'est pas utile de continuer l'exploration de ce noeud. Ce chemin complété n'est pas forcément réalisable mais il correspond au chemin « parfait » reliant le noeud courant au noeud terminal.

Soit la règle : $Si\ \exists l_f \in étiquettes(t)$ tel que $l_f \prec_p (d_l^1 + LB_i^1, d_l^2 + LB_i^2)$, alors l'étiquette peut être écartée.

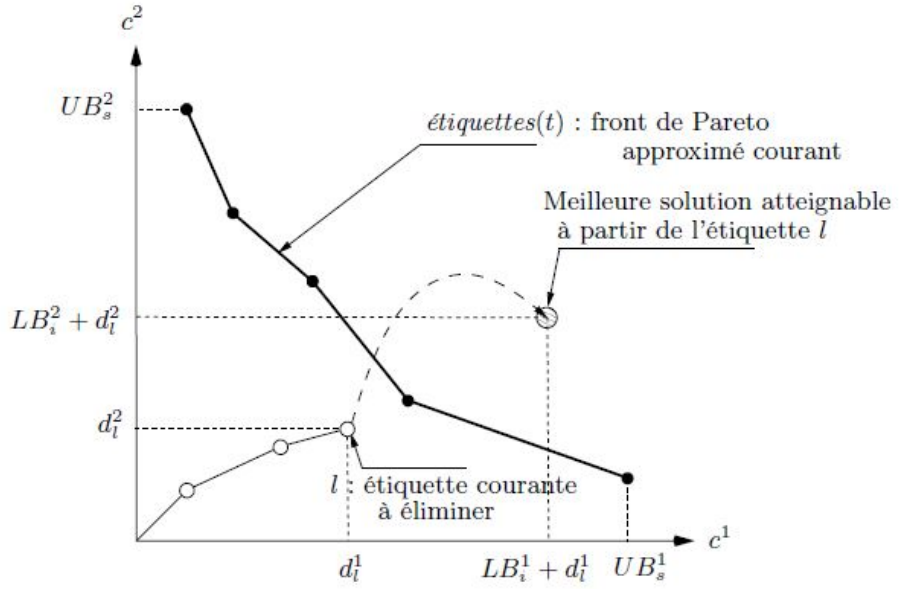

FIGURE 5.4 – Illustration de la propriété d'élimination en temps variable

Cette seconde règle englobe la première règle. Seulement, celle-ci étant plus longue en temps de trai-

tement, car il faut comparer le résultat du chemin complété à tout le front de Pareto actuel, on testera systématiquement la première règle d'abord puis en cas de retour négatif, on testera la seconde règle.

De plus, cette règle est très efficace lorsque le front de Pareto est déjà bien développé. Seulement, au départ il n'y a que deux solutions dans le front de Pareto (LB_1, UB_2) et (UB_1, LB_2) et les solutions complètes n'arrivent qu'après un certain nombre d'itérations. L'autre problème est l'ordre d'exploration lexicographique des étiquettes qui ne produit pas une construction homogènement répartie du front de Pareto.

5.3.3 Accélération de la construction du front de Pareto

Pour pouvoir tirer pleinement parti des propriétés précédentes et particulièrement de la seconde, nous allons mettre en place une technique d'approximation du front de Pareto en se servant des bornes calculées précédemment.

Pour cela, à chaque fois qu'une étiquette passera nos deux propriétés d'élimination sans être écartée, on créera deux nouvelles étiquettes $opt1$ et $opt2$ définies par : $opt1 = (LB_i^1, UB_i^2)$ le vecteur de coûts du sous-chemin reliant le noeud i au noeud t optimisant le premier critère et de manière équivalente $opt2 = (UB_i^1, LB_i^2)$ qui favorise le second critère.

Ainsi, le front de Pareto peut converger plus rapidement et permettre à la propriété 2 d'être bien plus efficace. Si les solutions $opt1$ et $opt2$ ne sont pas dominées par le front de Pareto, alors elles y sont ajoutées et on vérifie alors si elles dominent d'autres solutions. Ces chemins étant réalisables, le front de Pareto exact peut-être trouvé avant la fin de l'exécution de l'algorithme.

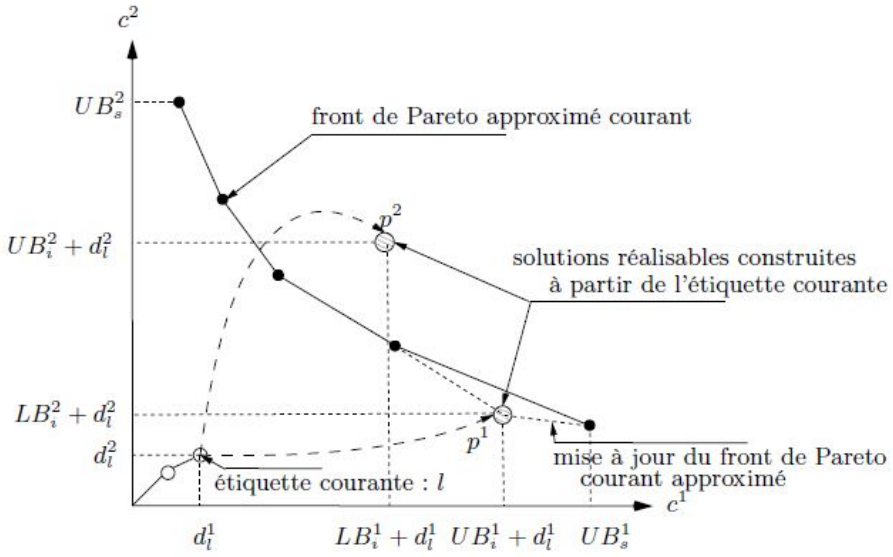

FIGURE 5.5 – Illustration de l'accélération du calcul du front de Pareto

5.4 Comparaison entre la bibliothèque Leda et Boost

Les travaux et résultats de Gaël Sauvanet ont été développés avec la bibliothèque Leda. Cependant, l'une des contraintes de ce projet était l'utilisation de Boost car elle devait être plus rapide.

Algorithm 3 Algorithme de *Label-Setting* amélioré : LSAP

1: $étiquettes(i) = \phi, i = \{1, ..., n\} \backslash \{s\}$
2: $étiquettes(s) = (0, 0)$
3: $Q \Leftarrow \{(0, 0) \rhd s\}$
4: **while** $Q \neq \phi$ **do**
5: $(l_i \rhd i) \Leftarrow Q.Min()$
6: $Q \Leftarrow Q \backslash (l_i \rhd i)$
7: **for all** $(i, j) \in A$ **do** // Pour chaque arc sortant du noeud i
8: $l_j = (d^1_{l_j}; d^2_{l_j}) \Leftarrow l_i + c_{ij}$
9: // règle 1
10: **if** $d^1_{l_j} + LB^1_j \leq UB^1_s$ **and** $d^2_{l_j} + LB^2_j \leq UB^2_s$ **then**
11: $isDominatedByJ \Leftarrow fusion(l_j, étiquettes(j))$
12: $isDominatedByT \Leftarrow false$
13: **if** $isDominatedByJ = false$ **then**
14: **for all** $l_t \in étiquettes(t)$ **do**
15: **if** $l \prec_p (d^1_{l_j} + LB^1_j, d^2_{l_j} + LB^2_j)$ **then**
16: $isDominatedByT \Leftarrow true$
17: **end if**
18: **end for**
19: **end if**
20: // règle 2
21: **if** $isDominatedByJ = false$ **and** $isDominatedByT = false$ **then**
22: $Q \Leftarrow Q \cup (l_j \rhd j)$
23: $étiquettes(j) \Leftarrow étiquettes(j) \cup l_j$
24: $l_{opt1} \Leftarrow (d^1_{l_j} + LB^1_j, d^2_{l_j} + UB^2_j)$
25: $l_{opt2} \Leftarrow (d^1_{l_j} + UB^1_j, d^2_{l_j} + LB^2_j)$
26: $opt1IsDominatedByT \Leftarrow fusion(l_{opt1}, étiquettes(t))$
27: $opt2IsDominatedByT \Leftarrow fusion(l_{opt2}, étiquettes(t))$
28: **if** $opt1IsDominatedByT = false$ **then**
29: $étiquettes(t) \Leftarrow étiquettes(t) \cup l_{opt1}$
30: **end if**
31: **if** $opt2IsDominatedByT = false$ **then**
32: $étiquettes(t) \Leftarrow étiquettes(t) \cup l_{opt2}$
33: **end if**
34: **end if**
35: **end if**
36: **end for**
37: **end while**

Malgré des difficultés importantes au départ, les implémentations sous Boost et les optimisations de code ont permis de réduire considérablement les temps de calcul. Le diagramme suivant montre les temps d'exécution des différentes classes de tests (expliquées plus loin dans le rapport) et comparent les résultats de la thèse par rapport à ceux de ce projet.

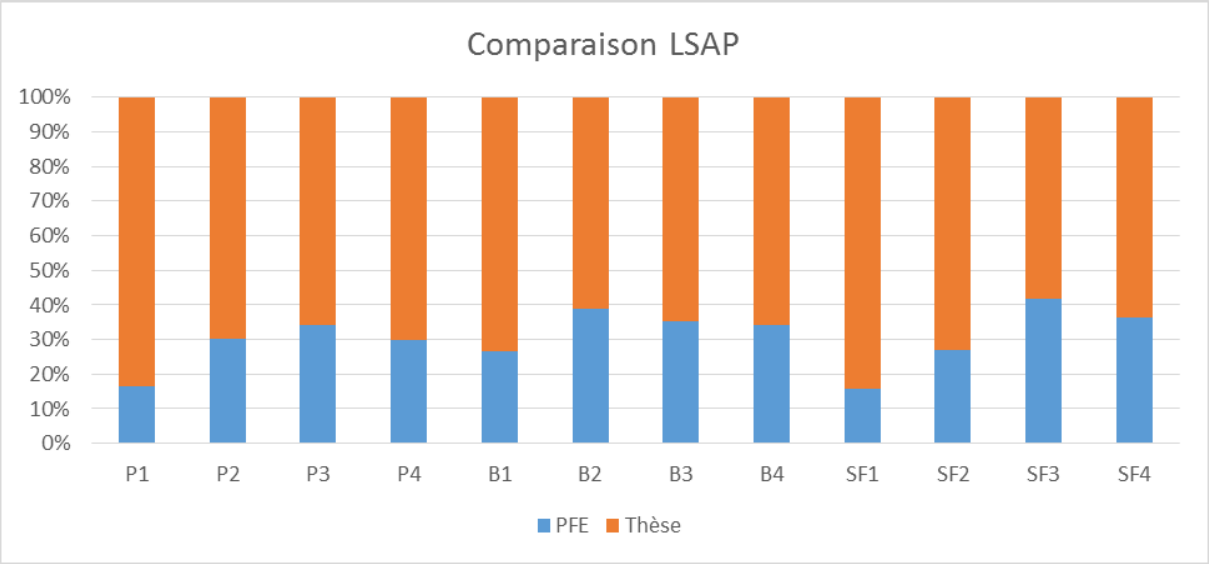

FIGURE 5.6 – Comparaison des temps de calcul entre les résultats de la thèse et ceux du PFE

Les instances de test sont presques équivalentes, certaines instances du PFE étant plus complexes que celles de la thèse et vice versa. Le nombre d'étiquettes générées pour chaques classes est éloigné au maximum de 5%, dans le positif comme dans le négatif.

L'ordinateur qui a réalisé les tests pour le PFE est équipé d'un intel celeron double coeur 2 x 1,6 GHz et l'ordinateur qui a réalisé les tests pour la thèse est équipé d'un intel double coeur 2 x 2,20 GHz. Il est utile de préciser que certaines améliorations m'ont été proposées par Gaël Sauvanet lors de nos échanges d'e-mails.

Approximation du front de Pareto par la méthode à 2 phases

6.1 Principe

L'approximation du front de Pareto est une excellente solution pour accélérer les temps de calcul. Le principe est de donner un front de Pareto approximé avant l'exécution de l'algorithme de Label-Setting pour que les propriétés d'élimination soient effectives. La première méthode implémentée et testée pour cela est inspirée de la méthode à deux phases.

La méthode originale consiste à découper le travail en deux phases : une première dans laquelle on calcule les solutions supportées, puis une seconde phase où on calcule les solutions non-supportées. Dans notre cas seule la première phase nous intéresse.

6.2 Implémentations

L'implémentation de cette méthode est guidée par les travaux de Gaël Sauvanet sur cette partie.
Pour avoir un front de Pareto le plus équilibré possible, on réalise une recherche dichotomique initialisée par les deux bornes de notre problème : (LB_s^1, UB_s^2) et (UB_s^1, LB_s^2). Le calcul des solutions se fait par une recherche monocritère où on réalise une combinaison linéaire des deux critères. Le poids des critères est attribué selon une variable (α) et $(1-\alpha)$.

A l'initialisation, on a nos deux solutions $a = (LB_s^1, UB_s^2)$ et $b = (UB_s^1, LB_s^2)$. Si une solution c est trouvée entre a et b, alors la recherche est relancée entre a et c mais aussi entre c et b.

L'inconvénient majeur de cette méthode est son temps d'exécution. Même si elle ne fait intervenir que des recherches monocritère, sur des instances complexes la durée peut être de plusieurs secondes. Pour palier à ce problème, on a développé un système de borne qui limite le nombre de solutions à calculer. Ainsi, sur de grandes instances l'algorithme s'arrête après un nombre fixe de solutions approximées.

6.3 Résultats

Pour chaque graphe testé, 200 chemins ont été choisis au hasard et ordonnés selon 4 classes en fonction du nombre de solutions trouvées pour le chemin.

Pour chaque graphe, les diagrammes suivant montrent :
- Le temps d'exécution de l'algorithme LSAP sans approximation
- Le temps d'exécution de l'algorithme LSAP avec en entrée le front de Pareto exact (préalablement calculé)
- Le temps d'exécution de l'algorithme LSAP avec en entrée un front approximé par la méthode à 2 phases, mais limité à 7 solutions
- Le temps d'exécution de l'algorithme LSAP avec en entrée un front approximé par la méthode à 2 phases, mais sans limite de solutions

Classe	Graphe	Nombre de solutions
P1	Paris	$0 \leq \|PF*\| < 50$
P2	Paris	$50 \leq \|PF*\| < 100$
P3	Paris	$100 \leq \|PF*\| < 200$
P4	Paris	$200 \leq \|PF*\|$
B1	Berlin	$0 \leq \|PF*\| < 100$
B2	Berlin	$100 \leq \|PF*\| < 200$
B3	Berlin	$200 \leq \|PF*\| < 300$
B4	Berlin	$300 \leq \|PF*\|$
SF1	San Francisco	$0 \leq \|PF*\| < 100$
SF2	San Francisco	$100 \leq \|PF*\| < 300$
SF3	San Francisco	$300 \leq \|PF*\| < 600$
SF4	San Francisco	$600 \leq \|PF*\|$

FIGURE 6.1 – Nombre de solutions non-dominées par classe d'instances

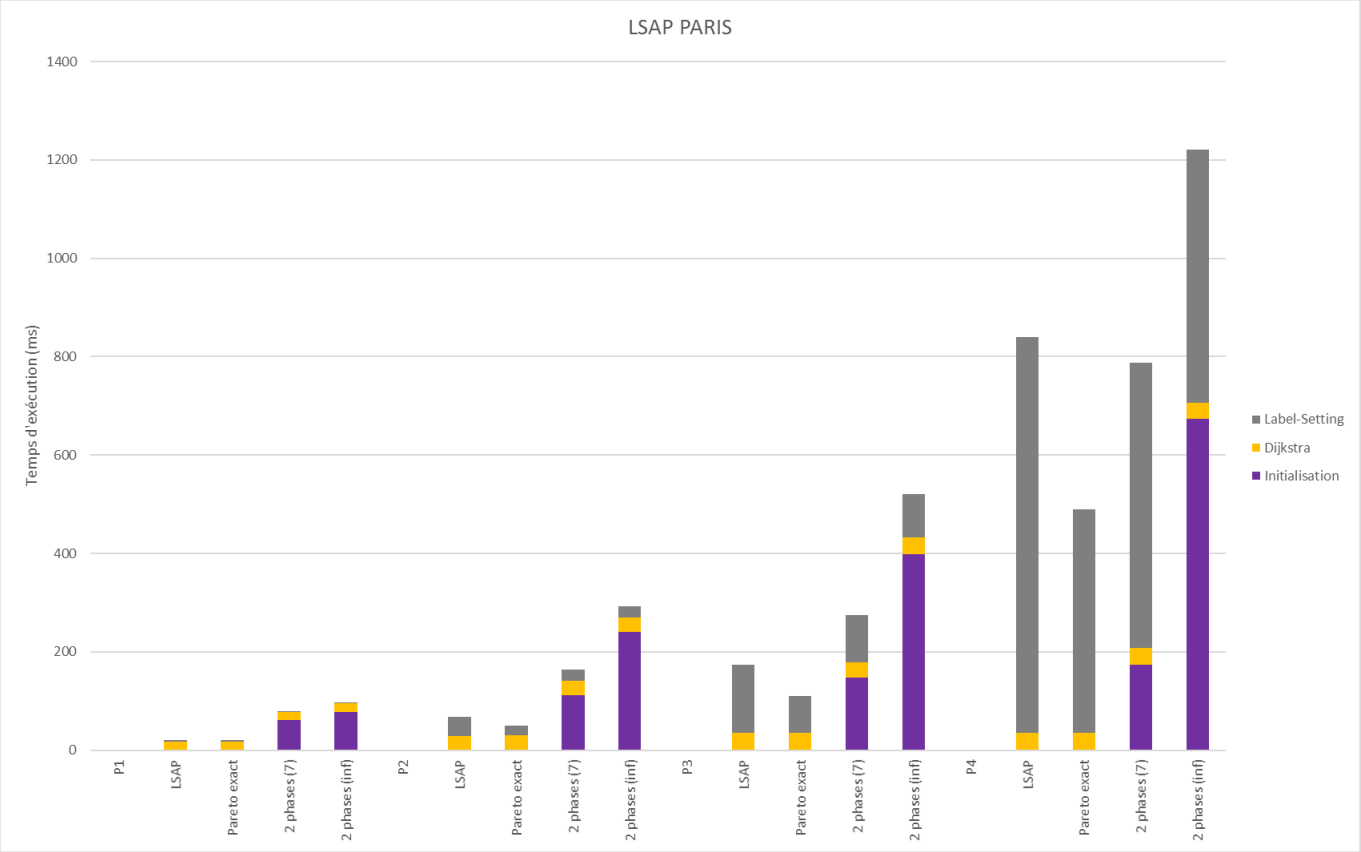

FIGURE 6.2 – Temps d'exécution pour le graphe de Paris

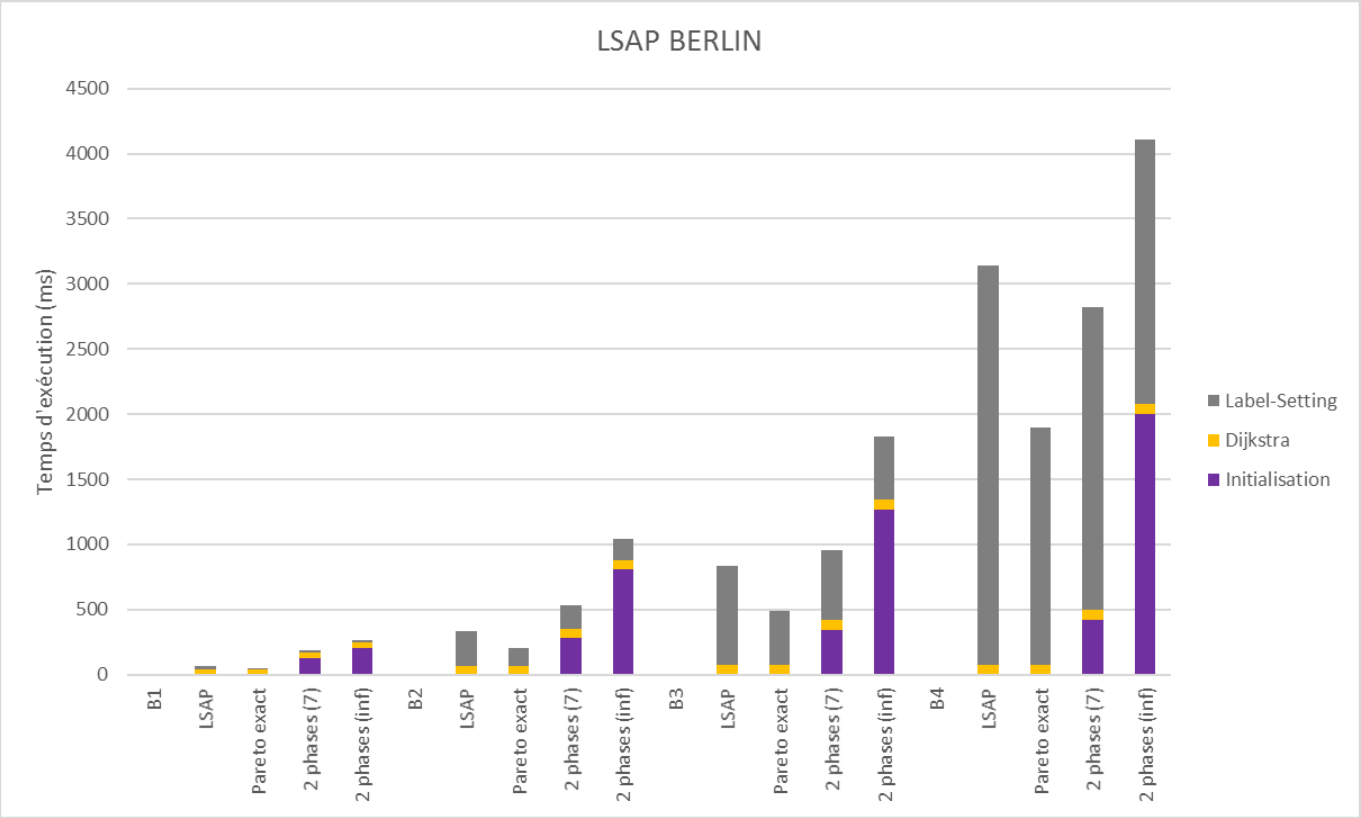

FIGURE 6.3 – Temps d'exécution pour le graphe de Berlin

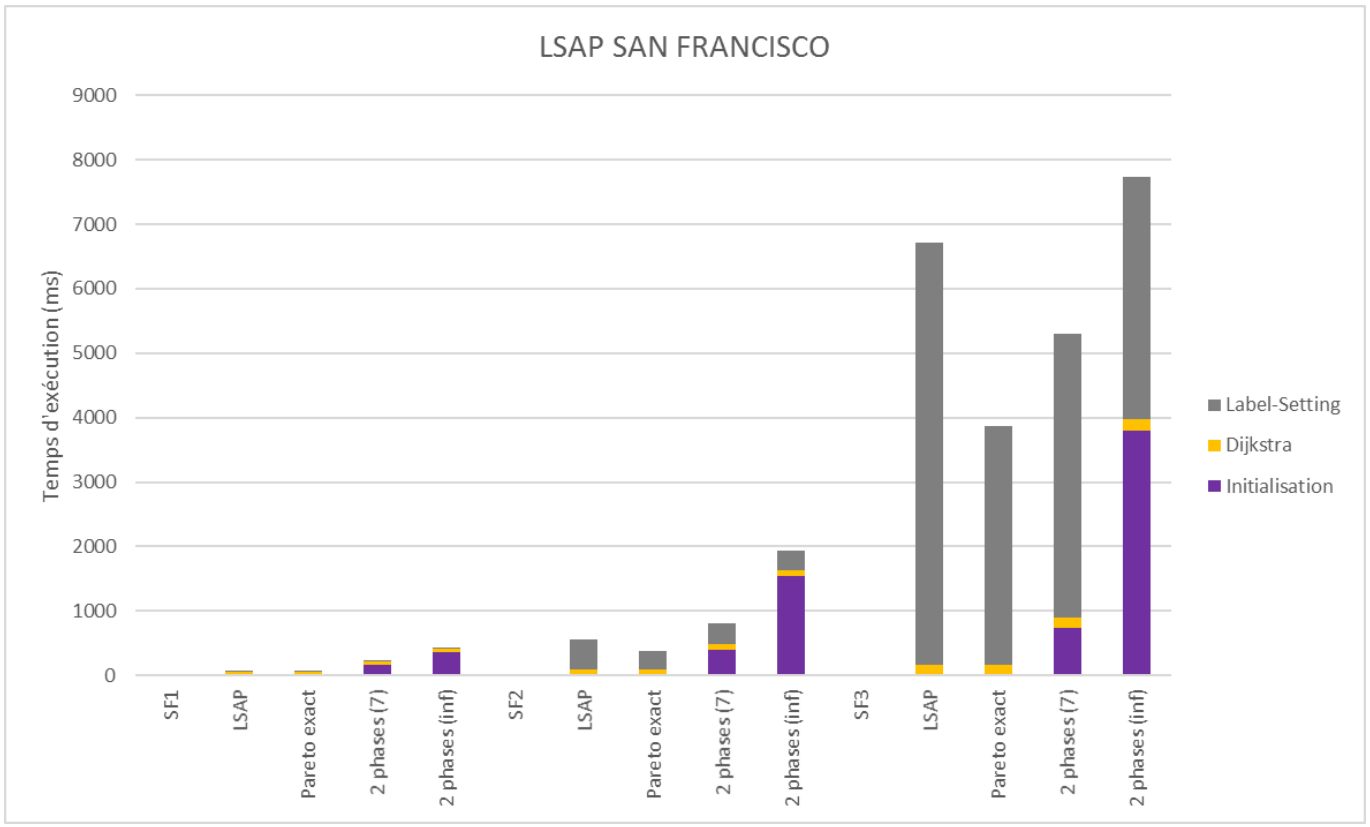

FIGURE 6.4 – Temps d'exécution pour le graphe de San Francisco : SF1, SF2 et SF3

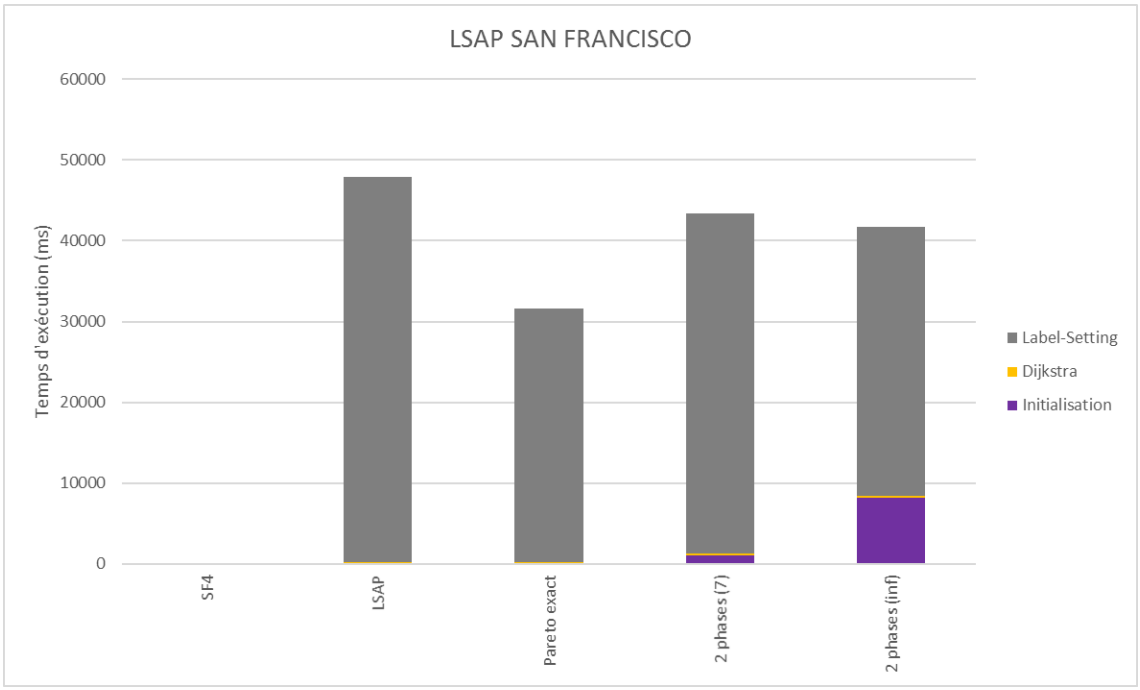

FIGURE 6.5 – Temps d'exécution pour le graphe de San Francisco : SF4

Instance	LSAP	front de Pareto exact	réduction	front méthode à 2 phases	réduction
P1	2 548	1 620	36%	1 904	25%
P2	23 167	11 055	52%	13 857	40%
P3	69 971	40 345	42%	49 244	30%
P4	287 768	166 523	42%	206 591	28%
B1	14 270	8 642	39%	10 712	25%
B2	118 398	65 793	44%	84 762	28%
B3	306 721	173 066	44%	210 344	31%
B4	898 092	570 587	36%	671 908	25%
SF1	15 364	9 269	40%	11 090	28%
SF2	204 855	122 465	40%	142 579	30%
SF3	1 679 710	1 028 550	39%	1 193 290	29%
SF4	8 330 210	5 745 980	31%	6 564 200	21%

FIGURE 6.6 – Nombre d'étiquettes générées par instance selon la méthode utilisée : Label-Setting amélioré sans approximation du front de Pareto, avec le front de Pareto exact en entrée, avec le front de Pareto approximé par la méthode à 2 phases sans limite de calcul

6.4 Analyse

On peut séparer l'analyse en deux parties : l'impact de la méthode à 2 phases sur les temps de calcul et sur la réduction du nombre d'étiquettes générées.

6.4.1 Impact sur les temps de calcul

Sur ce point, la méthode à 2 phases est intéressante uniquement sur de très grandes instances. Ceci s'explique par le fait que le temps passé par la méthode pour approximer le front de Pareto est compensé par la réduction de temps qu'elle opère. Cette méthode permet de racourcir l'exécution de P4, B4, SF3 et SF4. Sur de plus petites instances, cette méthode ralentit beaucoup les temps opératoires et n'a donc aucun intéret. Et ces résultats sont viables uniquement lorsqu'on limite la méthode à 2 phases à 7 solutions au maximum. Sinon cette méthode est intéressante uniquement pour SF4.

6.4.2 Impact sur la réduction du nombre d'étiquettes générées

Sur ce point, la méthode est relativement intéressante quelque soit l'instance testée. La réduction est comprise entre 25% et 40% avec une réduction de 1 766 010 étiquettes au maximum pour la méthode à 2 phases sans contrainte de limite. Lorsqu'on limite la méthode à 7 solutions, les résultats sont équivalents à 3% près. Cela fait de cette méthode une très bonne solution pour réduire le nombre d'étiquettes.

6.4.3 Conclusion

En conclusion, cette méthode se révèle très intéressante sur de grandes instances en terme de temps d'exécution ou sur toutes les instances en général concernant la réduction du nombre d'étiquettes générées. Seulement, cette méthode peut s'avérer handicapante lors du calcul d'instances plus simples (jusqu'à 5 fois plus long avec l'approximation que sans). Cette méthode n'étant pas suffisante, le projet s'oriente vers une nouvelle méthode d'approximation : l'utilisation de Landmarks. Une méthode proposée et étudiée par Gaël Sauvanet et approfondie dans ce projet de fin d'étude.

Approximation du front de Pareto par la méthode des Landmarks

La méthode proposée et les travaux présentés sont basés sur les travaux de Gaël Sauvanet dans la thèse [1] qui sont inspirés eux-même de la méthode ALT [Goldberg]. Les travaux d'approfondissement sont l'objet de ce projet ainsi que les propositions suivantes.

La méthode des Landmarks est un prétraitement important qui n'est effectué qu'une seule fois pour un graphe donné et qui peut donc prendre un temps très important. Une fois ce prétraitement réalisé, il n'est plus nécessaire de le faire. C'est donc la réalisation de ce prétraitement et son implémentation qui composent la complexité de cette méthode.

7.1 Principe et implémentation

Le principe de cette méthode est de calculer en amont un graphe composé uniquement des noeuds stratégiques et importants du graphe principal. Ce graphe simplifié permettra par la suite de calculer des approximations du front de Pareto pour accélérer les calculs et la convergence du front de Pareto terminal. Pour cela de nombreuses sous-méthodes sous nécessaires.

La création d'un graphe de Landmarks se réalise en plusieurs étapes :
- Extraction des noeuds stratégiques (landmarks)
- Choix des voisins dans le nouveau graphe
- Création des liaisons entre les nouveaux noeuds
- Enregistrement des données

Une fois le graphe réalisé, il est nécessaire d'intégrer le noeud de destination et de départ au graphe des Landmarks pour approximer le front de Pareto. Toutes les étapes seront approfondie dans la suite.

7.1.1 Extraction des noeuds stratégiques (landmarks)

Cette étape est très importante car si l'on sélectionne mal les landmarks, l'approximation finale sera très mauvaise.

Pour ce faire, le protocole mis en place est le suivant. Dans un premier temps, on initialise un compteur de passage sur chaque noeud du graphe initial (Paris, Berlin ou San Francisco). Ensuite on exécute un nombre important de recherches monocritère (Dijkstra) et pour chaque chemin calculé, on incrémente le compteur de passage des noeuds qui figurent dans le chemin terminal. Ainsi, on a exécuté 1 000 000 de recherches sur le graphe de Paris, 500 000 recherches sur le graphe de Berlin et environ 250 000 recherches sur le graphe de San Francisco. Cet ensemble de recherche permet de savoir quels sont les noeuds les plus importants du graphe.

La seconde étape est la sectorisation du graphe pour avoir une distribution le plus homogène possible. Pour cela nous nous basons sur les positions géographiques des noeuds (longitude/latitude) et on sélectionne le noeud le plus influent de chaque zone.

Une autre méthode implémentée et testée mais non approfondie est une sélection des noeuds uniquement sur la localisation. En calquant les noeuds sélectionnés par rapport à une grille, tous les noeuds sont à équidistance mais ne sont pas forcément des noeuds couramment empruntés. On se retrouve donc avec deux types de répartitions : par usage et localisation ou par localisation uniquement. A la vue des résultats expérimentaux, concordants avec ceux de Gaël Sauvanet, nous avons concentré nos travaux sur la répartition par usage et localisation.

Le nombre de landmarks à sélectionner est un paramètre important. Plus le nombre de landmarks sera important, plus la qualité de l'approximation sera satisfaisante. Cependant, un graphe de landmarks trop important entrainera des temps de calcul plus long et une approximation moins intéressante.

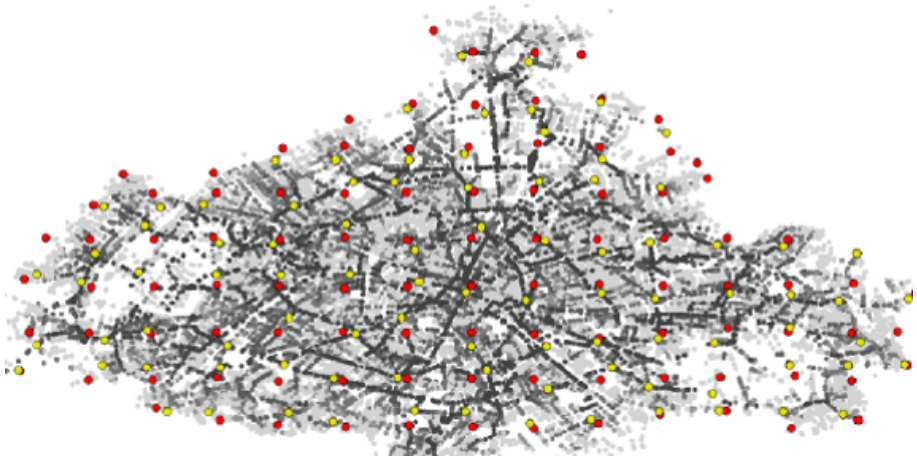

FIGURE 7.1 – Répartition selon deux modes sur le graphe de Paris : en rouge par répartition géographique stricte et en jaune par fréquence d'utilisation dans les zones géographiques

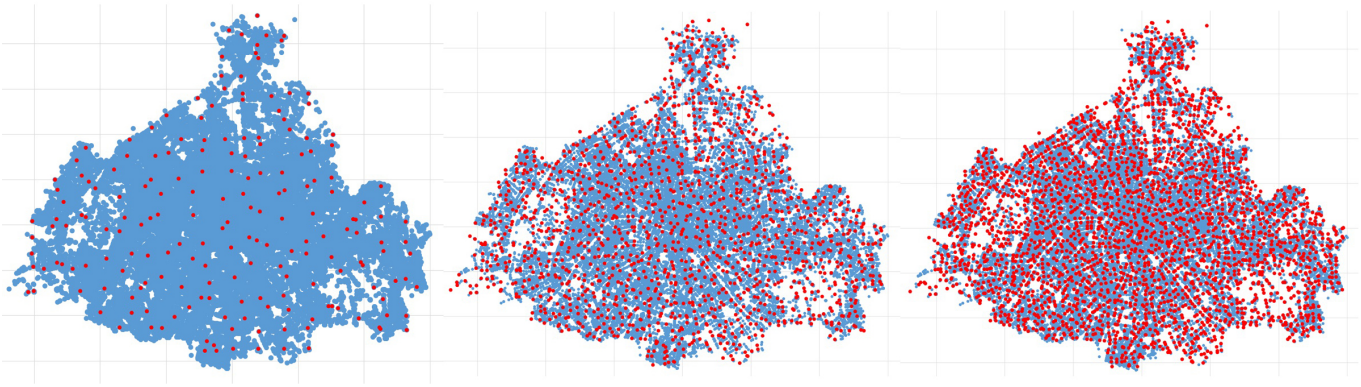

FIGURE 7.2 – Illustration du nombre de landmarks sélectionnés, de gauche à droite : 1%, 10% et 20% des noeuds sont des landmarks

7.1.2 Le choix des voisins dans le nouveau graphe

Cette étape influence directement la qualité du front de Pareto approximé. Les principales difficultés sont de choisir le nombre de voisins qu'a chaque noeud et surtout la stratégie de sélection des voisins.

Le nombre de voisins d'un landmark (le nombre de noeuds à qui il est relié) influence grandement la qualité des solutions approximées et permet même dans certains cas de trouver le front de Pareto exact lors de l'approximation. Le critère du nombre de voisins a nécessité une réelle étude et de nombreuses expérimentations. Le calcul d'un graphe de Landmark est assez long (entre 2 et 30 minutes selon le graphe d'origine) et le nombre de combinaisons de paramètres est assez important.

La stratégie mise en place pour choisir les voisins est basée sur une exploration du voisinage du landmark. Pour réaliser cette tâche on utilise une variante de Dijkstra qui alimente une liste qui contiendra les voisins du landmark courant. Pour cela, on fixe un nombre limite de voisins à trouver et on commence à étudier le voisinage. Lorsqu'un noeud du graphe original est aussi un landmark et qu'il n'est pas encore dans notre liste, on l'ajoute et on augmente le compteur. Ainsi, les voisins d'un landmark sont les landmarks les plus proches par rapport à un critère donné : dans notre cas la recherche s'effectue sur la distance entre les noeuds.
Une autre stratégie serait de prendre les noeuds des zones géographiques adjacentes. Seulement, cette méthode ne donne pas de bons résultats et certaines zones géographiques peuvent être proches à vol d'oiseau mais très éloignées par piste cyclable.

7.1.3 Création des liaisons entre les landmarks

Une fois tous les landmarks sélectionnés et les voisins pour chaque landmark sélectionnés, il faut calculer les arcs entre les landmarks et leurs voisins.

Pour réaliser cette opération, on utilise l'algorithme de Label-Setting pour calculer l'ensemble des solutions entre le landmarks et ses voisins. On a donc un nouveau paramètre à prendre en compte : le nombre d'arcs entre deux landmarks donnés. Ce paramètre n'influence pas particulièrement la précision du front de Pareto approximé mais plutôt sa dispersion. Un nombre d'arc important donnera un ensemble des solutions plus équilibré, ce qui favorise l'utilisation des propriétés d'élimination du LSAP.

Cette étape est la plus longue en terme d'exécution car le nombre de sous-chemins à calculer augmente considérablement lorsque le nombre de voisins par landmark est grand. Cependant, le code à développer est en grande partie recyclé des précédents développements.

7.1.4 Enregistrement des données

Pour faciliter certaines opérations et garder une cohérence entre les graphes de Landmarks et les graphes originaux, les indices des noeuds sont conservés ainsi que les localisations. Cela induit quelques difficultées supplémentaires lors de la création des graphes de Landmarks.

Les « sous-graphes » sont donc stockés au format CSV dans la même configuration que les graphes originaux. Cependant, les graphes de Landmarks correspondent dans le code source à une autre classe car certaines spécificités existent.

7.1.5 Utilisation des graphes de Landmarks pour approximer le front de Pareto

L'utilisation de ces graphes a pour but de donner une approximation du front de Pareto en entrée à l'algorithme de Label-Setting. Pour réaliser cette opération, il est nécessaire d'ajouter le noeud de départ ainsi que le noeud de destination au graphe des Landmarks si ces noeuds ne sont pas des landmarks. Dans le cas où ils ne le seraient pas, il est nécessaire de calculer des arcs entre ces noeuds et le reste du graphe pour les relier.
Pour ce faire, on execute Dijkstra dans le graphe original. Lorsqu'un noeud est étudié et qu'il est aussi un

landmark, on crée l'arc de distance et d'insécurité correspondant aux valeurs courantes dans l'exécution de Dijkstra et on relie ainsi nos noeuds de départ et d'arrivée au graphe des Landmarks. Pour équilibrer un maximum nos valeurs, on récupère les x premiers arcs en exécutant Dijkstra sur le critère de distance et les x premiers arcs en exécutant Dijkstra sur le critère d'insécurité. Ainsi on évite d'avoir un désequilibre de qualité sur le front de Pareto approximé.

Une fois les noeuds de départ et d'arrivée correctement reliés au graphe de Landmarks, on exécute l'algorithme de LSAP qui génère un front de Pareto approximé. Ce front sera ensuite fourni en entrée à l'exécution de l'algorithme LSAP sur le graphe original.

7.2 Etude, interprétations et résultats

De nombreuses idées et implémentations ont été testées sur les graphes de Landmarks. Comme précisé précédemment, il existe en réalité 3 paramètres influençant la création des graphes de Landmarks : le nombre de noeuds à retenir (exprimé en pourcentage), le nombre de voisins qu'a chaque landmark et le nombre de liaisons (d'arcs) entre chaques voisins.

Dans le cadre de ce projet, nous avons étudié l'impact de ces paramètres sur de larges intervalles (ensemble discret pour les 3 paramètres) :

- Pourcentage de noeuds à retenir : entre 1% et 20%
- Nombre de voisins : entre 5 et 30 (12 valeurs)
- Nombre d'arcs entre voisins : 4, 6 ou 8

La combinaison de ces paramètres génère un ensemble de 720 configurations possibles.
En plus des premières observations faites au début du développement, une longue série d'expérimentations a été réalisée pour tester les 720 configurations et synthétiser les résultats. Pour cela, nous avons utilisé la méthode de l'ACP (Analyse en Composantes Principales) pour synthétiser les résultats. L'objectif étant de pouvoir « prédire » quelle configuration serait la plus performante en ne connaissant que les bornes inférieures et supérieures du problème. De cette manière lors du calcul d'un itinéraire, une table permettrait de calculer la configuration à choisir selon la borne inférieure sur le critère de distance ou d'insécurité.
L'une des premières hypothèses était qu'un chemin court nécessiterait sûrement un graphe de Landmark au maillage fin alors qu'un long chemin nécessiterait plutôt un graphe avec des noeuds plus écartés.

7.2.1 Utilisation d'un seul graphe de Landmarks

L'utilisation d'une seul graphe de Landmarks pour calculer tous les itinéraires s'est révélé plutôt décevant mais bien plus régulière que la méthode à 2 phases. Cette méthode d'approximation permet d'améliorer les classes P3, P4, B2, B3, B4, SF2, SF3 et SF4. De ce point de vue c'est très intéressant mais les améliorations sont légères. D'un point de vue réduction des étiquettes générées, la méthode à 2 phases est plus efficace.

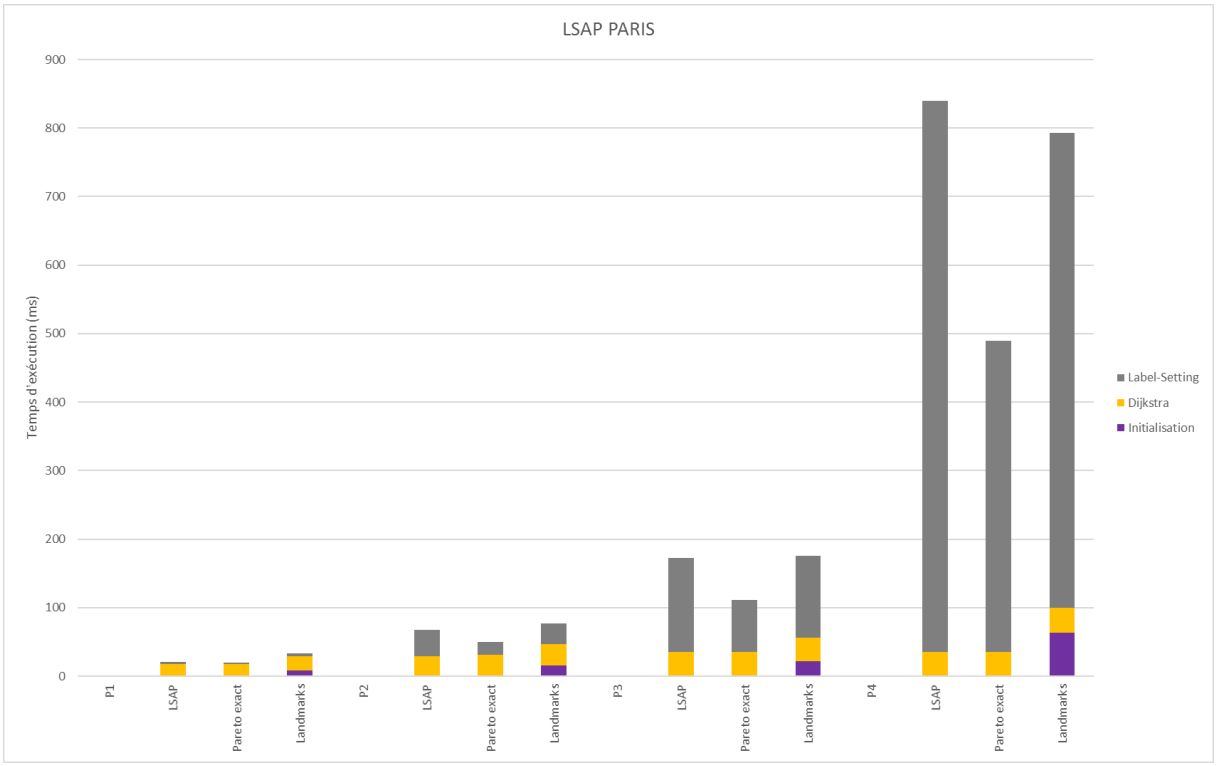

FIGURE 7.3 – Temps d'exécution du LSAP avec l'approximation par les Landmarks sur le graphe de Paris

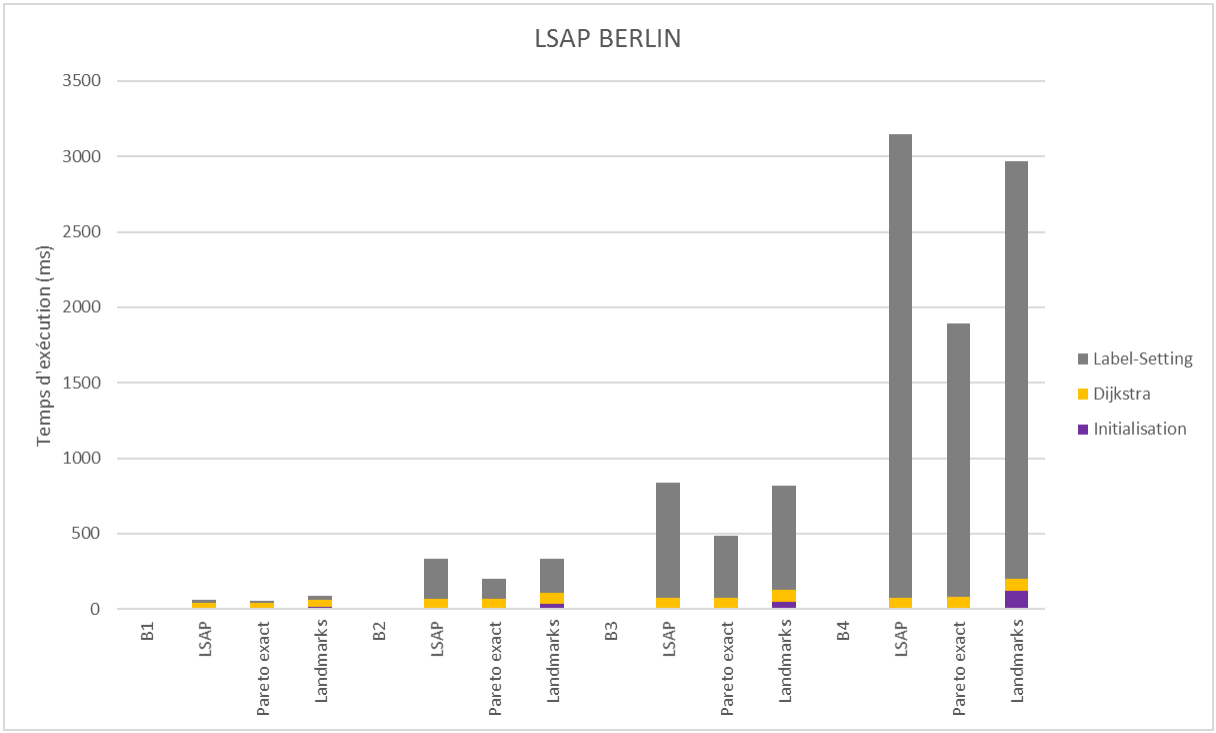

FIGURE 7.4 – Temps d'exécution du LSAP avec l'approximation par les Landmarks sur le graphe de Berlin

7.2.2 Table de configuration de Landmark

La proposition suivante était de créer une table permettant de faire le lien entre les données de notre problème et une configuration de Landmarks type. Nous avons pensé aux bornes inférieures et supérieures du problème qui sont des données disponibles et calculées en amont. Ainsi une table faisant le lien entre une borne inférieure sur la distance et une configuration de Landmarks permettrait une approximation optimale.

Après avoir testé les 720 configurations, en prenant en compte la meilleure approximation pour chaque itinéraire, les résultats sont très satisfaisant et le système des Landmarks améliore les résultats de toutes les instances.

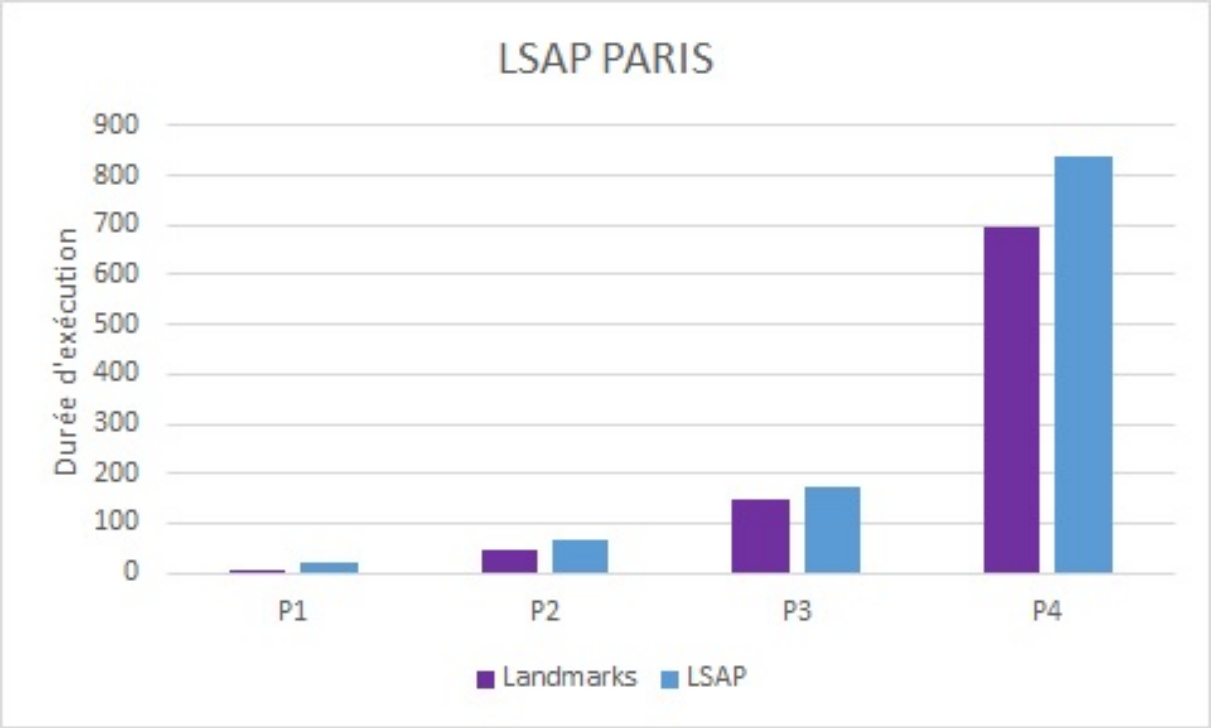

FIGURE 7.5 – Temps d'exécution du LSAP sans approximation et avec l'approximation par les Landmarks

Le seul problème est la partie concernant la prédiction des configurations à choisir. Il est facile de prendre pour chaque chemin la meilleure approximation sur les 720 calculées. Mais il est très difficile de savoir en amont laquelle choisir.

Pour tenter de trouver une corrélation entre la borne inférieure et la configuration du graphe de Landmark, nous avons réalisé deux ACP sur l'ensemble des données. La première correspond aux données qui réduisent le plus le temps d'exécution du LSAP. La seconde ACP correspond aux données des configurations de Landmarks qui réduisent le plus la distance entre le front de Pareto approximé et le front de Pareto exact.

7.2.3 ACP : données qui réduisent le temps d'exécution

Les classes de Paris ont été réorganisées et triées par LB distance croissante. Les classes sont toujours composées de 50 chemins chacune et l'ensemble des couples noeud de départ/noeud d'arrivée est le même.

Variable	Minimum	Maximum	Moyenne	Ecart type
LB	249	8119	5466	1945
nbPourcent	1	20	5,7	5,2
nbVoinsin	5	30	17	8,1
nbArc	4	8	5,7	1,7

FIGURE 7.6 – Informations de base de la classe 1

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	1	0,013	0,103	0,089
nbPourcent	0,013	1	-0,278	0,002
nbVoinsin	0,103	-0,278	1	-0,241
nbArc	0,089	0,002	-0,241	1

FIGURE 7.7 – Matrice de corrélation de la classe 1

Les résultats de l'ACP sont les suivants :

La totalité des résultats est disponible en annexe. Les résultats indiquent clairement qu'il n'existe aucune réelle corrélation entre les variables. L'écart type des variables nbVoisin et nbPourcent est très grand, ce qui n'est pas favorable à une tendance globale. Il n'est pas possible d'établir clairement un lien entre la borne inférieure et une configuration de graphe de Landmark. D'autres variables comme les bornes supérieures, l'écart entre la borne supérieure et la borne inférieure ont été testées mais les résultats sont identiques. Il n'y a pas de corrélation forte ou même significative. Les résultats des classes C2, C3 et C4 sont identiques et ne font pas apparaitre de corrélations entre variables.

7.2.4 ACP : données qui réduisent la surface entre le front approximé et exact

Les classes ont été réorganisées comme précédemment, par LB distance croissante.

Les résultats de l'ACP sont les suivants :

La totalité des résultats est disponible en annexe. Les résultats montrent clairement qu'il n'y a aucune corrélation significative entre les variables. Cependant, on peut noter une tendance forte qui est le nombre de voisin. L'écart type est relativement bas et permet de définir que le nombre de voisins doit se situer entre 28 et 30 pour que l'approximation génère un front de Pareto très proche du front de Pareto exact. De plus, par observation des résultats, le nombre d'arc n'influence pas de manière importante la qualité de l'approximation et fixer cette variable à 6 arcs semble être un bon choix. Le problème étant de définir le pourcentage de noeuds à garder. L'écart type est de 4 et les valeurs conseillées sont donc comprises entre 8,6 et 16,6. Cette partie est réellement problématique car un mauvais choix dans ce paramètre peut créer une très mauvaise approximation qui ralentirait ou ferait perdre tout l'intéret de cette méthode.

Variable	Minimum	Maximum	Moyenne	Ecart type
LB	15780	19176	17361	1027,6
nbPourcent	3	20	12,6	3,9
nbVoinsin	22	30	29,3	1,5
nbArc	4	8	6,5	1,5

FIGURE 7.8 – Informations de base de la classe 4

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	1	-0,007	0,008	0,066
nbPourcent	-0,007	1	-0,023	-0,100
nbVoinsin	0,008	-0,023	1	-0,161
nbArc	0,066	-0,100	-0,161	1

FIGURE 7.9 – Matrice de corrélation de la classe 4

7.3 Conclusion

En conclusion, la méthode des Landmarks est très prometteuse et donne de très bons résultats, comme heuristique par exemple. Cependant, il est difficile de prévoir à l'avance quelle configuration choisir en se basant sur des variables comme les bornes inférieures et supérieures ou la composition du chemin.
S'il était possible de prévoir avec une grande précision quelle configuration choisir, cette méthode serait aussi efficace et même supérieure à la méthode à 2 phases sur de grandes instances (comme pour P4 par exemple). Sur de petites instances, la méthode des Landmarks est peu gourmande en temps et permet encore d'améliorer les temps de calcul.

Si l'on souhaite réorienter les Landmarks pour en faire une heuristique, les fronts de Pareto générés sont très intéressants et parfois même la méthode trouve le front de Pareto complet.
L'exemple suivant montre la comparaison entre le front de Pareto approximé et le front de Pareto calculé par le LSAP. Le chemin testé contient plus de 600 solutions non-dominées et il faut environ 6 secondes pour le calculer en temps normal. L'approximation trouve ce front après 1 seconde. Ce chemin est le plus compliqué sur l'ensemble des instances de Paris.

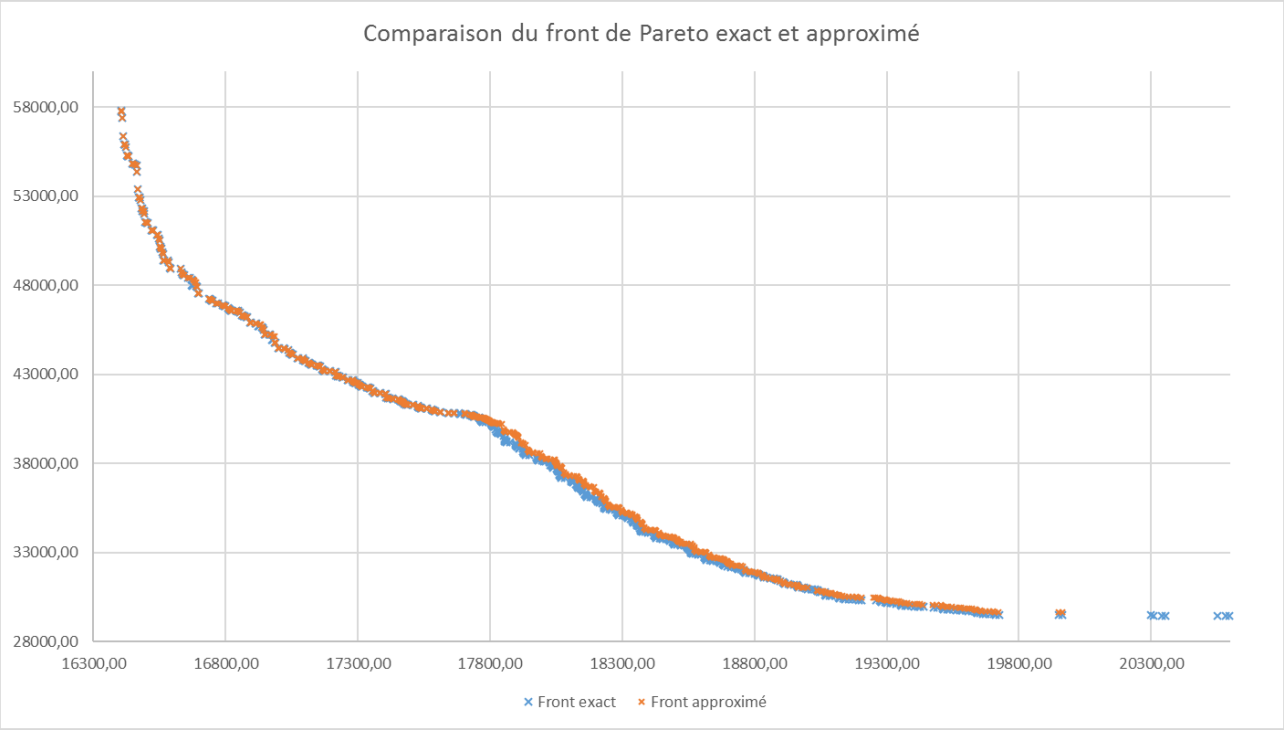

FIGURE 7.10 – Comparaison du front de Pareto approximé avec le front de Pareto trouvé par le LSAP

Conclusion

A la fin de ce projet de fin d'études, les principaux objectifs ont été remplis et le planning prévisionnel a été respecté. Les propositions de départ ont pu être développées et correctement testées et de nouvelles méthodes ont été implémentées.

D'un point de vue personnel, ce projet de fin d'études m'a permis de mettre en pratique mes cours de gestion de projet et de tester mes capacités à réaliser un projet dans son intégralité : information, spécifications, développements et tests. Ce projet s'inscrit dans ma volonté d'approfondir mes connaissances sur la théorie des graphes comme j'ai pu le faire en 3^{e} année et en 4^{e} année avec des projets de Monsieur Néron également. Ce projet est aussi complémentaire avec mon option logistique et optimisation et me permet d'élargir mes connaissances.

Ce projet a prouvé que la méthode des Landmarks pouvait être une solution intéressante que ce soit d'un point de vue heuristique ou approximation pour l'algorithme du Label-Setting. Cependant, prévoir les configurations des graphes de Landmarks à utiliser n'est pas trivial et certains travaux supplémentaires permettraient peut-être de faciliter cela.

Annexes

Variable	Minimum	Maximum	Mean	Std. deviation
LB	249,000	8119,000	5465,980	1944,580
nbPourcent	1,000	20,000	5,773	5,176
nbVoisin	5,000	30,000	16,987	8,150
nbArc	4,000	8,000	5,707	1,628

Correlation matrix (Pearson (n)):

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	**1**	0,013	0,103	0,089
nbPourcent	0,013	**1**	-0,278	0,002
nbVoisin	0,103	-0,278	**1**	-0,241
nbArc	0,089	0,002	-0,241	**1**

FIGURE 7.11 – ACP de la classe 1 pour les données qui réduisent le temps d'exécution

Variable	Minimum	Maximum	Mean	Std. deviation
LB	8197,000	11529,000	9698,540	857,829
nbPourcent	1,000	20,000	7,413	5,961
nbVoisin	5,000	30,000	14,567	8,098
nbArc	4,000	8,000	5,680	1,607

Correlation matrix (Pearson (n)):

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	**1**	0,126	-0,138	0,097
nbPourcent	0,126	**1**	-0,399	0,113
nbVoisin	-0,138	-0,399	**1**	-0,057
nbArc	0,097	0,113	-0,057	**1**

FIGURE 7.12 – ACP de la classe 2 pour les données qui réduisent le temps d'exécution

Variable	Minimum	Maximum	Mean	Std. deviation
LB	11539,000	15752,000	13371,640	1339,848
nbPourcent	1,000	20,000	6,973	5,305
nbVoisin	5,000	30,000	16,940	8,422
nbArc	4,000	8,000	5,573	1,598

Correlation matrix (Pearson (n)):

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	**1**	0,065	-0,055	0,120
nbPourcent	0,065	**1**	-0,502	0,045
nbVoisin	-0,055	-0,502	**1**	-0,055
nbArc	0,120	0,045	-0,055	**1**

FIGURE 7.13 – ACP de la classe 3 pour les données qui réduisent le temps d'exécution

Variable	Minimum	Maximum	Mean	Std. deviation
LB	15780,000	19176,000	17361,244	1027,605
nbPourcent	1,000	19,000	5,390	4,873
nbVoisin	5,000	30,000	20,000	8,163
nbArc	4,000	8,000	5,268	1,563

Correlation matrix (Pearson (n)):

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	**1**	0,119	-0,081	0,094
nbPourcent	0,119	**1**	-0,323	-0,078
nbVoisin	-0,081	-0,323	**1**	-0,101
nbArc	0,094	-0,078	-0,101	**1**

FIGURE 7.14 – ACP de la classe 4 pour les données qui réduisent le temps d'exécution

Variable	Minimum	Maximum	Mean	Std. deviation
LB	249,000	8119,000	5465,980	1944,580
nbPourcent	1,000	20,000	12,747	5,309
nbVoisin	11,000	30,000	28,047	3,443
nbArc	4,000	8,000	6,453	1,540

Correlation matrix (Pearson (n)):

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	**1**	0,301	-0,096	0,088
nbPourcent	0,301	**1**	-0,043	-0,030
nbVoisin	-0,096	-0,043	**1**	0,026
nbArc	0,088	-0,030	0,026	**1**

FIGURE 7.15 – ACP de la classe 1 pour les données qui réduisent la surface entre le front approximé et le front exact

Variable	Minimum	Maximum	Mean	Std. deviation
LB	8197,000	11529,000	9698,540	857,829
nbPourcent	1,000	20,000	13,020	5,071
nbVoisin	11,000	30,000	26,940	4,525
nbArc	4,000	8,000	6,533	1,549

Correlation matrix (Pearson (n)):

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	**1**	-0,109	-0,098	0,010
nbPourcent	-0,109	**1**	0,166	-0,169
nbVoisin	-0,098	0,166	**1**	-0,030
nbArc	0,010	-0,169	-0,030	**1**

FIGURE 7.16 – ACP de la classe 2 pour les données qui réduisent la surface entre le front approximé et le front exact

Variable	Minimum	Maximum	Mean	Std. deviation
LB	11539,000	15752,000	13371,640	1339,848
nbPourcent	2,000	20,000	12,600	5,639
nbVoisin	16,000	30,000	26,907	4,200
nbArc	4,000	8,000	6,387	1,514

Correlation matrix (Pearson (n)):

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	**1**	0,088	0,103	-0,094
nbPourcent	0,088	**1**	0,045	-0,170
nbVoisin	0,103	0,045	**1**	0,008
nbArc	-0,094	-0,170	0,008	**1**

FIGURE 7.17 – ACP de la classe 3 pour les données qui réduisent la surface entre le front approximé et le front exact

Variable	Minimum	Maximum	Mean	Std. deviation
LB	15780,000	19176,000	17361,244	1027,605
nbPourcent	3,000	20,000	12,642	3,915
nbVoisin	22,000	30,000	29,301	1,536
nbArc	4,000	8,000	6,537	1,517

Correlation matrix (Pearson (n)):

Variables	LB	nbPourcent	nbVoisin	nbArc
LB	**1**	-0,007	0,008	0,066
nbPourcent	-0,007	**1**	-0,023	-0,100
nbVoisin	0,008	-0,023	**1**	-0,161
nbArc	0,066	-0,100	-0,161	**1**

FIGURE 7.18 – ACP de la classe 4 pour les données qui réduisent la surface entre le front approximé et le front exact

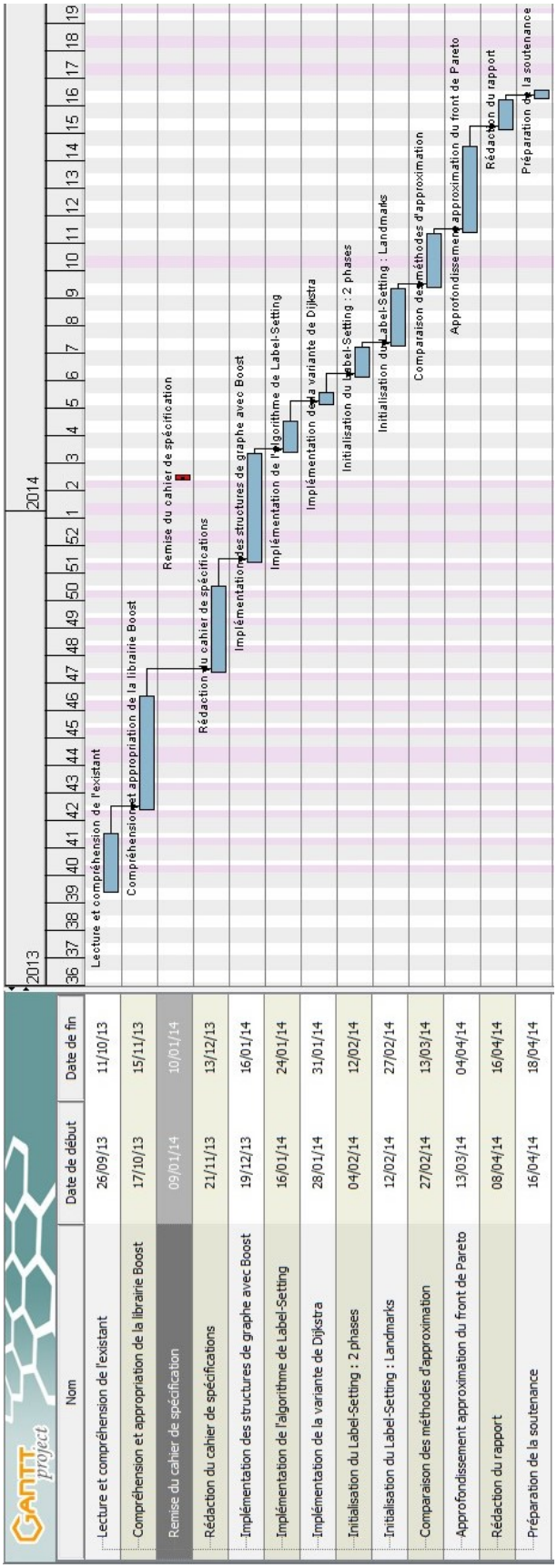

FIGURE 7.19 – Gantt prévisionnel du projet qui a été respecté

Bibliographie

[1] Gaël Sauvanet, *Recherche de chemins multiobjectifs pour la conception et la réalisation d'une centrale de mobilité destinée aux cyclistes* 2011.

Utilisation de Landmarks pour le calcul de chemins multicritères

Département Informatique
5e année
2013 - 2014

Rapport de projet de fin d'études

Résumé : Le but de ce projet de fin d'études est de reprendre une partie des travaux de la thèse de Gaël Sauvanet pour les approfondir et proposer de nouvelles méthodes d'optimisations. La méthode des Landmarks sera reprise à zéro et de nouvelles procédures seront testées. L'objectif parallèle est d'implémenter tous ces travaux en se basant sur la librairie Boost et de pouvoir constater les différences de performances avec Leda.

Mots clefs : multicritère, Label-Setting, Dijkstra, théorie des graphes, recherche de chemins, optimisation, sites d'intéret.

Abstract: The aim of this end of studies project is to continue a part of works of Gaël Sauvanet in his thesis to deepen it and propose new optimisations methods. The landmarks method will be taken from scratch and new process will be tested. The parallel objective is to implement all source code with the library Boost and to compare our results with the Leda library.

Keywords: multicriteria, Label-Setting, Dijkstra, graph theory, path finding, optimisation, Landmarks.

Encadrant
Emmanuel NÉRON
emmanuel.neron@univ-tours.fr

Étudiant
Raphaël ROGER
raphael.roger@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2013 - 2014