

# **Partitionnement de territoire à l'aide d'un algorithme génétique**

Harold Waterkeyn

Directeur : Professeur Yves De Smet

Assistant : Karim Lidouh



Mémoire présenté en vue de l'obtention

Du diplôme du Master en Sciences informatiques

Année académique 2012 - 2013



*The computer is incredibly fast, accurate, and stupid.  
Man is incredibly slow, inaccurate, and brilliant.  
The marriage of the two is a force beyond calculation.*

LEO CHERNE

*I have called this principle, by which each slight variation, if useful, is preserved, by the term of Natural Selection.*

CHARLES DARWIN

# Remerciements

J'aimerais adresser mes remerciements aux personnes qui m'ont aidé et soutenu et ont ainsi contribué à l'élaboration de ce mémoire.

Je tiens à remercier particulièrement M. Yves De Smet ainsi que M. Karim Lidouh pour leur suivi actif tout au long de ce travail. Leurs nombreux conseils pertinents et leur aide ont su guider mes recherches et mes travaux. Leur encadrement et leur grande disponibilité m'ont permis de mener ce projet à bien.

Je remercie les membres de mon jury, M. Bernard Fortz et M. Mauro Birattari, pour avoir accepté d'analyser et d'évaluer mon travail.

Je remercie M. Gilles Van Hamme et Mme Eléonore Wolff du département de géographie pour le temps qu'ils m'ont consacré et pour les données relatives aux communes wallonnes qu'ils m'ont fournies.

Je remercie Lucie Duchemin, pour son soutien dans mon travail, la motivation qu'elle m'a procurée et sa patience durant les nombreuses relectures de ce document.

Je remercie mes parents, Pascale Dulière et Pascal Waterkeyn, pour leur confiance et leurs encouragements.

De manière générale, je remercie l'ensemble de mes professeurs, qui m'ont formé durant ces cinq années et ont su éveiller ma curiosité dans de nombreux domaines.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>10</b>
1.1	Contexte et objectifs du mémoire . . . . .	10
1.2	Structure du mémoire . . . . .	12
1.3	Notations . . . . .	13
1.4	Vocabulaire . . . . .	14
<b>I</b>	<b>Etat de l'art</b>	<b>15</b>
<b>2</b>	<b>Optimisation multiobjectif</b>	<b>16</b>
2.1	Introduction . . . . .	16
2.2	Aide à la décision . . . . .	16
2.3	Problèmes d'optimisation mono-objectif . . . . .	17
2.4	Problèmes d'optimisation multiobjectif . . . . .	18
2.4.1	Problèmes discrets et continus . . . . .	20
2.4.2	Problèmes explicites et implicites . . . . .	21
2.4.3	Formulations . . . . .	21
2.5	Résolution de MOPs . . . . .	25
2.6	Techniques d'aide à la décision . . . . .	26
2.7	Conclusion . . . . .	27
<b>3</b>	<b>Algorithmes évolutionnistes</b>	<b>28</b>
3.1	Introduction . . . . .	28
3.2	Résolution de problèmes d'optimisation . . . . .	28
3.3	Métaheuristiques . . . . .	30
3.3.1	Assignation de fitness . . . . .	32
3.3.2	Préservation de la diversité . . . . .	37
3.3.3	Élitisme . . . . .	37
3.3.4	Évaluation des performances . . . . .	39
3.4	Algorithmes évolutionnistes . . . . .	42
3.4.1	NSGA-II . . . . .	44
3.5	Recherche locale et métahéuristiques hybrides . . . . .	46
3.6	Conclusion . . . . .	47
<b>4</b>	<b>Problème de partitionnement</b>	<b>48</b>
4.1	Introduction . . . . .	48
4.2	Système d'information géographique . . . . .	48
4.3	Big Data . . . . .	53

4.4	Partitionnement de territoire . . . . .	53
4.4.1	Partitionnement électoral . . . . .	56
4.4.2	Zone de travail pour des équipes de vendeurs . . . . .	56
4.4.3	Complexité . . . . .	58
4.4.4	Partitionnement de graphe . . . . .	58
4.4.5	Problème de regroupement . . . . .	60
4.5	Conclusion . . . . .	63
<b>II</b>	<b>Apports personnels</b>	<b>64</b>
<b>5</b>	<b>Lecture critique de l'article</b>	<b>65</b>
5.1	Introduction . . . . .	65
5.2	Enoncé du problème . . . . .	65
5.2.1	Contraintes . . . . .	68
5.2.2	Critères . . . . .	69
5.3	Algorithme génétique . . . . .	71
5.3.1	Représentation des solutions . . . . .	71
5.3.2	Assignation de fitness . . . . .	71
5.3.3	Crossover . . . . .	74
5.3.4	Mutation . . . . .	77
5.3.5	Procédure de fusion . . . . .	77
5.3.6	Population initiale . . . . .	81
5.3.7	Recherche locale . . . . .	82
5.4	Conclusion . . . . .	82
<b>6</b>	<b>Etude de cas : la Wallonie</b>	<b>84</b>
6.1	Introduction . . . . .	84
6.2	La Wallonie . . . . .	84
6.3	Algorithme . . . . .	88
6.3.1	Représentation des solutions . . . . .	90
6.3.2	Population initiale . . . . .	91
6.3.3	Crossover et mutation . . . . .	91
6.3.4	Recherche locale . . . . .	92
6.3.5	Réduction de la population . . . . .	93
6.3.6	Paramétrisation . . . . .	94
6.3.7	Tests unitaires . . . . .	95
6.4	Application de l'algorithme . . . . .	96
6.4.1	Visualisation . . . . .	96
6.4.2	Données . . . . .	98
6.4.3	Critères . . . . .	101
6.4.4	Paramétrage de l'algorithme . . . . .	103
6.5	Résultats . . . . .	109
6.5.1	Partitionnement électoral . . . . .	109
6.5.2	Agriculture et densité de population . . . . .	110
6.6	Conclusion . . . . .	111
<b>7</b>	<b>Extensions</b>	<b>116</b>
7.1	Introduction . . . . .	116

7.2	Représentation alternative . . . . .	116
7.2.1	Représentation . . . . .	116
7.2.2	Opérateurs . . . . .	117
7.2.3	Comparaison et résultat . . . . .	118
7.3	Partitions similaires . . . . .	124
7.3.1	Distance entre partition . . . . .	124
7.3.2	Recherche locale . . . . .	126
7.4	Parallélisation . . . . .	128
7.5	Conclusion . . . . .	130
<b>8</b>	<b>Conclusion</b>	<b>131</b>

# Table des figures

2.1	Concept de dominance . . . . .	23
2.2	Frontière Pareto optimale . . . . .	24
2.3	Point idéal et point nadir . . . . .	24
2.4	Solution supportée . . . . .	25
3.1	Techniques de résolution pour les problèmes d'optimisation . . . . .	31
3.2	Stratégie de rang de dominance . . . . .	34
3.3	Stratégie de profondeur de dominance . . . . .	35
3.4	Stratégie de compte de dominance . . . . .	35
3.5	Mesure de diversité : méthode noyaux . . . . .	38
3.6	Mesure de diversité : méthode du plus proche voisin . . . . .	38
3.7	Mesure de diversité : méthode histogramme . . . . .	38
3.8	Elitisme actif et passif . . . . .	39
3.9	Illustration de l'indicateur d'hypervolume . . . . .	41
3.10	Illustration de la procédure de <i>Crowding Distance</i> . . . . .	46
4.1	Google Earth . . . . .	50
4.2	GRASS GIS . . . . .	50
4.3	Technologie Read/Write World de Microsoft . . . . .	52
4.4	Notion de partition . . . . .	54
4.5	Partitionnement d'un territoire . . . . .	55
4.6	Notion de zones contiguës . . . . .	55
4.7	Zones de travail pour des équipes de vendeurs . . . . .	57
4.8	Partition de graphes . . . . .	59
4.9	Illustration du problème de regroupement . . . . .	61
4.10	Méthode des k-moyennes et diagramme de Voronoi . . . . .	62
4.11	Itération de l'algorithme des k-moyennes . . . . .	63
5.1	Graphe connexe et non connexe . . . . .	66
5.2	Graphe planaire et non planaire . . . . .	67
5.3	Graphe de contiguïté . . . . .	67
5.4	Zone incluse . . . . .	68
5.5	Encodage des solutions . . . . .	72
5.6	Crossover et relation d'équivalence . . . . .	75
5.7	Illustration de la procédure de crossover . . . . .	76
5.8	Procédure de crossover générant une solution non valide . . . . .	78
5.9	Critique de l'opérateur de crossover . . . . .	79
5.10	Illustration de la procédure de mutation . . . . .	80

5.11 Procédure de fusion et construction de solution . . . . .	81
5.12 Notion de solution voisine . . . . .	83
6.1 Les trois Régions composant la Belgique . . . . .	85
6.2 Partition actuelle de la Wallonie en cinq provinces . . . . .	86
6.3 Nombre d'habitants par province de Wallonie . . . . .	86
6.4 Surface de chaque province de Wallonie . . . . .	87
6.5 Densité de population de chaque province de Wallonie . . . . .	87
6.6 Diagramme UML de l'algorithme . . . . .	90
6.7 Représentation d'une solution (UML) . . . . .	91
6.8 Carte de la Wallonie . . . . .	97
6.9 Outil 1 : encodage des communes . . . . .	98
6.10 Outil 2 : adjacence des communes . . . . .	99
6.11 Retouches de la carte de Wallonie . . . . .	99
6.12 Conversion de la carte en image vectorielle . . . . .	100
6.13 Analyse du critère de compactage . . . . .	102
6.14 Modélisation alternative du critère de compactage . . . . .	103
6.15 Influence de la taille de la population sur l'hypervolume final . . . . .	105
6.16 Influence de la taille de la population sur le temps pris par l'algorithme . . . . .	106
6.17 Nombre de générations produites pour différentes tailles de population en un temps fixé. . . . .	107
6.18 Hypervolume final produit par différentes tailles de population en un temps fixé. . . . .	108
6.19 Influence de la probabilité de mutation sur le temps d'exécution . . . . .	109
6.20 Influence de la probabilité de mutation sur l'hypervolume final . . . . .	110
6.21 Ration hypervolume/temps pour différentes probabilités de mutation . . . . .	111
6.22 Influence de la recherche locale sur le nombre de générations produites . . . . .	112
6.23 Influence de la recherche locale sur l'hypervolume final . . . . .	113
6.24 Evolution de l'hypervolume sur 800 générations . . . . .	113
6.25 Front obtenu pour le partitionnement électoral . . . . .	114
6.26 Deux solutions pour le partitionnement électoral . . . . .	114
6.27 Front obtenu pour le problème densité/agriculture . . . . .	115
6.28 Deux solutions pour le problème densité/agriculture . . . . .	115
7.1 Front obtenu pour le problème du partitionnement électoral avec la nouvelle representation . . . . .	119
7.2 Solution obtenue avec le nouvel encodage . . . . .	120
7.3 Comparaison des trois représentations . . . . .	121
7.4 Comparaison des trois représentations avec un temps fixé . . . . .	122
7.5 Comparaison de la convergence des trois representations . . . . .	123
7.6 Solution non encodable avec la nouvelle représentation . . . . .	124
7.7 Partition actuelle de la Wallonie en vingt arrondissements . . . . .	125
7.8 Partitions similaires générées . . . . .	127
7.9 Partition actuelle et partition similaire . . . . .	127

# Chapitre 1

## Introduction

### 1.1 Contexte et objectifs du mémoire

Le problème de partitionnement de territoire est un problème d'optimisation de grande importance. Il consiste à grouper un ensemble d'éléments territoriaux en un ensemble de zones formant alors une partition du territoire répondant à un certain nombre de critères. Ce problème a de nombreuses applications pratiques.

Son application la plus populaire est connue sous le nom de partitionnement électoral. Celui-ci vise à agréger des zones administratives en un nombre donné de districts de façon à ce que chacun de ceux-ci soit compact et que le nombre d'habitants soit homogène entre les différents districts [21]. L'objectif est de diviser le territoire en zones tout en assurant une égalité de la force électorale.

Le terme *gerrymandering* fait référence à la manipulation du découpage des circonscriptions électORALES dans le but de favoriser un parti ou un candidat. Le terme prend naissance en 1811 lorsque le gouverneur américain *Elbridge Gerry* signe un redécoupage du Massachusetts donnant un net avantage à son parti. Cet exemple historique montre bien l'importance du problème de partitionnement et la motivation d'utiliser des techniques informatiques afin de mieux garantir l'équité du partitionnement.

De nombreuses autres applications, prenant naissance dans des domaines très variés, ont fait l'objet d'études. Nous pouvons par exemple citer :

- La définition de zones de travail pour des équipes de vendeurs. Chaque équipe est responsable des clients présents sur une partie du territoire et on peut alors chercher à définir les zones de façon à répartir la charge de travail entre les différentes équipes.[46, 114, 112]
- L'assignation de zones pour les différentes écoles publiques d'un territoire. Ce problème connu sous le nom de *school districting* consiste à spécifier les groupes d'enfants qui fréquentent chaque école [44, 50]. Ce partitionnement est nécessaire pour équilibrer le nombre d'inscriptions dans chaque école mais également pour déterminer les itinéraires de bus scolaires.
- Dans le domaine des télécommunications ou de l'énergie, le problème de partitionnement de territoire est souvent utilisé afin de trouver l'emplacement optimal d'antennes ou de relais.[91]

- Dans le domaine des transports, afin d'optimiser l'organisation des différentes lignes de métro/bus et répondre aux besoins des utilisateurs, le partitionnement d'un territoire en zones est souvent nécessaire.[112]

De par son utilité dans un large éventail d'applications et de par l'impact souvent capital de ses solutions, ce problème d'optimisation multicritère fait l'objet de nombreuses recherches.

La complexité inhérente au problème de partitionnement de territoire met à mal l'utilisation d'algorithmes classiques. L'utilisation d'heuristiques pour résoudre des problèmes d'optimisation complexes a toujours été très populaire. Ceci s'applique également au problème de partitionnement de territoire. On retrouve donc, par exemple, des recherches basées sur le recuit simulé [29, 22]. Il s'agit d'une heuristique inspirée du processus de thermodynamique des métaux recuits par un refroidissement lent. On trouve également des études utilisant la recherche tabou [26], basée sur la recherche locale et une heuristique basée sur la méthode de séparation et évaluation [78].

Les algorithmes évolutionnistes sont une grande famille de techniques inspirées de la théorie de l'évolution. Ces algorithmes ont su montrer de très bons résultats pour des problèmes d'optimisation de toute sorte [110, 41, 48, 115]. Il est donc naturel de s'intéresser à leur utilisation pour le problème de partitionnement de territoire.

Dans ce mémoire, nous évaluerons l'utilisation d'algorithmes évolutionnistes pour aborder la problématique du partitionnement de territoire. Il s'agit d'une piste peu explorée aujourd'hui [21, 112]. Nous analyserons en détail et critiquerons un algorithme génétique hybride proposé par Tavares et al. [112]. Nous proposerons ensuite une version revisitée de celui-ci offrant une plus grande flexibilité d'utilisation.

Nous œuvrerons dans le cas concret du partitionnement de la Wallonie. En utilisant les techniques citées plus haut, nous proposerons une méthode et des outils permettant d'obtenir un partitionnement territorial optimal de la Région wallonne, en réponse à une gamme étendue de besoins exprimés sous forme de critères à optimiser.

Outre l'application au cas spécifique de la Région wallonne, nous proposerons trois apports originaux :

Tout d'abord, l'évaluation comparative des deux principaux algorithmes génétiques envisagés aujourd'hui dans le contexte du partitionnement territorial. L'approche proposée par Tavares et al. [112] et celle de Baçao et al. [21] n'ayant, à notre connaissance, pas encore été mises en perspective à ce jour.

Notre second apport porte sur la prise en compte de la similitude entre les solutions optimales générées et un partitionnement de référence. Il est en effet fréquemment désirable que les solutions optimales fournies ne diffèrent pas fondamentalement du partitionnement territorial actuel. Bien que mentionné comme souhaitable par Tavares et al., la mise en œuvre de ce critère avec un algorithme génétique n'a pas encore été élaborée, à notre connaissance, dans la littérature.

Un dernier apport, présenté essentiellement sous forme de piste, porte sur la parallélisation des algorithmes utilisés en vue d'en accroître l'efficacité.

## 1.2 Structure du mémoire

Cette section vise à présenter l'organisation du mémoire et l'agencement des différents chapitres qui le composent. Le mémoire est divisé en deux parties. La première, appelée *état de l'art*, vise à introduire les différentes connaissances nécessaires à la compréhension de l'intégralité de ce travail. La seconde partie, nommée *apports personnels*, présente les résultats des recherches menées durant l'année.

La première partie est composée de trois chapitres. Le premier a pour objectif d'introduire le domaine de l'optimisation multiobjectif et les différentes notions qui lui sont associées. Le second chapitre introduit le concept de métaheuristique, technique utilisée pour résoudre les problèmes d'optimisation. En particulier, il traite des algorithmes évolutionnistes, une catégorie particulière de métaheuristique. Le troisième et dernier chapitre de la première partie aborde, tout d'abord, dans un cadre général, la notion de système d'information géographique. Il présente ensuite le problème d'optimisation connu sous le nom de partitionnement (*districting* en anglais) et aborde également brièvement le problème de regroupement.

La seconde partie est composée de trois chapitres. Le premier présente et critique les recherches [112] réalisées par Fernando Tavares-Pereira, José Rui Figueira, Vincent Mousseau et Bernard Roy. Ceux-ci proposent un algorithme évolutionniste hybride pour résoudre le problème de partitionnement. Le second chapitre de cette deuxième partie présente l'algorithme implémenté dans le cadre de ce mémoire ainsi que son utilisation pour un problème de partitionnement bien précis : celui de la Wallonie. Nous y présentons les différentes procédures composant de l'algorithme, les données et critères propres à la Wallonie. Une section couvre ensuite le paramétrage de l'algorithme et quelques résultats sont proposés. Enfin, le dernier chapitre s'inscrit dans la continuité du précédent en proposant des extensions à l'algorithme implémenté. La première extension est l'utilisation d'un encodage différent pour les solutions. Cet encodage est implémenté et un comparatif des résultats est proposé. La seconde extension aborde la génération de partitions similaires à la partition actuelle de la Wallonie. Enfin, la dernière extension propose des pistes pour la parallélisation de l'algorithme.

### 1.3 Notations

Le tableau ci-dessous reprend les différentes notations utilisées dans ce mémoire et leur correspondance. Celles-ci sont tirées de différents ouvrages [86, 110, 42, 107, 112].

Notation	Description
$\mathcal{X}$	Ensemble réalisable d'un problème d'optimisation
$\hat{\mathcal{X}}$	Ensemble Pareto optimale
$\mathcal{Y} \equiv F(\mathcal{X})$	Ensemble réalisable dans l'espace d'objectif
$\hat{\mathcal{Y}} \equiv F(\hat{\mathcal{X}})$	Frontière Pareto optimale
$\mathbb{R}^p$	Espace d'objectif
$x = (x_1, \dots, x_n)$	Vecteur variable
$y = (y_1, \dots, y_p)$	Vecteur de valeurs de fonctions objectif
$F = (f_1, \dots, f_p) : \mathcal{X} \rightarrow \mathbb{R}^p$	Vecteur de fonctions objectif
$Y = \{y_1, y_2, \dots, y_p\}$	Partition d'un territoire en $p$ zones.
$y_i = \{e_1, e_2, \dots, e_q\}$	Une zone composée de $q$ éléments territoriaux.

## 1.4 Vocabulaire

Dans le souci d'éviter toute ambiguïté, cette section introduit les différents termes francophones utilisés au travers de ce mémoire ainsi que leur équivalent que l'on trouve dans la littérature anglophone.

Terme anglais	Terme français utilisé
Total flow time	Temps d'écoulement total
Decision space	Espace de décision
Objective space or Criterion Space	Espace d'objectif
Criterion-based	Basé sur les critères
Dominance rank	Rang de dominance
Dominance depth	Profondeur de dominance
Dominance count	Compte de dominance
Kernel Methods	Méthodes noyau
Nearest-Neighbor Methods	Méthodes du plus proche voisin
Evolutionary algorithm	Algorithme évolutionniste
Spread	Propagation
Geographic information system	Système d'information géographique
Districting	Partitionnement
Clustering	Regroupement
k-means algorithm	Algorithme des k-moyennes
Encoding of a solution	Encodage/représentation d'une solution
Merging procedure	Procédure de fusion
Zone compactness	Degré de compactage des zones
Algorithm tuning	Paramétrage d'algorithme

**Première partie**

**Etat de l'art**

## Chapitre 2

# Optimisation multiobjectif

### 2.1 Introduction

Dans ce premier chapitre d'état de l'art, nous aborderons diverses notions de recherche opérationnelle. L'objectif est d'introduire les concepts de base nécessaires à la compréhension du reste du mémoire. Ce chapitre ne prétend pas couvrir l'intégralité du domaine. Il s'agit plutôt d'une introduction ciblée à cette discipline permettant d'aborder, dans le chapitre 3, les métahéuristiques et plus particulièrement les algorithmes évolutionnistes.

### 2.2 Aide à la décision

Depuis toujours, l'homme est confronté à la prise de décisions. Il doit choisir une alternative parmi plusieurs en vue de répondre de manière optimale à un problème particulier. Cela nécessite donc de comparer les différentes possibilités selon différents critères afin de sélectionner la meilleure solution ou, parfois, un ensemble de solutions satisfaisantes.

Bien souvent, nous nous basons sur notre intuition et nous donnons une importance plus ou moins grande à chacun des critères d'un problème de manière implicite.

Un critère est alors défini comme une fonction sur l'ensemble des solutions possibles du problème donnant une mesure de la qualité de celles-ci selon un point de vue. Cette notion de critère est illustrée à plusieurs reprises tout au long de ce chapitre. En particulier, un premier exemple est donné à la Section 2.3.

Toutefois, pour des choix de grande importance, il est évident que l'*intuition* est insuffisante. La discipline appelée *aide à la décision* fait partie de la recherche opérationnelle et vise à répondre de manière rationnelle à ce besoin. Cette science est utilisée dans de nombreux domaines, allant de l'informatique à la finance en passant par la politique.

L'abréviation *MCDM*, pour *Multiple Criteria Decision Making*, apparaît pour la première fois en 1979 dans l'article “MCDM—If Not a Roman Numeral, Then What?”[124] par Zions, S et a été largement adoptée dans la littérature scientifique depuis sa parution [66].

Le domaine de l'aide à la décision multicritère peut être considéré comme ancien ou récent [66] ; il est évident que l'homme a toujours été confronté à des choix mettant en scène de nombreux critères. On retrouve par exemple des discussions de Benjamin Franklin qui a vécu au 18<sup>ème</sup> siècle, mettant en évidence les différents compromis pour des décisions importantes qu'il devait prendre. Toutefois, on attribue les fondements de l'aide à la décision multicritère contemporaine aux années 1950.

Les auteurs Koksalan, M. and Wallenius, J. et Zions, S. mentionnent, dans leur ouvrage *Multiple Criteria Decision Making : From Early History to the 21st Century*, les domaines suivants comme faisant partie de l'aide à la décision multicritère :

Decision Analysis, Goal Programming, work of the *French School*, which includes outranking relations, Multiple Objective Mathematical Programming, Fuzzy Set Theory, the Analytic Hierarchy Process (AHP) and Evolutionary Multiobjective Optimization (EMO)

## 2.3 Problèmes d'optimisation mono-objectif

Cette section aborde brièvement les problèmes d'optimisation mono-objectif dans le but de mettre en évidence, dans la section suivante, les difficultés inhérentes aux problèmes d'optimisation multiobjectif.

Un problème mono-objectif est un problème où seul un critère est considéré. Il s'agit bien sûr du cas le plus simple de problème d'optimisation dans lequel une solution optimale, ou un ensemble de solutions équivalentes pour le critère étudié, est disponible. Un tel problème peut être formalisé mathématiquement de la manière suivante :

$$\begin{aligned} & \max f(x) \\ \text{s.c. } & x \in \mathcal{X} \end{aligned} \tag{2.1}$$

$f(x)$  est appelée fonction objectif et correspond au critère que l'on essaie de maximiser<sup>1</sup>.  $x$  est une solution possible au problème appartenant à  $\mathcal{X}$  qui est l'ensemble des solutions réalisables pour le problème étudié.

Prenons un exemple pour clarifier ces différentes notions. Un maquettiste désire réaliser des figurines en bois. Il a reçu de son fournisseur une liste des différents bois qu'il peut commander. Seuls certains bois sont néanmoins utilisables pour fabriquer ses figurines. Il faut par exemple que le bois soit assez robuste et puisse résister à l'eau. Ces deux contraintes réduisent son choix à un sous-ensemble du catalogue. Ce sous-ensemble est appelé l'espace de décision et correspond aux alternatives éligibles pour son problème. Le seul critère du maquettiste est le prix, qu'il va chercher à minimiser. On voit directement qu'il prendra simplement le bois le moins cher parmi l'ensemble des solutions. Il existe donc une solution optimale unique.

L'optimisation mono-objectif est une discipline étudiée depuis longtemps [107] et correspond à un cas particulier de d'optimisation.

---

1. ou de minimiser

Nom	Prix	Autonomie (appel)	Résolution caméra	Taille de l'écran	Poids
Apple iPhone 5	665\$	8h	8 MegaPixel	4"	112 Grams
Samsung Galaxy S3	550\$	22h	8 MegaPixel	4.8"	133 Grams
Nokia Limia 920	499\$	17h	8.7 MegaPixel	4.5"	185 Grams
LG Nexus 4	299\$	15h	8 MegaPixel	4.7"	139 Grams
Samsung Galaxy Note II	650\$	35h	8 MegaPixel	5.5"	180 Grams
Blackberry Torch 9850	500\$	7h	5 MegaPixel	3.7"	135 Grams

**Table 2.1** – Différents Smartphones et les valeurs de différents critères.

Données provenant du site [findthebest.com](http://findthebest.com) [45]

Nous allons maintenant nous intéresser aux problèmes d'optimisation multiobjectif. Ceux-ci traitent des problèmes ayant plusieurs objectifs<sup>2</sup>. Ces derniers étant souvent contradictoires comme nous allons le voir dans la section suivante.

## 2.4 Problèmes d'optimisation multiobjectif

Les problèmes d'optimisation multiobjectif font apparaître non plus un mais une série d'objectifs différents que l'on va chercher à maximiser ou minimiser. Dans la première partie de cette section, nous allons énoncer différents exemples. Ceux-ci nous permettront par la suite de mettre en évidence une classification des différents problèmes d'optimisation.

**Premier exemple** Un exemple simple pourrait être celui d'une personne désireuse d'acheter un nouveau Smartphone (un autre exemple similaire qui revient beaucoup dans la littérature est l'achat d'une nouvelle voiture [42]). Son choix quant à la marque et au modèle du téléphone dépend très certainement de plusieurs critères. Le prix, pourrait bien sûr être l'un des plus importants. L'autonomie du téléphone en communication, la résolution de l'appareil photo, la taille de l'écran ainsi que le poids total de l'appareil font également partie des critères pouvant être pris en compte. La table 2.1 met en évidence différentes possibilités d'achat avec la valeur des différents critères mentionnés ci-dessus.

**Second exemple** Reprenons l'exemple du maquettiste. Celui-ci vient d'acheter 400 unités de balsa et 200 unités d'acajou. Il veut utiliser ces bois pour fabriquer des avions et des voitures miniatures et les vendre. Chacune de ces deux figurines nécessite les deux bois dans des proportions différentes.

Son premier objectif est de maximiser ses bénéfices. Toutefois, le maquettiste vient d'acquérir une machine permettant de faire une grosse partie du travail à sa place. 70% du travail pour réaliser un avion peut être réalisé par la machine. Pour la voiture, seul 40% du travail peut être réalisé par la machine. Le maquettiste veut bien sûr maximiser le travail effectué par la machine.

---

2. Le terme critère est interchangeable avec le terme objectif dans le cadre de ce chapitre

Jouet	Balsa	Acajou	Prix de vente	Proportion par machine
Avion	6	1	4€	70%
Voiture	2	4	7€	40%

**Table 2.2** – Données relatives au problème d’optimisation du maquettiste.

Le choix que doit faire le maquettiste, qu’on appelle également les *variables* du problème, sont le nombre d’avions et le nombre de voitures qu’il va produire. Les *contraintes* du problème, quant à elles, sont les quantités limitées de balsa et d’acajou dont dispose le maquettiste.

La table 2.2 reprend les quantités de bois nécessaires pour chacune des figurines ainsi que leurs valeurs pour les objectifs.

Si à première vue l’ajout de plusieurs fonctions d’objectif peut sembler quelconque, cela implique des propriétés très importantes qui font des problèmes d’optimisation multiobjectif un domaine d’étude à part entière.

La première propriété qui aura un impact important est le fait que les différents objectifs d’un problème peuvent être conflictuels.

**Troisième exemple** Voici un exemple mettant en évidence la nature conflictuelle qui peut exister entre les différents objectifs d’un problème.

Une équipe d’ingénieurs, responsable de la construction d’une nouvelle centrale nucléaire envisage les différents matériaux de construction possibles pour celle-ci. Il est évident que cette équipe cherche à minimiser le coût de la construction, ce qui la dirige vers les matériaux les moins coûteux. D’autre part, la sécurité de la centrale est un autre critère qu’elle cherche à maximiser. On pourrait imaginer que le fait de rendre la centrale plus fiable nécessite l’utilisation de plus de matériaux ou de matériaux plus onéreux. On voit donc apparaître une propriété importante des problèmes d’optimisation multiobjectif due aux objectifs parfois conflictuels. Il n’existe plus, comme c’était le cas pour les problèmes mono-objectif, une seule solution optimale globale mais un ensemble de solutions optimales.

De plus, on peut montrer que plus le nombre d’objectifs sera important, plus le nombre de solutions optimales sera grand lui aussi. Par ailleurs, la majorité des problèmes d’optimisation multiobjectif concernent des problèmes dits NP-Hard [110]. Il est suffisant pour le moment de savoir que pour les problèmes de ce type, aucun algorithme de résolution exacte s’exécutant en un temps polynomial n’a été trouvé. Il est d’ailleurs fortement pensé que de tels algorithmes n-existent pas. Nous verrons au chapitre suivant que les métahéuristiques permettent d’approximer en un temps raisonnable l’ensemble des solutions optimales d’un problème d’optimisation. Elles sont donc souvent utilisées lorsque l’on s’intéresse aux problèmes NP-Hard.

**Quatrième exemple : Flow-Shop** Le problème de Flow-Shop est un problème bien étudié dans la littérature [94, 122, 85]. Ce problème, connu pour être NP-Hard, peut être formalisé comme suit : un ensemble de  $n$  travaux doivent être traités par  $m$  machines. Chaque travail a un temps d’exécution différent sur chacune des machines et doit passer

par chacune d'elle dans le même ordre. De plus, chaque machine ne peut traiter qu'un seul travail à la fois.

Nous utilisons pour cet exemple les notations suivantes [122] :

- $n$  est le nombre de travaux devant être traités.
- $m$  est le nombre de machines composant la chaîne d'exécution.
- $t(i, j)$  est le temps de traitement du travail  $i$  sur la machine  $j$ .  $i \in \{1, \dots, n\}, j \in \{1, \dots, m\}$
- $\{\pi_1, \pi_2, \dots, \pi_n\}$ , l'ordonnancement des  $n$  travaux.

(2.2)

On peut alors définir  $C(\pi_i, j)$ , le moment où le travail  $i$  a terminé son traitement sur la machine  $j$  comme suit [94, 122] :

$$\begin{aligned} C(\pi_1, 1) &= t(\pi_1, 1) \\ C(\pi_i, 1) &= C(\pi_{i-1}, 1) + t(\pi_i, 1) \quad i \in \{2, \dots, n\} \\ C(\pi_1, j) &= C(\pi_1, j-1) + t(\pi_1, j) \quad j \in \{2, \dots, m\} \\ C(\pi_i, j) &= \max(C(\pi_{i-1}, j), C(\pi_i, j-1)) + t(\pi_i, j) \quad i \in \{2, \dots, n\}; j \in \{2, \dots, m\} \end{aligned} \quad (2.3)$$

Résoudre le problème revient donc à trouver un ordonnancement pour ces  $n$  travaux qui optimisent un ou plusieurs objectif(s). On peut par exemple citer la minimisation du makespan. Le makespan est le temps total de complétion. Il s'agit donc du temps écoulé entre le début du traitement du premier travail sur la première machine et la fin du traitement du dernier travail sur la dernière machine. La fonction d'objectif associée est alors donnée par :

$$f_1 = C(\pi_n, m) \quad (2.4)$$

D'autres critères comme la minimisation du temps d'inactivité des machines ou encore le temps d'écoulement total sont envisageables. Le temps d'écoulement total peut être calculé comme suit :

$$f_2 = \sum_{i=1}^n C(\pi_i, m) \quad (2.5)$$

#### 2.4.1 Problèmes discrets et continus

Les problèmes d'optimisation multiobjectif peuvent être classés selon un premier critère, ils peuvent être discrets, continus ou mixtes.

Si l'on reprend les exemples énoncés précédemment, on peut constater que le premier exemple, relatif au choix d'un Smartphone, est un problème discret. La personne doit choisir parmi un ensemble fini de possibilités.

Il y a toutefois une distinction importante à faire ici. Certains problèmes discrets ont un petit ensemble de solutions, que l'on peut facilement énumérer. C'est d'ailleurs le cas de l'exemple du Smartphone. Par contre, d'autres problèmes discrets ont un ensemble de solutions possibles tellement grand qu'il est impensable de faire une recherche exhaustive afin de trouver les meilleures solutions. Nous reviendrons sur cette distinction et l'illustrerons à la Section 3.2.

Si l'on considère le troisième exemple, relatif au choix de matériaux de construction d'une centrale nucléaire, on peut constater qu'il s'agit d'un problème continu puisqu'un

nombre infini, ou tout du moins non dénombrable, de formes et matériaux peut être considéré pour la construction de la centrale nucléaire.

Un problème d'optimisation est dit mixte lorsque certaines de ses variables de décision sont discrètes tandis que d'autres sont continues. Nous reviendrons sur la notion de variable de décision dans la section 2.4.3.

### 2.4.2 Problèmes explicites et implicites

Un second critère de classification des problèmes d'optimisation classe ces derniers en deux catégories, selon que les différentes choix possibles du problème soient explicites ou implicites [42].

Le premier exemple est un problème explicite puisque les différents modèles de Smartphones sont présentés explicitement.

Par contre le second exemple met en évidence un problème implicite puisque les alternatives ne sont pas listées explicitement mais sont données de façon implicite par les contraintes du problème.

Les problèmes discrets et explicites, comme l'exemple du choix de Smartphone, relèvent de l'aide à la décision et non de l'optimisation multiobjectif [42].

Par contre, les problèmes discrets où les contraintes sont données sous forme de fonctions mathématiques, sont appelés *problèmes combinatoires* et relèvent bien de l'optimisation multiobjectif.

### 2.4.3 Formulations

Cette section vise à formaliser les différents concepts relatifs aux problèmes d'optimisation multiobjectif.

Le domaine de l'optimisation multiobjectif est une branche de l'aide à la décision multicritère qui fait elle-même partie comme mentionné précédemment de la recherche opérationnelle. La différence entre l'optimisation multiobjectif et l'aide à la décision multicritère est que la première a pour but de trouver l'ensemble des solutions optimales à un problème donné tandis que l'aide à la décision vise à sélectionner une solution acceptable parmi celles-ci.

**Définition 1. Problème d'optimisation multiobjectif** Un MOP (*Multiobjective Optimization Problem*) peut s'écrire sous la forme suivante [42] :

$$MOP = \begin{cases} \min F(x) = (f_1(x), f_2(x), \dots, f_p(x)) \\ s.c. \quad x \in \mathcal{X} \end{cases} \quad (2.6)$$

$F(x)$  est le vecteur de fonction objectif.  $f_1(x)$  à  $f_n(x)$  sont les différents objectifs que l'on cherche à minimiser. Sans perte de généralité, nous ne considérons que le cas où l'on cherche à minimiser les fonctions objectifs. En effet, maximiser  $f_i(x)$  revient à minimiser  $-f_i(x)$ <sup>3</sup>.

---

3. pour autant que l'on puisse exprimer  $f_i(x)$  par des nombres

$\mathcal{X}$  est l'ensemble des solutions réalisables du problème étudié et  $x = \{x_1, \dots, x_n\}$  correspond au vecteur de décision d'une alternative réalisable.

Dans la littérature, on appelle l'*espace de décision* du problème, l'espace dont  $\mathcal{X}$  est un sous-ensemble.

$F(x) = (f_1(x), \dots, f_p(x)) : \mathcal{X} \rightarrow \mathbb{R}^p$  envoie l'espace de décision sur l'espace d'objectif dans lequel on peut comparer la qualité de chaque possibilité.

Si l'on reprend le second exemple mentionné plus haut, l'espace  $\mathbb{N}^2$  est l'espace de décision. Il contient l'ensemble des solutions<sup>4</sup>  $x = \{\text{nombre d'avions, nombre de voitures}\}$ .

L'espace d'objectif, est l'espace  $\mathbb{R}^2$ . Grâce au vecteur objectif  $F(x)$ , on peut passer de l'espace de décision à l'espace d'objectif. Dans ce cas-ci :  $F(x) = \{\text{bénéfices}(x), \text{travail machine}(x)\}$

Afin de pouvoir comparer différentes solutions entre elles, on utilise la notion de *dominance Pareto*.

**Définition 2. Dominance Pareto** Nommée de par l'économiste italien Vilfredo Pareto, cette notion a été introduite pour caractériser un système sous satisfaction économique maximale c'est à dire, un système ou personne ne peut être rendu plus heureux sans rendre quelqu'un d'autre moins heureux.

L'image d'une solution dans l'espace d'objectif domine une autre si la valeur de chacun des objectifs est meilleure<sup>5</sup>.

Le concept de dominance est illustré à la Figure 2.1 pour le problème biobjectif de Flow-Shop que nous avons décrit précédemment.

Une distinction est souvent faite entre les notions de *dominance faible* et de *dominance stricte*[110].

**Définition 3. Dominance faible** Un vecteur objectif  $u = \{u_1, \dots, u_p\}$  domine faiblement un autre vecteur objectif  $v = \{v_1, \dots, v_p\}$  si et seulement si :

$$\forall i \in \{1, \dots, p\}, u_i \leq v_i \quad (2.7)$$

**Définition 4. Dominance stricte** Un vecteur objectif  $u = \{u_1, \dots, u_p\}$  domine strictement un autre vecteur objectif  $v = \{v_1, \dots, v_p\}$  si et seulement si :

$$\forall i \in \{1, \dots, p\}, u_i < v_i \quad (2.8)$$

Lorsque nous ne précisons pas si une dominance est faible ou stricte, nous parlons implicitement de dominance faible. Une solution dont le vecteur objectif n'est pas dominé est appelée Pareto optimale. Il s'agit d'une solution telle qu'il est impossible de trouver une autre solution qui améliore un des objectifs sans en dégrader un autre. Le terme *solution efficace* est également utilisé dans la littérature [42] et est interchangeable avec l'expression Pareto optimal.

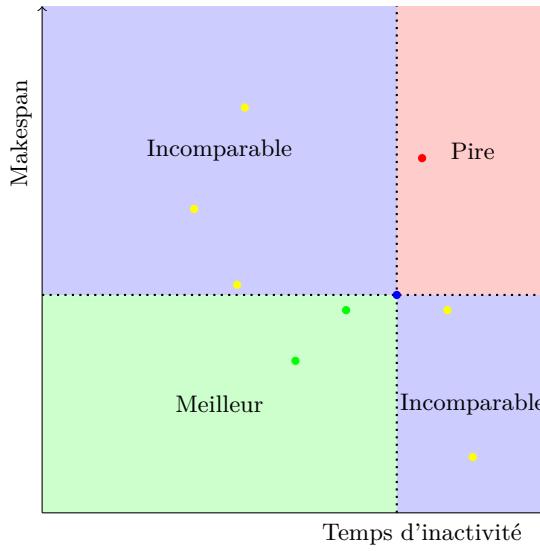
**Définition 5. Pareto optimal** Une solution  $x_u \in \mathcal{X}$  est dite Pareto optimale si et seulement si il n'existe pas de solution  $x_v \in \mathcal{X}$  avec  $v = F(x_v)$  qui domine  $u = F(x_u)$ . C'est-à-dire, il n'existe pas de  $x_v$  tel que :

$$\forall i \in \{1, \dots, p\}, v_i \leq u_i \wedge \exists i \in \{1, \dots, p\} \mid v_i < u_i \quad (2.9)$$

---

4. réalisables mais également non réalisables

5. Plus petite dans notre cas puisque nous cherchons à minimiser les objectifs



**Figure 2.1** – Les points dans la zone verte sont sans équivoque meilleurs que le point bleu puisque le dernier travail est achevé plus tôt et les machines sont inactives moins longtemps. Les points dans la zone rouge sont, eux, clairement moins bons. Toutefois, puisque les points dans les deux zones bleues sont meilleurs selon une dimension mais moins bons selon l'autre, ils ne peuvent être qualifiés de meilleurs ou moins bons.

**Définition 6. Ensemble Pareto optimal** *Cet ensemble de solutions Pareto optimales est appelé l'ensemble Pareto optimal. Nous le notons  $\hat{\mathcal{X}}$ .*

**Définition 7. Frontière Pareto** *L'image dans l'espace d'objectif de l'ensemble Pareto optimal est appelée la frontière Pareto ou encore l'ensemble non dominé.*

$$\hat{\mathcal{Y}} = F(\hat{\mathcal{X}}) \quad (2.10)$$

La Figure 2.2 illustre cette dernière notion, toujours pour un problème bi-objectif.

**Définition 8. Point idéal et point nadir** *Le point idéal est défini par :*

$$y^i = \left\{ \min_{\hat{x} \in \hat{\mathcal{X}}} f_1(\hat{x}), \dots, \min_{\hat{x} \in \hat{\mathcal{X}}} f_p(\hat{x}) \right\} \quad (2.11)$$

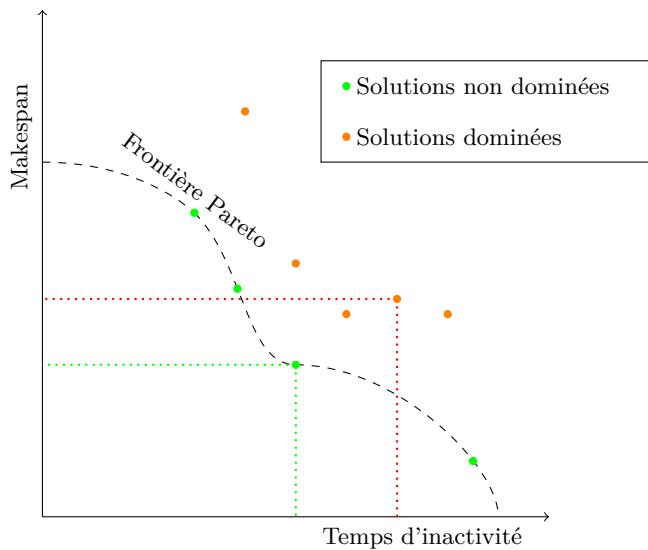
Le point idéal correspond au point dans l'espace d'objectif ayant, pour chaque objectif, la plus petite valeur parmi les solutions de la frontière Pareto.

Le point nadir est défini par :

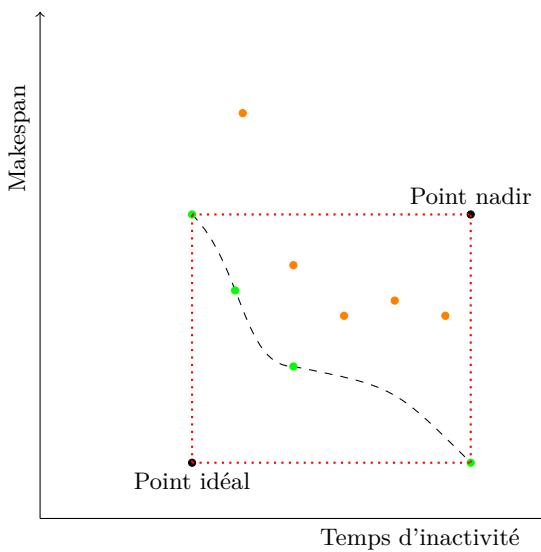
$$y^n = \left\{ \max_{\hat{x} \in \hat{\mathcal{X}}} f_1(\hat{x}), \dots, \max_{\hat{x} \in \hat{\mathcal{X}}} f_p(\hat{x}) \right\} \quad (2.12)$$

Le point nadir correspond au point dans l'espace d'objectif ayant, pour chaque objectif, la plus grande valeur parmi les solutions de la frontière Pareto. Ces deux points sont souvent utilisés pour avoir une idée des bornes de la frontière Pareto. La Figure 2.3 illustre ces deux points pour le problème de Flow-Shop.

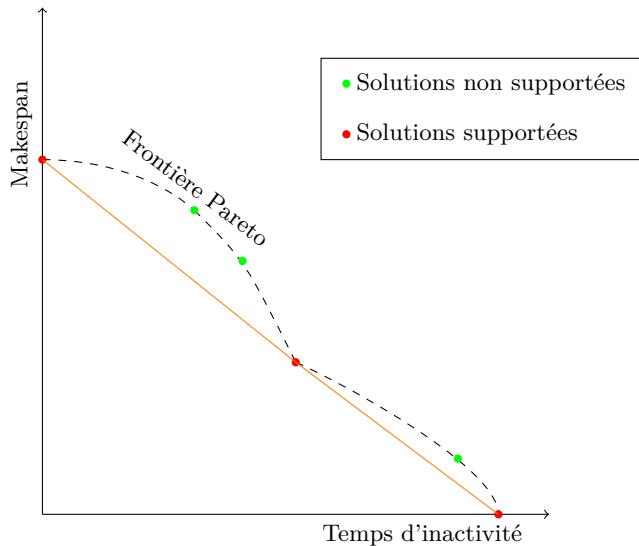
**Définition 9. Solution supportée** *Les solutions supportées sont les solutions trouvées lorsque l'on résout le problème d'optimisation multiobjectif en agrégeant les différents*



**Figure 2.2** – Illustration de la frontière Pareto optimale : les points verts le long de la ligne pointillée (la frontière Pareto) sont tous Pareto optimaux : aucun autre point ne les domine. Les points en orange sont eux dominés : pour chacun d'eux au moins un point est meilleur pour tous les objectifs.



**Figure 2.3** – Illustration du point idéal et du point nadir. Ces deux points donnent une bonne idée des bornes de la frontière Pareto optimale.



**Figure 2.4** – Illustration de la notion de solution supportée. Les points rouges sont des solutions supportées. Elles seront trouvées par une méthode d’agrégation. Les solutions en vert quant à elles sont non supportées et ne seront pas trouvées par une méthode d’agrégation, peu importe les coefficients  $\lambda_i$  choisis.

objectifs. Le problème devient alors un problème mono-objectif. La fonction objectif à minimiser devient alors :

$$\begin{aligned} \min F(x) &= \sum_{i=1}^p \lambda_i f_i(x) \\ \text{s.c. } x &\in \mathcal{X} \end{aligned} \tag{2.13}$$

Les solutions optimales de ce problème, pour différents coefficients  $\lambda_i$ , sont appelées solutions supportées du problème multiobjectif initial. La Figure 2.4 illustre la notion de solution supportée en prenant l'exemple du Flow-Shop bi-objectif.

Dans cette section nous avons défini les problèmes d’optimisation multiobjectif ainsi que les principaux concepts qui y sont inhérents. Dans la section suivante, nous allons nous intéresser brièvement à la notion de résolution d’un MOP et aux différentes techniques qui existent pour y parvenir.

## 2.5 Résolution de MOPs

Résoudre un problème d’optimisation multiobjectif peut avoir différentes significations. Comme nous l’avons vu à la section précédente, ces problèmes ont une multitude de solutions Pareto optimales et celles-ci sont incomparables mathématiquement.

Dans le cadre de ce mémoire nous définirons *résoudre un MOP* comme équivalent à trouver la frontière Pareto optimale du problème considéré. Il est alors du ressort de l'aide à la décision, MCDA<sup>6</sup>, d'assister le preneur de décision dans sa sélection d'une solution parmi celles qui composent la frontière et qui sont considérées comme

---

6. MultiCriteria Decision Aid

mathématiquement équivalentes au niveau de leur optimalité, nous reviendrons sur ceci dans la Section 2.6.

Cette section ne se veut pas exhaustive dans l'énumération des techniques de résolution qui existent. Nous nous concentrerons sur un type de MOP particulier.

Comme nous l'avons défini à la Section 2.4.1, les problèmes d'optimisation peuvent être continus. Dans ce cas, la frontière Pareto peut être de taille infinie. Autrement, ils sont combinatoires (discrets). Dans ce dernier cas, l'espace de décision peut toutefois être trop grand pour être énuméré par des techniques d'analyse informatique. Nous avons également vu à la Section 2.4 que la majorité des MOPs traitent de problèmes NP-Hard, nous allons nous restreindre à ce type de problèmes d'optimisation pour lesquels des techniques d'approximation sont parfois préférées aux techniques de résolution exactes.

Les techniques de résolution misent en place pour résoudre ce type de MOPs font l'objet du chapitre suivant. Nous nous intéresserons en particulier aux algorithmes évolutionnistes qui se montrent en général efficaces pour résoudre de tels problèmes [57].

## 2.6 Techniques d'aide à la décision

Les techniques d'aide à la décision ont pour objectif d'aider le preneur de décision, c'est-à-dire un être humain, dans la sélectionner d'une solution dans l'ensemble Pareto optimal trouvé par un algorithme de résolution en fonction de ses préférences.

**Définition 10. Préférence du preneur de décision** *On entend par préférence, l'importance que va donner le preneur de décision à chaque objectif du problème par rapport aux autres.*

Ces techniques sont généralement rangées dans trois catégories, les méthodes *a priori*, les méthodes *a posteriori* et les méthodes *interactives*.

**Méthode a priori** Dans les méthodes a priori, le preneur de décision exprime ses préférences avant d'appliquer les techniques d'optimisation.

Un exemple très simple de méthode a priori utilise la notion de fonction d'utilité. Celle-ci est définie par le preneur de décision au préalable et définit un ordre total sur l'ensemble  $\mathcal{Y}$ . Ainsi, le problème multiobjectif peut être ramené à un problème mono-objectif où l'on cherche à maximiser l'utilité. On remarque cependant [110] qu'il est rarement aisé, voire impossible, de définir de telles fonctions d'utilité qui reprennent toutes les préférences du preneur de décision.

**Méthode a posteriori** Les méthodes a posteriori sont naturellement l'opposé des techniques a priori. Elles consistent à aider le preneur de décision à sélectionner une solution parmi l'ensemble Pareto optimal, après avoir trouvé ou, plus généralement, approximé ce dernier en utilisant différentes techniques de résolution.

**Méthode interactive** Les méthodes interactives font intervenir les choix du preneur de décision tout au long de la recherche des alternatives optimales. Ces décisions intermédiaires permettent de guider la recherche des solutions tout en tenant compte des préférences du preneur de décision.

Une façon très simple de résumer ces trois approches est proposée dans [115] :

- **A priori** : Décision → Recherche
- **A posteriori** : Décision ← Recherche
- **Interactive** : Décision ↔ Recherche

## 2.7 Conclusion

Dans ce chapitre nous avons introduit et défini les problèmes d'optimisation multiobjectif ainsi que les principaux concepts et notions qui leur sont inhérents.

Les problèmes d'optimisation apparaissent dans de très nombreux domaines, de l'économie à la finance, aux sciences de l'ingénierie, aux transports, à la gestion de l'environnement ou même simplement dans la conception d'un produit dans n'importe quelle industrie.

Ces notions vont nous permettre, dans le chapitre suivant, d'aborder les mét heuristicques. Celles-ci sont de puissantes techniques permettant de résoudre des problèmes d'optimisation parfois très complexes.

## Chapitre 3

# Algorithmes évolutionnistes

### 3.1 Introduction

L'objectif de ce chapitre est tout d'abord d'aborder la notion de métaheuristique ainsi que les concepts clés qui y sont liés. Ceci permettra, par la suite, d'aborder les algorithmes évolutionnistes qui font partie de la famille des métaheuristiques et sont utilisés pour résoudre des problèmes d'optimisation multiobjectif.

Le domaine de l'optimisation ou même des métaheuristiques étant extrêmement vaste, ce chapitre ne se veut pas exhaustif, mais vise plutôt à introduire les connaissances qui sont nécessaires pour pouvoir comprendre l'entièreté de ce mémoire mais également pour pouvoir situer les différentes disciplines abordées les unes par rapport aux autres.

### 3.2 Résolution de problèmes d'optimisation

Il existe bien entendu de très nombreuses manières de résoudre des problèmes d'optimisation. En effet, ceux-ci pouvant être de natures très différentes, une méthode de résolution sera tantôt plus adaptée tantôt moins. Il convient donc d'utiliser une méthode adéquate au problème que l'on considère.

De la manière la plus générale, un problème d'optimisation est composé de deux éléments[110] : Le premier  $S$ , appelé espace de recherche, contient toutes les alternatives possibles au problème étudié. Cet espace peut être fini ou infini. Le second élément est une ou des fonction(s) d'objectif  $f$  qui assigne à chaque  $s \in S$  une valeur  $f(s) \in \mathbb{R}$ . La ou les fonction(s)  $f$  définissent aussi un ordre total sur les éléments de  $S$ . Résoudre un tel problème d'optimisation revient à trouver les valeurs  $\hat{s} \in S$  qui ont la plus grande valeur d'objectif pour chacune des fonctions  $f$ .

**Définition 11.** *Ordre total* Une relation  $\leq$  est un ordre total sur un ensemble  $S$  si elle satisfait les propriétés suivantes [19] :

- Réflexivité :  $a \leq a \quad \forall a \in S$
- Antisymétrique :  $a \leq b \wedge b \leq a \rightarrow a = b$
- Transitivité :  $a \leq b \wedge b \leq c \rightarrow a \leq c$
- Pour chaque  $a, b \in S$ , soit  $a \leq b$  ou  $b \leq a$

De très nombreuses classifications des problèmes d'optimisation sont proposées dans la littérature [86]. Celles-ci se complètent souvent puisqu'elles proposent chacune une classification faite selon un aspect particulier des problèmes d'optimisation.

La classification basée sur le type d'équations qui composent le problème est fréquemment utilisée. On retrouve alors les catégories suivantes :

**Problème d'optimisation linéaire** Les fonctions d'objectif ainsi que les différentes contraintes de ce type de problèmes sont exprimées sous forme d'équations linéaires. Le second exemple, vu au premier chapitre 2.4, fait partie de cette catégorie.

**Problème d'optimisation non linéaire** Il s'agit d'une forme très générale de problème d'optimisation qui contient tout problème dont au moins une équation (de contrainte ou objectif) n'est pas linéaire.

**Problème d'optimisation géométrique** Un problème est dit géométrique si la fonction d'objectif et les différentes contraintes sont exprimées sous la forme d'équations polynomiales. Il s'agit d'une sous-classe des problèmes d'optimisation non linéaire.

**Problème d'optimisation quadratique** Il s'agit d'un cas particulier de problème non linéaire pour lesquels les fonctions d'objectif sont quadratiques alors que les différentes contraintes sont linéaires.

Comme nous l'avons déjà vu au chapitre précédent une seconde classification peut être réalisée selon que le problème soit continu ou discret.

**Problème continu** Pour un tel problème, les variables de décision prennent des valeurs réelles. Ce type de problème d'optimisation est très étudié et de nombreux algorithmes de résolution existent [110].

**Problème discret** Également appelés problèmes entiers, ces problèmes sont caractérisés par des variables de décision discrètes. Le problème de Flow-Shop qui a été présenté au chapitre précédent en est un exemple. Si les variables de décision sont, pour chaque travail, sa position dans l'ordonnancement, elles prennent bien des valeurs discrètes.

**Problème mixte** Les problèmes mixtes sont des problèmes comportant à la fois des variables de décision discrètes et des variables de décision continues.

Nous allons maintenant nous intéresser aux techniques de résolution des différents problèmes d'optimisation.

Les problèmes linéaires sont des problèmes bien étudiés et pour lesquels des techniques de résolution exacte sont disponibles. Nous citerons par exemple, à titre d'information, le très populaire algorithme du Simplex [36], proposé par George Dantzig en 1947.

Si des techniques de résolution exacte existent pour certains problèmes non linéaires, pour de grandes instances de tels problèmes, ces techniques se révèlent souvent inefficaces de par leur lenteur. Des techniques d'approximation donnant plus rapidement des solutions satisfaisantes sont alors utilisées. Les métahéuristiques en sont un exemple.

Considérons maintenant les problèmes discrets et plus généralement les problèmes appelés *problèmes d'optimisation combinatoire*. Ceux-ci sont caractérisés par des variables de décision prenant des valeurs discrètes et par un espace de recherche fini.

Pour des problèmes d'optimisation combinatoire de petite taille, des méthodes de résolution exacte peuvent être utilisées. On peut par exemple imaginer un algorithme qui parcourrait l'entièreté de l'espace de recherche afin de trouver la solution optimale. En d'autres termes, un tel algorithme envisagerait toutes les solutions possibles du problème et évaluerait, pour chacune d'entre elles, les valeurs des différents objectifs du problème. Si une telle approche est possible pour des problèmes de petite taille, elle devient tout simplement impraticable pour des problèmes où l'espace de recherche explose très rapidement.

Le problème du voyageur de commerce [40, 69, 70, 95] est exemple de problème NP-Hard très connu. Il consiste à trouver le chemin le plus court passant une et une seule fois par un nombre donné de villes sur un plan. Parcourir l'espace de recherche de manière exhaustive consiste donc à évaluer toutes les permutations possibles des villes données<sup>1</sup>. Si pour 6 villes, cela revient à considérer 720 possibilités. Pour 30 villes, la taille de l'espace de recherche est de  $2.6 \times 10^{32}$  qu'il est impossible d'énumérer en un temps raisonnable.

La théorie de complexité nous permet de classer les problèmes selon leur degré de complexité. L'objectif n'est pas ici d'aborder les bases de la théorie de la complexité mais plutôt d'en exploiter les résultats et de les mettre en lien avec les métaheuristiques. Le lecteur intéressé pourra se référer aux ouvrages suivants [98, 102] pour plus de détails sur les notions utilisées par la suite.

De nombreux problèmes d'optimisation sont NP-Hard et ne possèdent donc pas d'algorithme permettant de trouver une solution en un temps polynomial<sup>2</sup>. Citons par exemple : le voyageur de commerce, la coloration de graphe [56, 64] ou le partitionnement de graphe [24], etc. Il est évident que pour ce type de problème la résolution exacte ne peut que rarement être appliquée et les techniques d'approximation sont alors utilisées.

Tout au long de cette section, au-delà de la classification des différents problèmes d'optimisation, nous avons également mentionné certaines techniques de résolution qui sont applicables à certaines classes de problèmes. La Figure 3.1 permet de situer les différentes méthodes citées plus haut, les unes par rapport aux autres.

### 3.3 Métaheuristiques

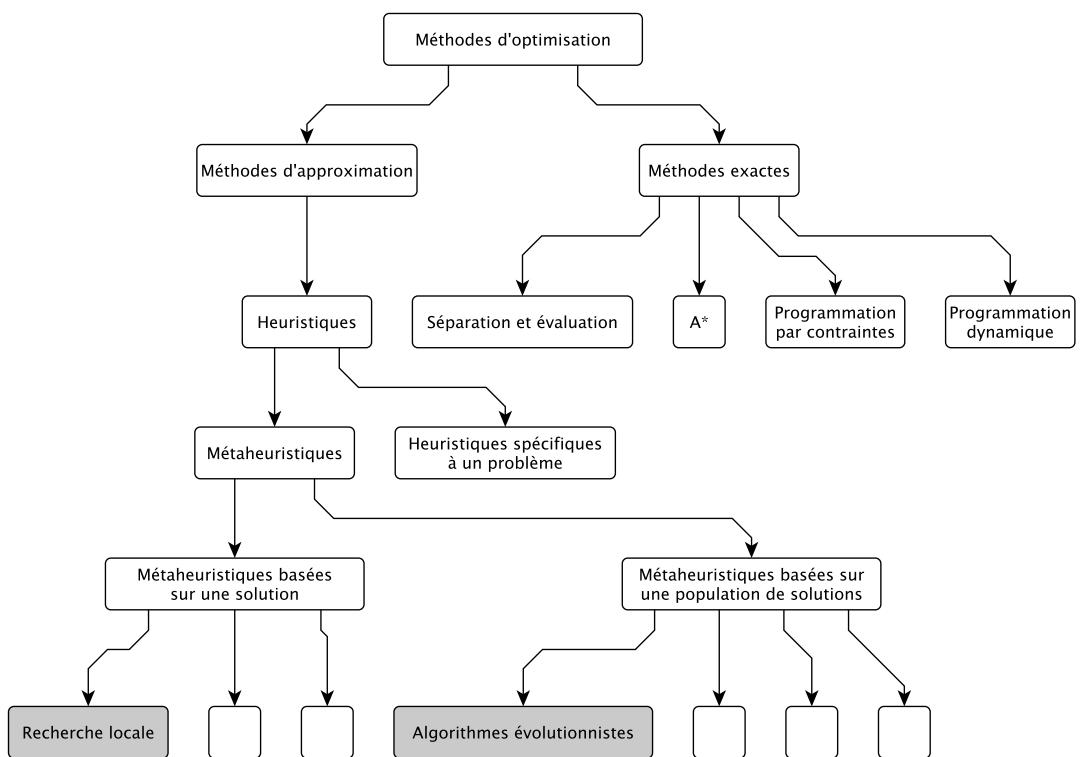
Comme nous venons de le voir dans la section précédente, les méthodes permettant de trouver la frontière Pareto optimale exacte d'un problème ne sont pas toujours applicables. Nous aurons alors recours à des techniques d'approximation.

Parmi les techniques d'approximation, nous retrouvons entre autre les heuristiques. Elles se divisent en deux catégories. D'un côté, on trouve les techniques spécifiques à un problème particulier (par exemple le problème du voyageur de commerce) et de l'autre, les techniques générales mises au point pour résoudre un grand nombre de problèmes d'optimisation de types différents. Les métaheuristiques font partie de cette seconde catégorie et nous intéressent en particulier. Elles sont conçues pour aborder un grand

---

1. qui est donné par  $n!$

2. pour autant que  $P \neq NP$



**Figure 3.1** – Ce diagramme, inspiré d'un diagramme proposé par Talbi [110], permet de mieux visualiser les différentes techniques de résolution existantes pour les problèmes d'optimisation. Ce diagramme ne se veut pas exhaustif.

nombre de problèmes différents, sans nécessité d'adaptation trop importante par rapport à leur manière de fonctionner. Cette propriété des métaheuristiques en font des outils très populaires et énormément étudiés.

Considérons à nouveau les problèmes d'optimisation multiobjectif tels qu'ils ont été définis dans le chapitre précédent. Nous continuerons d'appeler *Résolution de ces problèmes*, la recherche de la frontière Pareto optimale.

Les métaheuristiques restent, dans ce cas-ci, des techniques d'approximation très utilisées car en plus de l'éventuelle nature NP-Hard d'un problème, l'aspect multiobjectif peut faire exploser l'espace de recherche. En effet, comme nous l'avons vu au chapitre précédent, chaque ajout d'un objectif ne peut qu'augmenter la taille de la frontière Pareto optimale.

Quand nous parlons de métaheuristiques en tant que techniques d'approximation dans le cadre de l'optimisation multiobjectif, cela signifie que l'algorithme ne garantit pas de trouver la frontière Pareto optimale exacte du problème mais une bonne approximation de celle-ci.

Les métaheuristiques sont des méthodes itératives qui font évoluer une ou plusieurs solution(s) vers la ou les solution(s) optimale(s) recherchée(s). Il existe donc deux grandes familles de métaheuristiques ; les métaheuristiques basées sur une seule solution qui font évoluer une solution en la faisant passer à une solution voisine, et les métaheuristiques basées sur une population de solutions. Ces dernières contiennent à tout instant un ensemble de solutions et font évoluer cette population vers les solutions optimales du problème.

Lors de l'utilisation de métaheuristiques pour résoudre des problèmes d'optimisation multiobjectif, il y a trois questions principales qui doivent être adressées [86, 125] :

**L'assignation de fitness** Cette procédure a pour objectif d'assigner à chaque solution une valeur scalaire, appelée fitness, qui aidera la métaheuristique à faire évoluer l'ensemble de solutions trouvées vers l'ensemble Pareto optimal.

**La préservation de la diversité** Les procédures pour maintenir une diversité au sein des solutions trouvées sont importantes. Elles augmentent les chances de trouver de nouvelles solutions et que l'ensemble final soit le plus varié possible.

**L'élitisme** L'élitisme permet de garder les meilleures solutions trouvées tout au long des itérations de l'algorithme. Il permet d'éviter de perdre les bonnes solutions qui ont été trouvées.

Les différentes techniques existantes pour réaliser les trois points cités ci-dessus sont abordées plus en détail dans les trois sous-sections qui suivent.

### 3.3.1 Assignation de fitness

Nous allons, dans un premier temps, nous intéresser aux techniques permettant d'attribuer à chaque solution une valeur scalaire reflétant sa qualité.

Pour un problème mono-objectif, la fonction d'assignation de fitness est souvent identique à la fonction d'objectif du problème considéré. Par exemple, si l'objectif d'un problème d'optimisation est de maximiser les quantités produites d'une marchandise, la

valeur de fitness d'une solution particulière peut être la quantité produite en utilisant cette solution.

Pour les problèmes multiobjectif cette approche n'est évidemment plus possible. Différentes méthodes sont alors utilisées pour mesurer la qualité d'une solution. Cette valeur scalaire assignée à chaque solution est alors utilisée par la métaheuristique pour guider la recherche vers l'ensemble Pareto-optimal du problème.

On retrouve trois catégories principales de stratégies d'assignation [86, 125] :

**Basé sur l'agrégation** Également appelée approche scalaire, cette technique consiste à combiner les différents objectifs en un seul, transformant alors le problème multiobjectif en un problème mono-objectif. Une méthode très simple consiste à faire une somme pondérée des différents objectifs du problème.

$$\begin{aligned} \text{fitness}(x) &= \sum_{i=1}^p \lambda_i f_i(x) \\ \text{avec } \lambda_i &\in [0...1] \text{ et } \sum_{i=1}^p \lambda_i = 1 \end{aligned} \quad (3.1)$$

Différentes variantes de cette technique existent. Une de celles-ci consiste à fixer les différents  $\lambda_i$  a priori. Si l'on prend l'exemple du Flow-Shop que nous avons introduit au chapitre précédent, on pourrait imaginer une fonction de fitness définie par  $\text{fitness}(x) = \text{makespan}(x) + \text{idletime}(x)$ , qu'on cherchera à minimiser. Il est évident qu'une très bonne connaissance du problème est alors nécessaire. Une autre variante appelée pondération dynamique fait varier les différents  $\lambda_i$  pendant la recherche.

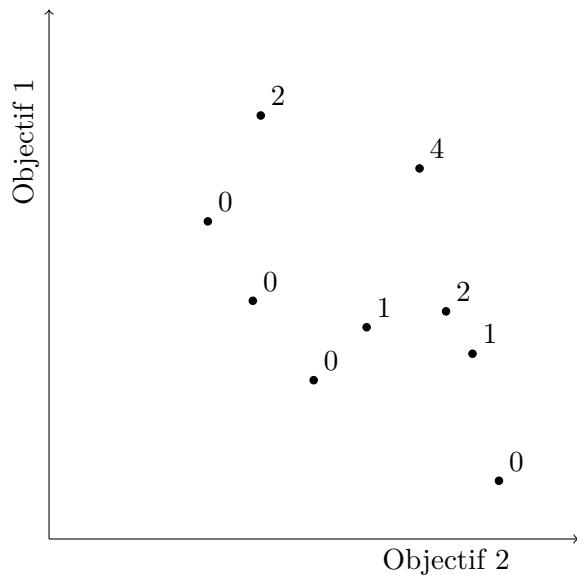
Cette technique a toutefois des limites puisqu'elle ne permet de trouver que les solutions supportées du problème comme nous l'avons vu à la Figure 2.4.

**Basé sur les critères** Contrairement à l'approche précédente, celle-ci considère les différents objectifs séparément. La méthode appelée lexicographique nécessite qu'un ordre de préférence soit précisé pour les différents objectifs. Le problème multiobjectif devient alors un ensemble de problèmes mono-objectif résolus séparément.

L'approche basée sur les critères est également utilisée avec les métaheuristiques qui maintiennent une population de solutions. A chaque itération de la métaheuristique, la population est divisée en autant de sous-ensembles qu'il y a d'objectifs. Les solutions sont alors sélectionnées dans chaque sous-population selon un seul des objectifs.

**Basé sur la dominance** L'approche basée sur la dominance est certainement la plus utilisée. On y a recours pour les métaheuristiques travaillant sur un ensemble de solutions et consiste à assigner une valeur scalaire à chaque solution en se basant sur la notion de dominance Pareto telle que nous l'avons définie dans la section 2.4.3. Cette approche a également comme avantage de comparer entre elles les solutions trouvées au lieu de calculer une valeur absolue ne dépendant que de la valeur des objectifs. La grande majorité des algorithmes évolutionnistes utilise ce type d'approche. Le reste de cette section y sera donc dédié.

**Basé sur des indicateurs** Cette méthode consiste à utiliser des indicateurs de performance afin de guider la recherche. Nous verrons différents indicateurs de performance par la suite. Le lecteur intéressé peut se référer à l'article [126] pour plus de détails sur cette méthode.



**Figure 3.2** – Illustration de la stratégie de rang de dominance pour un problème bi-objectif.

Comme expliqué, les deux premières techniques nécessitent qu'un ordre de préférence soit défini sur les différents objectifs. Ce n'est pas le cas pour les deux dernières méthodes mentionnées.

La fin de cette section est consacrée aux différentes méthodes qui font partie de la dernière classe de stratégies citées plus haut : la classe de techniques assignant une valeur de fitness aux solutions en se basant sur la notion de dominance.

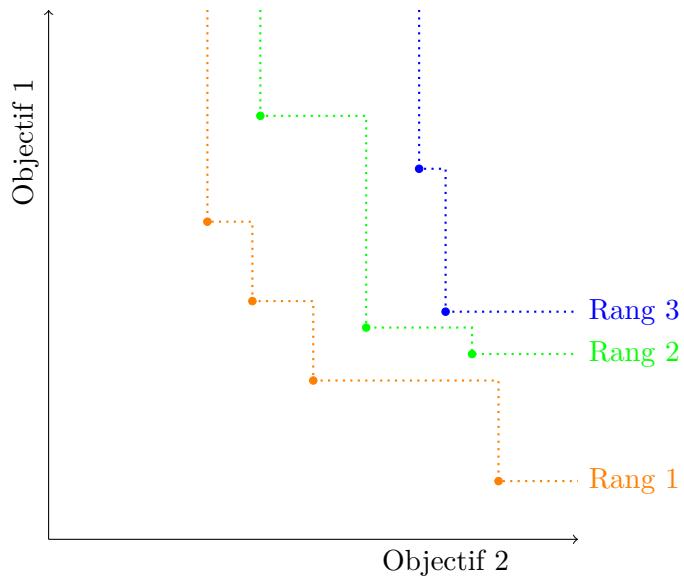
On retrouve dans cette catégorie trois méthodes principales détaillées ci-dessous :

**Rang de dominance** Cette stratégie associe à chaque solution un rang proportionnel au nombre de solutions qui la dominent. Elle est illustrée à la Figure 3.2. Une solution sera alors meilleure qu'une autre si elle a un rang plus petit.

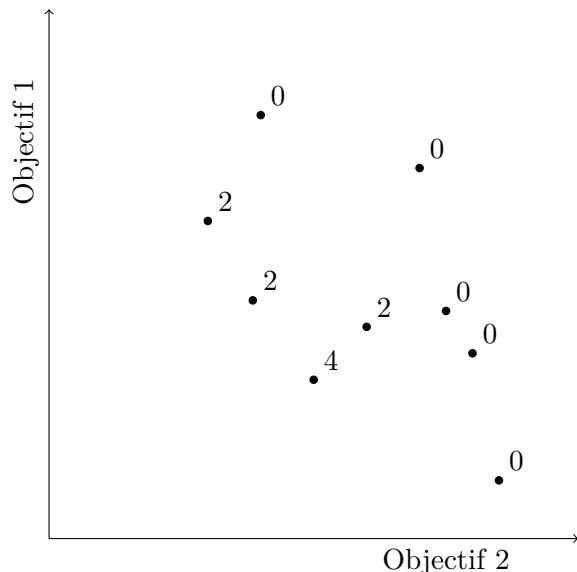
**Profondeur de dominance** L'ensemble des solutions est divisé en plusieurs fronts. Le premier front est composé des solutions non-dominées de la population. Le second front est l'ensemble des solutions non-dominées de la population à laquelle on a retiré le premier front. Le rang d'une solution particulière est alors le front dans laquelle elle est située. Cette stratégie est illustrée à la Figure 3.3. Plus petit sera le rang d'une solution, meilleure sera celle-ci.

Cette approche est utilisée dans l'algorithme évolutionniste NSGA-II[38], que nous verrons dans la Section 3.4.1, et peut être calculé comme illustré dans l'algorithme 1

**Compte de dominance** Cette stratégie associe à chaque solution un nombre proportionnel au nombre de solution qu'elle domine. Elle est illustrée à la Figure 3.4. A l'inverse du rang de dominance, une solution sera meilleure qu'une autre si elle a un rang plus élevé.



**Figure 3.3** – Illustration de la stratégie de profondeur de dominance pour un problème bi-objectif.



**Figure 3.4** – Illustration de la stratégie de compte de dominance pour un problème bi-objectif.

---

**Algorithme 1:** Procédure de *fast non dominated sort* utilisée dans l'algorithme NSGA-II[38].

---

```

input : population
output : fronts
for  $p \in$  population do
    // Nombre de solutions qui dominent  $p$ 
     $n_p \leftarrow 0$ 
    // Ensemble des solutions dominées par  $p$ 
     $S_p \leftarrow \emptyset$ 
end
for  $p \in$  population do
    for  $p \in$  population do
        if  $p$  domine  $q$  then
            |  $S_p \leftarrow S_p \cup \{q\}$ 
        end
        else if  $q$  domine  $p$  then
            |  $n_p \leftarrow n_p + 1$ 
        end
    end
    if  $n_p = 0$  then
        |  $\text{fronts}_i \leftarrow \text{fronts}_i \cup \{p\}$ 
    end
end
 $i \leftarrow 1$ 
while  $\text{fronts}_i \neq \emptyset$  do
     $H \leftarrow \emptyset$ 
    for  $p \in \text{fronts}_i$  do
        for  $q \in S_p$  do
            |  $n_q \leftarrow n_q - 1$ 
            if  $n_q = 0$  then
                | |  $H \leftarrow H \cup \{q\}$ 
            end
        end
    end
     $i \leftarrow i + 1$ 
     $\text{fronts}_i \leftarrow H$ 
end

```

---

### 3.3.2 Préservation de la diversité

L'assignation d'une valeur de fitness à chaque solution permet de guider la métaheuristique vers la frontière Pareto-optimale mais ne garantit malheureusement pas que les solutions trouvées soient suffisamment diversifiées.

Les métaheuristiques basées sur une population de solutions sont très sensibles au choix initial de la population et sont sujettes à une détérioration de la diversité si aucune précaution n'est prise par rapport au problème.

C'est pourquoi la majorité des métaheuristiques inclut un facteur tenant compte de la diversité dans le processus d'assignation de fitness. Il existe bien sûr plusieurs techniques pour mesurer cette notion de diversité.

**Méthodes noyaux** L'estimation de la densité est la somme d'une fonction  $K$ , appelée noyau, mappée sur la distance entre une solution et les autres solutions de la population. La distance entre deux solutions est la distance entre leur image dans l'espace d'objectif. On peut par exemple utiliser la distance euclidienne ou la distance de Manhattan. La Figure 3.5 illustre cette méthode avec la distance euclidienne.

La technique appelée *Partage de fitness* est souvent utilisée et fait partie des méthodes noyaux. Celle-ci est par exemple utilisée dans l'algorithme génétique NSGA, précurseur de NSGA-II. Cette technique consiste à dégrader la valeur de fitness  $f(x)$  associée à chaque solution en tenant compte du nombre de solutions se trouvant à proximité dans l'espace d'objectif. La nouvelle valeur de fitness devient alors :

$$f'(x) = \frac{f(x)}{m(x)} \quad (3.2)$$

où  $m(x)$  est parfois appelé compteur de niche. Il se calcule de la manière suivante :

$$m(x) = \sum_{y \in Population} sh(d(x, y)) \quad (3.3)$$

$$sh(d(x, y)) = \begin{cases} 1 - \frac{d(x, y)}{\sigma} & \text{si } d(x, y) \leq \sigma \\ 0 & \text{sinon} \end{cases} \quad (3.4)$$

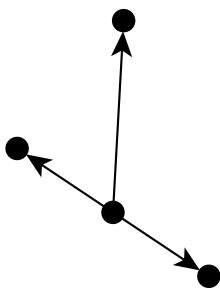
Le paramètre  $\sigma$  est un seuil au-delà duquel deux solutions sont considérées comme suffisamment distantes et ne faisant pas partie d'une même niche.

**Méthodes du plus proche voisin** L'estimation de la densité est donnée par la distance entre une solution et ses  $k$  solutions voisines les plus proches ou par le volume de l'hypersphère définie par ses  $k$  solutions voisines les plus proches. Cette technique est illustrée à la Figure 3.6. L'algorithme NSGA-II que nous verrons en Section 3.4.1 utilise une telle méthode.

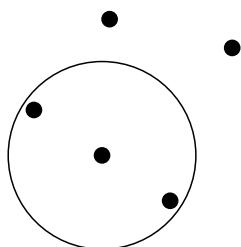
**Histogramme** L'espace de recherche est divisé en hypercubes de même tailles. L'estimation de la densité est alors le nombre de solutions se trouvant dans le même hypercube. Cette technique est illustrée à la Figure 3.7.

### 3.3.3 Élitisme

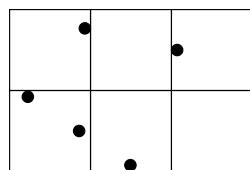
L'élitisme est un mécanisme mis en place dans certaines métaheuristiques afin de ne pas perdre les bonnes solutions. En effet, lors des itérations successives d'une métaheuristique,



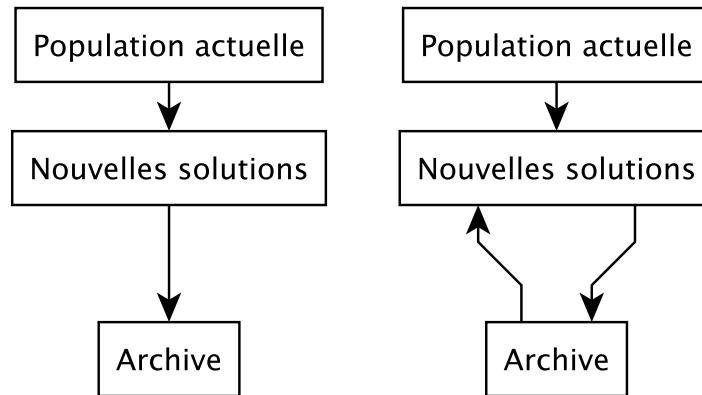
**Figure 3.5** – Ce diagramme illustre les méthodes noyaux de mesure de la diversité.



**Figure 3.6** – Ce diagramme illustre les méthodes du plus proche voisin de mesure de la diversité.



**Figure 3.7** – Ce diagramme illustre la méthode histogramme de mesure de la diversité.



**Figure 3.8** – Ce diagramme, illustre la différence entre l’élitisme passif (à gauche) et l’élitisme actif (à droite).

de bonnes solutions trouvées jusqu’à un certain point peuvent être perdues lors des itérations suivantes. L’élitisme permet d’éviter cette perte de solutions.

L’utilisation la plus simple de l’élitisme consiste à garder, dans un ensemble appelé l’archive, les meilleures solutions trouvées. Ce type d’élitisme est dit passif et consiste donc simplement en une mémoire de meilleures solutions.

Un autre type d’élitisme, dit actif, utilise cette archive de meilleures solutions lors des itérations successives de la météahéuristiche dans l’espérance d’augmenter la vitesse de convergence de la météahéuristiche. La différence entre l’élitisme passif et actif est illustrée à la Figure 3.8

### 3.3.4 Évaluation des performances

Lors du design d’une météahéuristiche il est important de pouvoir s’assurer de son bon fonctionnement et de pouvoir quantifier ses performances.

Les performances d’une météahéuristiche peuvent être analysées sous trois angles différents[110] : la qualité des solutions, l’effort de calcul et la robustesse. Nous allons principalement nous attarder sur le premier.

L’effort de calcul est souvent difficile à quantifier car il dépend de la machine sur laquelle s’exécute l’algorithme. Il peut aussi varier d’une itération à l’autre de l’algorithme. La complexité asymptotique est souvent utilisée ainsi que la complexité moyenne pour avoir une idée de l’effort de calcul d’une météahéuristiche.

La robustesse d’une météahéuristiche est une mesure de la variation de la qualité des solutions en fonction de la variation de paramètres ou du type d’instance fournie. Un météahéuristiche ayant une faible variation de qualité pour différents types d’instance ou pour des changements de paramètres sera dite robuste. La robustesse peut également être la mesure de variations sur différentes courses de la météahéuristiche sur une même instance.

Comme dit plus haut, le reste de cette section est dédié à la mesure de la qualité

des solutions. De manière générale, on peut mesurer la qualité du front trouvé par une météaheristique à la fin de son exécution en mesurant sa distance par rapport au front Pareto optimal exact. Celui-ci n'est cependant pas toujours disponible et une série d'indicateurs de performance ont été mis au point pour mesurer la qualité des solutions trouvées par une météaheristique.

Pour rappel, nous avions mentionné en section 3.3.1 que ces indicateurs étaient utilisés dans certaines météaheristiques pour l'assiguation de fitness.

Nous allons brièvement décrire certains de ceux-ci dans le reste de cette section en se basant sur l'ouvrage [110]. L'objectif est de donner une idée de leur fonctionnement. Le lecteur intéressé est invité à se référer aux ouvrages suivants [127, 110] pour plus de détails sur les différents indicateurs et leur fonctionnement.

Les indicateurs de performance peuvent être rangés en trois catégories. La première catégorie permet de quantifier la convergence des solutions trouvées vers le front Pareto optimal. La seconde catégorie permet de mesurer la diversité au sein des solutions trouvées. Enfin, la troisième catégorie, appelée hybride, regroupe la mesure de convergence et de diversité.

### Indicateurs de convergences

**Contribution** La contribution[81] est un opérateur binaire qui permet de comparer deux fronts. Soit deux fronts,  $P_1$  et  $P_2$ . La contribution de  $P_1$  par rapport à  $P_2$  est la proportion de solutions non dominées dans le front  $P_u = P_1 \cup P_2$  qui proviennent de  $P_1$ .

**Distance générationnelle** La distance générationnelle[61, 110] calcule la distance moyenne entre un front  $A$  donné et un front de référence  $R$ . Le front exact est souvent utilisé comme front de référence lorsque celui-ci est disponible. Cette distance générationnelle est calculée comme suit :

$$I_{GD}(A, R) = \frac{(\sum_{u \in A} (\min_{v \in R} d(u, v)))^{\frac{1}{2}}}{|R|} \quad (3.5)$$

où  $d(u, v)$  est la distance euclidienne entre la solution  $u$  et  $v$  dans l'espace d'objectif.

**$\epsilon$ -Indicateur** Cet indicateur est également basé sur la distance. Il se mesure donc par rapport à un front de référence  $R$ . L' $\epsilon$ -Indicateur correspond au facteur minimum par lequel un front  $A$  doit être translaté pour faiblement dominer un front de référence  $R$ . Il est calculé comme suit :

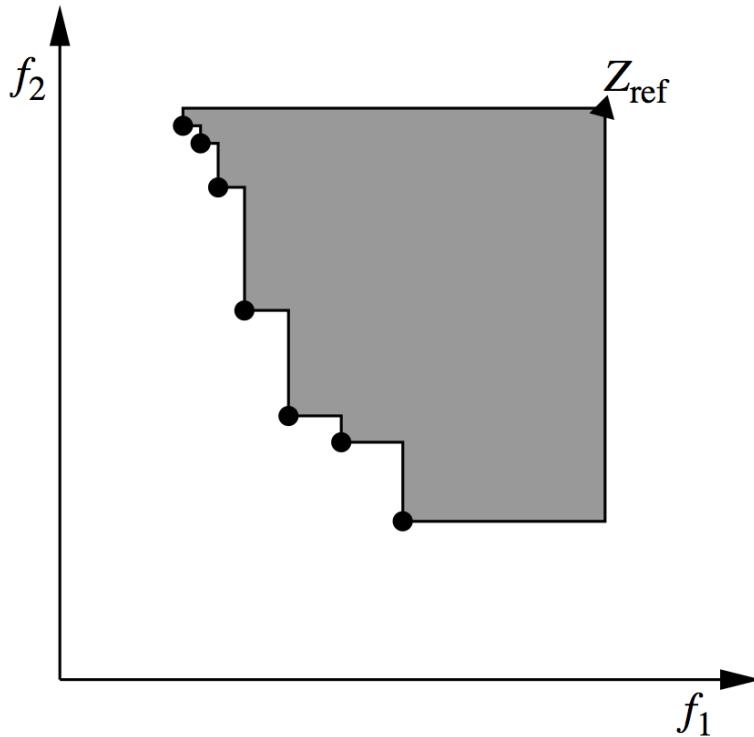
$$I_\epsilon(A, R) = \min_{\epsilon \in \mathbb{R}} \{ \forall v \in R \ \exists u \in A : u_i - \epsilon \leq v_i \quad \forall i \in \{1, \dots, p\} \} \quad (3.6)$$

### Indicateurs de diversités

**Propagation** La propagation[110] mesure la distribution mais également la cardinalité d'un front approché. Il se calcule de la manière suivante :

$$I_S = \frac{\sum_{u \in A} |\{v \in A : d(u, v) > \sigma\}|}{|A| - 1} \quad (3.7)$$

Plus la valeur de  $I_S$  sera proche de 1 meilleur sera la dispersion des solutions.



**Figure 3.9** – Diagramme repris de E. G. Talbi [110] illustrant la notion d’hypervolume, pour deux objectifs dans ce cas ci. L’hypervolume est donc une aire.

**Entropie** Cet indicateur est une généralisation de la notion d’entropie présente en théorie de l’information. L’espace d’objectif est divisé en hypergrilles de même taille. L’entropie[81] mesure alors la dispersion des solutions au sein de ces hypergrilles. L’entropie maximale sera de 1 si il n’y a pas plus d’une solution par hypergrille.

### Indicateurs hybrides

**Hypervolume** L’hypervolume[116, 74] est un indicateur hybride qui combine la mesure de convergence et de diversité. Celui-ci nécessite un point de référence dans l’espace d’objectif. Par exemple le point Nadir. La Figure 3.9 permet de visualiser à quoi il correspond.

C’est un indicateur très apprécié dans la littérature car il comporte une propriété importante, il est strictement conforme au sens de Pareto. Ceci signifie que si un front  $A$  domine un front  $B$ , alors l’hypervolume de  $A$  sera strictement supérieur à l’hypervolume de  $B$ .

Ces différents indicateurs peuvent être utilisé afin de comparer les performances de plusieurs algorithmes ou pour comparer plusieurs résultats obtenus. Il convient toutefois de les utiliser correctement[81] en tenant compte des préférences du preneur de décision. De plus, certains d’entre eux nécessitent de choisir des points ou fronts de références. Il convient donc de sélectionner ceux-ci de manière adéquate.

La section suivante 3.4 est dédiée aux algorithmes évolutionnistes qui font partie de la famille des métaheuristiques basées sur une population de solutions. Nous reviendrons ensuite brièvement sur la recherche locale qui appartient à l'autre famille de métaheuristique c'est-à-dire basée sur une solution.

Les algorithmes évolutionnistes, que nous allons aborder dans la section suivante font partie de la classe des métaheuristiques basées sur une population de solutions.

### 3.4 Algorithmes évolutionnistes

Cette section est organisée de la manière suivante ; après avoir introduit quelques notions de bases sur les métaheuristiques basées sur une population, nous aborderons les algorithmes évolutionnistes et plus particulièrement les algorithmes dits génétiques. Nous terminerons cette section par l'étude d'un des grands standards dans le domaine, l'algorithme NSGA-II[38].

Les métaheuristiques basées sur une population de solutions partagent certaines caractéristiques. Leur façon de fonctionner peut être schématisée comme suit : une première population de solution est d'abord générée, on l'appelle population initiale. Le processus itératif prend alors place : une nouvelle population est générée à partir de la population actuelle. Cette dernière est alors fusionnée à la population courante en utilisant un mécanisme de sélection propre à chaque métaheuristique. Ce processus itératif s'arrête alors lorsqu'une condition spécifique est rencontrée.

Le pseudocode ci-dessous illustre le mécanisme général d'une métaheuristique basée sur une population de solutions :

---

**Algorithme 2:** Pseudo-code d'une métaheuristique basée sur une population de solutions.

---

```

 $P_0 \leftarrow \text{InitialPopulation}$ 
 $i \leftarrow 0$ 
while  $\neg \text{StopCondition}$  do
     $N_i \leftarrow \text{Generation}(P_i)$ 
     $P_{i+1} \leftarrow \text{Selection}(P_i \cup N_i)$ 
     $i \leftarrow i + 1$ 
end

```

---

Les métaheuristiques basées sur une population de solutions sont très bien adaptées pour résoudre des MOPs. En effet, puisqu'elles traitent un ensemble de solutions simultanément, plusieurs membres de la frontière Pareto-optimale peuvent être trouvés en une seule itération [110, 38].

En particulier, les algorithmes évolutionnistes (EA pour Evolutionary Algorithm) sont très souvent utilisés pour résoudre des problèmes d'optimisation multiobjectif [115]. Leur capacité à s'adapter à des problèmes très complexes les rendent bien plus performants que des techniques traditionnelles [48] pour plusieurs raisons. Tout d'abord, comme mentionné plus haut, à chaque itération de l'algorithme, plusieurs individus peuvent chercher plusieurs nouvelles solutions en tirant avantage des similarités disponibles dans

le reste de la population. De plus, les algorithmes évolutionnistes s'adaptent bien aux discontinuités de l'espace de décision réalisable et sont peu sensible à la forme de la frontière Pareto. Enfin, les algorithmes évolutionnistes restent applicables même pour des problèmes faisant intervenir des fonctions qui ne sont même pas analytiques comme par exemple une température.

Les algorithmes évolutionnistes s'inspirent de la nature et plus particulièrement de la théorie de l'évolution des espèces pour résoudre des problèmes d'optimisation. Nous verrons en particulier dans le reste de cette section les algorithmes génétiques qui sont une classe plus spécifique d'algorithmes évolutionnistes.

Les algorithmes génétiques utilisent des procédés de sélection et de génération bien particuliers appelés dans la littérature opérateurs génétiques. Ceux-ci sont combinés pour implémenter les procédures de sélection et de générations mentionnés précédemment.

Dans le jargon des algorithmes génétiques, les solutions sont appelées chromosomes. De manière générale, un algorithme génétique fonctionne comme suit à chaque itération : une procédure de sélection est appliquée sur la population actuelle afin d'en choisir un sous-ensemble qui est appelé chromosomes *parents*. Ces parents sont alors utilisés pour générer de nouveaux chromosomes au moyen de deux opérateurs génétiques appelés *crossover* et *mutation*. Nous discuterons de ceux-ci plus bas. Les chromosomes fils sont ensuite fusionnés avec la population pour former la nouvelle population. Enfin, une dernière procédure réduit la taille de la population en gardant uniquement les meilleures solutions qui la composent. Cette procédure générique est illustrée dans l'algorithme 3.

---

**Algorithme 3:** Pseudo-code d'un algorithme génétique.

---

```

population ← InitializePopulation(populationSize)
Parents ← SelectParents(population)
children ← GeneticOperators(Parents)
population ← population ∪ children
while ¬StopCondition do
    population ← KeepBests(population, populationSize)
    Parents ← SelectParents(population)
    children ← GeneticOperators(Parents)
    population ← population ∪ children
end
return NonDominated(population)

```

---

Nous allons maintenant brièvement décrire les opérateurs génétiques cités plus haut, leur utilité et ensuite passer à un exemple concret d'algorithme génétique.

**Opérateur de crossover** Cette procédure est utilisée pour générer un nouveau chromosome à partir de deux<sup>3</sup> chromosomes parents. Cette procédure est appliquée dans l'espoir que le chromosome résultant combine les bons aspects de chacun des deux parents pour former un meilleur chromosome.

---

3. ou de manière générale  $n$

**Opérateur de mutation** La procédure de mutation est un opérateur binaire qui est appliquée sur un chromosome. Elle consiste à induire de légères variations aléatoires au sein de celui-ci, exactement comme les mutations apparaissent dans la nature.

Ces deux opérateurs génétiques s'inscrivent en fait, dans le cadre plus général des métahéuristiques, dans deux classes de méthode ayant un but bien précis.

Les deux buts poursuivis sont l'intensification et la diversification parfois appelés respectivement exploitation et exploration.

Au début de ce chapitre (3.2) nous avions introduit le problème du voyageur de commerce, problème NP-Hard. Nous avions insister sur le fait qu'il était impensable d'énumérer l'ensemble des solutions possibles du problème en un temps raisonnable, celles-ci étant beaucoup trop nombreuses. Pour parcourir 50 villes,  $3 \times 10^{64}$  solutions existent. Dans le cadre d'une métahéuristique, il convient donc de générer de nouvelles solutions de manière intelligente.

Le processus d'intensification consiste à utiliser de bonnes solutions trouvées jusque là afin d'en générer de nouvelles. Par exemple, si un grand nombre de bonnes solutions trouvées jusqu'ici visite la ville  $j$  juste après avoir visité la ville  $i$ , il paraît cohérent de garder ce tronçon lors de la génération de nouvelles solutions. Au même titre, si le tronçon allant de la ville  $k$  à la ville  $m$  n'apparaît dans aucune des meilleures solutions, il semble inutile de générer de nouvelles solutions le contenant.

Toutefois, en utilisant uniquement le processus d'intensification, l'algorithme converge vers un optimum local. Si aucune des bonnes solutions trouvées jusque là ne contient le tronçon  $k, m$ , cela ne signifie pas pour autant qu'il n'existe aucune meilleure solution le contenant. La meilleure solution du problème le contient même peut-être. C'est pour palier à ce problème que le processus de diversification est utilisé. Celui-ci consiste à ajouter une composante aléatoire lors de la génération de nouvelles solutions afin d'éviter de rester bloqué dans un optimum local.

Trop d'intensification risque de bloquer l'algorithme dans un optimum local du problème. Au contraire, trop de diversification risque de rendre l'algorithme trop lent à converger. Dans le design d'une métahéuristique, il convient donc de trouver une bonne balance entre intensification et diversification.

Nous allons maintenant décrire plus en détail un des grands standards dans le domaine des algorithmes génétiques, *NSGA-II*.

### 3.4.1 NSGA-II

NSGA-II est une version améliorée de l'algorithme NSGA (Non-dominated Sorting Genetic Algorithm) initialement proposé par Deb et Srinivas en 1994 [103] auquel on reprochait : une complexité trop importante à la procédure de tri des chromosomes, un manque d'élitisme et la nécessité de préciser un paramètre pour une procédure de la métahéuristique [38].

Nous allons d'abord présenter le pseudocode de NSGA-II et ensuite passer en revue la façon dont sont implémentées les procédures caractéristiques de cette métahéuristique, à savoir l'assignation de fitness, la préservation de la diversité et l'élitisme.

---

**Algorithme 4:** Pseudo-code de l'algorithme NSGA-II.

---

```

population ← InitializePopulation(populationSize)
fronts ← FastNonDominatedSort(population)
for  $f_i \in$  fronts do
| ComputeCrowdingDistance ( $f_i$ )
end
selected ← SelectParentsByRank(population, populationSize)
children ← CrossoverAndMutation(selected)
population ← population  $\cup$  children
while  $\neg$ StopCondition do
| fronts ← FastNonDominatedSort(population)
| for  $f_i \in$  fronts do
| | ComputeCrowdingDistance ( $f_i$ )
| end
| population ← KeepBests(population, populationSize)
| selected ← SelectParentsByRank(population, populationSize)
| children ← CrossoverAndMutation(selected)
| population ← population  $\cup$  children
end
return NonDominated(population)

```

---

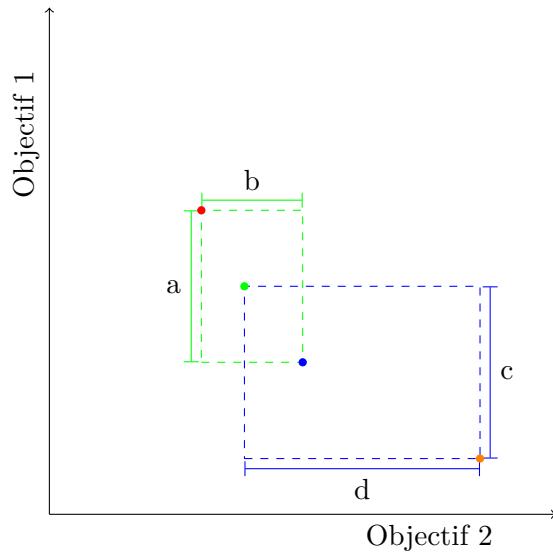
**Pseudocode de NSGA-II**

**L'assignation de fitness** La procédure `FastNonDominatedSort` trie l'ensemble des chromosomes en une hiérarchie de fronts de solutions non-dominées tel quel nous l'avons défini dans la Section 3.3.1 sous le nom de *profondeur de dominance*. Chaque chromosome de la population se voit assigné un rang qui correspond à l'index du front dont il fait partie.

**Préservation de la diversité** La procédure `computeCrowdingDistance` calcule une mesure de la diversité pour chaque chromosome. Pour chaque chromosome, la distance par rapport à ses voisins les plus proches partageant la même valeur de rang est calculée comme illustré à la Figure 3.10. Cette procédure fait partie de la catégorie que nous avions nommée *Méthodes du plus proche voisin*. Les chromosomes ayant une valeur élevée de *crowding distance* sont bien sur les meilleurs.

**Nouveaux chromosomes** Un groupe de parents est construit à partir de la population en utilisant un tournoi binaire : deux chromosomes sont choisis aléatoirement dans la population et celui avec le plus haut rang est gardé. En cas d'égalité, la mesure de *crowding distance* est alors utilisée. De nouveaux chromosomes sont alors générés en appliquant la procédure de *crossover* sur deux chromosomes choisis aléatoirement dans l'ensemble de parents fraîchement créé. La procédure de *mutation* est également appliquée à ces chromosomes résultants de la procédure de *crossover*.

**Elitisme** Bien que NSGA-II n'utilise pas d'archive à proprement parlé, l'algorithme est toutefois bien élitiste. En effet, lorsque la nouvelle population est créée, elle est ajoutée à la population actuelle et seuls les meilleurs chromosomes se trouvant dans cette union sont gardés. Ceci signifie que les meilleurs chromosomes sont nécessairement gardés.



**Figure 3.10** – La procédure de *Crowding Distance* présente dans NSGA-II. La valeur de crowding distance pour le point vert est  $a+b$ . Pour le point bleu, elle est de  $c+d$ . Quant aux points orange et rouge, ils ont une valeur de *crowding distance* de  $\infty$ .

### 3.5 Recherche locale et mét heuristicques hybrides

La recherche locale fait partie de la famille des mét heuristicques basées sur une solution unique. Nous avons vu au travers des sections précédentes que les mét heuristicques basées sur une population de solutions sont généralement plus efficaces pour résoudre des MOPs. Toutefois, cette section est dédiée à la recherche locale car celle-ci peut être combinée à une mét heuristicque afin d'en améliorer l'efficacité. La mét heuristicque résultant de cette combinaison est alors appelée mét heuristicque hybride.

De nombreuses recherches montrent que l'ajout d'une procédure de recherche locale à un algorithme évolutif, juste après l'application des procédures de crossover et mutation augmente de manière significative les performances de l'algorithme [84, 80, 113, 79, 108, 28, 57]. Nous confirmerons d'ailleurs ce résultat dans un chapitre ultérieur à la Section 6.4.4.

Cette section ne vise pas à donner une étude approfondie de la recherche locale mais plutôt à donner une idée de son fonctionnement pour pouvoir l'aborder dans le cadre de l'hybridation.

La recherche locale est une mét heuristicque fort simple. L'algorithme débute avec une solution initiale. A chaque itération, des solutions voisines à la solution actuelle sont générées. La solution est alors remplacée par une solution voisine de meilleur qualité. Lorsque aucune solution voisine n'est meilleure que la solution courante, l'algorithme termine et un optimum local a été trouvé.

La notion de solutions voisines dépend bien évidemment du problème considéré. De manière générale, on peut considérer comme solutions voisines, les solutions dont les différentes variables de décision diffèrent très peu de celles de la solution considérée. L'objectif sous-jacent à se restreindre aux solutions voisines, est d'accélérer la recherche.

Pour illustrer cette notion de solution voisine, reprenons le problème du voyageur de commerce. Une solution à ce problème est un ordonnancement des différentes villes qu'il doit parcourir. Une solution voisine peut alors par exemple être ce même ordonnancement où deux villes ont été échangées. Par exemple une solution  $a$  est la suivante  $[1, 3, 4, 2, 7, 6, 5, 1]$ <sup>4</sup>. Une solution voisine peut alors être  $[1, 3, 2, 4, 7, 6, 5, 1]$  où la ville 2 et 4 ont échangé leur place.

Plusieurs méthodes existent pour remplacer la solution actuelle par une des meilleures solutions voisines, elles sont brièvement expliquées ci-dessus :

**Meilleur amélioration** Toutes les solutions voisines sont d'abord générées et la solution améliorant le plus la solution actuelle est alors sélectionnée.

**Première amélioration** La génération des solutions voisines s'arrête dès qu'une meilleure solution est trouvée. Celle-ci remplace alors la solution actuelle.

**Amélioration aléatoire** Après génération de l'ensemble des solutions voisines, une solution est choisie aléatoirement parmi les solutions qui sont meilleures que la solution actuelle.

Un problème inhérent à la recherche locale est, comme son nom l'indique, qu'elle s'arrête sur un optimum local et non global. Elle est donc très sensible à la solution initiale. Plusieurs méthodes peuvent alors être mises en place pour palier à ce problème comme par exemple lancer plusieurs recherches locales avec des solutions initiales différentes afin d'augmenter la diversification. Nous n'aborderons toutefois pas ces techniques dans ce mémoire car nous ne considérons la procédure de recherche locale que dans le cadre d'une hybridation de métahéuristiques basées sur une population de solutions. Dans ce cas, la métahéuristique hybride ne souffre donc pas de ce problème.

Nous verrons plus en détail un exemple d'hybridation dans la seconde partie de ce mémoire.

### 3.6 Conclusion

Dans ce chapitre, nous avons d'abord abordé les différentes classifications de problèmes d'optimisation, ce qui nous a permis par la suite de justifier l'utilisation des métahéuristiques. Nous avons ensuite introduit cette notion de métahéuristique comme technique d'approximation pour résoudre des problèmes d'optimisation multiobjectif. Nous en avons vu les différentes caractéristiques et principales composantes. Nous avons ensuite abordé les algorithmes évolutionnistes, en particulier les algorithmes génétiques que nous avons illustrés par NSGA-II. Nous avons enfin traité de la recherche locale tout en mentionnant son utilité dans le cadre d'une hybridation avec un algorithme évolutionniste.

Dans le chapitre suivant, nous aborderons les systèmes d'information géographique que nous raffinerons pour finalement traiter d'un problème d'optimisation particulier : le partitionnement de territoire.

---

4. Remarquer la double présence de la ville 1 puisque le voyageur doit faire un tour complet.

# Chapitre 4

## Problème de partitionnement

### 4.1 Introduction

Ce chapitre vise à introduire le problème d'optimisation qu'est le partitionnement de territoire. La première partie de ce chapitre vise à offrir un cadre à ce problème d'optimisation. Nous allons donc dans un premier temps aborder le sujet des systèmes d'informations géographiques avec divers exemples ainsi qu'une réflexion sur l'accumulation de données et la formation de *Big Data*.

Nous verrons ensuite plus en détail le problème de partitionnement sous différentes formes ; le problème de districting, le partitionnement de graphe et le problème de regroupement.

### 4.2 Système d'information géographique

La notion de Système d'Information Géographique, abrégé SIG, est une notion très générale. Comme le souligne K. C. Clarke [34], toute bonne science commence par des définitions claires. Cependant, dans le cadre des SIG, cet objectif est complexe tant la notion englobe d'éléments.

Dans le souci d'avoir une idée claire de ce que représente un SIG avant d'introduire des définitions plus formelles, nous nous baserons sur l'introduction de Longley et al. [72].

Au cœur d'un SIG se trouve une base de données géo-référencées. Une telle base de données se caractérise par le fait que des coordonnées sont associées à chaque entrée. Dans le cas des SIG, il s'agit de coordonnées sur la surface de la terre.

Par exemple, une société de taxi, maintenant dans une base de données la position de chaque véhicule est une base de données géo-référencées. Au même titre qu'un carnet d'adresses contenant l'adresse postale de chaque personne puisque chacune de ces adresses postales peut être convertie en une latitude et une longitude.

L'*Open Geospatial Consortium* [12] a d'ailleurs défini un ensemble de spécifications pour ces bases de données. De nombreux fournisseurs proposent donc des systèmes de base de

données implémentant ces spécifications. On peut par exemple citer : Oracle Spatial [89], IBM DB2 Spatial Extender [59] et SpatiaLite [16].

Au dessus de cette base de données géo-référencées s'ajoute un ensemble de logiciels permettant de manipuler, d'analyser et de visualiser ces données. Grâce à l'informatique, des tâches autrefois très lourdes pour des géographes sont maintenant exécutées en une fraction de seconde sur des ordinateurs. Parmi ces différentes tâches, on peut par exemple citer [72] : la génération de cartes, le calcul de distance, de route à suivre, le traçage d'actif, la détection de motifs, etc.

Les systèmes d'informations géographiques sont donc d'une certaine façon le mariage entre la géographie et l'informatique.

Un exemple maintenant très connu de système d'information géographique est Google Earth [53] que l'on peut voir en Figure 4.1. Ce programme permet différentes visualisations de la terre ainsi que l'affichage de très nombreuses informations la concernant. On retrouve bien sur des informations comme des images satellites ou bien l'affichage des frontières et nom des différents pays, régions et villes. Mais bien que moins connues, Google Earth contient énormément plus d'informations géo-référencées. Il est possible d'afficher des photos prises par divers acteurs, que ce soit des photos amateurs ou même des clichés de la NASA. Il est également possible de visualiser des bâtiments en 3D, d'afficher la météo, le trafic routier, et bien d'autres types d'informations. Comme nous l'avons expliqué plus haut, un SIG contient non seulement des données mais également des outils permettant de les visualiser, les traiter et les analyser. Google Earth entre donc bien dans cette première définition.

Google Earth est bien sur un logiciel grand public. Mais de nombreux autres outils à utilisation professionnelle existent sur le marché. On peut par exemple citer *SuperGIS Desktop* permettant de visualiser des phénomènes spatiaux complexes et de résoudre des problèmes spatiaux.

Parmi les logiciels les plus célèbres, on peut citer : GRASS GIS [8] (Figure 4.2), gvSIG [9], Quantum GIS [14], JUMP GIS [10] et ArcGIS [1], des SIG complets. La majorité d'entre eux sont sous licence *GNU General Public License*.

Les quelques exemples d'information disponible dans Google Earth citées plus haut donne une idée de l'étendue des domaines qui peuvent profiter des SIG ; l'épidémiologie, la criminologie, l'archéologie, écologie, la géologie, et bien d'autres.

Nous sommes maintenant en mesure de mieux aborder différentes définitions de SIG présentes dans la littérature :

“Un ensemble puissant d’outils pour stocker et retrouver à volonté, transformer et afficher des données spatiales du monde réel pour un ensemble de buts spécifiques”[30]

“Des systèmes automatisés pour la capture, le stockage, la récupération, l’analyse et l’affichage de données spatiales.”[33]

“Un système d’information conçu pour fonctionner avec des données référencée par des coordonnées spatiales ou géographiques. En d’autres mots, un SIG est à la fois un système de base de données avec des capacité spéciale pour les données spatiales, et un ensemble d’opérations pour travailler avec ces données.”[105]

“Une collection intégrée de logiciels et de données utilisées pour afficher et gérer des in-



Figure 4.1 – Illustration de Google Earth, logiciel développé par Google.

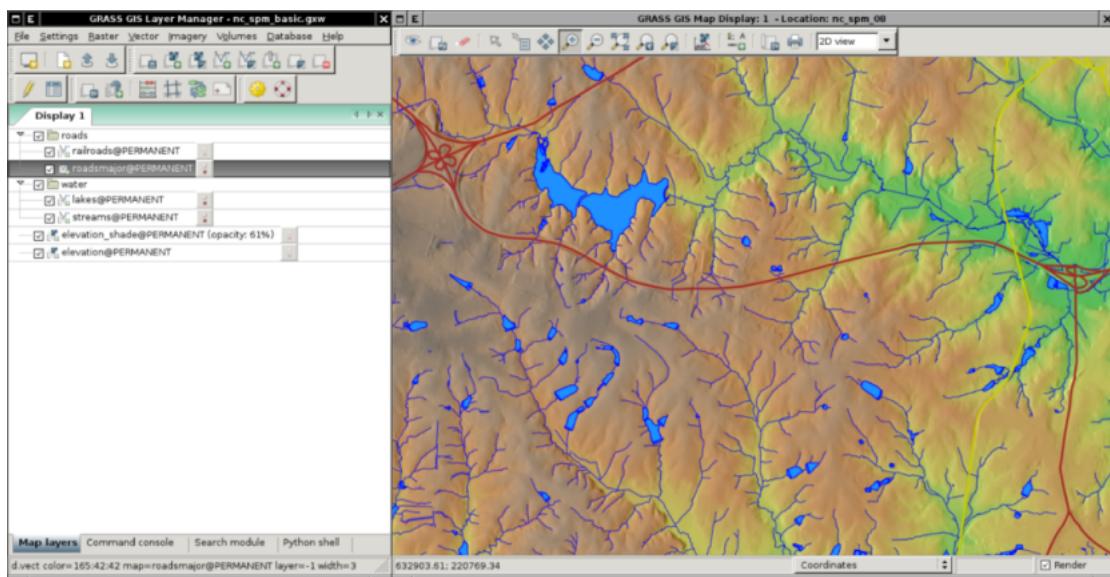


Figure 4.2 – Illustration de GRASS GIS et de son interface.

formations sur des lieux géographiques, d'analyser les relations spatiales, et de modéliser des processus spatiaux. Un SIG constitue un cadre pour la collecte et l'organisation des données spatiales et des informations connexes afin qu'elles puissent être visualisées et analysées.”[52]

Lorsque l'on s'intéresse à l'histoire des SIG [72], on constate naturellement que celle-ci prend naissance dans les années 1960 avec l'apparition des premiers ordinateurs offrant des capacités de traitement de l'information plus rapide mais surtout exacte. Des tâches qui étaient alors faites à la main jusque là sont automatisées. Si l'on reprend cette dernière phrase, on peut même remarquer que les idées de base des SIG sont apparues bien avant les ordinateurs ou divers systèmes étaient mis au point pour aider à collecter ou traiter des informations géo-référencées.

Lorsqu'on s'intéresse aux premières années des SIG [109, 72, 34], il est souvent mentionné dans la littérature les traitements formidables qu'il est possible de réaliser avec des ordinateurs une fois les données entrées dans le système. La situation est quelque peu différente aujourd'hui. Il est d'abord utile de rappeler que la collecte d'informations géo-référencées fait partie intégrante du SIG et qu'elle a un tout autre sens aujourd'hui que dans les années 1960. En effet, nous vivons actuellement dans un monde de plus en plus instrumentalisé. Les différents équipements qui nous entourent, participent à la collecte de données. Nos téléphones par exemple, aujourd'hui équipés de GPS, récupèrent régulièrement leur position et la transmettent à des systèmes comportant d'immenses bases de données et de nombreux outils d'analyse. Ces positions sont utilisées à de nombreuses fins, comme la détection et prédition du trafic routier, l'amélioration de la réception téléphonique en identifiant les zones moins couvertes par des antennes GSM.

Les appareils photo sont eux aussi de plus en plus souvent équipés de GPS et anote chaque photo prise par sa position géographique. Les possibilités de traitement et d'analyse possible une fois ces données accessibles paraissent parfois futuristes mais permettent, par exemple, de produire ce que l'on appelle la réalité augmentée où des informations supplémentaires sont ajoutées à une image.

Ce type de base de données existe bien. On peut par exemple citer Flickr [47]. Mr Blaise Agüera y Arcas [121] a démontré en 2012 lors d'une conférence TED [111] une technologie mise au point par Microsoft [82] permettant, à partir d'images géo-référencées disponibles sur Flickr de réaliser des compositions de photos permettant une immersion poussée dans de nombreux endroits de la planète. La Figure 4.3 donne un aperçu de cette technologie.

Ce qu'il est important de réaliser avec de telles technologies, c'est qu'elles utilisent des quantités colossales d'informations provenant de nombreuses sources différentes. A la différence de Google Street View où seules les images prisent par les caméras de Google sont présentes, il est possible, en réalisant du crowdsourcing de photos géo-référencées de fournir également des images à l'intérieur des bâtiments, ou même de pouvoir voyager dans le temps, tellement le nombre de photos disponibles est important.



**Figure 4.3** – Illustration de la technologie Read/Write World de Microsoft.

### 4.3 Big Data

Avant d'aborder dans la section suivante le problème de partitionnement de territoire, nous ferons une rapide introduction à la notion de *Big Data* tant celle-ci est importante dans notre société contemporaine.

Comme nous l'avons mentionné plus haut et comme le souligne le MIKE2 [83] : Les machines modernes comme les voitures, les trains, les stations électriques, les avions , les Smartphones ont toutes un nombre grandissant de capteurs qui amassent constamment de l'information. Il est d'ailleurs courant d'avoir des milliers voir des centaines de milliers de capteurs qui collectent de l'information à propos des performances et des activités d'une machine.

Comme le mentionne IBM [58] : dénormes quantités de données sont générées chaque jour. Pour donner une idée, 90% des données présentent dans le monde aujourd'hui ont été générées ces deux dernières années. Toutes ces données proviennent de capteurs, de médias sociaux, de photos et vidéos digitales, de transactions bancaires, de données GPS provenant des téléphones, etc. Toutes ces données forment ce qu'on appelle *Big Data* aujourd'hui.

Pour quantifier ces volumes importants de données, nous utiliserons l'exemple du très célèbre réseau social Facebook [5]. Facebook contient aujourd'hui plus d'un milliard d'utilisateurs accédant au site via leur ordinateur ou téléphone. Tous ces utilisateurs généraient en 2012 plus de 500 téraoctets de nouvelles données chaque jour [6]. Plus de 105 téraoctets de données sont scannées par leur système de recherche toutes les 30 minutes. Une grande partie de ces données sont d'ailleurs géo-référencées.

La complexité de traitements de toute cette information dont les volumes sont très importants et le type de données très varié est un sujet contemporain [73, 117, 35, 71, 68].

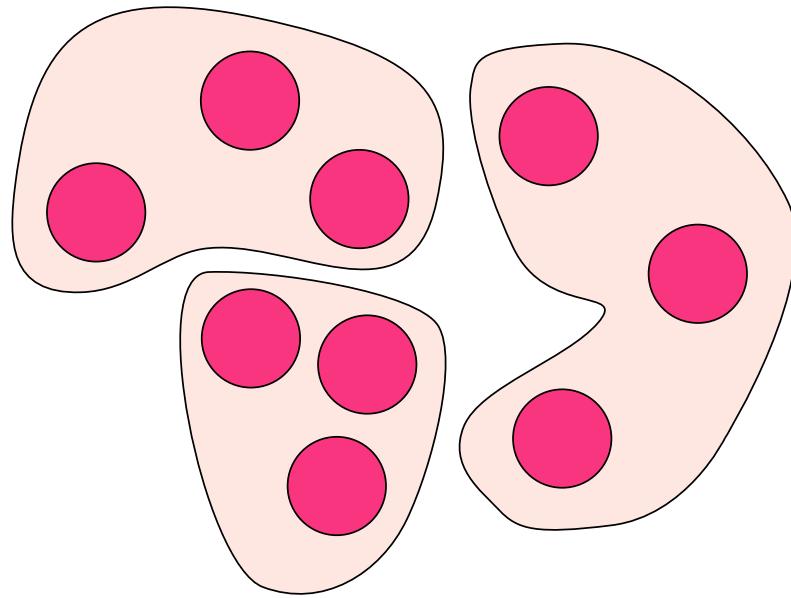
La notion de Big Data s'inscrit bien sûr dans un contexte plus large que les données géo-référencées mais celles-ci en font néanmoins bien partie.

Dans le début de ce chapitre, nous avons présenté la notion de SIG, de puissants outils d'analyse de données géo-référencées. Dans les sections suivantes, nous allons introduire et formaliser un des problèmes d'optimisation relevant de l'analyse spatiale. Il s'agit du problème de partitionnement de territoire.

### 4.4 Partitionnement de territoire

Nous allons maintenant nous intéresser au problème de partitionnement de territoire. De manière générale, le partitionnement d'un ensemble consiste, comme son nom l'indique à créer une partition de celui-ci. Une partition d'un ensemble  $S$  est alors définie comme une division de cet ensemble en sous-ensembles non vides, qui ne se chevauchent pas et qui couvrent entièrement l'ensemble  $S$ . La notion de partition est illustrée à la Figure 4.4 et définie ci-dessous.

**Définition 12. Partition** Soit un ensemble  $S$  quelconque. Un ensemble  $P$  de sous-



**Figure 4.4** – Cette figure illustre la notion de partition. L’ensemble des cercles rouge  $S$  est partitionné en trois sous-ensembles. La partition  $P$  est formé des trois sous-ensembles de couleur crème.

ensemble de  $S$  est appelé une partition de  $S$  si et seulement si[67, 23] :

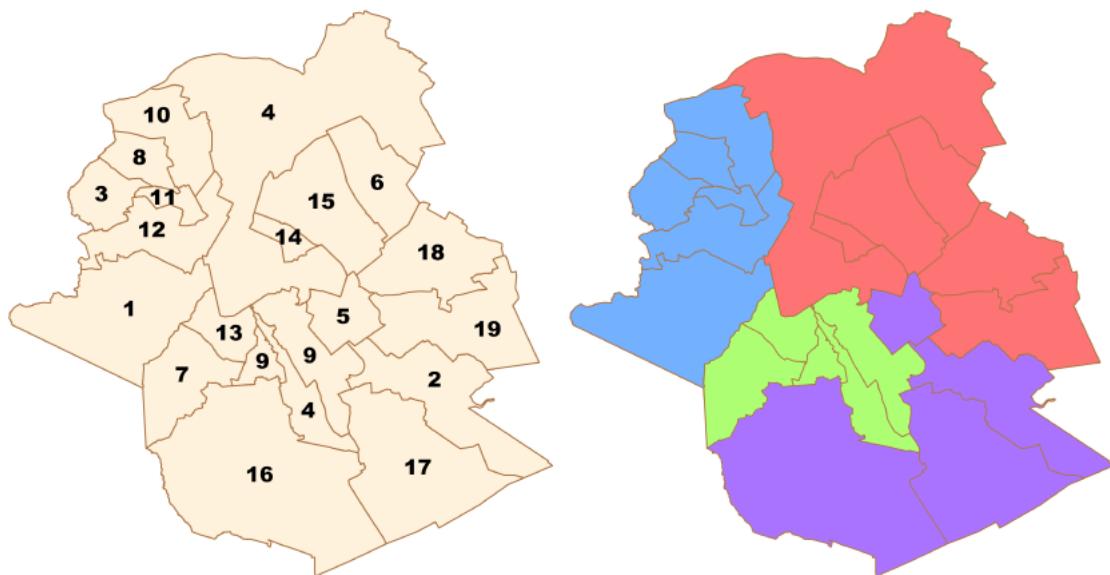
$$\begin{aligned} \emptyset &\notin P \\ \bigcup P &= X \\ J \cap K &= \emptyset \quad \forall J, K \in P, \quad J \neq K \end{aligned} \tag{4.1}$$

Le problème consistant à grouper de petites unités ou zones géographiques en de plus grands regroupements géographiques selon certains critères est connu dans la littérature sous le nom de *districting*. La Figure 4.5 permet de mieux visualiser à quoi correspond un districting.

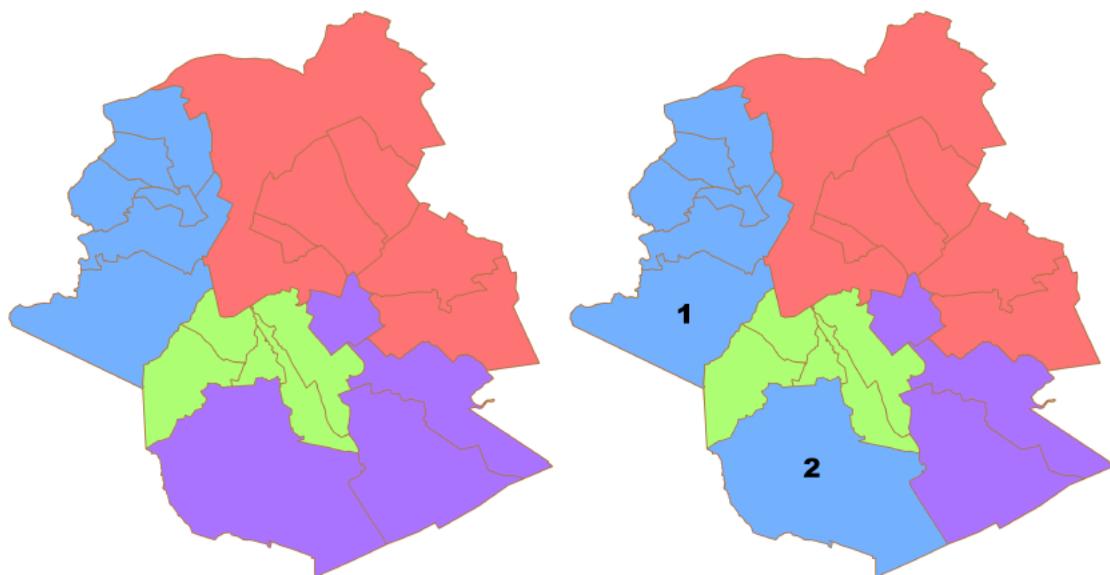
Plus précisément, le problème de districting consiste à agréger  $n$  unités surfaciques en  $k$  zones de manière à optimiser une ou plusieurs fonctions d’objectif tout en étant sujet à des contraintes sur la topologie des zones. En particulier, la connectivité de celles-ci.

Cette dernière notion est importante, en effet, celle-ci stipule que le problème de districting contient une contrainte supplémentaire aux trois contraintes du problème de partitionnement tel que nous l’avons défini en Définition 12. Ceci rend le problème de districting un problème plus complexe [21] que le problème de partitionnement puisque cette nouvelle contrainte limite le nombre de solutions acceptables. Concrètement, cette contrainte consiste à assurer une contiguïté entre les différentes unités surfaciques qui constituent une zone. La contiguïté au sein d’une zone signifie que chaque unité surfacique au sein d’une zone est connectée à toutes les autres unités surfaciques qui constituent cette zone via des unités surfaciques faisant également partie de cette même zone. Ceci est illustré à la Figure 4.6.

Le problème de districting a de très nombreuses applications et répond à des questions pratiques de différents domaines. Les sous-sections suivantes illustrent quelques-unes de



**Figure 4.5** – La figure de gauche est une carte de la région de Bruxelles-Capitale et de ses 19 communes. Ces dernières peuvent être considérées comme des unités surfaciques et un possible partitionnement de celles-ci en 4 zones est illustré sur la figure de droite.



**Figure 4.6** – Illustration de la notion de contiguïté des zones du districting. Sur le partitionnement de gauche, toutes les zones sont contiguës, ce n'est pas le cas sur le partitionnement de droite où la zone bleue n'est pas contiguë. En effet, bien que la commune d'Anderlecht (1) et la commune d'Uccle (2) fasse toutes les deux parties de la zone bleue, il est impossible d'accéder à la commune d'Uccle en partant de la commune d'Anderlecht sans passer par une commune ne faisant pas partie de la zone bleue.

ces applications et visent à donner une idée de l'intérêt du problème de districting.

#### 4.4.1 Partitionnement électoral

Le partitionnement électoral (electoral districting) est très certainement le cas de problème de districting le plus connu et qu'on retrouve le plus dans la littérature [21].

Celle-ci vise à agréger  $n$  zones administratives, en  $k$  zones ou districts de façon à ce que chacun de ceux-ci soit contiguë, compacte et que le nombre d'habitants soit homogène entre les différents districts.

L'idée derrière le partitionnement électoral peut-être illustrée par le fameux principe, “un homme, une voix”. Avoir des districts homogènes au niveau du nombre d'habitants permet alors une équité entre les différents districts au niveau du vote.

Le terme *gerrymandering* fait référence à la manipulation du découpage des circonscriptions électORALES dans le but de favoriser un parti ou un candidat. Le terme prend naissance en 1811 lorsque le gouverneur américain *Elbridge Gerry* signe un redécoupage du Massachusetts donnant un net avantage à son parti. Cet exemple historique montre bien l'importance du problème de partitionnement électoral et la motivation d'utiliser des techniques informatiques afin de mieux garantir l'équité du partitionnement.

#### 4.4.2 Zone de travail pour des équipes de vendeurs

Une autre application du problème de districting est la définition de zones de travail pour des équipes de vendeurs [46, 114, 112].

Le problème peut être formulé comme suit : Une compagnie vend des produits à travers un territoire. Cette compagnie dispose de différentes équipes de vendeurs, chacune responsable d'une partie des clients. De manière plus concrète, à chaque équipe de vendeurs est assignée une partie du territoire. L'équipe est alors responsable de tout les clients situés dans la partie du territoire à laquelle elle est associée.

La distribution du territoire doit alors répondre à certains critères et maximiser certains objectifs comme par exemple la balance de la charge de travail entre les différentes équipes. Le temps de trajet entre une équipe et chacun de ses clients devrait également être minimisé. La Figure 4.7 permet de se donner une idée de ce problème particulier.

De nombreuses autres applications existent pour le problème de districting. Nous citerons par exemple :

- Le problème consistant à spécifier les groupes d'enfants qui fréquentent chaque école est connu sous le nom de *school districting* [44, 50].
- Des applications dans le domaine de l'épidémiologie [55].
- Dans le domaine des télécommunications, le problème de districting est utilisé pour trouver l'emplacement optimal d'antennes ou de HUB internet [91].

Pour un problème de districting donné, la désignation des différentes unités surfaciques utilisées est un choix complexe qui ne doit pas être négligé. S'il peut sembler naturel d'utiliser des zones administratives pour le problème de districting électoral, le choix se



**Figure 4.7** – Illustration d'une possible découpe du territoire de la région de Bruxelles-Capitale en zones de travail pour 4 équipes de vendeurs. Chaque équipe, représentée ici par un cercle brun se voit assigner un territoire et est responsable de chacun des clients en faisant partie. Le lien entre une équipe et ses clients est représenté ici par des droites.

révèle bien plus complexe pour les autres types de problèmes mentionnés précédemment. Pour le lecteur intéressé, les questions d'échelles et d'agrégations dans le cadre des problèmes districting sont traitées dans l'article [88].

#### 4.4.3 Complexité

Une des raisons qui fait du problème de districting un problème complexe est la taille de l'espace de solution qui rend impraticable une énumération exhaustive de celles-ci [21].

Si l'on ne tient pas compte de la contrainte de contiguïté, la taille de l'espace de solution est donnée par le nombre de Stirling du second type. Celui-ci correspond aux nombres de façon de partitionner un ensemble de  $n$  objets en  $k$  sous-ensembles non-vides. Il peut être calculé comme suit [100] :

$$\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n. \quad (4.2)$$

Pour donner une idée de l'étendue de l'espace de recherche, on peut noter que le nombre de Stirling du second type correspondant au partitionnement de 100 éléments en 4 sous-ensembles est de  $6.7 \times 10^{58}$ . On constate donc bien qu'une énumération des solutions n'est pas envisageable.

Pour le problème de partitionnement de territoire, la contrainte de contiguïté des zones doit être respectée et le nombre de solutions réalisables est donc inférieur au nombre de Stirling du second type. Celui-ci donne toutefois une borne supérieure.

De plus, le problème de districting a été montré comme NP-complet en terme de complexité [20]. Il semble donc naturel de se tourner vers des heuristiques afin de produire des solutions de qualité en un temps raisonnable.

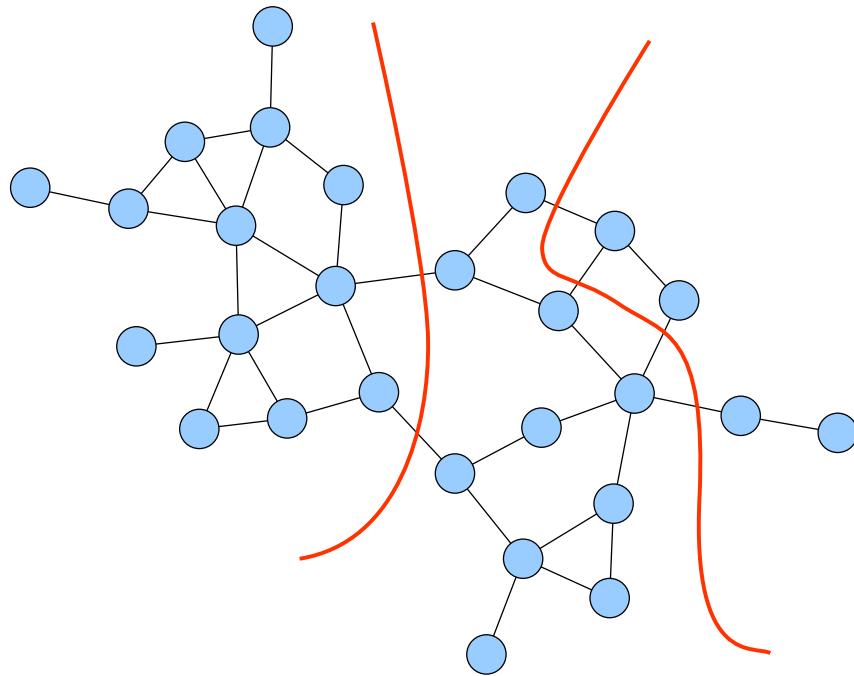
De nombreuses méthodes d'optimisation ont été mises au point pour aborder le problème de districting [119], plusieurs d'entre-elles basées sur des métahéuristiques. Quelques exemples de techniques mises au points sont : le recuit simulé [29], la recherche tabou [26], une heuristique basée sur la méthode de séparation et évaluation [78], ou même des techniques d'énumérations implicites [51] qui n'ont de sens que pour de petites instances.

Toutefois l'utilisation d'algorithme génétique pour résoudre le problème de districting reste rare dans la littérature comme le mentionne les articles [21, 112].

#### 4.4.4 Partitionnement de graphe

Le problème du partitionnement de graphe est un problème qui peut être apparenté au problème de districting. Il s'agit d'un problème très étudié dans la littérature [24].

Dans le cadre du partitionnement d'un graphe, on peut considérer deux types de partitionnements. Le partitionnement des noeuds ou le partitionnement des arêtes. Nous nous restreindrons au premier type, celui-ci étant le plus commun.



**Figure 4.8** – Sur cette figure, le graphe est partitionné en trois parties. Les traits rouges représentent les endroits de coupe. La première coupe, séparant la partie gauche de la partie centrale, sectionne deux arêtes. La coupe de droite, séparant la partie centrale de la partie de droite, sectionne quatre arêtes. Le coût de coupe total est donc de 6.

Soit un graphe non dirigé  $G = (N, A)$ , où  $N$  est l'ensemble des nœuds et  $A$  est l'ensemble des arêtes. On appelle, en anglais, une *k-way partition*, une partition de l'ensemble des sommets en  $k$  sous-ensemble connexes. Une bonne partition est alors définie comme une partition où les différentes parties sont de tailles similaires et le coup de coupe est minimal.

Le coup de coupe est le nombre d'arêtes reliant les différentes parties. Si les arêtes sont valuées, on utilise alors la somme des valeurs d'arêtes reliant les différentes parties [25]. Plus formellement, soit  $P = \{N_1, N_2, \dots, N_k\}$ , un partitionnement de  $N$  en  $k$  parties :

$$\begin{aligned} \text{coupe}(N_i, N_j) &= \sum_{n_i \in N_i} \sum_{n_j \in N_j} \text{poidsArete}(n_i, n_j) \\ \text{coupe}(P) &= \sum_{i < j} \text{coupe}(N_i, N_j) \end{aligned} \quad (4.3)$$

La Figure 4.8 illustre le partitionnement d'un graphe ainsi que la notion de coupe.

De nombreuses méthodes de résolution existent pour le problème du partitionnement de graphe défini ci-dessus. La nature du problème étant très complexe (NP-Hard), les solutions pratiques sont souvent basées sur des heuristiques. Parmi les approches les plus célèbres, on peut citer [24] : les méthodes inertielles, la classification hiérarchique, l'expansion de région, la méthode spectrale, la méthode multi-niveaux [65], les différentes variantes de l'algorithme de Kernighan-Lin ainsi que les métahéuristiques en général. Le lecteur intéressé pourra se référer aux ouvrages suivants [24, 25] pour plus de détails sur ces méthodes.

Le problème présenté ci-dessus est le cas le plus classique et le plus étudié de partitionnement de graphe. D'autres variantes de ce problème existent également avec des contraintes et des fonctions d'objectifs différentes. Le partitionnement de graphe a de nombreuses applications. Parmi celles-ci [25] : la répartition de charge dans les machines parallèles, la conception de circuits intégrés, la segmentation d'image, l'exploration de données, etc.

Si le problème de districting peut être vu comme un problème de partitionnement de graphe, comme nous le verrons dans la suite de ce mémoire, il est toutefois important de ne pas confondre les deux problèmes tels que nous les avons définis jusqu'ici. En effet, certaines techniques de résolution du problème de partitionnement de graphe peuvent être appliquées au problème de districting mais il ne s'agit pas de l'unique façon de procéder.

#### 4.4.5 Problème de regroupement

Le problème de regroupement (*clustering* en anglais) est un autre problème d'optimisation très étudié dans la littérature [63, 120]. Celui-ci, que nous allons définir ci-dessous peut-être vu comme un problème plus général au problème de districting.

Le regroupement de données consiste à regrouper un ensemble de  $n$  données en différents groupes homogènes par rapport à certains critères définis. On peut remarquer qu'il s'agit bien d'un problème de partitionnement tel que nous l'avons défini en Définition 12. Toutefois, on remarquera également que la contrainte additionnelle de contiguïté des zones, présente dans le problème de districting, ne fait pas partie du problème de regroupement qui est donc plus général. La Figure 4.9 illustre le regroupement de données.

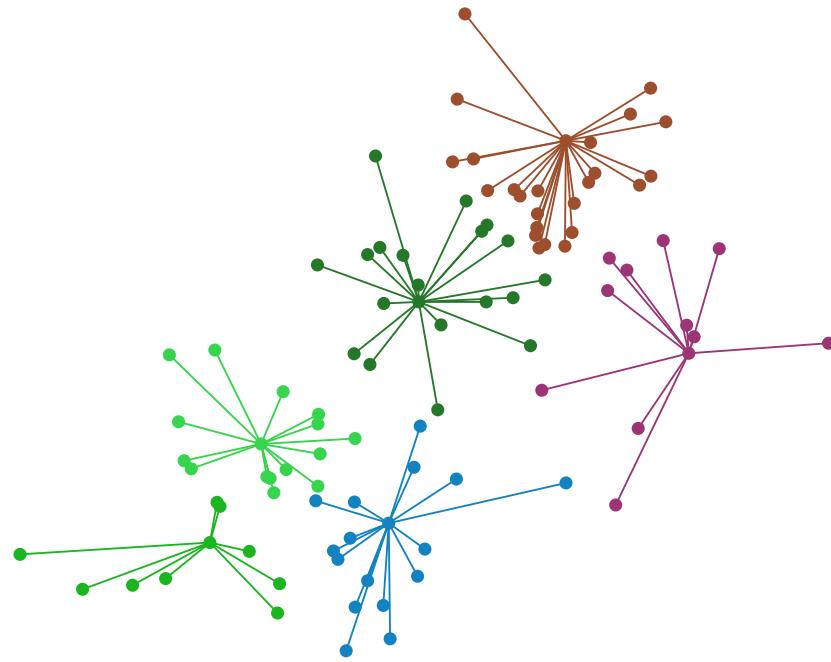
La notion d'homogénéité des zones peut être vue de différentes façons [112, 63]. On fera la distinction entre l'homogénéité *intra-zone* et l'homogénéité *inter-zone*.

**Intra-zone** L'homogénéité intra-zone vise à avoir des groupes au sein desquels les données sont les plus homogènes possible.

**Inter-zone** L'homogénéité inter-zone vise à avoir des groupes les plus homogènes entre eux. On pourra chercher à maximiser ou minimiser cette homogénéité en fonction du problème étudié. Par exemple, en la minimisant, on obtiendra des groupements bien distincts.

Le problème de regroupement a de nombreuses application dans le domaine de l'intelligence artificielle et du *data mining*. Nous mentionnons plus tôt l'émergence des *big-data* (Section 4.3). Le volume titanique de données produites aujourd'hui nécessite l'apparition de nouvelles méthodes de traitement de ces données. Le regroupement de celle-ci et la détection de pattern au sein de données jouent bien évidemment un rôle primordial dans le domaine [63].

Les lecteurs intéressés par le problème de regroupement, ses différentes applications, ainsi que les différentes techniques existantes pourront se référer aux ouvrages [43, 97, 120, 63, 62].



**Figure 4.9** – Cette figure, provenant de [123], illustre le regroupement de données. Les données, représentées sur le graphique par des points ont été regroupées (ou divisées) en 6 groupe distincts. La configuration de ces regroupements peut laisser penser que la proximité des données au sein d'un groupe (homogénéité intra-zone) faisait partie des objectifs recherchés.

### Algorithme des k-moyennes

L'algorithme des k-moyennes est une des approches existantes au problème de regroupement. Elle fait partie des méthodes les plus utilisées et les plus simples [62]. Elle a été introduite pour la première fois par MacQueen J. en 1967[75].

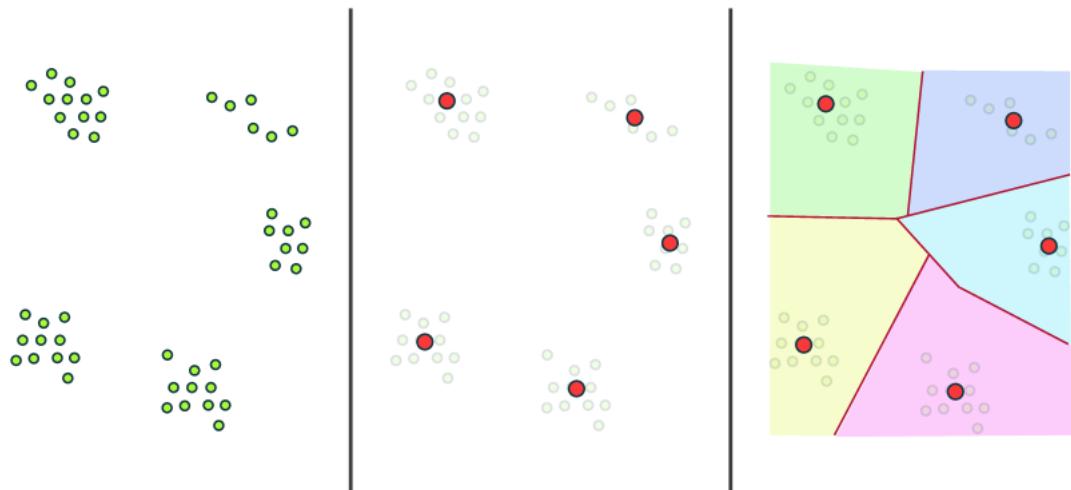
Le principe consiste à regrouper  $n$  éléments en  $k$  groupes, où chaque élément appartient au groupe ayant la plus proche moyenne, de manière à ce que la distance au carré entre chaque élément et la moyenne de son groupe soit minimale. Ceci est illustré à la Figure 4.10.

Soit  $(x_1, x_2, \dots, x_n)$ , un ensemble de  $n$  vecteurs de données de même dimension. La méthode des k-moyennes vise à regrouper ces données en  $k$  groupes,  $S = \{S_1, S_2, \dots, S_k\}$  de manière à minimiser la fonction d'objectif suivante [11] :

$$J = \sum_{i=1}^k \sum_{x_j \in S_i} \|x_j - c_i\|^2 \quad (4.4)$$

où  $c_i$  est la moyenne des points faisant partie du groupe  $i$ .

Le problème d'optimisation visant à minimiser cette valeur  $J$  est un problème NP-Hard. L'algorithme des k-moyennes est donc une méthode approchée qui converge vers un optimum local. Le pseudo-code [11] de cet algorithme itératif est donné à l'Algorithm 5. La Figure 4.11 montre une itération de l'algorithme.



**Figure 4.10** – Cette figure, provenant de [99], illustre la méthode des  $k$ -moyennes. Sur la partie gauche de la figure, on peut voir les données représentées par des points dans un espace à deux dimensions. Sur la partie centrale de la figure, on voit les 5 moyennes des groupes considérés. Enfin, sur la partie droite de la figure, on voit de diagramme de Voronoi qui délimite les 5 régions composées des points les plus proches des moyennes.

---

**Algorithme 5:** Pseudo-code de l'algorithme des  $k$ -moyennes.
 

---

Prendre  $k$  points dans l'espace, de même dimension que les données. Ces points,  $c_1, c_2, \dots, c_k$  seront une première estimation des moyennes des  $k$  groupes.

**repeat**

Calculer les groupes en assignant chaque solution au groupe dont la moyenne est la plus proche.

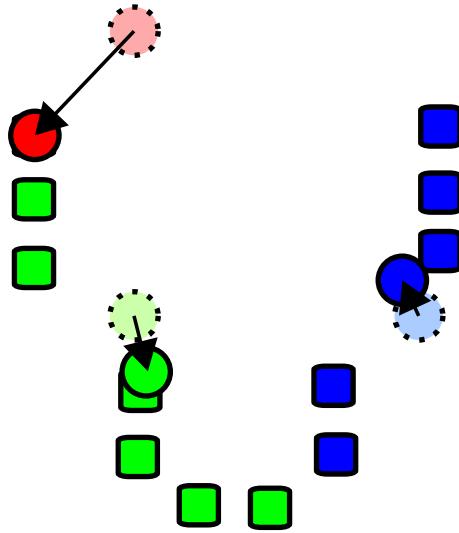
**for**  $i = 0$  **to**  $k$  **do**

| Recalculer la moyenne  $c_i$

**end**

**until** *Plus aucune moyenne n'a changé*

---



**Figure 4.11** – Cette figure [118] illustre une itération de l’algorithme des  $k$ -moyennes. Les moyennes (ou centres) de chaque groupe sont mises à jour.

Comme expliqué plus tôt, l’algorithme des  $k$ -moyennes est un algorithme très simple, il souffre toutefois de plusieurs faiblesses. L’algorithme est en effet très sensible aux conditions initiales. De plus, sa tendance à créer des sous-ensembles de même taille mène parfois à de mauvais résultats. On lui préférera donc d’autres méthodes comme l’algorithme *Expectation-maximization*. Le lecteur intéressé par cet algorithme pourra par exemple se référer aux ouvrages [90, 27].

## 4.5 Conclusion

Au cours de ce chapitre, nous avons pu introduire de nombreuses notions. Tout d’abord, les systèmes d’informations géographiques, véritable mariage entre la géographie et l’informatique. Ceci nous a permis d’introduire le problème de *Big Data*, problème au cœur de notre société contemporaine.

La seconde partie de chapitre était l’objet du problème de partitionnement de territoire, également connu sous le nom de districting. Nous avons alors défini la notion de partitionnement et traité de sa complexité. Finalement, nous nous sommes intéressé à deux autres problèmes proche du problème de districting ; le partitionnement de graphe et le problème de regroupement.

Nous sommes maintenant en possession de tous les outils nécessaires pour aborder les prochains chapitres de ce mémoire. Ceux-ci concernent les différents apports personnels. Après une lecture critique d’un article proposant l’utilisation d’un algorithme génétique pour résoudre le problème de districting que nous avons présenté dans ce chapitre, nous étudierons un cas particulier de partitionnement de territoire, celui de la Wallonie.

## **Deuxième partie**

# **Apports personnels**

# Chapitre 5

## Lecture critique de l'article

### 5.1 Introduction

Ce chapitre vise à décrire et fournir une analyse critique de l'article *Multiple criteria districting problems : The public transportation network pricing system of the Paris region*[112] écrit par Fernando Tavares-Pereira, José Rui Figueira, Vincent Mousseau et Bernard Roy. Nous allons premièrement passer en revue leur formulation du problème de districting et ensuite présenter leur algorithme tout en faisant des remarques et critiques lorsque cela est pertinent.

Ce chapitre visant à présenter le travail de Tavares et al., il contient des passages traduits de leur article ainsi que de très nombreuses citations. La référence vers leur article [112] est parfois omise volontairement pour ne pas être trop redondante. Il convient donc de bien garder en mémoire que ce chapitre présente et critique le résultat de leurs recherches.

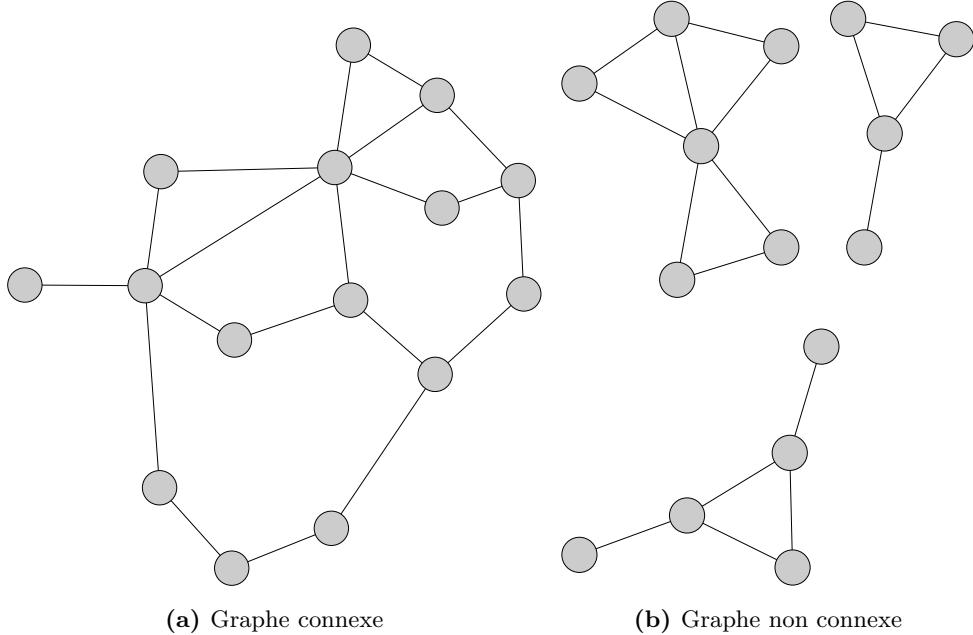
### 5.2 Enoncé du problème

Les auteurs ont décidé de s'intéresser au problème de districting multicritère. Celui-ci correspond généralement plus aux besoins réels que sa version monocritère.

Comme nous l'avons vu en Section 4.4.3, l'utilisation de métahéuristiques pour résoudre le problème de districting a déjà fait l'objet de nombreuses recherches. Toutefois, l'utilisation d'algorithmes génétiques pour résoudre ce problème reste peu exploitée.

En Section 4.4.4, nous avions évoqué le problème de partitionnement de graphe en précisant que, bien que ce ne soit pas l'unique façon de procéder, il était possible de formuler le problème de districting en utilisant la théorie des graphes. C'est le choix qu'ont fait les auteurs de l'article.

“Soit, un territoire composé d’éléments territoriaux indivisibles. On définit un *graphe de contiguïté* comme un graphe planaire non dirigé connexe  $G = (N, A)$ , où  $N = \{1, 2, \dots, n\}$  est l’ensemble des noeuds, chacun représentant un élément territorial et  $A = \{a_1, a_2, \dots, a_m\} \subset N \times N$  est l’ensemble des arêtes, où  $a_i = (i, j)$  représente deux



**Figure 5.1** – Cette figure illustre la notion de graphe connexe. La figure de droite est composée de trois parties connexes mais elle ne le sont pas entre elles.

éléments territoriaux adjacents  $i$  et  $j$ .”[112] La Figure 5.3 illustre la construction d’un graphe de contiguïté.

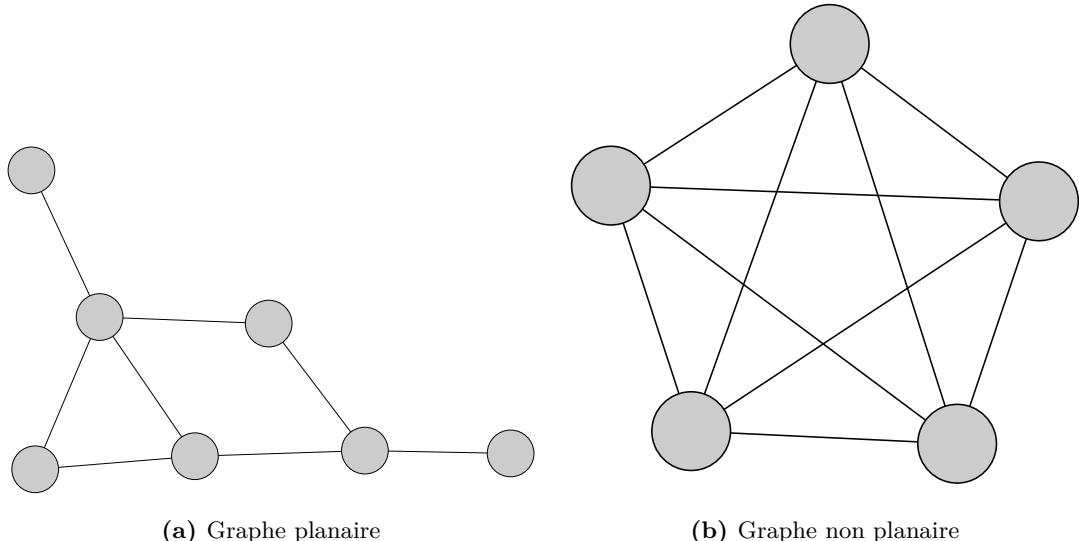
**Définition 13. *Graphe connexe*** [87] *En théorie des graphes, un graphe non orienté est dit connexe s'il existe un chemin entre tout couple de sommets de ce graphe. Cette notion est illustrée à la Figure 5.1.*

**Définition 14. *Graphe planaire*** [87] *En théorie des graphes, un graphe est dit planaire s'il est possible de le représenter dans le plan sans qu'aucune de ses arêtes ne se croisent. Cette notion est illustrée à la Figure 5.2.*

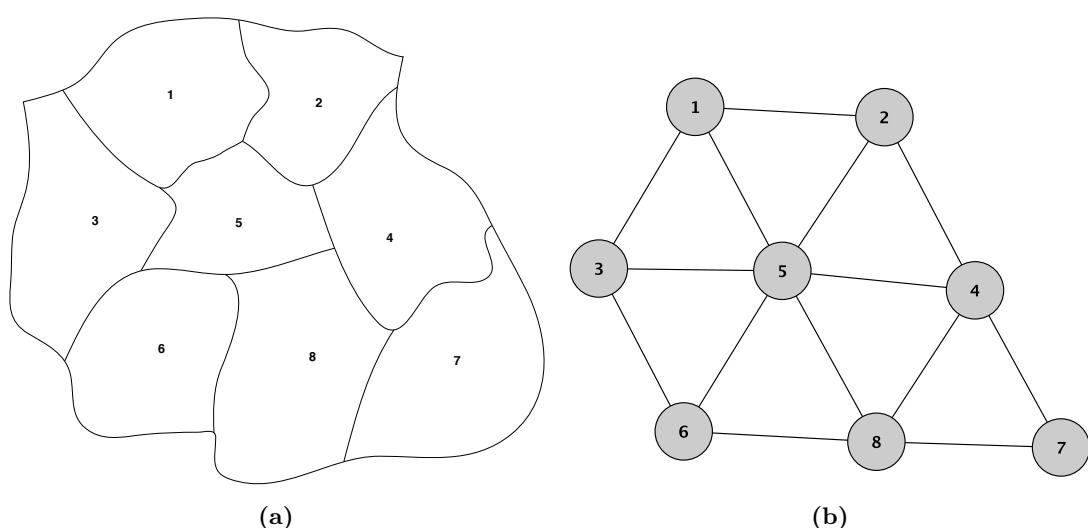
“Une fois le graphe de contiguïté  $G = (N, A)$  défini, une carte de districts peut être considérée comme une partition de  $N$  en sous-ensembles de noeuds connexes. De plus, toutes les valeurs associées aux éléments territoriaux (population, surface, etc.) peuvent être associées aux noeuds correspondants. De façon similaire, les valeurs associées à une paire d’éléments territoriaux contigus (par exemple la longueur de la frontière commune) peuvent être associées à l’arête correspondante.”[112]

Une solution  $Y$  au problème de districting sera alors représentée par une partition de  $N$  en  $K$  sous-ensembles. Ces derniers respectant les propriétés que nous avons définies en Définition 12. Pour rappel :

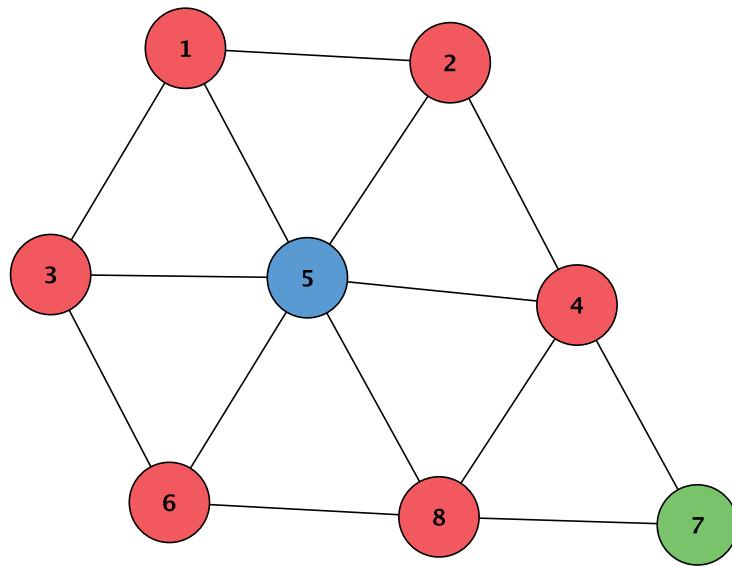
$$Y = \{y_1, y_2, \dots, y_k\} \quad (5.1)$$



**Figure 5.2** – Cette figure illustre la notion de graphe planaire. Il est impossible de dessiner le graphe de droite dans le plan sans que deux arêtes ne se croisent.



**Figure 5.3** – Cette figure illustre l'utilisation d'un graphe non dirigé, connexe et planaire pour modéliser un territoire. A chaque noeud correspond un élément territorial et chaque arête modélise la contiguïté de deux éléments. Nous continuerons d'utiliser cet exemple dans la suite de ce chapitre.



**Figure 5.4** – Cette figure illustre la notion de zone incluse. La zone bleue est incluse dans la zone rouge. Ceci est une solution que nous voulons éviter.

où

$$\begin{aligned} \emptyset &\notin Y \\ \bigcup_{1 \leq i \leq k} y_i &= V \\ y_u \cap y_v &= \emptyset, u \neq v \end{aligned} \tag{5.2}$$

### 5.2.1 Contraintes

Dans le cas du problème de districting, des contraintes supplémentaires sont nécessaires. Les auteurs de l'article mentionnent les contraintes suivantes :

**Intégrité** Un nœud ne peut appartenir à plusieurs sous-graphes. Chaque nœud appartient à exactement un sous-graphe. Cette contrainte est toutefois déjà présente dans la définition de partition.

**Contiguïté** Il doit être possible de trouver un chemin entre deux nœuds appartenant à une même zone sans passer par une zone différente.

**Absence de trous** Il ne doit pas être possible d'obtenir un sous-graphe inclus dans un autre sous-graphe. La notion de graphe inclus est illustrée à la Figure 5.4

La modélisation de ces différentes contraintes au sein de l'algorithme est également adressée dans l'article. La contrainte d'intégrité est triviale à modéliser ; une variable binaire est définie pour chaque couple nœud - zone. La contrainte de contiguïté requiert un nombre exponentiel de contraintes dans le modèle. Enfin, la contrainte d'absence de trou n'est pas adressée dans l'article et ne fait donc pas partie de l'algorithme. Toutefois, des solutions comportant des zones incluses sont très rarement générées. Nous verrons dans la section concernant la présentation de l'algorithme comment ces différentes contraintes sont prises en compte.

### 5.2.2 Critères

Les auteurs de l'article abordent également les critères les plus courants pour le problème de districting. Ceux-ci sont au nombre de trois.

**Critères d'homogénéisation** Les critères de type homogénéisation peuvent être classés en deux catégories : l'homogénéisation intra-zone et l'homogénéisation inter-zone.

**Homogénéisation intra-zone** Les critères d'homogénéisation intra-zone visent à créer des zones composées d'éléments territoriaux les plus similaires possible. Un exemple pourrait être l'homogénéisation du revenu. Ceci signifierait que au sein de chaque zone, les différents éléments territoriaux aient un revenu le plus similaire possible. Soit une partition  $Y = \{y_1, y_2, \dots, y_k\}$ , en gardant l'exemple du revenu, la valeur de la fonction objectif peut se calculer comme suit :

$$f_{revenu}(Y) = \sum_{y_u \in Y} H(y_u) \quad (5.3)$$

où

$$H(y_u) = \max_{i \in y_u} revenu_i - \min_{i \in y_u} revenu_i \quad (5.4)$$

où  $revenu_i$  est la valeur du revenu pour la commune  $i$ . Le problème consiste alors à minimiser cet objectif.

**Homogénéisation inter-zone** Au contraire de l'homogénéisation intra-zone qui vise à avoir des éléments territoriaux similaires au sein de chaque zone, l'homogénéisation inter-zone, vise à avoir des zones similaires entre elles. Un exemple classique d'un tel critère est l'homogénéisation de la population visant à avoir des zones comportant le même nombre d'habitants. Pour cet exemple, la valeur de la fonction d'objectif peut se calculer comme suit :

$$f_{population}(Y) = \max_{y_u \in Y} H(y_u) - \min_{y_u \in Y} H(y_u) \quad (5.5)$$

où

$$H(y_u) = \sum_{i \in y_u} population_i \quad (5.6)$$

où  $population_i$  est le nombre d'habitants dans la commune  $i$ . Le problème consiste alors à minimiser cet objectif.

**Critique** La modélisation mathématique des critères d'homogénéisation décrite ci-dessus est celle proposée par les auteurs de l'article. Toutefois, il est intéressant de noter que dans le cadre d'une utilisation au sein d'une métaheuristique, cette modélisation peut souffrir d'un défaut.

En effet, l'utilisation de l'opérateur min et max induit que seule une variation des valeurs extrêmes fera varier la fonction objectif. Dans le cadre d'un processus itératif construisant les solutions par étapes successives, ceci peut être un problème puisque de petites variations au sein des zones ne feront pas nécessairement changer les valeurs des fonctions d'objectif ce qui peut nuire à la convergence de l'algorithme et à la découverte de nouvelles bonnes solutions.

Une autre façon de procéder pourrait être de considérer la somme des différences par rapport à la moyenne à la place des extrema. Ceci permet de rendre

la fonction objectif sensible à chaque changement et non plus ceux des extrêmes uniquement.

La fonction d'objectif d'homogénéisation intra-zone devient alors :

$$f_{revenu}(Y) = \sum_{y_u \in Y} H(y_u) \quad (5.7)$$

$$H(y_u) = \sum_{i \in y_u} |revenu_i - \mu| \quad (5.8)$$

$$\mu = \frac{\sum_{i \in y_u} revenu_i}{\#y_u} \quad (5.9)$$

Et la fonction d'objectif inter-zone devient :

$$f_{population}(Y) = \sum_{y_u \in Y} |S(y_u) - \mu| \quad (5.10)$$

$$\mu = \frac{\sum_{y_u \in Y} S(y_u)}{k} \quad (5.11)$$

$$S(y_u) = \sum_{i \in y_u} population_i \quad (5.12)$$

**Critère de compactage** Ce critère vise à créer des zones compactes visuellement. Il s'agit d'un critère très courant dans les problèmes de districting. Le degré de compactage d'une partition sera égal au plus mauvais degré de compactage des zones qui la composent. Ce critère peut se mesurer grâce à la fonction d'objectif suivante :

$$f_{compact}(Y) = \max_{y_u \in Y} C(y_u) \quad (5.13)$$

$$C(y_u) = \frac{\pi r^2}{\sum_{i \in y_u} surface_i} \quad (5.14)$$

où  $r$  est le rayon du plus petit cercle contenant l'entièreté de la zone et  $surface_i$  est la surface de la commune  $i$ .

**Critique** Le même problème que celui énoncé pour les critères d'homogénéisation est valable. En effet, seul le degré de compactage de la moins bonne zone se reflète dans l'évaluation du partitionnement considéré. Dans ce cas, l'amélioration du degré de compactage d'autres zones peut ne pas se refléter dans la fonction d'objectif. Nous reviendrons sur ce problème dans le chapitre suivant.

**Critère de similarité** Les auteurs citent également un autre critère qu'ils considèrent comme régulièrement utilisé, le critère de similarité. Celui-ci vise à mesurer la similarité entre un partitionnement et un autre partitionnement dit de référence. Ce critère peut par exemple être utilisé pour trouver des partitionnements proches d'un partitionnement existant. Nous y reviendrons dans un chapitre ultérieur en Section 7.3.

### 5.3 Algorithme génétique

Nous allons maintenant aborder l'algorithme proposé par les auteurs de l'article. Cette section vise à en faire la description mais également une analyse critique. Les différentes étapes de l'algorithme seront accompagnées d'exemples. Pour ceux-ci, nous continuerons d'utiliser le territoire simple que nous avions introduit en Figure 5.3.

L'algorithme proposé par les auteurs de l'article est une météuristiche hybride. Plus précisément, il s'agit d'un algorithme évolutionniste hybridé par une recherche locale.

Dans cette section, nous allons adopter une approche *top-down* pour présenter l'algorithme, après avoir présenté le fonctionnement global de l'algorithme, nous aborderons en détail les différentes étapes de son fonctionnement.

Le pseudo-code de l'algorithme proposé est disponible en Figure 6. Il suit le schéma classique d'un algorithme génétique que nous avons présenté précédemment, auquel est ajouté la procédure de recherche locale.

---

**Algorithme 6:** Pseudo-code de l'algorithme[112].

---

```

 $t \leftarrow 0$ 
Generate  $\mathcal{P}_0$ 
Compute the Fitness value for each individual in  $\mathcal{P}_0$ 
Initialize the set of potential non-dominated solutions,  $Z^{nd}$ , from  $\mathcal{P}_0$ 
while  $\neg$ StopCondition do
    Children  $\leftarrow$  Crossover ( $\mathcal{P}_t$ )
    Mutant  $\leftarrow$  Mutation (Children)
    Update  $Z^{nd}$  with Children  $\cup$  Mutant
    LocalSearch ( $Z^{nd}$ )
    Compute the Fitness value for each individual in  $\mathcal{P}_t \cup$  Children  $\cup$  Mutant
     $\mathcal{P}_{t+1} \leftarrow$  the best of  $\mathcal{P}_t \cup$  Children  $\cup$  Mutant
     $t \leftarrow t + 1$ 
end

```

---

#### 5.3.1 Représentation des solutions

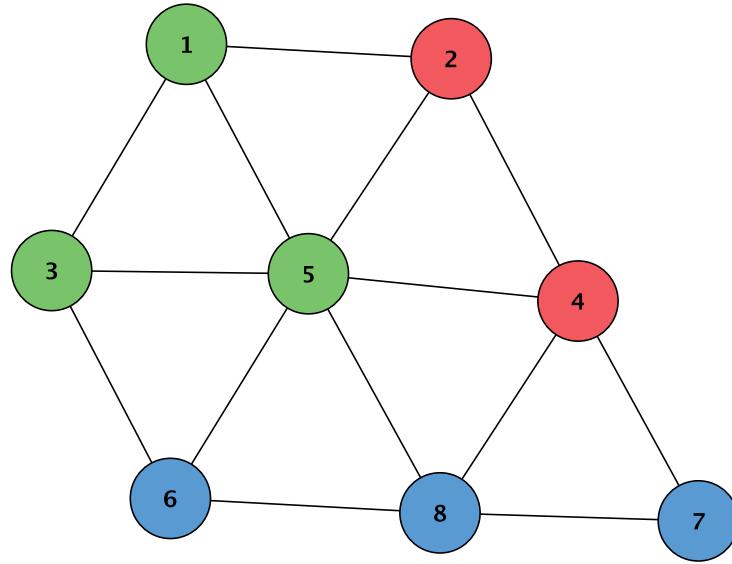
Les auteurs proposent de représenter une partition du territoire en  $k$  régions comme une liste de listes de nœuds du graphe de contiguïté construit précédemment.

De cette manière, une solution  $Y = \{y_1, y_2, \dots, y_k\}$  pour le graphe  $G = (N, A)$  est représentée comme une liste de graphes connexes, eux-mêmes représentés comme une liste de nœuds du graphe  $G$ .

Par exemple,  $Y = [[1, 3, 5], [2, 4], [6, 7, 8]]$  est illustrée à la Figure 5.5.

#### 5.3.2 Assiguation de fitness

La première procédure que nous allons aborder est l'assiguation de fitness. Nous avions abordé les différentes techniques d'assiguation de fitness dans la Section 3.3.1.



**Figure 5.5** – Cette figure vise à illustrer la façon dont sont représentées les solutions. La solution montrée est représentée par la liste de listes  $[[1, 3, 5], [2, 4], [6, 7, 8]]$ .

Les auteurs ont choisi d'utiliser une assignation de fitness basée sur la notion de dominance, en particulier, la technique de profondeur de dominance. Pour rappel, cette technique consiste à ranger les solutions de la population en fronts. Le premier front contient les solutions non-dominées. Le second front contient les solutions non-dominées si les solutions du premier front sont retirées de la population. Et les fronts suivants contiennent les solutions non-dominées lorsque les solutions de tous les fronts précédents sont retirées de la population. Cette technique est une technique classique utilisée par exemple dans l'algorithme NSGA-II [38] que nous avons présenté en Section 3.4.1.

Une technique de préservation de la diversité est ensuite appliquée au sein de chaque front. Celle-ci est reprise de l'algorithme NSGA [103] et fonctionne comme suit : Une fois la valeur de fitness assignée à chaque solution (en fonction du front auquel elle appartient), cette valeur est réduite si la solution est trop proche d'autres solutions existantes.

La nouvelle valeur obtenue, notée  $\mathcal{F}$ , est alors appelée la valeur de partage. Elle est obtenue en divisant la valeur de fitness de départ,  $F$ , par la valeur  $\alpha$ .

$$\mathcal{F} = \frac{F}{\alpha} \quad (5.15)$$

La quantité  $\alpha$  pour une solution  $i$  est calculée comme suit :

$$\alpha_i = \sum_{j \in Front} sh(d_{ij}) \quad (5.16)$$

où  $d_{ij}$  est la distance euclidienne dans l'espace d'objectif entre deux solutions  $Y_i$  et  $Y_j$  et  $sh(d_{ij})$  vaut :

$$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^2, & \text{if } d_{ij} < \sigma_{share} \\ 0 & \text{otherwise} \end{cases} \quad (5.17)$$

Plus le nombre de solutions proches d'une solution  $i$  sera grand, plus la valeur de fitness de cette solution  $i$  sera réduite. Il est important de noter que la valeur de partage de toutes solutions faisant partie du front  $i$  restera plus grande que la valeur de partage de toutes solutions occupant le front  $i + 1$ . La valeur de partage permet donc de différencier les solutions d'un même front tout en les gardant supérieures à celles des fronts suivants.

L'algorithme 7 est le pseudo-code proposé par les auteurs pour trier les solutions en fronts et calculer leur valeur de partage.

---

**Algorithme 7:** Pseudo-code de l'assignation de fitness[112].

---

```

input :  $\mathcal{P} = \{Y_1, Y_2, \dots, Y_N\}$ 
output :  $\mathcal{F}_i$  for each  $Y_i$ 
 $\mathcal{P}_{aux} \leftarrow \mathcal{P}$ 
 $F \leftarrow$  a large dummy fitness value
while  $\mathcal{P}_{aux} \neq \emptyset$  do
     $Z^{nd} \leftarrow$  all potential non-dominated individuals in  $\mathcal{P}_{aux}$ 
    for  $Y_i \in Z^{nd}$  do
        | Calculate  $\alpha_i$ 
        |  $\mathcal{F}_i \leftarrow \frac{F}{\alpha_i}$ 
    end
     $\mathcal{P}_{aux} \leftarrow \mathcal{P}_{aux} - Z^{nd}$ 
     $F \leftarrow \min\{\mathcal{F}_i : Y_i \in Z^{nd}\} \times p$ 
end

```

---

A chaque nouveau front, la plus petite valeur de partage du front précédent est multipliée par  $p \in ]0, 1[$  afin de garantir une valeur de partage plus petite pour toutes solutions du front en construction.

Avant d'appliquer les opérateurs génétiques, il est nécessaire de sélectionner des solutions dans la population. Pour ce faire, les auteurs ont choisi d'opter pour une technique dite de roulette. Celle-ci consiste à sélectionner aléatoirement deux solutions dans la population et à garder celle ayant la plus grande valeur de partage (de fitness). De cette manière, bien que chaque solution puisse potentiellement être sélectionnée, les solutions résidant dans les premiers fronts ont une plus grande chance de l'être.

### Critique

Comme mentionné précédemment, cette technique d'assignation de fitness est celle utilisée dans l'algorithme NSGA [103]. Elle lui est d'ailleurs reprochée [38] car elle nécessite le tuning du paramètre  $\sigma_{share}$ . Cette méthode de mesure de la diversité est d'ailleurs remplacée par une autre technique dans NSGA-II [38] comme nous l'avons vu dans la Section 3.4.1.

### 5.3.3 Crossover

Lors du design d'un algorithme évolutionniste, une étape importante est l'élaboration des différents opérateurs génétiques, c'est-à-dire l'opérateur de crossover et l'opérateur de mutation.

Pour rappel, l'opérateur de crossover vise à tirer parti de deux solutions et de les combiner pour en créer une nouvelle. Soit, deux solutions :

$$Y = \{y_1, y_2, \dots, y_k\} \quad (5.18)$$

et

$$Y' = \{y'_1, y'_2, \dots, y'_{k'}\} \quad (5.19)$$

L'opérateur de crossover fonctionne comme suit [112] : la première étape consiste à sélectionner un nombre  $j \leq k$  de zones de la solution  $Y$ ,  $S_Y = \{\bar{y}_1, \bar{y}_2, \dots, \bar{y}_j\} \subseteq Y$ . Ces zones, composées des noeuds  $N_c = \bigcup S_Y \subseteq N$ , feront partie de la solution *fille* en cours de construction.

Les noeuds restants, c'est-à-dire  $N \setminus N_c$ , seront copiés de la seconde solution,  $Y'$ . Toutefois, comme l'illustre la Figure 5.6, il n'est pas suffisant de copier les zones de la solution  $Y'$  en ignorant les noeuds déjà présents dans la solution en construction, c'est-à-dire  $N_c$ .

Il est donc nécessaire de définir une relation d'équivalence sur les noeuds  $N \setminus N_c$  afin de compléter la solution en construction. Cette relation d'équivalence est définie comme suit :

Soit  $v_1, v_2 \in (N \setminus N_c)$ , on dira que  $v_1 \sim v_2$  si  $\exists j \in [1, \dots, k']$  tel qu'il existe un chemin entre  $v_1$  et  $v_2$  dans  $y'_j \setminus N_c$ .

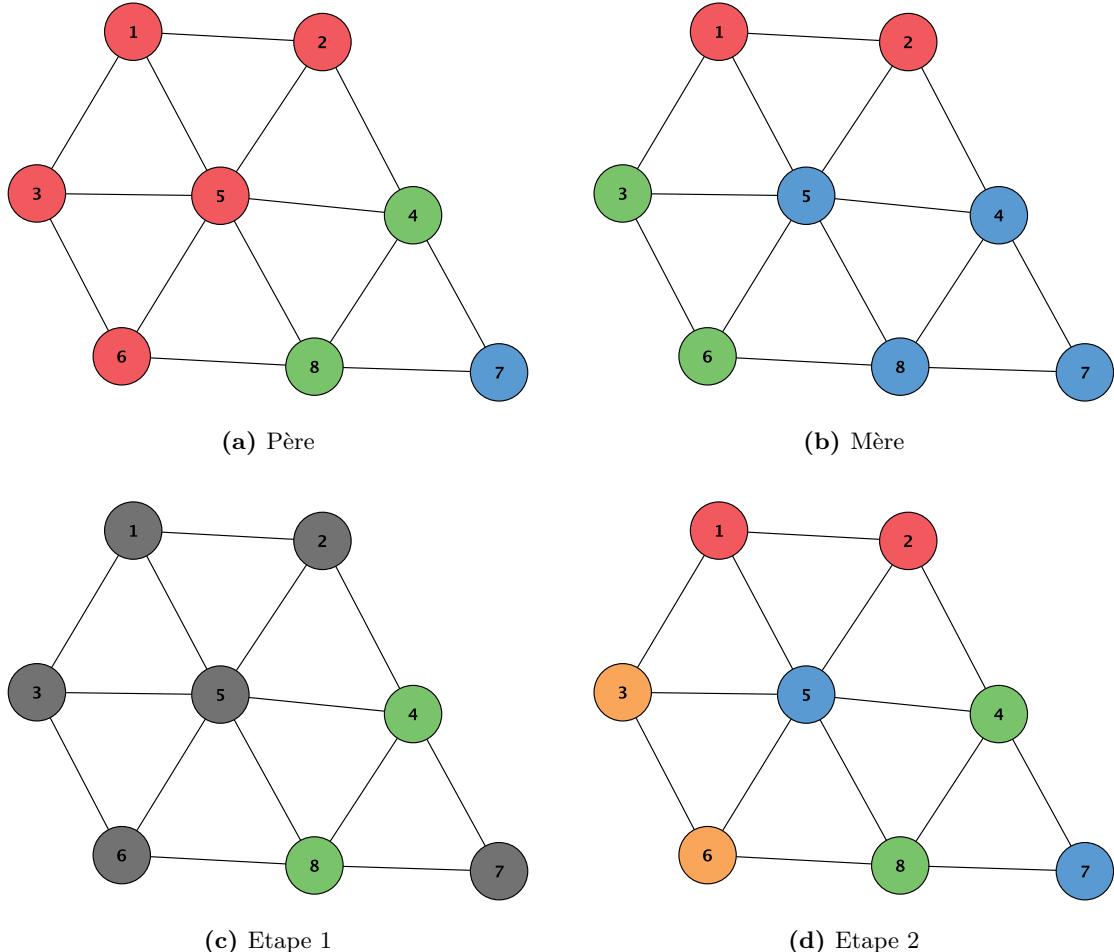
En d'autres termes, comme expliqué dans l'article, on dira que  $v_1$  est équivalent à  $v_2$  si il existe un chemin entre ces deux noeuds dans une des zones de  $Y'$  sans tenir compte des noeuds de  $N_c$ .

La solution résultant du crossover sera donc composée des zones  $\bar{y}_1, \bar{y}_2, \dots, \bar{y}_j$ , sélectionnées dans la première solution, ainsi que de la classe d'équivalence de la relation d'équivalence, définie sur  $N \setminus N_c$ . Ceci est illustré à la Figure 5.7.

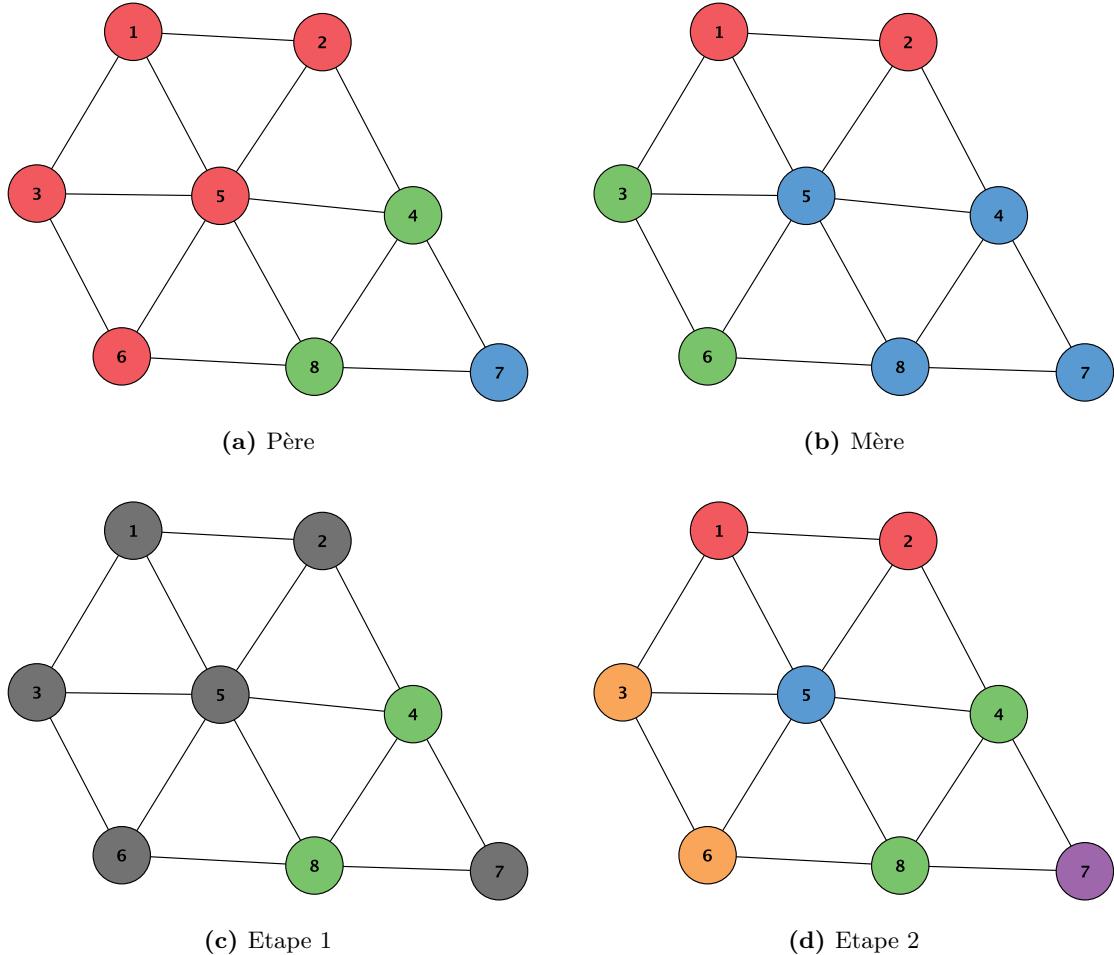
Après cette seconde étape, la solution générée contient souvent plus de zones que les solutions parents. C'est d'ailleurs le cas sur la Figure 5.7. Une troisième étape est donc nécessaire. Cette dernière consiste à appliquer une procédure appelée *Merging* (fusion) sur la solution générée afin de réduire le nombre de zones qui la composent. Nous verrons en détail cette procédure dans la Section 5.3.5.

### Critique

Imaginons le problème de districting d'un territoire en  $k$  zones. Lors de la génération de nouvelles solutions via la procédure de crossover, il est possible que des solutions contenant plus de  $k$  zones soient générées. Les auteurs de l'article adressent ce problème grâce à la procédure de fusion qui permet de ramener le nombre de zones à  $k$ .



**Figure 5.6** – Cette figure illustre la nécessité de définir une relation d'équivalence avant de copier les zones de la seconde solution. En effet, à la première étape (c), la zone verte est copiée de la première solution (a) vers la solution fille. La deuxième étape consiste à utiliser la seconde solution (b) pour partitionner les nœuds restants. La figure (d) montre qu'il ne suffit pas de copier les zones de la solution (b) en ignorant les nœuds déjà présents dans la solution fille. On peut voir que la solution résultante n'est pas valide car la zone bleue, copiée de la solution (b) n'est pas connexe.



**Figure 5.7** – Cette figure illustre la création d'une solution fille à partir des deux solutions (a) et (b). A la première étape, la zone verte de la solution (a) est copiée dans la solution fille. Ensuite, la classe d'équivalence est calculée sur la solution (b) en ne tenant compte que des noeuds restants c'est-à-dire  $\{1, 2, 3, 5, 6, 7\}$ . La classe d'équivalence définie par la relation d'équivalence  $\sim$  est  $\{1, 2\}, \{3, 6\}, \{5\}$  et  $\{7\}$ . Ces zones sont alors ajoutées à l'étape 2 (d).

Toutefois, les auteurs oublient de mentionner que des solutions contenant moins de  $k$  zones peuvent également être générées comme le montre la Figure 5.8. Ces solutions sont bien évidemment des solutions non réalisables pour le problème de partitionnement en  $k$  zones et doivent donc être rejetées lorsqu'elles sont générées.

Il est également pertinent de remarquer que l'ordre des solutions utilisées dans le crossover est important puisqu'il générera très souvent des solutions différentes. Si nous appelons le *père* la solution utilisée à l'étape 1 du crossover et la *mère*, la solution utilisée à l'étape 2 et qui est utilisée pour appliquer la relation d'équivalence sur les noeuds restants, on peut constater que bien qu'on retrouve souvent dans la solution *fille*, les zones du père sélectionnées à l'étape 1, il est souvent plus rare de retrouver les zones de la mère dans la solution finale. En effet, ces dernières sont souvent en plus grand nombre et la recombinaison des zones peut fortement modifier la solution, la rendant alors fort différente de la mère et du père. Ceci est illustré à la Figure 5.9 où la solution générée ne contient aucune des zones des parents et diffère fortement de chacun de ceux-ci.

### 5.3.4 Mutation

La procédure de mutation consiste, comme nous l'avons vu à la Section 3.4, à modifier de manière aléatoire une solution.

La procédure proposée par les auteurs de l'article est très simple, elle consiste à sélectionner un certain nombre de zones dans la solution et à les diviser. Ceci est illustré à la Figure 5.10.

Diviser une zone constituée de  $n$  noeuds consiste à la remplacer par  $n$  zones, chacune constituée d'un noeud unique.

Après ce processus, si au moins une zone a été divisée, le nombre de zones est évidemment plus grand que celui de départ. La procédure de fusion est alors appliquée pour retourner au nombre de zones voulu.

#### Remarque

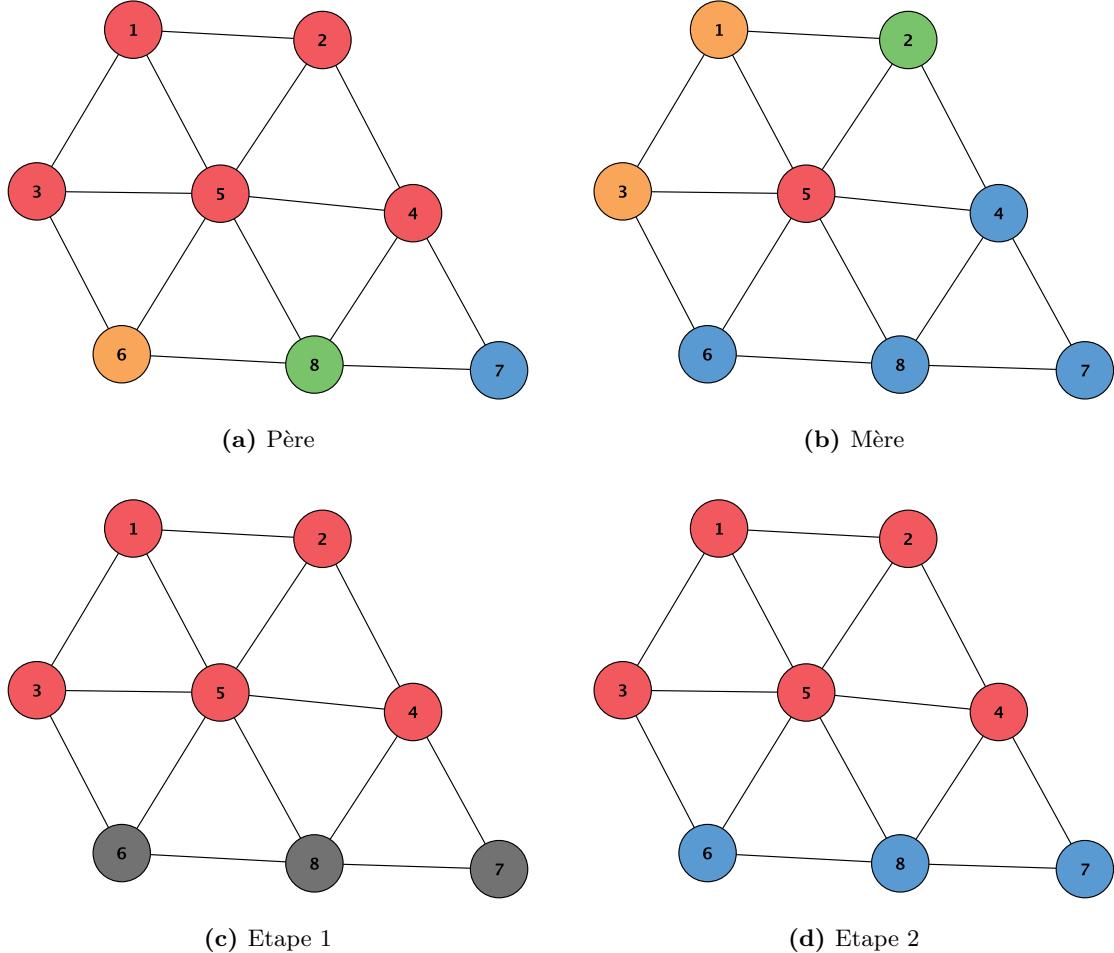
Deux probabilités sont à prendre en compte pour l'opérateur de mutation. La première, présente dans la majorité des algorithmes évolutionnistes, est la probabilité d'appliquer l'opérateur de mutation sur une solution de la population (souvent résultante de l'opérateur de crossover).

La seconde probabilité intervient lors de l'application de l'opérateur de mutation sur une solution. Il s'agit de la probabilité de sélectionner une zone pour la diviser.

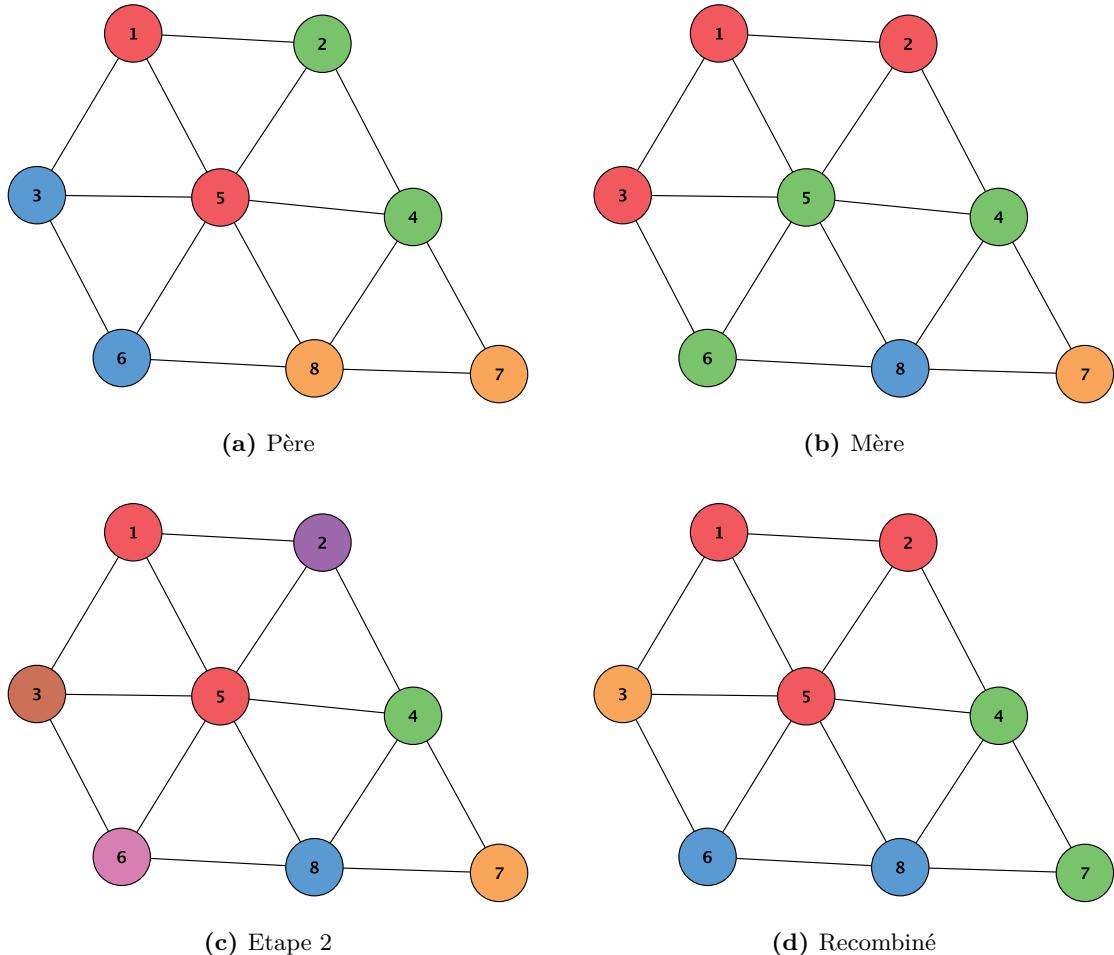
### 5.3.5 Procédure de fusion

L'objectif de la procédure de fusion, (*merging*) que nous avons mentionnée plusieurs fois précédemment, est de réduire le nombre de zones d'une solution si celui-ci est trop grand.

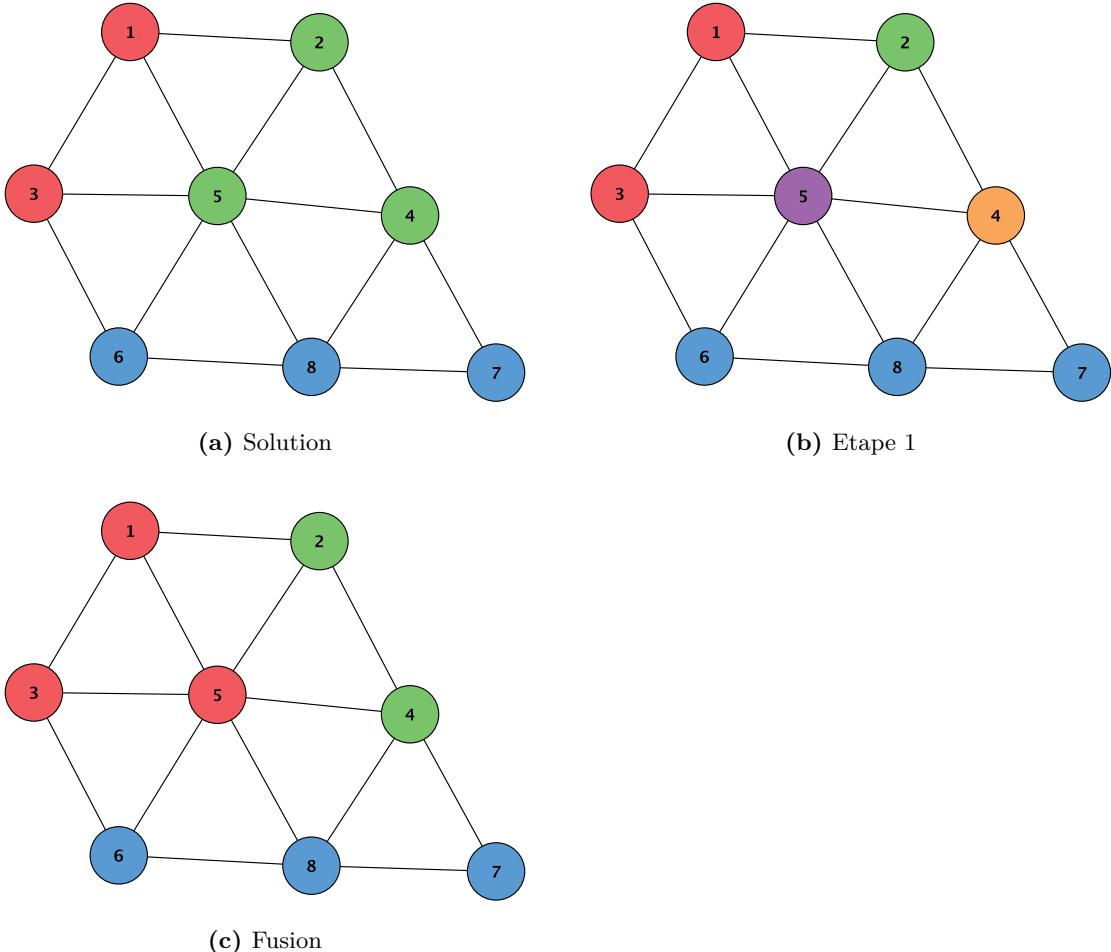
Cette procédure consiste à fusionner deux zones adjacentes tant que le nombre de zones est supérieur au nombre de zones souhaité,  $k$ . L'algorithme 8 est le pseudo-code de cette



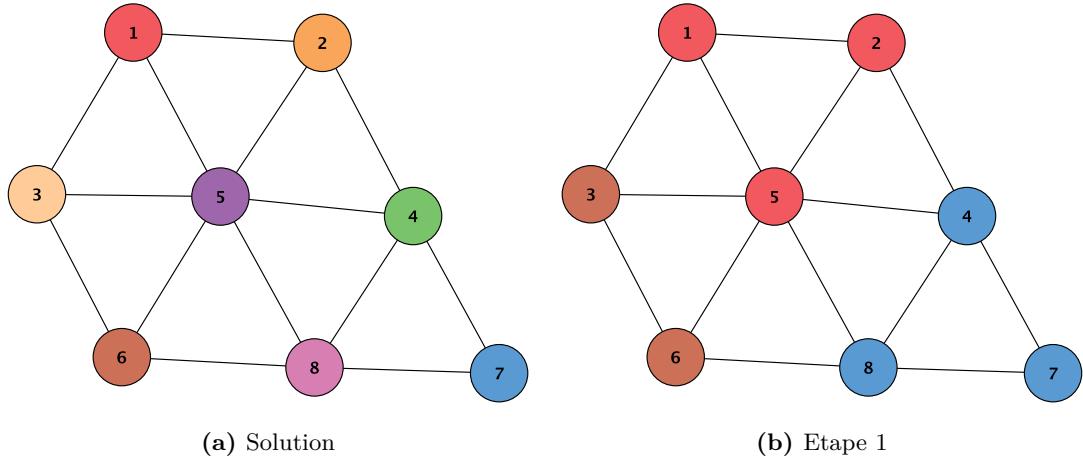
**Figure 5.8** – Cette figure montre que l'opérateur de crossover peut générer des solutions ayant trop peu de zones. Imaginons le problème consistant à partitionner le territoire en 4 zones. L'opérateur de crossover est appliqué sur la solution (a) et (b). A la première étape (c), la zone rouge de la solution (a) est sélectionnée et copiée dans la solution fille. A la seconde étape, la classe d'équivalence de la relation d'équivalence  $\sim$  appliquée sur les nœuds restants, c'est-à-dire  $\{6, 7, 8\}$ , est  $\{6, 7, 8\}$ . La solution finale est donc composée de 2 zones et non 4. Il s'agit donc d'une solution non réalisable pour le problème de partitionnement en 4 zones.



**Figure 5.9** – Cette figure met en évidence que les solutions générées par l’opérateur de crossover peuvent être très différentes des solutions parents sélectionnées. La figure (c) résulte du crossover de la solution (a) et (b) avant d’appliquer la procédure de fusion. La zone rouge de la solution (a) a été sélectionnée et copiée dans la solution fille. La classe d’équivalence résultant de la relation d’équivalence  $\sim$  appliquée sur les noeuds restants  $\{2, 3, 4, 6, 7, 8\}$  est  $\{2\}, \{3\}, \{4\}, \{6\}, \{7\}$  et  $\{8\}$ . A ce stade, il est déjà possible de remarquer qu’aucune des zones de la solution (b) ne se retrouve dans la solution générée. La procédure de fusion réduit ensuite le nombre de zones de cette solution (7) au nombre de zones voulu, 4 dans le cas présent. On peut finalement remarquer que la solution finale (d) n’a aucune zone en commun avec les solutions (a) et (b).



**Figure 5.10** – Cette figure illustre la procédure de mutation appliquée sur la solution (a). À la première étape (b), la zone verte est divisée, chaque noeud devient une zone à lui tout seul. Le nombre de zones a alors augmenté et la procédure de fusion permet de retourner à un partitionnement en 3 zones.



**Figure 5.11** – Cette figure illustre la création d'une nouvelle solution. A partir de la solution triviale (a), où chaque zone est composée de un et un seul nœud, la solution est construite en appliquant la procédure de fusion qui va fusionner deux à deux des zones jusqu'à obtenir le nombre de zones désiré.

procédure, modifiant une zone  $Y$  composée de  $K > k$  zones en une solution  $Y'$  composée de  $k$  zones.

---

**Algorithme 8:** Pseudo-code de la procédure de fusion [112].

---

```

input :  $Y = \{y_1, y_2, \dots, y_K\}$ 
output :  $Y' = \{y_1, y_2, \dots, y_k\}$ 
while  $K > k$  do
    Let  $y_i, y_j$  be two neighbors zones, chosen according to a heuristic rule
    defined for each type of criteria
    Merge  $y_i$  and  $y_j$ 
     $K \leftarrow K - 1$ 
end
 $Y' \leftarrow Y$ 

```

---

### 5.3.6 Population initiale

Comme nous l'avons vu au Chapitre 3, il est nécessaire de générer une population initiale de solutions au lancement d'un algorithme évolutionniste.

La procédure générant une nouvelle solution afin d'intégrer celle-ci dans la population initiale fonctionne comme suit : La solution triviale où chaque noeud constitue à lui tout seul une zone est générée. La procédure de fusion décrite à la section précédente lui est ensuite appliquée jusqu'à obtenir une zone ayant le nombre de zones souhaité. Ceci est illustré à la Figure 5.11.

### 5.3.7 Recherche locale

La procédure de recherche locale est appliquée sur les solutions non dominées de la population à chaque itération. Cette procédure vise à améliorer les meilleures solutions en explorant leur voisinage comme expliqué en Section 3.5.

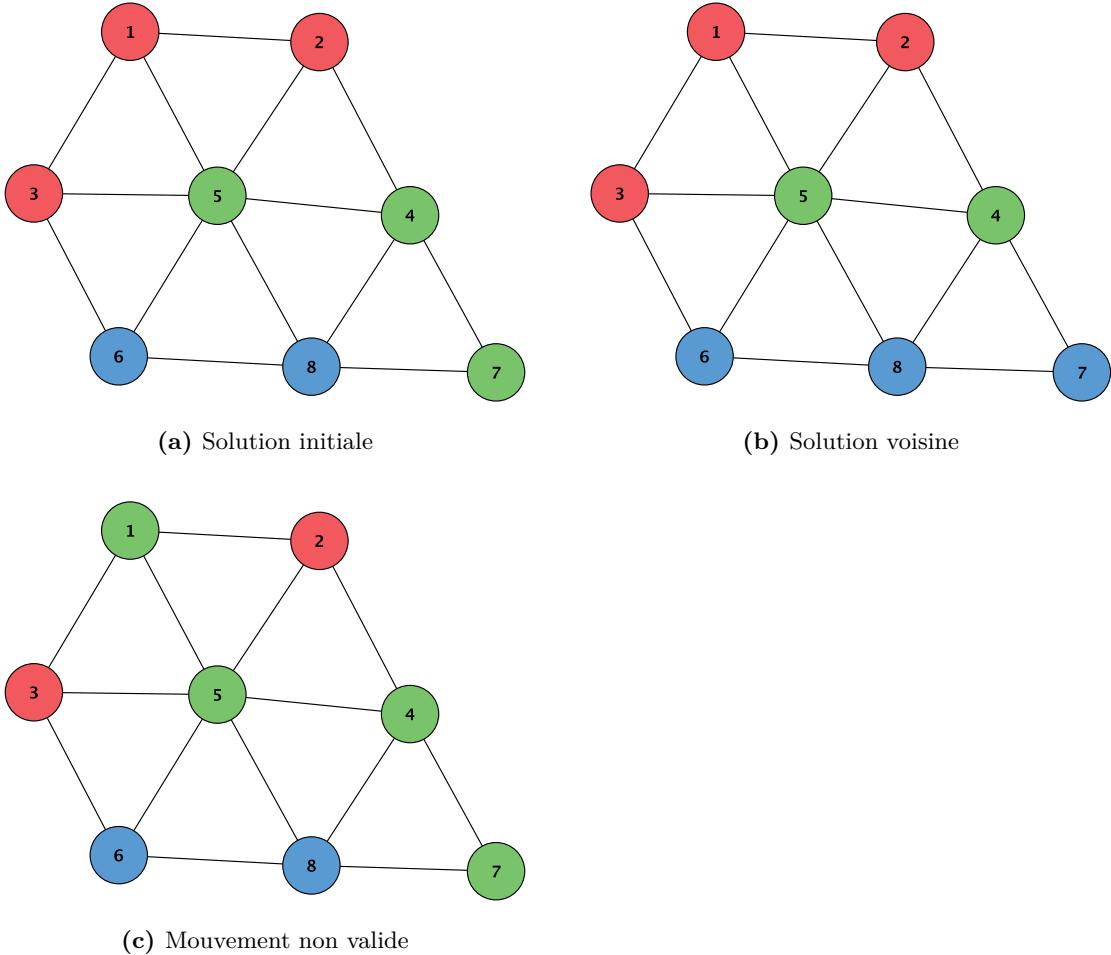
Pour se faire, il est nécessaire de définir la notion de voisinage et la manière de trouver les solutions voisines d'une solution donnée. Toutes les solutions voisines d'une solution donnée sont trouvées en déplaçant au plus un noeud d'une des zones à une zone adjacente. Ceci est illustré en Figure 5.12.

Il est important de remarquer que le déplacement n'est pas toujours possible car il peut rompre la contrainte de contiguïté. Ceci est également illustré en Figure 5.12.

## 5.4 Conclusion

Au cours de ce chapitre nous avons présenté les recherches de M. Tavares-Pereira, M. Figueira, M. Mousseau et de Mr. Roy sur le problème de districting. Nous avons décrit et revu de manière critique leur modélisation du problème ainsi que l'algorithme qu'ils proposent pour le résoudre, à savoir, un algorithme évolutionniste avec recherche locale.

Ce chapitre va servir de fondement au chapitre suivant qui abordera un cas concret de problème de partitionnement, l'implémentation d'un algorithme pour ce dernier ainsi qu'une discussion sur les résultats obtenus et une série d'extensions possibles.



**Figure 5.12** – Cette figure illustre la notion de solution voisine utilisée dans la procédure de recherche locale. La solution (b) est une solution voisine à la solution (a) où le noeud 7 est passé de la zone verte à la zone bleue qui lui était adjacente. La figure (c) illustre un mouvement interdit. En effet, bien que la zone rouge soit adjacente à la zone verte dans la solution initiale (a), le noeud 1 ne peut être déplacé de la zone rouge à la zone verte car cela briserait la contrainte de contiguïté, la zone rouge n'étant plus connexe.

# Chapitre 6

## Etude de cas : la Wallonie

### 6.1 Introduction

Dans ce chapitre, nous proposons une étude de cas pour le problème de districting, à savoir, la Wallonie. Dans un premier temps, nous contextualiserons le problème avec une courte introduction sur la Région wallonne. Nous présenterons la partition actuelle de la Wallonie en cinq provinces. Nous mettrons alors en évidence certaines disparités entre celles-ci. Cela nous permettra de motiver la recherche de nouvelles partitions du territoire wallon.

Nous présenterons ensuite en détail l'algorithme implémenté, en insistant sur sa générnicité, le rendant utilisable pour de nombreux problèmes d'optimisation. Après la présentation des différentes procédures composant l'algorithme, nous détaillerons les différents paramètres de ce dernier et parlerons brièvement de tests unitaires implémentés pour s'assurer de son bon fonctionnement.

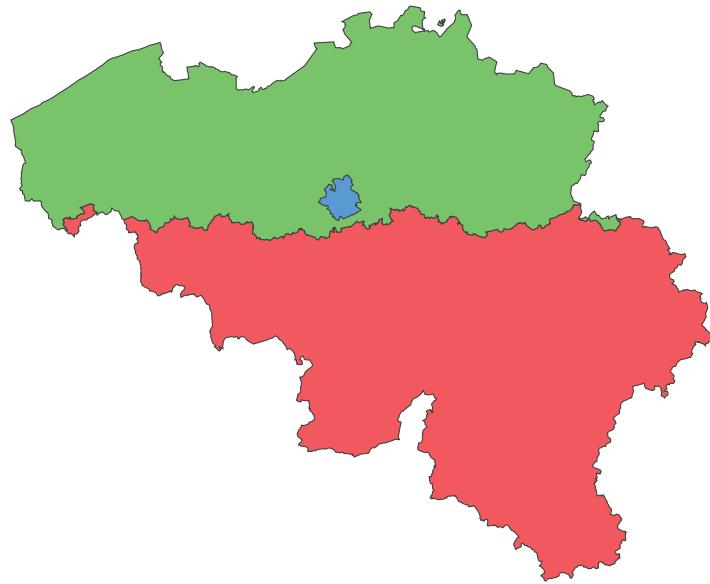
Dans une seconde partie, nous nous attarderons sur l'utilisation de l'algorithme dans le cadre du partitionnement de la Région wallonne. Nous présenterons donc les moyens mis en œuvre pour visualiser les partitionnements générés et les divers outils qui ont été nécessaires pour y parvenir. Nous présenterons ensuite les différentes données dont nous disposons pour chaque commune wallonne et les critères que nous jugeons pertinents pour le problème. Une section est ensuite dédiée au réglage des différents paramètres de l'algorithme. Enfin, plusieurs résultats de partitionnement sont présentés.

### 6.2 La Wallonie

La Wallonie ou Région wallonne, fait partie avec la Région flamande et la Région de Bruxelles-Capitale des trois régions formant la Belgique comme le montre la Figure 6.1.

**La Région wallonne** La Wallonie est composée de 262 communes, rassemblées en cinq provinces : Brabant wallon, Hainaut, Liège, Luxembourg et Namur.

**La Région de Bruxelles-Capitale** La Région de Bruxelles-Capitale couvre 19 communes, ces dernières ne sont pas divisées en provinces.



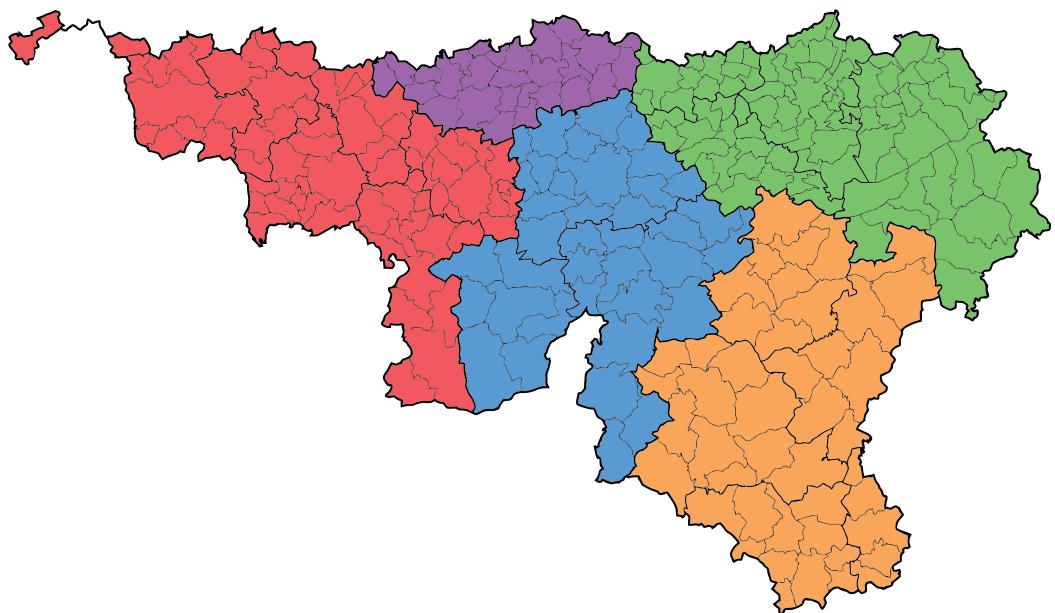
**Figure 6.1** – Cette figure montre les trois Régions composant la Belgique. La zone rouge est la Région wallonne, la zone verte est la Région flamande et la zone bleue est la Région de Bruxelles-Capitale. Cette illustration a originellement été réalisée par Ssolbergj [104].

**La Région flamande** La Flandre est, elle, composée de 308 communes qui se répartissent en cinq provinces également : Anvers (Antwerpen), Brabant flamand (Vlaams-Brabant), Flandre occidentale (West-Vlaanderen), Flandre orientale (Oost-Vlaanderen) et Limbourg (Limburg).

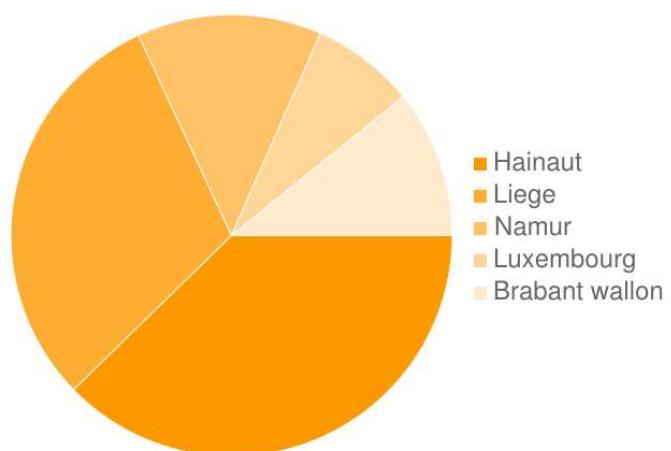
Dans le cadre de cette étude de cas, nous nous intéresserons particulièrement à la Région wallonne et à ses cinq provinces existantes. Celles-ci sont illustrées à la Figure 6.2.

Il est intéressant de remarquer que, bien que la superficie de la Wallonie couvre 60% de la superficie de la Belgique, seul un tiers de la population belge y réside. En effet : “La répartition des Belges sur les différentes régions est la suivante : 6 251 983 vivent en Région flamande, 3 498 384 (dont 75 222 issus de la Communauté germanophone) vivent en Région wallonne et 1 089 538 résident dans la Région de Bruxelles-Capitale.”[2]

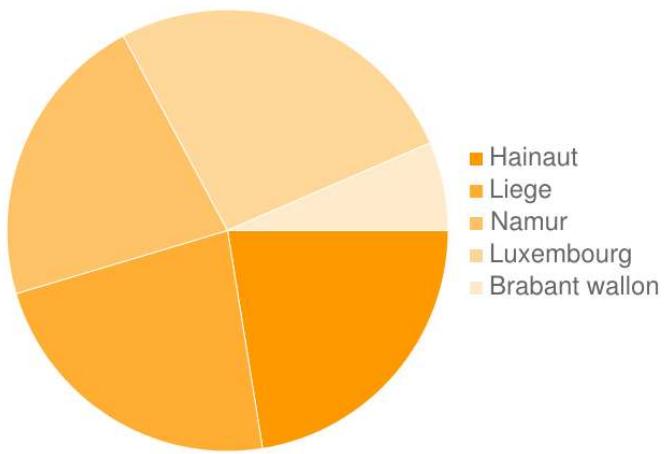
Afin de motiver la recherche de nouvelles partitions, intéressons nous à différentes caractéristiques de la partition actuelle de la Wallonie en cinq provinces. La Figure 6.3 montre la répartition de la population au sein des cinq provinces. Il est possible de remarquer une grande disparité au niveau du nombre d'habitants entre les différentes provinces. La province du Luxembourg par exemple, contient le plus petit nombre d'habitants avec 260,983 habitants. La province du Hainaut, quant à elle, contient 1,292,925 habitants soit près de cinq fois plus. Si l'on s'intéresse ensuite à la Figure 6.4 montrant la répartition de la surface entre les provinces, on peut remarquer que bien que la province du Luxembourg contienne le moins d'habitants, il s'agit de la province la plus vaste. Ceci se répercute évidemment sur la densité de population des cinq provinces (voir Figure 6.5) et accentue encore plus les disparités existantes. Il est donc pertinent de rechercher de nouvelles partitions de la Wallonie. On peut par exemple rechercher une partition reflétant une meilleure homogénéisation du nombre d'habitants.



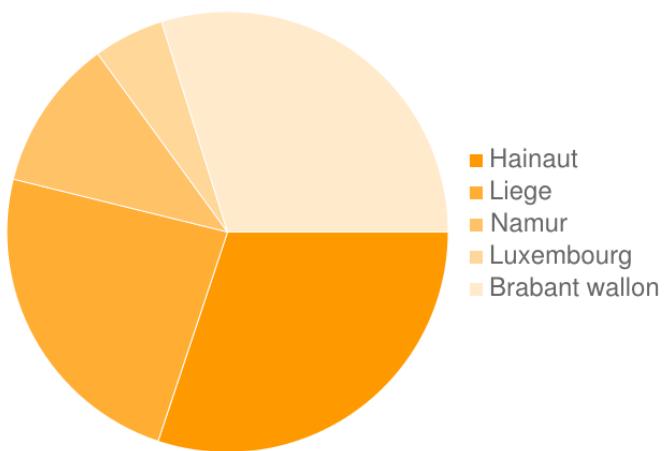
**Figure 6.2** – Cette figure montre le partitionnement actuel de la Wallonie en cinq provinces. La zone rouge est la province du Hainaut, la zone mauve est la province du Brabant wallon, la zone bleue est la province de Namur, la zone verte est la province de Liège et enfin, la zone orange est la province du Luxembourg.



**Figure 6.3** – Répartition de la population au sein des cinq provinces de Wallonie.



**Figure 6.4** – Répartition de la surface du territoire wallon entre les différentes provinces.



**Figure 6.5** – Densité de population de chacune des cinq provinces de la Wallonie.

Comme mentionné précédemment, la Région wallonne est composée de 262 communes pouvant être considérées comme des unités territoriales élémentaires. Dans ce contexte, le regroupement de ces communes en cinq provinces correspond bien à un partitionnement (ou districting) de celles-ci tel que nous l'avons défini dans les chapitres précédents. L'objectif de ce chapitre est donc de proposer des critères, jugés pertinents, et de trouver, à l'aide d'un algorithme génétique, de nouveaux partitionnements de ces 262 communes qui forment la Wallonie.

### 6.3 Algorithme

L'algorithme implémenté pour être appliqué sur le territoire wallon est inspiré de l'algorithme [112] mis au point par Fernando Tavares-Pereira, José Rui Figueira, Vincent Mousseau et Bernard Roy que nous avons présenté au chapitre précédent. Les remarques et critiques que nous avions émises à son égard ont bien sûr été prises en compte.

Cette section vise à présenter l'algorithme de manière générale avant d'aborder, à la Section 6.4, son utilisation dans le contexte de la Wallonie.

Lors du design de l'algorithme, une attention particulière a été portée à le rendre le plus générique possible ; c'est-à-dire à rendre l'algorithme applicable à un grand nombre de problèmes de districting sans que cela ne nécessite de profonds changements de celui-ci.

Toujours dans l'optique de la générnicité, l'algorithme a été paramétrisé un maximum. Nous y reviendrons dans la Section 6.3.6.

L'algorithme a été implémenté en Ruby. Ruby se définit comme suit [15] :

“Ruby est un langage open-source dynamique qui met l’accent sur la simplicité et la productivité. Sa syntaxe élégante en facilite la lecture et l’écriture.”

Ruby est un langage interprété et est donc généralement plus lent qu'un langage compilé comme le C ou le C++. Toutefois, il a été choisi pour sa syntaxe, sa simplicité d'utilisation et son élégance. Ceci permet entre autre d'écrire un programme correct plus aisément qu'avec un langage comme le C. Durant le développement de l'algorithme, la version 2.0.0 de Ruby a été utilisée. Celle-ci est disponible depuis le 24 février 2013.

L'algorithme développé est basé sur l'algorithme NSGA-II [38] que nous avons déjà décrit dans le Chapitre 3 à la Section 3.4.1. Toutefois, l'algorithme est hybride puisqu'une procédure de recherche locale y est ajoutée comme l'ont fait les auteurs de l'article étudié au chapitre précédent.

Le pseudo-code de l'algorithme implémenté est illustré à l'Algorithme 9. On retrouve la procédure de rangement de la population en fronts, nommée `FastNonDominatedSort`, que nous avons décrite à la Section 3.3.1 qui traitant des techniques d'assignation de fitness. On retrouve également la technique de préservation de la diversité, calculée par la procédure `computeCrowdingDistance` qui a été expliquée et illustrée à la Figure 3.10. La procédure `SelectParentsByRank` est également la procédure classique de tournoi binaire présente dans l'algorithme NSGA-II et que nous avions présentée à la Section 3.4.1.

Les autres procédures de l'algorithme sont détaillées dans le cadre du problème de districting dans les sous-sections qui suivent.

**Algorithme 9:** Pseudo-code de l'algorithme implémenté.

---

```

population ← InitializePopulation(populationSize)
fronts ← FastNonDominatedSort(population)
while ¬StopCondition do
    selected ← SelectParentsByRank(population, populationSize)
    children ← CrossoverAndMutation(selected)
    population ← population ∪ children
    population ← population ∪ LocalSearch(fronts[0])
    fronts ← FastNonDominatedSort(population)
    for  $f_i \in$  fronts do
        | ComputeCrowdingDistance ( $f_i$ )
    end
    population ← KeepBests(population, populationSize)
end
return NonDominated(population)

```

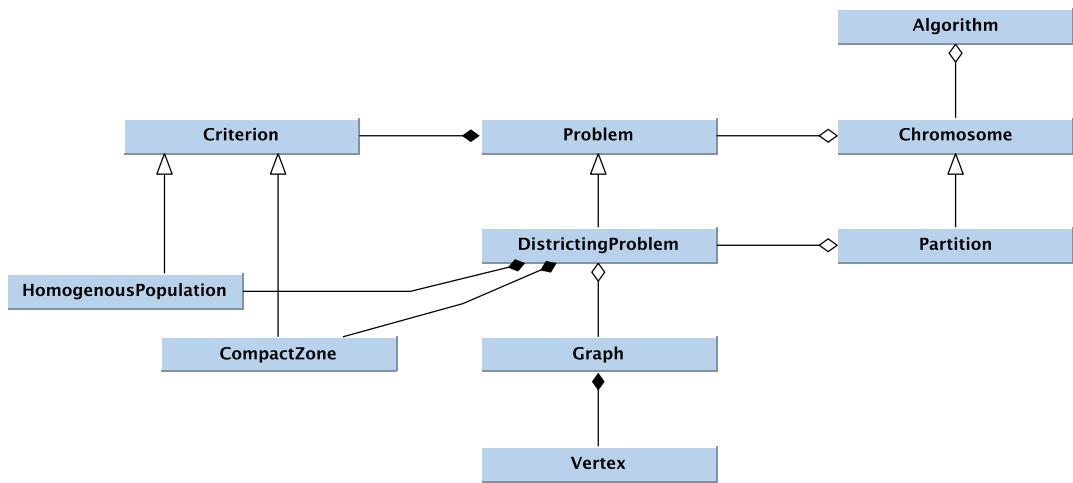
---

Plus tôt dans cette section, nous évoquions l'accent mis sur la générnicité de l'algorithme. Comme il est possible de le voir dans le pseudo-code de l'algorithme, rien ne fait mention d'une spécificité liée au problème de districting. En effet, l'implémentation a été divisée en deux parties bien distinctes. Dans un premier temps, l'algorithme hybride résultant de la combinaison de NSGA-II et d'un appel à une procédure de recherche locale a été développé. Celui-ci est complètement indépendant du problème étudié et peut très facilement être utilisé pour d'autres fins que le problème de districting. Dans un second temps, les différents opérateurs ainsi que la représentation d'une solution qui sont propres au problème de districting ont été implémentés.

La Figure 6.6 est un diagramme UML simplifié du programme et permet de mieux comprendre cette séparation existante entre l'algorithme et un problème d'optimisation particulier, le problème de districting dans notre cas.

Sur cette figure, on peut voir que l'**Algorithme** utilise des objets de type **Chromosome**. Ceux-ci sont des objets présentant des méthodes comme **Mutate**, provoquant la mutation du chromosome, **Crossover**, permettant de combiner deux chromosomes et **Neighbours**, permettant de retrouver tous les chromosomes voisins au chromosome considéré. Ces méthodes de **Chromosome** sont vides par défaut et doivent être redéfinies pour chaque problème étudié. C'est d'ailleurs le rôle de l'objet de type **Partition**, qui ré-implémente ces méthodes pour le problème de districting. Chaque chromosome a accès à un objet de type **Problem** qui modélise le problème étudié et contient toutes les informations le concernant, en particulier, les différents critères et les fonctions d'objectif du problème considéré. Dans le cas du problème de districting, l'objet **Problem** est raffiné en un objet **DistrictingProblem**. Ce dernier contient le graphe représentant le territoire étudié ainsi que les informations sur chacun des éléments territoriaux et leurs connexions. Il contient également des implémentations de la classe **Criterion** qui permettent d'évaluer une solution.

Ce qu'il est important de retenir de cette explication et de cette figure est que l'algorithme est indépendant du problème étudié. Celui-ci fait appel à des opérateurs génétiques et manipule des solutions sans avoir conscience de leur signification. Ceci offre un avantage conséquent, une très grande flexibilité. Il est par exemple possible



**Figure 6.6** – Cette figure est un diagramme UML simplifié du programme implémenté. Il vise à illustrer la séparation entre l'algorithme, gardé très générique et la modélisation du problème de districting.

d'ajouter une nouvelle représentation d'une partition. Il suffit alors de redéfinir les trois opérations : crossover, mutation et voisins pour cette nouvelle représentation et l'algorithme peut directement fonctionner avec cette dernière. Nous reviendrons sur cette flexibilité par la suite.

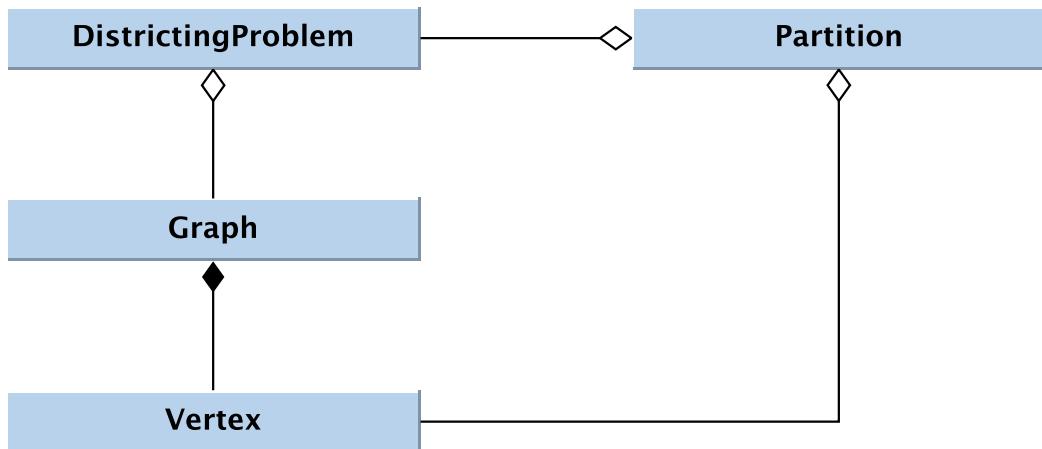
Dans les sous-sections suivantes, nous allons aborder les différentes opérations et étapes de l'algorithme qui sont propres au problème étudié, c'est-à-dire le problème de districting. Nous ferons régulièrement référence au diagramme UML présenté en Figure 6.6 afin de permettre une meilleure compréhension de la séparation entre la modélisation du problème étudié d'une part et l'algorithme générique d'autre part.

### 6.3.1 Représentation des solutions

La représentation des solutions que nous utiliserons est celle proposée par les auteurs de l'article [112]. Nous l'avions présentée à la Section 5.3.1.

Chaque **Partition** contient la représentation d'une solution, c'est-à-dire une liste de listes de noeuds du graphe. En effet, pour rappel, la représentation d'un partitionnement est une liste des zones qui composent ce dernier. Chacune de ces zones est également une liste des éléments territoriaux la composant. Et comme les éléments territoriaux sont des noeuds du graphe de contiguïté, un partitionnement est bien représenté par une liste de liste de noeuds du graphe.

La Figure 6.7 permet de contextualiser cette représentation par rapport au diagramme de classe que nous avions présenté à la Figure 6.6. Une **Partition** a accès à l'instance du problème. Cette instance contient le graphe de contiguïté tel que nous l'avons défini au chapitre précédent en Section 5.2 et en particulier les noeuds de ce graphe. Chaque objet **Partition** contient donc une liste de listes d'objets de type **Vertex**. Ceci est illustré à Figure 6.7.



**Figure 6.7** – Cette figure montre que la représentation d'une solution fait partie de l'objet *Partition* qui contient une liste de listes d'objets de type *Vertex*.

### 6.3.2 Population initiale

Pour générer une nouvelle solution afin de placer celle-ci dans la population initiale de l'algorithme, nous utilisons la technique expliquée au chapitre précédent à la Section 5.3.6. Celle-ci utilise la procédure de fusion que nous avions décrite à la Section 5.3.5 afin de fusionner deux à deux des zones de la solution triviale jusqu'à atteindre le nombre de zones souhaité.

### 6.3.3 Crossover et mutation

Après la sélection de solutions dans la population grâce à la technique du tournoi binaire, les opérateurs génétiques leur sont appliqués.

Le fonctionnement de ces opérateurs génétiques est le même que celui présenté au chapitre précédent. Pour le fonctionnement de l'opérateur de crossover, se référer à la Section 5.3.3, pour l'opérateur de mutation, se référer à la Section 5.3.4.

L'Algorithme 10 présente le pseudo-code de la génération de nouvelles solutions à partir d'un pool de parents. On peut remarquer un détail inhabituel pour un algorithme génétique classique. En effet, les solutions générées à partir de l'opérateur de crossover sont ajoutées à la population avant de leur appliquer l'opérateur de mutation. Généralement, lorsqu'une solution est générée, soit une mutation lui est appliquée et la solution mutante est ajoutée à la population, soit aucune mutation n'est appliquée et la solution est directement ajoutée à la population. Dans notre cas, une solution peut être ajoutée à la population, puis mutée et le mutant est également ajouté à la population. Ceci est un choix qu'ont fait les auteurs de l'article que nous avons présenté au chapitre précédent et qui a été gardé. Toutefois, comme nous le verrons à la Section 6.3.6, il est possible d'exécuter l'algorithme en mode *classique* c'est-à-dire de façon à ce que seule la solution résultant du crossover ou la solution résultant de la mutation soit ajoutée à la population et non les deux.

Une autre remarque importante par rapport au pseudo-code 10 est que ce dernier est complètement indépendant du problème étudié et de la représentation des solutions. L'algorithme fait simplement appel aux méthodes `Crossover`, `Mutate`, `IsValid` qui doivent être redéfinies pour le problème considéré. Dans le cas du problème de districting, les méthodes `Crossover` et `Mutate` sont celles que nous avons présentées. La méthode `IsValid` vérifie simplement que la solution  $x$  est bien composée de  $k$  zones.

En effet, nous avions vu au chapitre précédent, lors de l'analyse critique de l'opérateur de crossover en Section 5.3.3, que celui-ci pouvait générer des solutions contenant un nombre de zones inférieur à celui escompté. Ces solutions doivent donc être rejetées.

L'opérateur de mutation ne souffrait pas d'un tel problème et génère toujours des solutions valides. Toutefois, le test a été ajouté pour les solutions produites par cet opérateur car, comme nous le verrons par la suite, d'autres représentations des solutions du problème de districting, utilisant un opérateur de mutation différent, pourraient produire des solutions non valides qu'il faut alors exclure.

---

**Algorithme 10:** Pseudo-code de la génération de nouvelles solutions à partir des solution sélectionnées dans la population par tournoi binaire.

---

```

input : population, parents
output : children
children  $\leftarrow \emptyset$ 
for  $i = 1$  to populationSize do
    dad  $\leftarrow$  Sample(parents)
    mom  $\leftarrow$  Sample(parents)
    son  $\leftarrow$  Crossover(dad, mom)
    if IsValid(son) then
        children  $\leftarrow$  children  $\cup \{son\}$ 
        if Random() < mutationProbability then
            mutant  $\leftarrow$  Mutate(son)
            if IsValid(mutant) then
                children  $\leftarrow$  children  $\cup \{mutant\}$ 
            end
        end
    end
end
return children

```

---

### 6.3.4 Recherche locale

Au chapitre précédent, nous avons défini la notion de solution voisine. Pour rappel, une solution  $n$  est dite voisine d'une solution  $x$  s'il est possible de passer de la solution  $x$  à la solution  $n$  en ne déplaçant qu'un élément d'une des zones vers une zone adjacente.

La procédure de recherche locale est appliquée sur les solutions non-dominées de la population. C'est-à-dire, les solutions résidant dans le premier front. A chaque itération, et pour chacune des solutions du premier front, toutes les solutions voisines sont calculées, et celles dominant la solution dont elles sont voisines sont ajoutées à la population.

L’Algorithme 11 contient le pseudo-code de cette procédure qui est exécutée après l’application des opérateurs génétiques.

La procédure de recherche locale est une opération très coûteuse. En effet, il est important de réaliser que pour calculer les solutions voisines d’une solution, le déplacement de chacun des éléments territoriaux à chacune de ses zones adjacentes est effectué. Ensuite un test est appliqué sur chacune des solutions générées pour vérifier que la contrainte de contiguïté des zones n’a pas été brisée durant la manipulation. Il n’est donc pas envisageable de réaliser cette opération sur chacune des solutions non-dominées à chaque itération de l’algorithme. Une borne maximum au nombre de solutions traitées à chaque itération est donc présente dans l’algorithme, comme le montre l’Algorithme 11, afin de limiter l’application de la recherche locale.

---

**Algorithme 11:** Pseudo-code de l’application de la recherche locale sur les solutions non-dominées de la population.

---

```

 $i \leftarrow 0$ 
for  $nd \in \text{fronts}[0]$  do
    if  $nd$  not marked as searched then
        for  $n \in \text{Neighbours}(nd)$  do
            if  $n$  dominates  $nd$  then
                | population  $\leftarrow$  population  $\cup \{n\}$ 
            end
        end
        mark  $nd$  as searched
         $i \leftarrow i + 1$ 
    end
    if  $i \geq \text{maximumIteration}$  then
        | break
    end
end

```

---

### 6.3.5 Réduction de la population

Après l’application des opérateurs génétiques et de la procédure de recherche locale, la taille de la population a grandi. Il est donc nécessaire de la réduire avant de passer à l’itération suivante.

Lorsque la taille de la population est plus grande que le paramètre `PopulationSize` fourni à l’algorithme, NSGA-II propose de garder les meilleures solutions. En d’autres termes, de garder les solutions appartenant au premiers fronts. Pour rappel, il est possible de comparer des solutions au sein d’un même front en regardant leur *crowding distance*.

Cependant, lorsque le nombre de critères d’un problème d’optimisation augmente, le nombre de solutions non-dominées, et donc présentes dans le premier front, augmente également. Il est donc possible que le nombre de solutions non-dominées soit plus grand que le paramètre `PopulationSize` et dans ce cas, l’algorithme perd des solutions.

Pour éviter ce problème, il est donc possible de modifier l'algorithme pour réduire la taille de la population à `PopulationSize` tout en le forçant à garder l'intégralité du premier front. En procédant de la sorte, la taille de la population restera plus grande que désiré mais aucune solution ne sera perdue. Le choix d'opter pour cette méthode se résume à la spécification d'un paramètre de l'algorithme comme nous le verrons dans la section suivante.

### 6.3.6 Paramétrisation

Nous insistons dans la première partie de cette section sur l'importance donnée à la générnicité de l'algorithme. Afin de permettre cette dernière, un accent a été mis sur sa paramétrisation.

Pour le lecteur soucieux, la paramétrisation se définit comme suit [18] :

“(Informatique) Action de modifier la structure d’un élément numérique (document, texte, image, formule mathématique ou logiciel) pour lui ajouter des degrés de liberté supplémentaires sous forme de paramètres, ce qui lui donne une générnicité accrue.”

Nous allons maintenant passer en revue les différents paramètres permettant de configurer l'algorithme.

**Problème** Comme nous l'avons vu sur le diagramme UML à la Figure 6.6, l'algorithme nécessite un objet de classe `Problem` pour fonctionner. Toutes les solutions générées au cours de l'exécution de l'algorithme y auront accès. Cette classe contiendra toutes les informations nécessaires pour le problème d'optimisation étudié, par exemple, les différents critères à optimiser.

**Représentation** Le nom de la classe à utiliser pour représenter une solution. A chaque instanciation, la classe recevra l'instance du problème que nous avons décrit ci-dessus. C'est également cette classe qui implémente les différents opérateurs génétiques, `Crossover`, `Mutation` et également la procédure `Neighbours` utilisée dans la recherche locale. Fournir ceci comme paramètre permet de très facilement changer de représentation interne des solutions ou d'opérateurs génétiques.

**Nombre de génération** Le nombre de générations que doit générer l'algorithme. Ceci est utilisé comme condition d'arrêt.

**Taille de la population** La taille de la population, est également un paramètre important pour un algorithme évolutionniste.

**Archive** L'archive est un paramètre booléen, spécifiant si une archive doit être utilisée ou non. Comme nous l'avions expliqué dans la Section 6.3.5, lorsque le nombre de solutions dans le premier front est supérieur au paramètre `Taille de la population`, des solutions non dominées sont perdues lors de la réduction de la taille de la population à la fin de chaque itération. Toutefois, si le paramètre `Archive` est activé, l'algorithme gardera toujours l'intégralité du premier front de la population même si sa taille est supérieure à `Taille de la population`.

**Probabilité de mutation** Il s'agit de la probabilité d'appliquer une mutation sur une solution résultante de l'opérateur de crossover afin d'ajouter la solution mutante à la population.

**Probabilité de muter une zone** Lorsqu'une mutation est appliquée sur un partitionnement, chaque zone le composant a, à son tour, une probabilité d'être divisée comme nous l'avons expliqué à la Section 5.3.4. Remarque : Ce paramètre étant spécifique au problème de partitionnement, il ne fait pas partie de l'**Algorithme** mais se trouve dans le **Problème**.

**Ajout avant mutation** Ce paramètre est également un paramètre booléen. Comme nous l'avions expliqué en Section 6.3.3, si ce paramètre est *vrai*, chaque solution résultante de l'opérateur de crossover sera toujours ajoutée à la population et si une mutation lui est appliquée, la solution mutante sera également ajoutée. Par contre, si ce paramètre est *faux*, un fonctionnement plus classique est utilisé. Si aucune mutation n'est appliquée, la solution résultante du crossover est ajoutée à la population, par contre, si une mutation est appliquée, seule la solution mutante résultant de l'opération de mutation sera ajoutée à la population.

**Recherche locale** Ce paramètre est un simple paramètre booléen permettant de spécifier si l'opération de recherche locale doit être utilisée ou non. Ce paramètre permet donc d'activer ou non l'hybridation de l'algorithme.

**Limite de recherche locale** Comme expliqué à la Section 6.3.4, ce paramètre permet de limiter l'application de la recherche locale à un certain nombre de solutions à chaque itération de l'algorithme.

Dans la Section 6.4.4, nous aborderons le paramétrage de l'algorithme. En d'autres termes, nous trouverons pour chacun des paramètres présentés ci-dessus, les valeurs optimales produisant les meilleurs résultats au regard des critères étudiés et des données utilisées.

### 6.3.7 Tests unitaires

Afin de garantir un fonctionnement correct de l'algorithme, de nombreux tests unitaires ont été implémentés. Ceux-ci permettent de s'assurer du bon fonctionnement de l'algorithme en général mais surtout de la fiabilité des différents opérateurs propres à la représentation des solutions choisie.

Le principal avantage des tests unitaires est que ceux-ci peuvent être lancés à tout moment, en particulier après chaque changement effectué au code source du programme, afin de s'assurer que l'algorithme a toujours le comportement souhaité. Ceci procure une confiance dans le code source et permet de travailler plus efficacement. Pour réaliser cette tache, nous avons utilisé le framework de test faisant partie de la bibliothèque standard de Ruby [15].

Les deux paragraphes suivant reprennent à titre d'illustration quelques exemples de tests unitaires implémentés.

#### Algorithme

- Lors du tri de la population dans les différents fronts, plusieurs tests unitaires vérifient qu'une solution se trouvant dans le front  $i$  domine bien toutes les solutions se trouvant dans le front  $i + 1$ . Un autre test s'assure que toutes les solutions de la population font bien partie d'un et un seul front.

- Un test unitaire s'assure que toutes les solutions générées à partir des opérateurs génétiques et de la recherche locale et qui sont ajoutées à la population à chaque itération sont bien des solutions valides.

**Partition** Les tests unitaires décrits ci-dessous sont propres à la représentation des solutions que nous avons choisie et décrite dans les sections précédentes.

- Un test unitaire vérifie qu'une solution générée aléatoirement, par exemple pour être ajoutée à la population initiale, est toujours valide.
- Un test vérifie qu'une solution mutée reste toujours valide. Bien que ceci ne soit pas vrai dans l'absolu, c'est-à-dire pour une représentation lambda, c'est bien le cas pour la représentation que nous avons choisie et la façon dont fonctionne l'opérateur de mutation implémenté.
- Après l'application de l'opérateur de crossover, nous avons vu qu'une solution peut être constituée de trop peu de zones. Toutefois, un test vérifie que, si une solution générée lors de la procédure de crossover contient le nombre escompté de zones, alors, elle répond également à d'autres critères : chaque élément territorial fait partie d'une et une seule zone et chaque fonction d'objectif a bien une valeur.
- Un test vérifie que toutes les solutions générées à partie de la procédure `Neighbours` sont bien valides.
- Un autre test vérifie que toutes les solutions provenant de l'opération `Neighbours` sont bien des solutions voisines à la solution de départ. C'est-à-dire qu'elles ne diffèrent que par un élément territorial ayant changé de zone.
- Un test vérifie également le bon fonctionnement du test de dominance. En d'autres termes, si une solution  $x$  domine une solution  $y$ , alors  $y$  ne peut dominer  $x$ .

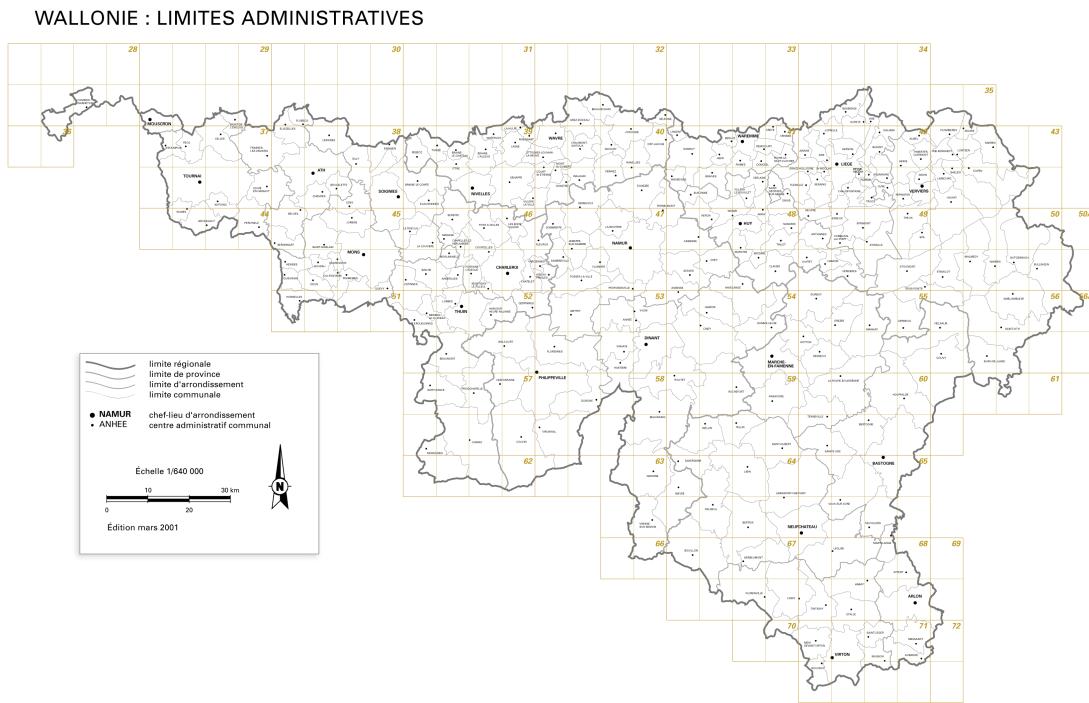
## 6.4 Application de l'algorithme

Au travers des sections précédentes, nous avons présenté l'algorithme de manière générale. Dans cette section, nous allons nous intéresser à son utilisation pour un cas bien précis, celui de la Wallonie. Dans un premier temps, nous allons aborder le problème de la visualisation des résultats. Nous verrons ensuite les données à notre disposition et les différents critères que nous pouvons utiliser. Enfin, nous aborderons le paramétrage de l'algorithme et les résultats.

### 6.4.1 Visualisation

Dès l'implémentation des premières versions de l'algorithme, il a été nécessaire de pouvoir visualiser les partitionnements produits. Il était donc nécessaire de pouvoir placer sur une carte chacune des 262 communes de la Wallonie afin de pouvoir les colorier. De plus, pour générer le graphe de contiguïté nécessaire à l'algorithme, il était nécessaire de connaître les communes adjacentes à chacune des communes de Wallonie. Pour rendre cette tâche plus simple, différents outils ont été développés.

La première étape nécessitait de pouvoir placer sur une carte chaque commune de Wallonie. Afin de produire des images de qualité, une image de taille suffisante était nécessaire. L'image en Figure 6.8, provenant du portail de la Wallonie [13], a donc été choisie.



**Figure 6.8** – Cette carte, provenant du portail de la Wallonie, a été utilisée dans les différents outils dans le but de pouvoir être utilisée pour représenter les partitions générées par l'algorithme.

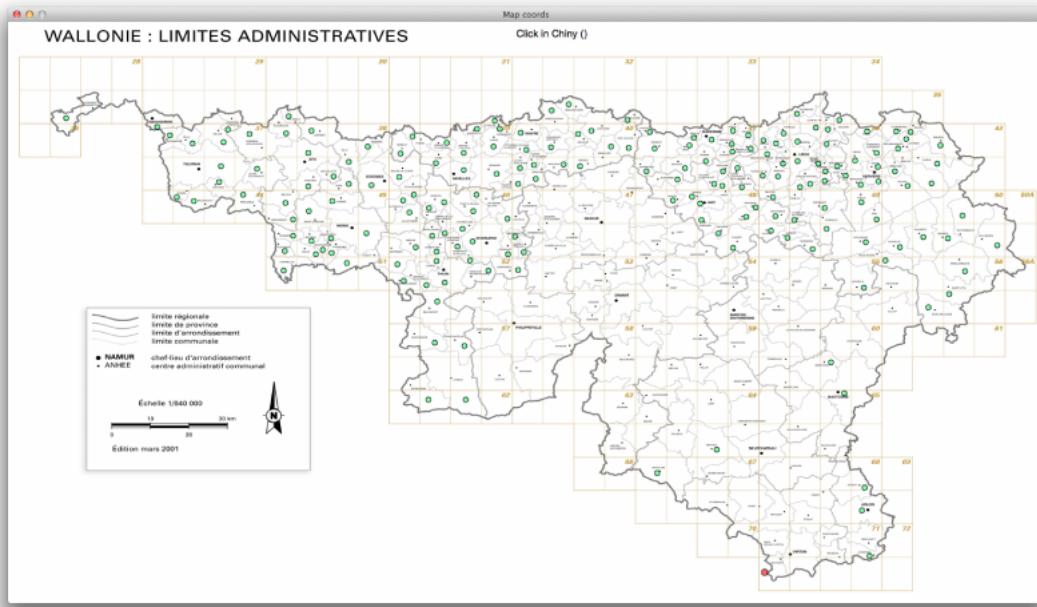
Le premier outil développé propose donc une interface très simple afin d'enregistrer un emplacement sur l'image pour chacune des différentes communes. La Figure 6.9 illustre l'enregistrement sur la carte des communes.

La seconde étape nécessaire était de connaître l'adjacence des différentes communes. Un second outil, illustré en Figure 6.10, a donc été développé afin de faciliter l'enregistrement des communes adjacentes.

Enfin, il était nécessaire de pouvoir colorier la carte afin de pouvoir visualiser les partitionnements générés par l'algorithme. Cependant, la carte utilisée jusqu'ici ne se prête pas à cette tâche puisqu'elle contient de nombreuses annotations ainsi qu'une grille couvrant la carte et rendant les procédures de remplissage inefficaces. Un considérable travail de retouche d'image a donc été réalisé sur l'image de base afin de la rendre utilisable. La Figure 6.11 illustre les retouches effectuées.

La carte utilisée est une image matricielle, *bitmap* en anglais. Les images matricielles souffrent d'un défaut important, lorsqu'elles sont trop agrandies, on peut voir apparaître les pixels qui la composent. Une façon de contourner ce problème est de se tourner vers des images vectorielles qu'il est possible d'agrandir à souhait. Les cartes vectorielles sont donc souvent préférables à leur équivalent matriciel puisqu'elles s'adaptent à n'importe quelle échelle et peuvent être imprimées en haute qualité.

Toutefois, ces cartes de très haute qualité sont rarement disponibles librement sur internet. La société Comersis [3] propose par exemple une carte vectorielle des communes de Belgique au prix de 89.90€. Néanmoins, à partir de la carte matricielle que nous avons



**Figure 6.9** – Cette figure illustre le premier outil dont l’objectif est d’enregistrer un pixel sur la carte pour chaque commune. Les petites sphères vertes montrent les communes déjà enregistrées. Le programme affiche le nom d’une commune et l’utilisateur doit simplement cliquer sur la carte afin d’enregistrer la position de la commune.

présentée précédemment et grâce à des outils de vectorisation [17], une image vectorielle des communes de Wallonie a pu être produite. La différence entre la version matricielle et vectorielle est illustrée à la Figure 6.12. Nous utiliserons cette image vectorielle par la suite pour représenter les partitionnements générés par l’algorithme.

#### 6.4.2 Données

Avant d’aborder les différents critères utilisés pour générer de nouveaux partitionnements, nous allons aborder dans cette section les différentes données disponibles pour le problème.

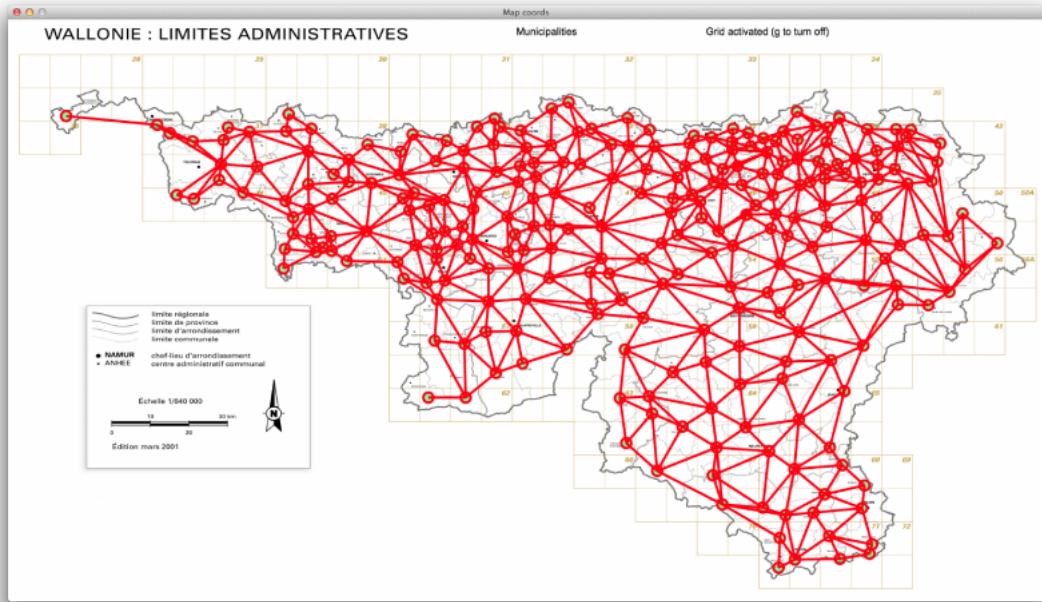
Les données récoltées pour chacune des 262 communes de Wallonie sont les suivantes :

**Code INS** Le code INS de la commune. Cette donnée ne sera pas utilisée mais est toutefois présente par souci de compatibilité.

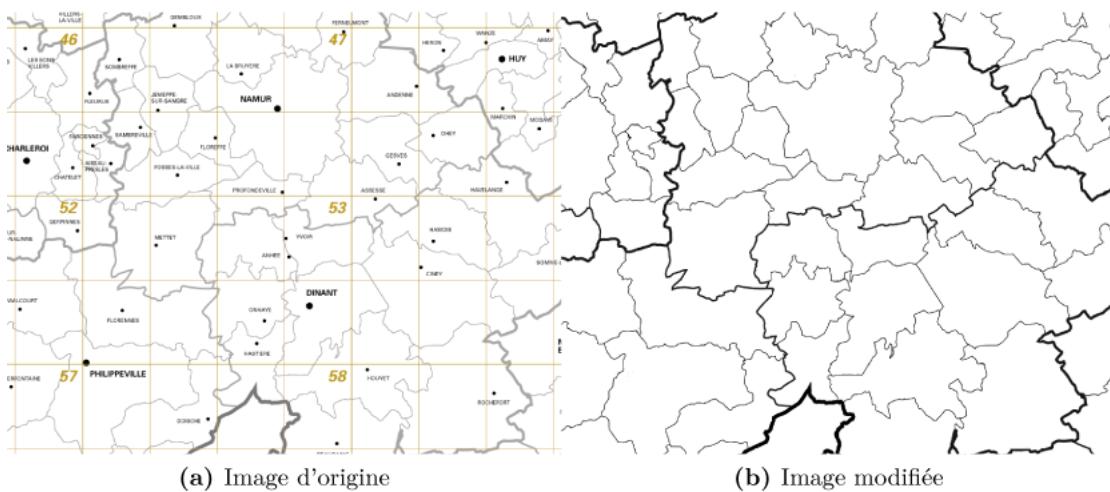
**Revenus** Le montant moyen des revenus totaux (transferts, patrimoine, travail) par habitant en 2001 (en €) [4].

**Province** Le nom de la province à laquelle la commune appartient dans le partitionnement actuel de la Wallonie tel que nous l’avons vu à la Figure 6.2.

**Arrondissement** Nous avons déjà mentionné précédemment que la Wallonie était divisée en cinq provinces. Celles-ci sont, elles-mêmes, divisées en arrondissements. La Wallonie en totalise 20. Ce champ contient donc le nom de l’arrondissement auquel appartient la commune.



**Figure 6.10** – Cette figure illustre le second outil dont l'objectif est d'enregistrer l'adjacence des différentes communes. L'outil passe en revue chacune des communes de Wallonie et invite l'utilisateur à cliquer sur les communes qu'il considère comme adjacentes. La figure montre également la grille permettant de mieux visualiser les connexions réalisées.



**Figure 6.11** – Afin de pouvoir automatiquement colorier les différentes communes, il a été nécessaire de *nettoyer* la carte d'origine visible en (a). Ces retouches ont été effectuées avec le logiciel Photoshop et consistent à supprimer la grille jaune, les chiffres ainsi que le nom des communes puis à augmenter le contraste. Après la suppression de la grille, le contour de chaque commune a dû être vérifié puisque celui-ci devait être clos. Le résultat est visible à l'image (b).



**Figure 6.12** – La figure (a) est une partie de l'image matricielle que nous avons présentée précédemment. L'image (b) correspond à la même partie de l'image dans sa version vectorielle.

**Code postal** Le code postal de la commune en Belgique.

**Latitude et longitude** Les coordonnées du centre de chaque commune. Ceci permet par exemple de mesurer la distance entre deux communes et peut être utilisé dans de nombreux critères. Pour récupérer les coordonnées géographiques de chaque commune, un script a été implémenté. Celui-ci utilise le plugin Ruby Geocoder [96] pour réaliser cette tâche.

**Population** Le nombre d'habitants de la commune en 2010 [106].

**Surface** La superficie en hectare de la commune.

**Densité** Grâce au nombre d'habitants et à la superficie de chaque commune, il est possible de calculer la densité de population.

**Valeur ajoutée** Grâce à M. Gilles Van Hamme et Mme Éléonore Wolff, chercheurs à L’Institut de Gestion de l’Environnement et d’Aménagement du Territoire (IGEAT), département de la Faculté des Sciences de l’Université Libre de Bruxelles, le pourcentage de la valeur ajoutée produite par différents secteurs a pu être récupéré pour chaque commune. Parmi les différents secteurs, on peut par exemple citer : le secteur de l’agriculture, de l’industrie agricole, de la construction, de l’HORECA, des transports et de l’éducation.

#### 6.4.3 Critères

De très nombreux critères peuvent être imaginés pour les différentes données disponibles. Dans cette section, nous allons définir et formaliser ceux que nous utiliserons avec l’algorithme présenté à la Section 6.3. Les critères que nous allons utiliser ont déjà été présentés à la Section 5.2.2. Cette section est donc un rappel au niveau du formalisme et vise surtout à placer les différents critères dans le contexte de la Wallonie.

**Zones compactes** Au chapitre précédent, nous avons introduit le critère de compactage d’un partitionnement et sa fonction d’évaluation proposée par les auteurs de l’article. Nous avons également fait remarquer que celle-ci pouvait souffrir d’un problème puisque seule la zone la moins compacte était prise en compte dans le calcul de la fonction d’objectif.

La Figure 6.13 illustre deux partitions générées en utilisant cette fonction d’objectif, c'est-à-dire, pour rappel :

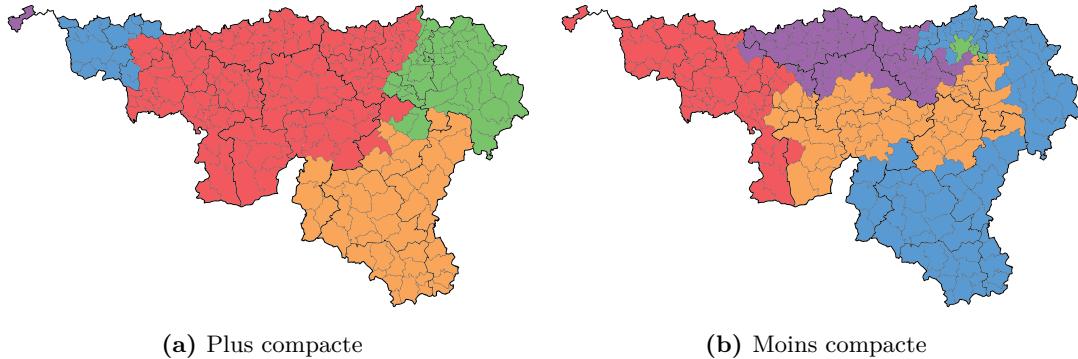
$$f_{compact}(Y) = \max_{y_u \in Y} C(y_u) \quad (6.1)$$

$$C(y_u) = \frac{\pi r^2}{\sum_{i \in y_u} surface_i} \quad (6.2)$$

où  $r$  est le rayon du plus petit cercle contenant l’intégralité de la zone. L’algorithme visera à minimiser la valeur de cette fonction d’objectif pour générer des zones compactes. Un second critère a également été utilisé pour cette figure. Ce dernier vise à créer des zones homogènes entre elles au niveau du nombre de communes les composant.

Une autre façon de calculer la fonction d’objectif liée au critère de compactage est présentée dans l’article [21]. Elle se calcule comme suit :

$$f_{compact}(Y) = \sum_{y_u \in Y} \sum_{i \in y_u} distance(c_{y_u}, i) \quad (6.3)$$



**Figure 6.13** – L’illustration (a) est une solution non-dominée lorsque le critère de compactage a été minimisé, c’est-à-dire, les partitions les plus compactes possibles ont été générées. L’illustration (b) correspond au même problème mais où le critère a été maximiser, il s’agit donc d’une partition non compacte. Les valeurs de l’objectif pour ces deux partitions sont 0.62 et 66.9.

où  $c_{y_u}$  correspond à la latitude et longitude du centre géographique de la zone  $y_u$  et  $distance(c_{y_u}, i)$  est la distance en km entre la commune  $i$  et le centre de la zone à laquelle elle appartient. L’avantage de cette méthode est qu’elle tient compte de chaque zone du partitionnement, c’est pourquoi nous avons choisi d’utiliser cette méthode dans l’algorithme. La Figure 6.14 est semblable à la Figure 6.13 mais en utilisant cette nouvelle fonction d’objectif.

**Homogénéisation intra-zone** La fonction d’objectif pour le critère d’homogénéisation intra-zone, que nous avons introduit à la Section 5.2.2, se calcule, par exemple pour le revenu, comme suit :

$$f_{revenu}(Y) = \sum_{y_u \in Y} H(y_u) \quad (6.4)$$

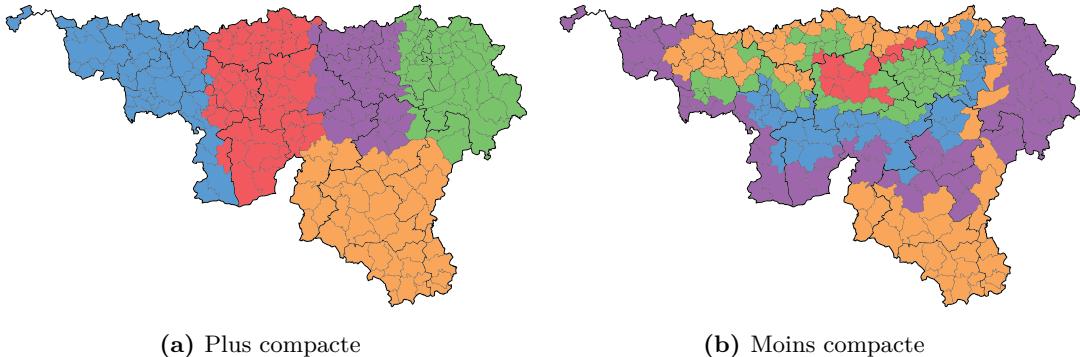
$$H(y_u) = \sum_{i \in y_u} |revenu_i - \mu| \quad (6.5)$$

$$\mu = \frac{\sum_{i \in y_u} revenu_i}{\#y_u} \quad (6.6)$$

L’algorithme visera à minimiser la valeur de cette fonction d’objectif. Différents critères d’homogénéisation intra-zone peuvent être considérés avec les données disponibles. Par exemple l’homogénéisation du revenu, de la densité de population ou bien de la valeur ajoutée générée par un secteur en particulier, au sein de chaque zone.

**Homogénéisation inter-zone** Pour les critères d’homogénéisation inter-zone, la fonction d’objectif sera calculée comme suit :

$$f_{population}(Y) = \sum_{y_u \in Y} |S(y_u) - \mu| \quad (6.7)$$



**Figure 6.14** – La figure (a) correspond à une solution non-dominée lorsque l’objectif est minimisé et la figure (b) correspond à l’équivalent lorsque l’objectif est maximisé. Les valeurs de l’objectif pour ces deux partitions sont 5924 et 14573.

$$\mu = \frac{\sum_{y_u \in Y} S(y_u)}{k} \quad (6.8)$$

$$S(y_u) = \sum_{i \in y_u} population_i \quad (6.9)$$

L'algorithme visera à minimiser la valeur de cette fonction d'objectif. Avec les données présentées à la Section 6.4.2, plusieurs critères inter-zone peuvent être définis : générer des zones homogènes au niveau du nombre d'habitants, de la densité de population ou de la surface totale peut par exemple être pertinent.

#### 6.4.4 Paramétrage de l'algorithme

Dans le Chapitre 3, en Section 3.3.4 nous avons abordé le sujet de la mesure des performances d'une métaheuristique. En particulier, nous avions évoqué différents indicateurs permettant de mesurer la qualité des solutions.

Afin de déterminer le paramétrage optimal pour l'algorithme, nous allons utiliser ces indicateurs. Nous utiliserons en particulier l'hypervolume qui, pour rappel, est un indicateur hybride combinant la mesure de la convergence et de la diversité du front Pareto trouvé par l'algorithme.

Pour calculer la valeur de l'hypervolume d'un front Pareto généré par l'algorithme, nous utiliserons l'outil *hv* [74] développé dans les laboratoires d'IRIDIA de l'ULB.

Nous allons dans un premier temps aborder le paramétrage de chaque paramètre de l'algorithme considéré individuellement. Nous parlerons ensuite d'une extension développée permettant le paramétrage tout en tenant compte de plusieurs paramètres simultanément.

## Mono-paramétrage

Comme expliqué précédemment, dans cette section, nous analyserons chaque paramètre de manière individuelle. Afin de déterminer la valeur de chaque paramètre permettant

Paramètre	Valeur
Problème	Le problème utilisé pour le paramétrage de l'algorithme est le problème du partitionnement de la Wallonie en 5 zones. Les données utilisées sont celles présentées à la Section 6.4.2.
Critères	Quatre critères ont été utilisés lors du paramétrage de l'algorithme : <ul style="list-style-type: none"> <li>– Générer des zones compactes.</li> <li>– Générer des zones homogènes entre elles au niveau du nombre d'habitants (inter-homogénéisation).</li> <li>– Générer des zones homogènes au niveau de la densité de population (intra-homogénéisation).</li> <li>– Générer des zones homogènes au niveau du pourcentage de valeur ajoutée produite par le secteur de l'agriculture (intra-homogénéisation).</li> </ul>
Chromosome	La représentation des solutions ainsi que le fonctionnement des opérateurs génétiques et de recherche locale sont ceux que nous avons présentés précédemment dans ce chapitre et qui sont basés sur la théorie des graphes et sur l'article [112].
Nombre de générations	200
Taille de la population	70
Archive	<i>false</i>
Probabilité de mutation	0.2
Probabilité de muter une zone	0.1
Ajout avant mutation	<i>false</i>
Recherche locale	<i>true</i>
Limite de recherche locale	5

**Table 6.1** – Cette table montre les valeurs de base utilisées pour chaque paramètre. Lorsque nous ferons varier la valeur d'un paramètre dans la suite de cette section, la valeur des autres paramètres sera celle reprise dans cette table.

le plus grand hypervolume après un nombre fixe de générations, nous ferons prendre à celui-ci plusieurs valeurs.

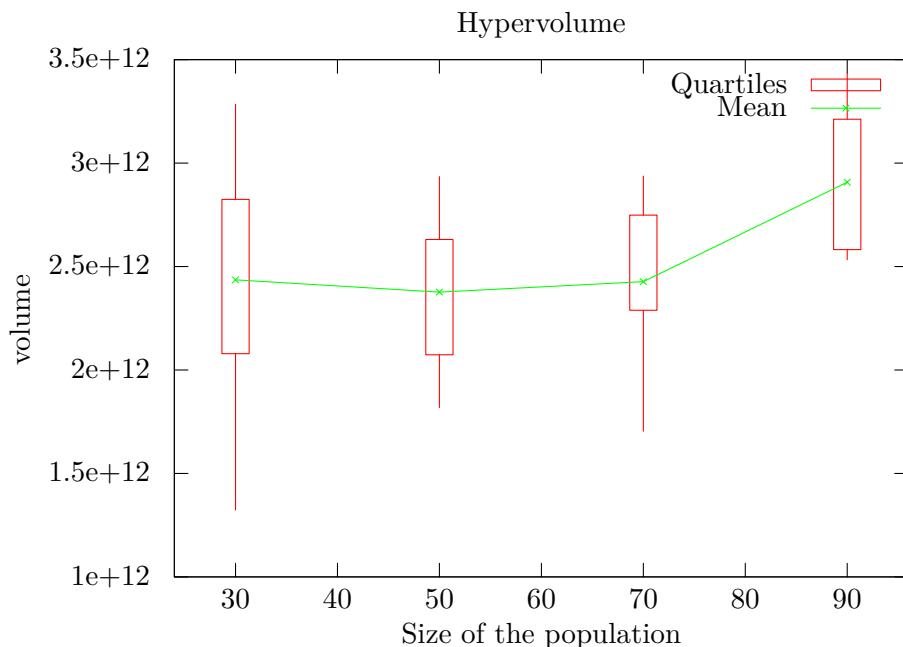
Pour chaque valeur d'un paramètre donné, l'algorithme sera lancé plusieurs fois afin d'avoir un hypervolume moyen mais également les valeurs minimum, maximum et les différents quartiles. Ceci est nécessaire étant donné la nature non déterministe de l'algorithme.

Les valeurs de base données au différents paramètres sont reprisent dans la Table 6.1. Pour rappel, la signification des différents paramètres a été introduite précédemment à la Section 6.3.6.

**Taille de la population** Dans un premier temps, nous allons nous intéresser à l'influence de la taille de la population sur les résultats de l'algorithme. Dans cette section, nous ferons donc varier la taille de la population tout en gardant la valeur des autres paramètres fixe. Pour chaque valeur de la taille de la population, l'algorithme est lancé

10 fois.

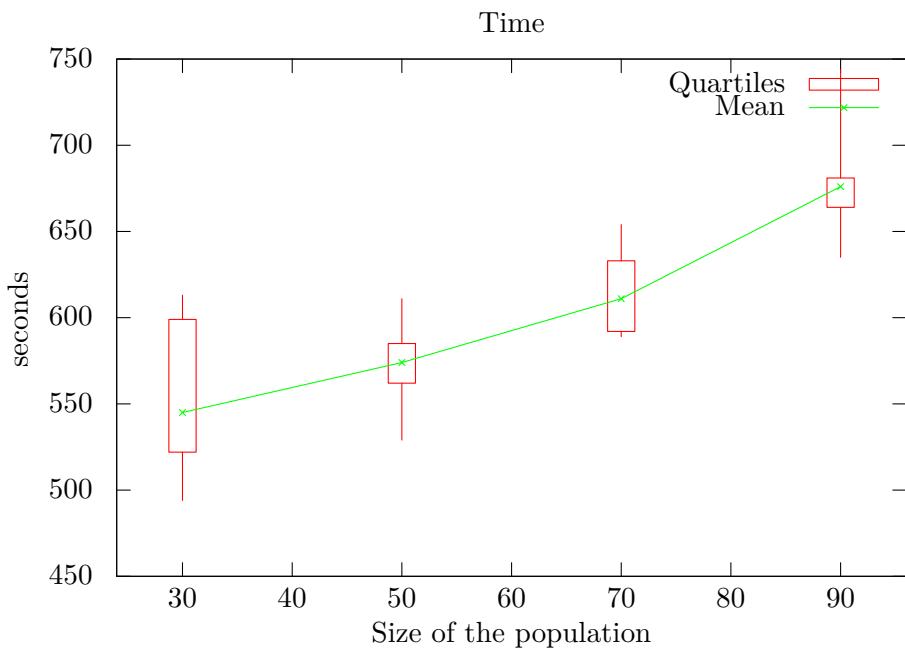
Il est évident que lorsque la condition d'arrêt de l'algorithme est le nombre de générations, plus la taille de la population est grande, plus l'hypervolume final sera élevé. Ceci est illustré à la Figure 6.15. Bien sûr, le temps total pris par l'algorithme sera également plus important, le nombre de solutions traitées et générées étant supérieur. Ceci est montré à la Figure 6.16.



**Figure 6.15** – Influence de la taille de la population sur l'hypervolume final. La condition d'arrêt de l'algorithme est le nombre de générations. Celui-ci est fixé ici à 200 générations.

Il est donc intéressant d'analyser l'influence de la taille de la population lorsque la condition d'arrêt de l'algorithme est un nombre de secondes fixe. Une population plus petite générera donc sur plus de générations comme le montre la Figure 6.17. Il est alors pertinent de trouver la valeur pour la taille de la population générant un hypervolume maximal en un temps donné, ce que permet la Figure 6.18. On peut y voir que, malgré que la Figure 6.15 nous suggérait qu'une population plus grande induisait un hypervolume final plus grand, le coût causé par la population de grande taille dépasse le bénéfice qu'apporte celle-ci et une population de 70 individus semble optimale.

**Probabilité de mutation** Nous allons ici nous intéresser à l'influence de la probabilité de mutation sur l'hypervolume final. Pour rappel, à chaque itération de l'algorithme, de nouvelles solutions sont générées grâce à la procédure de crossover. Chacune de ces solutions générées a alors une probabilité  $p$  d'être mutée avant d'être ajoutée à la population. La première chose qu'il est possible de constater est alors que plus la probabilité de mutation est importante, plus le temps d'exécution de l'algorithme sera grand (si la condition d'arrêt est un nombre fixé de générations). En effet, la procédure consistant à muter une solution générée avant de l'ajouter à un coût, plus sa fréquence



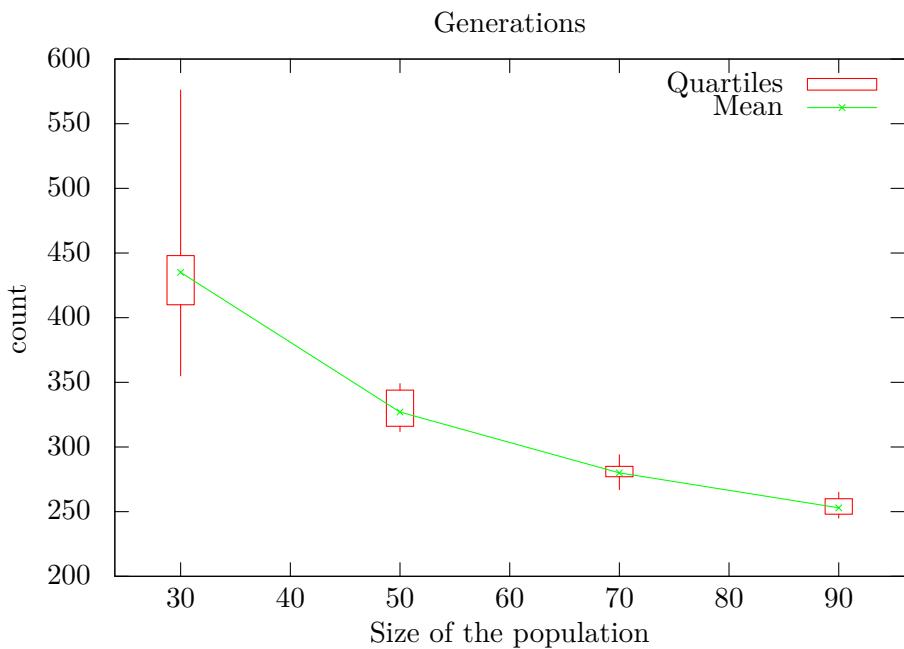
**Figure 6.16** – Influence de la taille de la population sur le temps pris par l’algorithme. La condition d’arrêt de celui-ci est le nombre de générations, fixé à 200. Ce graphique met bien en évidence que le traitement d’une population plus importante a un coût.

d’application est importante, plus le coût sera important. La Figure 6.19 reflète d’ailleurs ce résultat.

La Figure 6.20 montre l’hypervolume final pour différentes valeurs du paramètre. Cette figure suggère qu’une probabilité de mutation de 0.1 fournit les meilleurs résultats. L’effet est donc double si l’on considère le temps comme condition d’arrêt. En effet, le fait que la probabilité de 0.1 aie un temps total d’exécution plus petit s’additionne au fait qu’elle produit un hypervolume plus important. La Figure 6.21 montre le ratio hypervolume par seconde qui marque la différence de façon encore plus flagrante.

**Recherche locale** Nous allons ici nous intéresser à la procédure de recherche locale et à son influence sur les résultats obtenus. Cette section est particulièrement pertinente car cette procédure à un coût colossal qui justifie sa remise en question.

Les résultats ont ici été produits en activant et en désactivant la recherche locale. La condition d’arrêt de l’algorithme utilisée est le temps. Celui-ci a été fixé à 20 minutes. Le premier Graphique 6.22 permet de visualiser le coût de la procédure en montrant le nombre de générations produites en 20 minutes par l’algorithme avec et sans recherche locale. Si presque cinq fois moins de générations sont produites avec la recherche locale, l’hypervolume final est toutefois supérieur lorsque la procédure est utilisée comme le témoigne le Graphique 6.23. Il en résulte que l’utilisation de cette procédure est tout à fait pertinente comme l’avaient suggéré Tavares et al. [112].



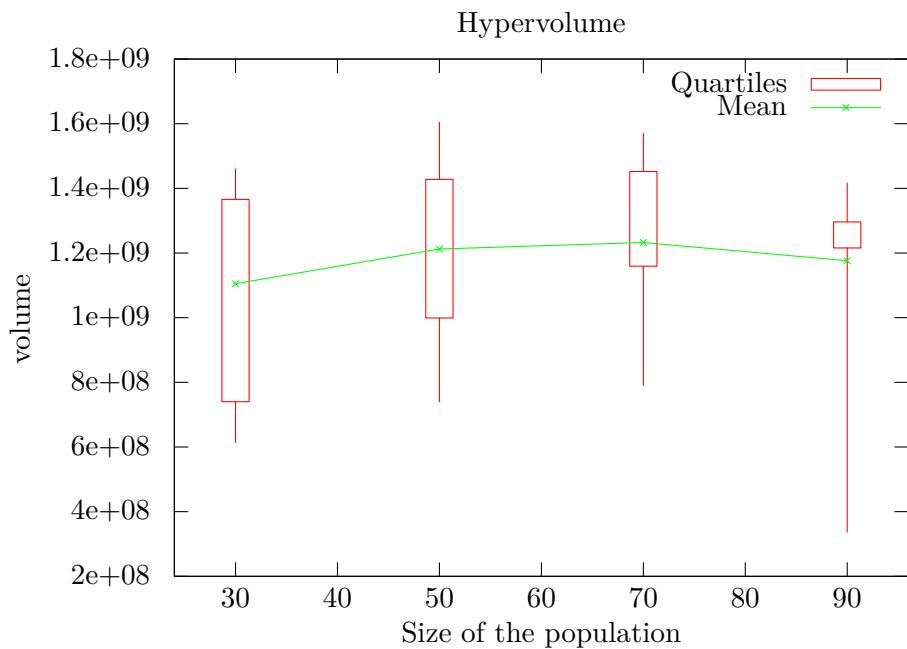
**Figure 6.17** – Sur ce graphique, on peut visualiser le coût engendré par une population plus grande. Pour chaque taille de population, l'algorithme a été lancé 10 fois pendant 12 minutes. On peut constater que plus la taille de la population est importante, moins le nombre de générations produites est volumineux.

**Nombre de générations** Puisque souvent le nombre de générations est utilisé comme condition d'arrêt de l'algorithme, il est pertinent de s'intéresser au gain d'hypervolume à chaque itération. Ceci permet de trouver le nombre de générations minimal nécessaire pour obtenir des résultats satisfaisants et permet également de vérifier la convergence de l'algorithme.

Lorsque la condition d'arrêt de l'algorithme est le nombre de générations, il est évident que l'analyse du temps mis par l'algorithme n'est pas pertinente puisque produire des générations supplémentaires nécessitera forcément un temps plus important. Il est également évident que laisser l'algorithme s'exécuter sur un nombre plus important de générations produira un hypervolume final plus important. L'analyse de l'hypervolume final n'est donc pas pertinente non plus.

Toutefois, l'analyse de l'évolution de l'hypervolume tout au long de l'exécution de l'algorithme est, elle, pertinente puisqu'elle permet de visualiser la convergence de celui-ci.

Nous allons, pour cette expérience, utiliser les résultats précédents. Par rapport aux valeurs reprises dans la Table 6.1, nous allons ici appliquer une probabilité de mutation de 0.1. La taille de la population sera conservée à 70 et la recherche locale sera toujours activée. La Figure 6.24 montre l'évolution de l'hypervolume sur 800 générations. Pour produire ce graphique, l'algorithme a été lancé 10 fois. La convergence de l'algorithme est bien visible sur le graphique et on peut voir que l'hypervolume se stabilise assez rapidement.



**Figure 6.18** – Ce graphique permet de visualiser que le coût induit par le traitement d'une population plus grande finit par dépasser le gain que celle-ci produit sur l'hypervolume finale. L'algorithme a été lancé dans les mêmes conditions que pour la Figure 6.17 c'est à dire 10 fois pendant 12 minutes pour chaque taille de population.

### Multi-paramétrage

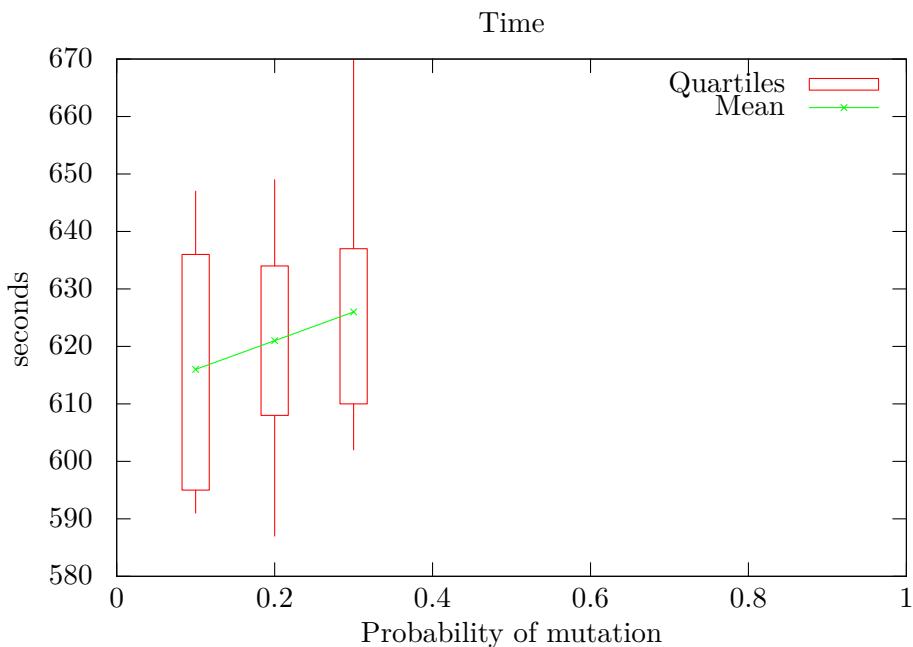
Le problème consistant à trouver la valeur de chaque paramètre, permettant une convergence rapide du front approché vers le front Pareto optimal ainsi qu'une bonne diversité des solutions le composant, est lui-même un problème d'optimisation. En particulier, on peut considérer le problème d'optimisation multicritère composé de deux fonctions d'objectifs. Le premier objectif est de maximiser l'hypervolume du front approché après l'exécution de l'algorithme. Le second objectif est de minimiser le temps d'exécution de l'algorithme.

Le second objectif permet de mesurer le coût induit par le choix de la valeur de certains paramètres. Il est évident que laisser tourner l'algorithme pendant 6 milliards d'itérations produira un hypervolume final plus important. Il est donc nécessaire d'ajouter ce second critère de mesure du temps afin d'en mesurer le coût.

Puisqu'un effort important a été fourni afin de garder l'algorithme générique, celui-ci est facilement adaptable pour traiter ce nouveau problème d'optimisation multicritère que nous venons de décrire. En effet, une solution de ce problème est l'association d'une valeur à chaque paramètre de l'algorithme. Les opérateurs de mutation, de crossover et de recherche locale peuvent alors facilement être implémentés :

**Mutation** La mutation d'une solution consiste à faire varier aléatoirement une ou plusieurs valeurs associées au paramètre de l'algorithme.

**Crossover** La procédure de crossover consiste alors à générer une nouvelle solution dont la valeur associée aux différents paramètres provient tantôt de la mère et



**Figure 6.19** – Lorsque le nombre de générations est fixé (200 dans ce cas ci), une probabilité plus importante d’appliquer la procédure de mutation sur une solution générée via la procédure de crossover avant son ajout à la population se reflétera par un temps d’exécution total plus important.

tantôt du père. De nombreuses techniques de crossover existent [76] et peuvent être envisagées.

**Recherche locale** Une solution  $x$  est considérée comme voisine d’une solution  $y$  si seule la valeur d’un paramètre a changé.

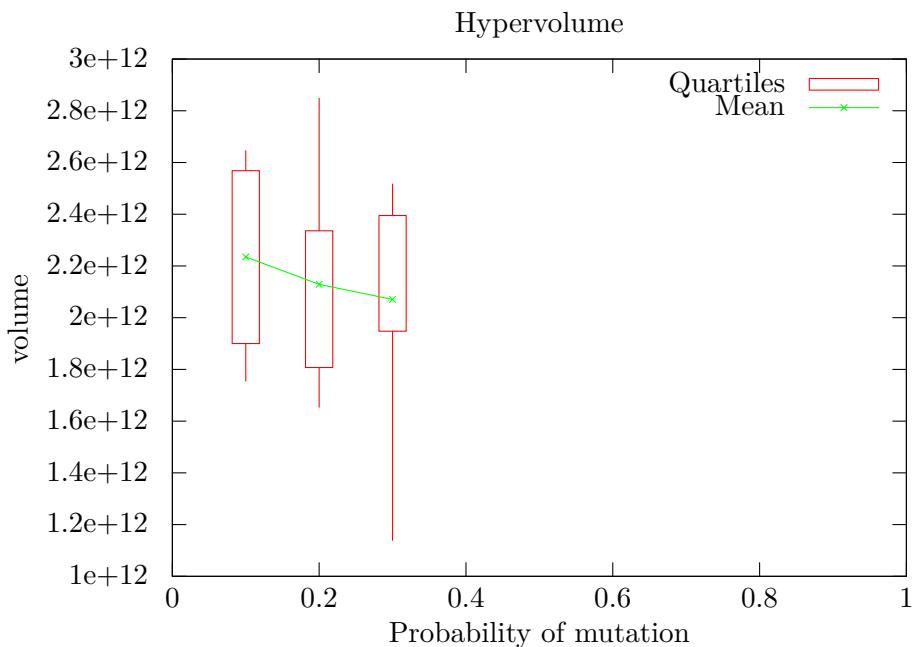
Il est alors possible d’exécuter l’algorithme sur ce nouveau problème d’optimisation afin de trouver des valeurs pour le paramétrage de l’algorithme dans le cadre du problème de districting.

## 6.5 Résultats

Dans cette section, nous allons présenter quelques résultats obtenus avec l’algorithme. Bien que nous pourrions considérer l’ensemble des critères simultanément, nous présenterons ici des résultats pour des couples de critères afin de pouvoir visualiser les fronts de solutions non-dominées obtenus dans le plan. Nous considérerons deux paires de critères.

### 6.5.1 Partitionnement électoral

Comme premier problème, nous nous intéresserons au partitionnement électoral. Pour rappel, celui-ci est composé du critère visant à générer des zones compactes ainsi que du critère ayant pour objectif l’homogénéisation de la population entre les différentes zones. La Table 6.2 reprend les différentes valeurs de paramètre utilisées. La Figure 6.25



**Figure 6.20** – Ce graphique montre l'hypervolume final obtenu avec différentes valeurs pour le paramètre de probabilité de mutation. Une probabilité de 0.1 semble fournir les meilleurs résultats. Le nombre de générations était ici utilisé comme condition d'arrêt et fixé à 200.

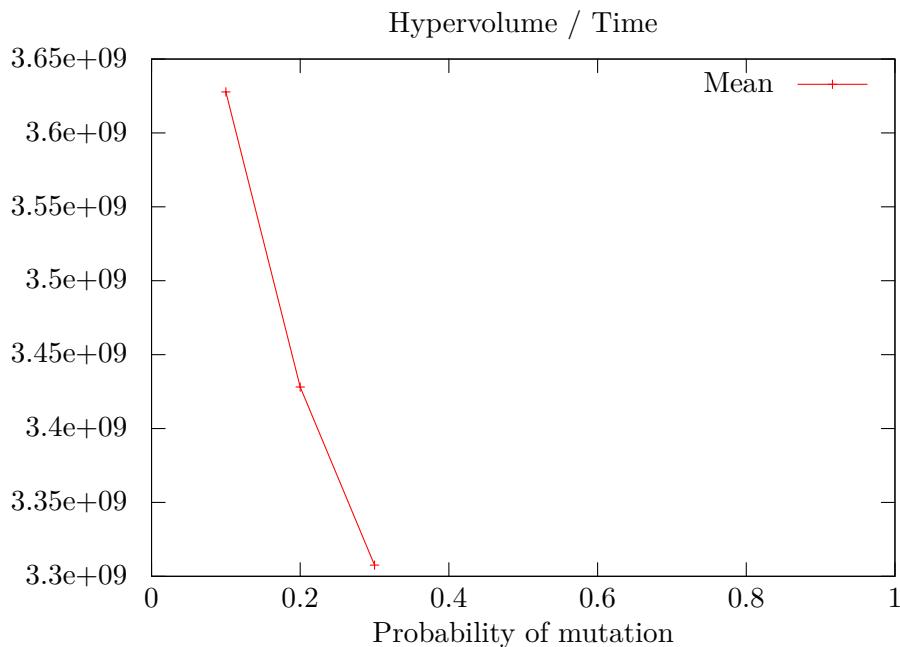
est le front de solutions non-dominées obtenu et la Figure 6.26 contient deux de ces solutions.

### 6.5.2 Agriculture et densité de population

Le second problème auquel nous allons nous intéresser est composé du critère d'homogénéisation de la densité de population au sein de chaque zone ainsi que de l'homogénéisation du pourcentage de valeur ajoutée produite par le secteur de l'agriculture. Nous utiliserons, pour chaque paramètre, les mêmes valeurs que celle présentée à la Table 6.2. La Figure 6.27 contient le front obtenu et la Figure 6.28 contient les deux so-

Paramètre	Valeur
Nombre de générations	300 (condition d'arrêt)
Taille de la population	70
Archive	<i>false</i>
Probabilité de mutation	0.1
Probabilité de muter une zone	0.1
Ajout avant mutation	<i>false</i>
Recherche locale	<i>true</i>
Limite de recherche locale	5

**Table 6.2** – Cette table reprend les différentes valeurs utilisées par l'algorithme discuté en Section 6.5.



**Figure 6.21** – Ce graphique montre le ratio  $\frac{\text{hypervolume}}{\text{temps d'exécution}}$  pour les valeurs moyennes de ces deux données. L'hypervolume plus important généré par la valeur de probabilité de mutation additionné à son temps d'exécution plus faible se reflète par un ratio bien plus important que pour les autres valeurs données à ce paramètre.

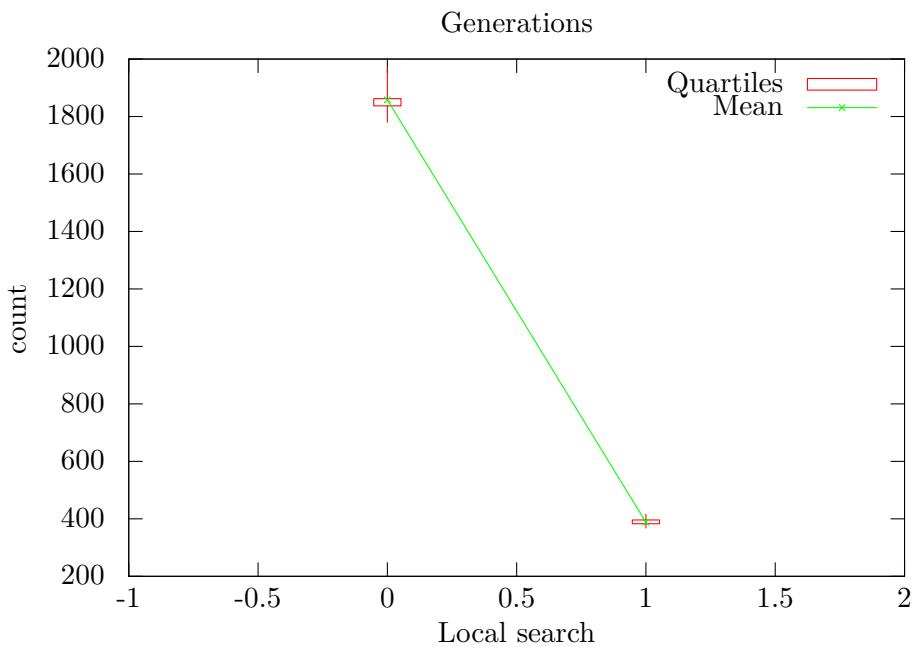
lutions extrêmes du front. Il est intéressant de remarquer la forme particulière du front obtenu et la similitude entre les deux solutions illustrées suggérant qu'un lien existe entre la densité de population et le pourcentage de valeur ajouté générée par le secteur de l'agriculture.

## 6.6 Conclusion

Dans ce chapitre, après une introduction au territoire wallon, nous avons pu présenter l'algorithme implémenté résultant d'une hybridation entre NSGA-II [38] et une procédure de recherche locale. Nous avons insisté sur la générnicité de l'algorithme qui est un atout majeur comme nous le verrons dans le prochain chapitre. Nous avons ensuite pu présenter les différentes procédures propres au problème de partitionnement de territoire.

Nous avons ensuite pu nous concentrer sur l'étude de cas de la Wallonie en présentant les différentes données disponibles pour chacune des communes de la Région et les nombreux critères pertinents. Nous avons également décrit la création de cartes ainsi que les différents outils développés dans ce but pour que l'algorithme puisse dessiner les partitions générées.

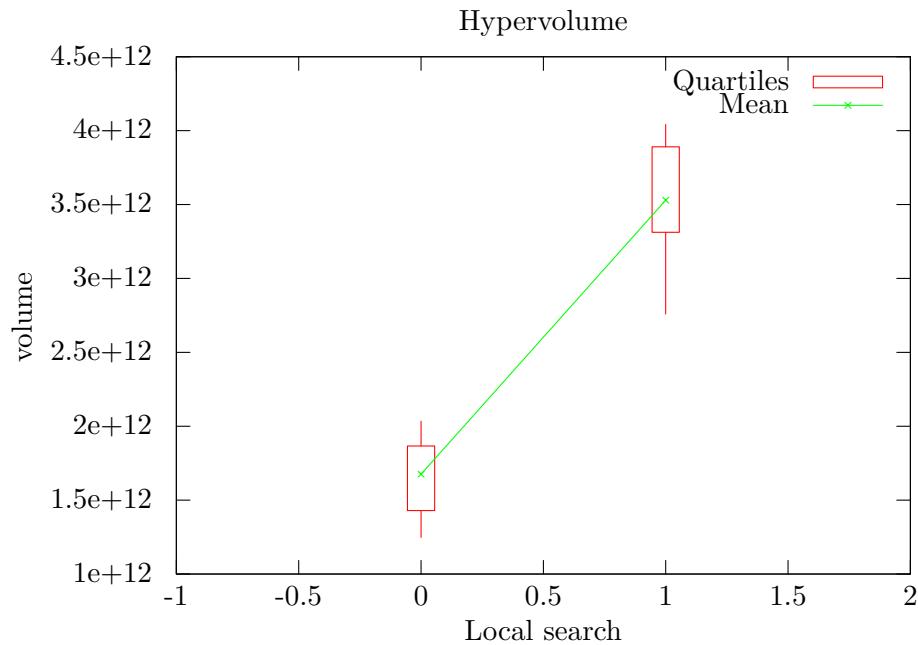
Nous avons analysé en profondeur le paramétrage de l'algorithme afin d'optimiser ses performances et même suggéré une technique utilisant l'algorithme lui-même pour se paramétrier. Nous avons alors pu produire quelques résultats en proposant deux problèmes de partitionnement de la Wallonie, le classique problème de partitionnement électoral et



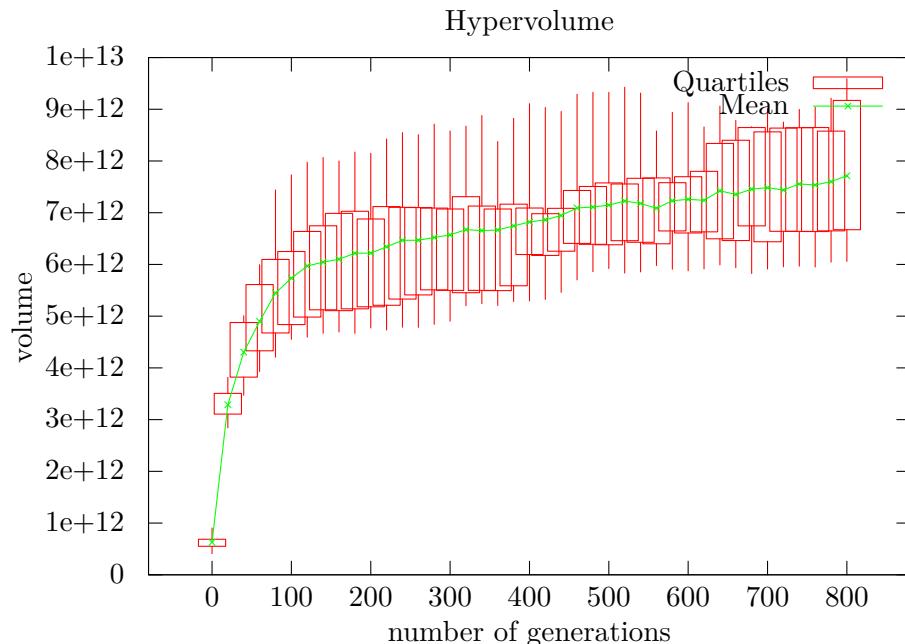
**Figure 6.22** – Ce graphique montre le nombre de générations produites par l'algorithme lorsque en 20 minutes avec et sans recherche locale. Il met également en avant le coût de la procédure de recherche locale puisque presque cinq fois plus de générations sont produites dans le même intervalle de temps lorsque celle-ci est désactivée.

un problème mettant en évidence le lien entre la densité de population et le pourcentage de valeur ajoutée produit par le secteur de l'agriculture.

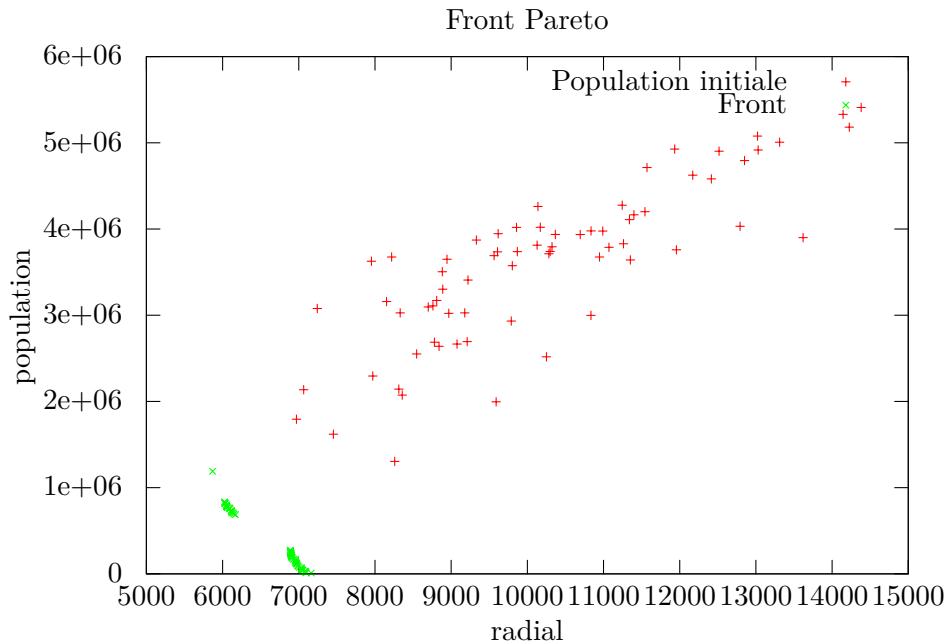
Dans le chapitre suivant nous présenterons trois extensions à l'algorithme. La première tire parti de la généricité de l'algorithme et propose une nouvelle représentation des solutions. Cette dernière est alors comparée à la représentation que nous utilisions jusqu'ici. La seconde extension donne deux techniques pouvant être utilisées pour produire des partitions proche de la partition actuelle de la Wallonie. Enfin, nous introduirons la notion de parallélisation d'algorithme évolutionniste.



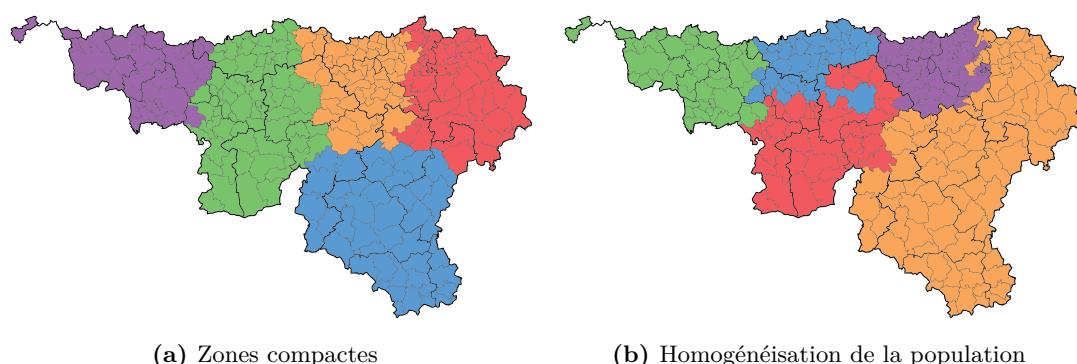
**Figure 6.23** – Malgré le coût de la procédure de recherche locale, celle-ci permet, comme le montre ce graphique, une amélioration importante de l'hypervolume finale même lorsque le nombre de générations est significativement moins important.



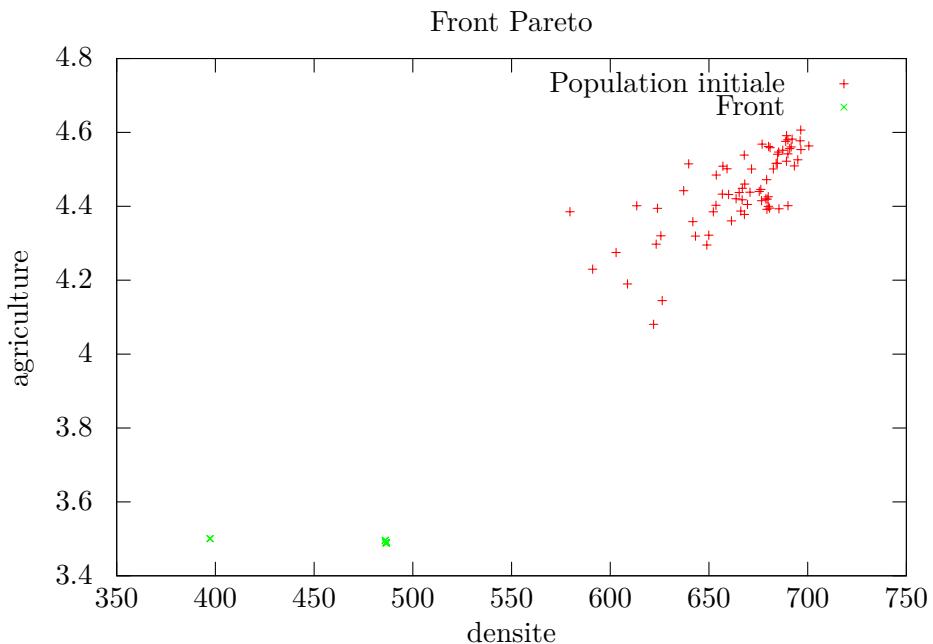
**Figure 6.24** – Cette figure montre l'évolution de l'hypervolume tout au long de l'exécution de l'algorithme. La condition d'arrêt a ici été fixée à 800 générations.



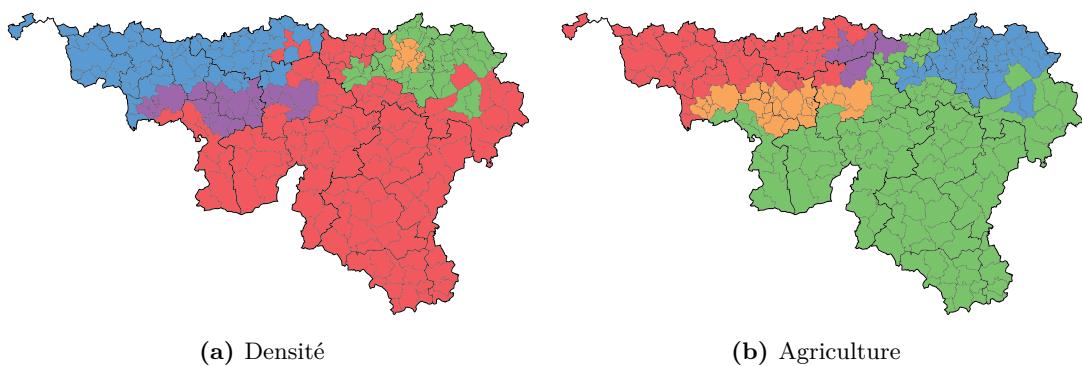
**Figure 6.25** – Ce graphique montre le front de solutions non-dominées obtenu en utilisant les critères du partitionnement électoral. Il contient 73 solutions.



**Figure 6.26** – La figure (a) correspond à la solution du front présenté en Figure 6.25 ayant les zones les plus compactes. La valeur des objectifs vaut 5868 et 1,189,796. La figure (b) est la solution ayant la meilleure homogénéisation du nombre d'habitants. Ses objectifs valent 7164 et 10,050.



**Figure 6.27** – Ce graphique contient le front de solutions non-dominées obtenu en utilisant les critères d’homogénéisation de la densité de population et de la valeur ajoutée par le secteur de l’agriculture. Ce front contient 78 solutions.



**Figure 6.28** – Les deux figures ci-dessus sont des solutions du front illustré en Figure 6.27. La figure (a) est la solution avec la meilleure homogénéisation de la densité de population au sein de chaque zone. La figure (b) est l’équivalent pour le second critère. Les valeurs d’objectif de la solution (a) sont 397 et 4 et celles de la solution (b) sont 487 et 3.

# Chapitre 7

## Extensions

### 7.1 Introduction

Ce chapitre s'inscrit dans la continuité du chapitre précédent et vise à proposer des extensions ou des pistes d'extension à l'algorithme implémenté. Trois principales extensions seront passées en revue dans ce chapitre. Dans un premier temps, nous présenterons une représentation alternative des solutions du problème de districting ainsi que des opérateurs génétique associés. Nous pourrons alors nous intéresser aux résultats obtenus et les comparer avec la représentation précédemment utilisée. Dans un second temps, nous aborderons le problème consistant à générer des partitionnements similaires au partitionnement existant de la Wallonie mais étant toutefois supérieur par rapport aux critères étudiés. Enfin, nous proposerons une piste d'extension en introduisant le sujet de la parallélisation d'algorithme génétique.

### 7.2 Représentation alternative

L'article [21] aborde également le problème de districting et l'utilisation d'un algorithme génétique pour traiter ce dernier. Bien que l'article ne considère pas le problème sous sa forme multicritère puisque les différents critères sont agrégés sous une fonction d'objectif unique, il propose toutefois une approche intéressante au niveau de la représentation des solutions et des opérateurs génétiques utilisés.

Dans cette section, nous allons dans un premier temps décrire leur approche vis-à-vis de l'encodage des solutions et des différents opérateurs génétiques. Ce qui nous permettra ensuite de comparer les résultats obtenus avec ce nouvel encodage des solutions et celui que nous utilisions précédemment.

#### 7.2.1 Représentation

Comme l'explique très justement les auteurs de cet article [21] :

“Avec assez de temps, un algorithme génétique commodément codé trouvera toujours une solution optimale. Toutefois, pour obtenir des solutions raisonnables dans un délai raisonnable, il faut prendre soin de l'encodage du

problème dans les chromosomes et du choix de la fonction de fitness qui sera optimisée.”

Il est donc intéressant d'essayer et de comparer différents encodages pour un même problème. Les auteurs de cet article proposent deux encodages différents décrits ci-dessous.

Le premier encodage d'une solution proposé par les auteurs consiste à forcer chaque zone à être centrée sur un point. Ce dernier étant le centre d'un des éléments territoriaux composant la zone. Une solution  $Y$ , représentant un partitionnement en  $k$  zones est alors encodé par une liste de  $k$  éléments territoriaux correspondant au centre de chaque zone. Pour déterminer à quelle zone appartient un élément territorial, la distance entre ce dernier et chacun des  $k$  centres est calculée. L'élément territorial appartiendra à la zone dont le centre est le plus proche. Pour s'assurer que le partitionnement résultant est bien valide, il faut ensuite vérifier que la contrainte de contiguïté est bien respectée.

Le second encodage proposé est fort semblable au premier.  $k$  centres sont toujours sélectionnés, toutefois, ceux-ci ne doivent plus correspondre aux coordonnées d'un élément territorial mais peuvent être n'importe quelle coordonnées à l'intérieur de l'espace de recherche (la Wallonie dans notre cas).

Il est important de remarquer que ces deux encodages auront naturellement tendance à créer des zones compactes. Comme il s'agit souvent d'un critère recherché, cela ne pose aucun problème.

Ce type d'encodage est régulièrement utilisé pour les problèmes de clustering que nous avons présenté à la Section 4.4.5. En effet, les  $k$  centres choisis divisent l'espace en  $k$  régions que l'on peut représenter sur un diagramme de Voronoï, comme nous l'avons fait à la Figure 4.10. Nous illustrerons à nouveau le diagramme de Voronoï dans ce contexte ci à la Section 7.2.3 regroupant les résultats.

### 7.2.2 Opérateurs

Dans le cadre des algorithmes génétiques, une fois un encodage établi pour les solutions, il est nécessaire de définir les opérateurs génétiques agissant sur celles-ci. Avec la représentation décrite ci-dessus, les opérateurs de mutation et de crossover peuvent être implémenté très simplement.

#### Mutation

L'opérateur de mutation consiste à changer certains bits d'information dans le chromosome de manière à introduire des variations aléatoires. Ceci est très simplement réalisable dans le cadre de la représentation expliquée plus haut. Il suffit de modifier un ou plusieurs des  $k$  centres.

Tout comme dans la représentation que nous utilisions précédemment, deux probabilités sont associées aux mutations. La première est la probabilité de muter une solution pour créer grâce à l'opérateur de crossover. La seconde est, lorsqu'une solution est mutée, la probabilité de muter chacune de ses zones. Dans le cas présent, cette probabilité s'applique à chaque centre.

## Crossover

L'opérateur de crossover consiste à créer une nouvelle solution à partir d'informations provenant de deux parents. Les auteurs proposent ici d'utiliser un crossover uniforme. C'est-à-dire, pour chaque centre de la solution générée, il y a 50% de chance de sélectionner le centre du père et 50% de chance de sélectionner le centre de la mère.

## Recherche locale

Baçao et al. [21] utilisent un algorithme génétique classique, c'est-à-dire sans recherche locale. Toutefois, il est aisément de définir l'opérateur pour cet encodage.

Pour le premier encodage consistant à sélectionner une commune comme centre, on peut trouver les solutions voisines en déplaçant chaque centre à une commune adjacente. Pour le second encodage où chaque centre est un point de l'espace de recherche, il est moins trivial de définir la notion de solutions voisines, nous n'utiliserons donc pas la recherche locale avec ce second encodage.

### 7.2.3 Comparaison et résultat

Au chapitre précédent, nous avons décrit l'algorithme implémenté en insistant sur sa généralité et sa capacité à être adapté à d'autres problèmes. C'est donc avec une grande simplicité que cette nouvelle représentation ainsi que les opérateurs génétiques qui y sont associés ont pu être incorporés au programme.

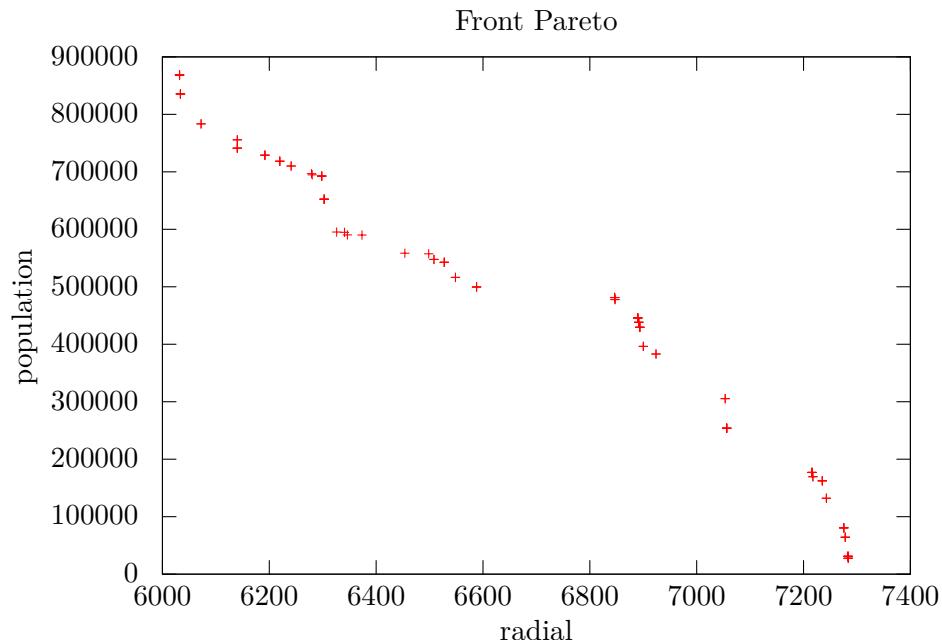
Dans un premier temps, afin de pouvoir visualiser le comportement de l'algorithme avec cette nouvelle implémentation, nous avons exécuté celui-ci avec les valeurs de chaque paramètre repris dans la Table 7.1. La condition d'arrêt utilisée était le nombre de générations produites, fixé à 200. La Figure 7.1 montre le front de solutions non-dominées obtenu et la Figure 7.2 est une des solutions de ce front.

Dans un second temps, il est évidemment pertinent de comparer les performances de l'algorithme avec les différents encodages qui sont maintenant à notre disposition. Comme première comparaison, nous avons exécuter l'algorithme avec chacun des trois encodages disponibles : l'encodage proposé par Tavares et al. que nous avions présenté au Chapitre 5 ainsi que les deux nouveaux encodages proposés par Baçao et al. L'algorithme a été lancé avec comme condition d'arrêt un nombre fixe de génération, fixé ici à 200. Les valeurs des différents paramètres sont celles que nous avions présentées dans la Table 7.1 à l'exception de la recherche locale qui est désactivée pour le second encodage de Baçao et al. car elle n'est pas implémentée pour cette représentation comme expliqué précédemment. La Figure 7.3 illustre le front de solutions non-dominées trouvé pour chacune des trois représentations. Il est intéressant de constater que le premier encodage semble fournir un front contenant des trous alors que les deux autres représentations fournissent un front plus continu. Il est également important de remarquer que chacun des trois fronts contient des solutions non dominées par les deux autres fronts.

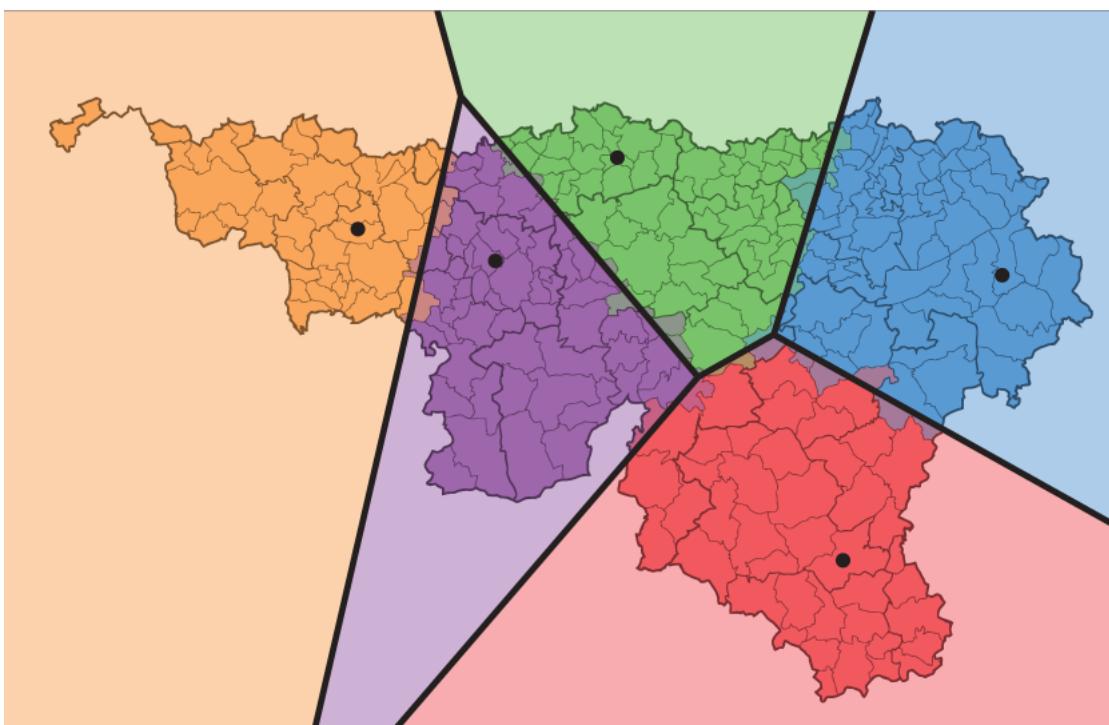
L'algorithme ne s'exécute pas à la même vitesse avec les différents encodages. Ceci est dû aux différentes procédures qui ont des temps de traitement plus ou moins long. Un second comparatif a donc été réalisé en changeant la condition d'arrêt de l'algorithme

Paramètre	Valeur
Problème	Le problème utilisé avec cette nouvel représentation est le problème du partitionnement de la Wallonie en 5 zones. Les données utilisées sont celles présentées à la Section 6.4.2.
Critères	Les critères classiques du partitionnement électoral ont été utilisé, c'est-à-dire générer des zones compactes et homogènes entre elles au niveau du nombre d'habitant (inter-homogénéisation).
Chromosome	Le premier encodage proposé par les auteurs de l'article [21] et que nous avons décrit plus haut (7.2.1) a été utilisé.
Nombre de générations	200
Taille de la population	70
Archive	<i>false</i>
Probabilité de mutation	0.1
Probabilité de muter une zone	0.1
Ajout avant mutation	<i>false</i>
Recherche locale	<i>true</i>
Limite de recherche locale	5

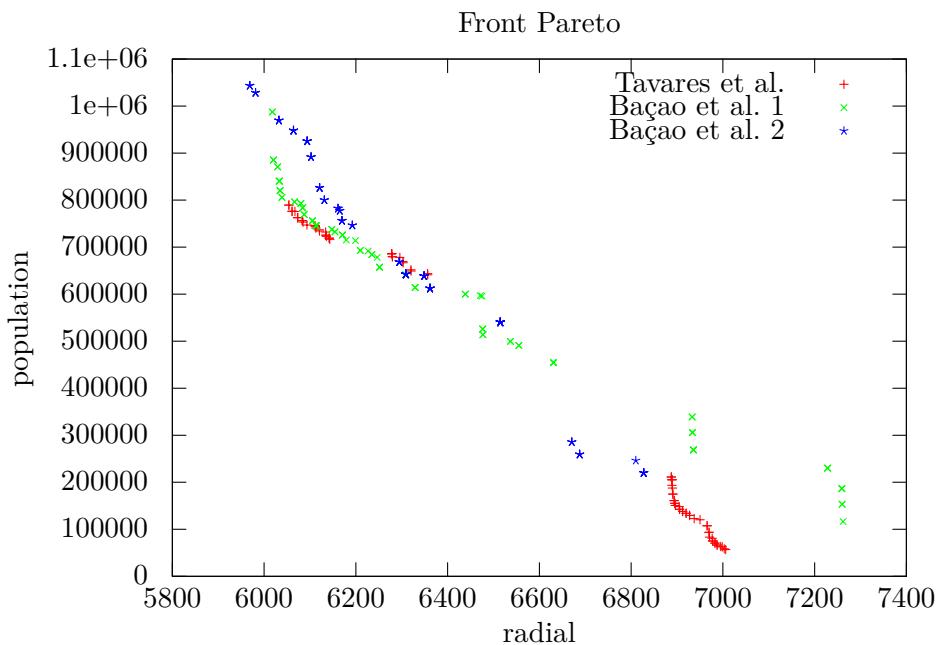
**Table 7.1** – Cette table reprends les différents paramètres utilisés pour générer les résultats en utilisant l'encodage proposé dans l'article [21].



**Figure 7.1** – Ce graphique représente le front de solutions non-dominées obtenu avec le premier des deux nouveaux encodages et les valeurs de paramètre repris dans la Table 7.1. Celui-ci est composé de 107 solutions. La valeur de l'objectif concernant le compactage des zones est reprise en abscisse et celle concernant l'homogénéisation du nombre d'habitants entre les zones est reprise en ordonnée. A titre d'information, les valeurs de ces deux objectifs pour la partition actuelle de la Wallonie sont 6500.05 et 1,925,212.



**Figure 7.2** – Sur cette figure, on peut visualiser une des solutions non-dominées du front présenté à la Figure 7.1. Les cinq points correspondent aux cinq centres encodés dans la solution. Il est également possible de visualiser le diagramme de Voronoi délimité par ces cinq centres.



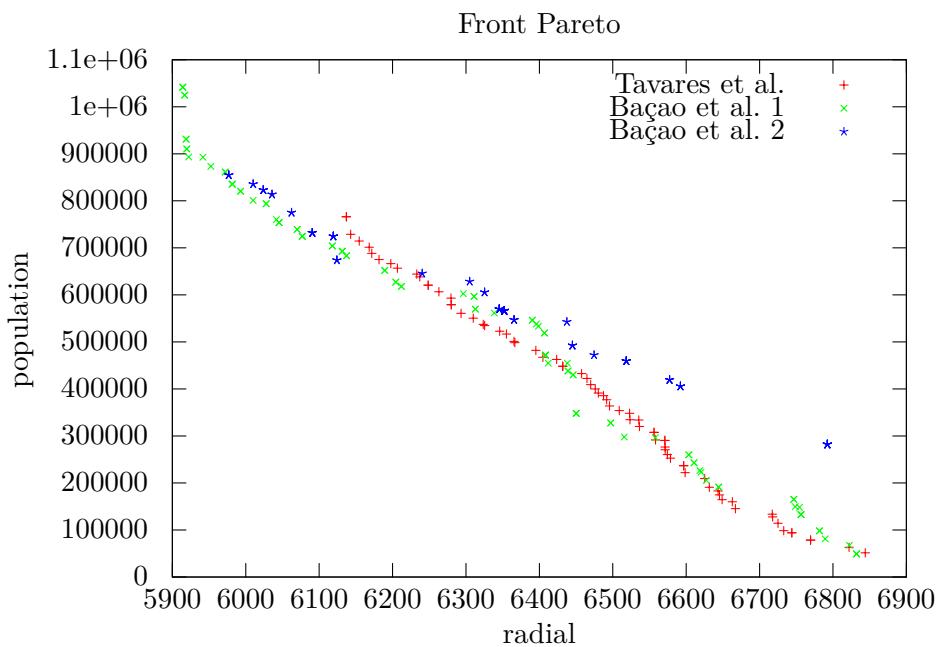
**Figure 7.3** – Ce graphique reprend les fronts obtenus après 300 générations pour chacun des trois encodages du problème de partitionnement. La taille de chacun des fronts est respectivement 77, 89 et 113 solutions.

à un temps fixé. Le résultat est visible en Figure 7.4, l'algorithme s'est exécuté pour chaque encodage pendant 30 minutes. A nouveau, il est possible de voir que chaque front contient des solutions qui sont non-dominées par les deux autres fronts.

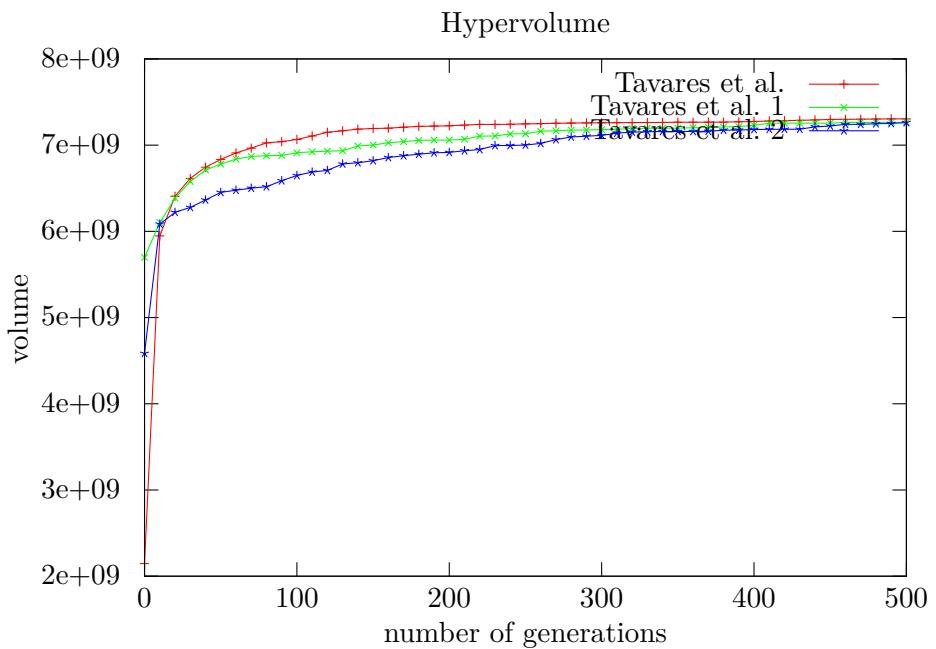
Enfin, comme dernier comparatif, nous nous sommes intéressé à l'évolution de l'hypervolume pour les différents encodages. De cette manière, nous pouvons visualiser la convergence de chacune des trois représentations et comparer celles-ci. Le résultat est visible en Figure 7.5. On peut remarquer que les deux nouveaux encodages semblent fournir un hypervolume initial plus important que la représentation utilisée précédemment. Ce résultat est toutefois à utiliser avec précaution car nous utilisons ici les critères du partitionnement électoral c'est-à-dire entre autre le critère de zones compactes et comme expliqué à la section précédente, de par leur nature, les deux nouveaux encodages ont naturellement tendance à générer des partitions compactes.

Comme on a pu le constater au travers de cette section, les résultats de l'algorithme sont relativement sensibles à la représentation des solutions adoptées comme nous l'avait suggéré la citation en début de Section 7.2.1.

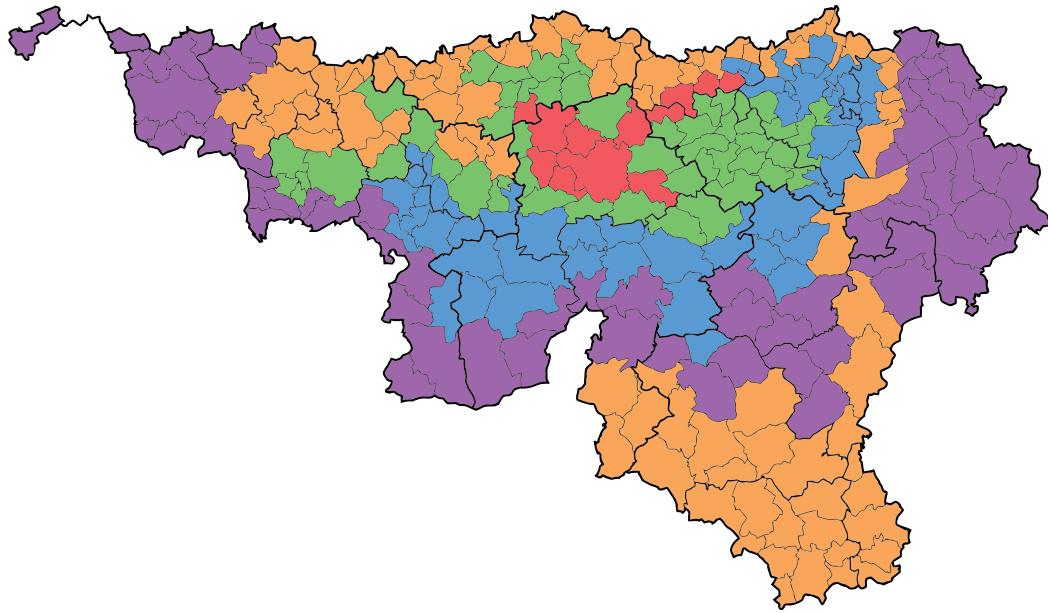
Toutefois, au cours des nombreuses comparaisons effectuées, nous avons pu constater que chacune des trois représentations de solutions fournissent de très bons résultats. En particulier, les deux nouveaux encodages implémentés ont de très bonnes performances puisqu'ils sont plus rapides que le premier encodage. Leur implémentation est également triviale et bien plus simple que le premier encodage que nous avions présenté au chapitre précédent. Il est toutefois à noter que au regard du fonctionnement de ces deux nouveaux encodages, et contrairement à la représentation précédente, toutes les



**Figure 7.4** – Sur ce graphique, il est possible de visualiser les fronts générés en utilisant les trois encodages implémentés. Le critère d’arrêt utilisé est une limite de temps, fixée à 30 minutes par encodage. Dans ce délais, la première représentation a généré 760 générations, le second 1098 générations et le dernier 938 générations. Les tailles des fronts sont respectivement, 72, 96 et 125.



**Figure 7.5** – Sur cette figure, il est possible de comparer les trois encodages au niveau de leur convergence. Ce graphique reprend en effet l'évolution de l'hypervolume tout au long de l'exécution de l'algorithme. La valeur d'hypervolume pour chaque encodage est une valeur moyenne effectuée sur 10 exécutions de l'algorithme. On peut voir que pour chacune des trois représentations, l'algorithme converge et semble stable dès 300 générations.



**Figure 7.6** – Cette figure illustre une partition qu'il n'est pas possible d'encoder avec les deux nouvelles représentations proposées. En effet, il est impossible de choisir cinq centres qui définiraient les cinq zones illustrées.

partitions ne peuvent être encodées. Par exemple, la Figure 7.6 est une partition qui ne peut être encodée avec aucune des deux nouvelles représentations. Toutefois, le critère de zones compactes est souvent désiré et nous ne considérerons donc pas ceci comme un problème.

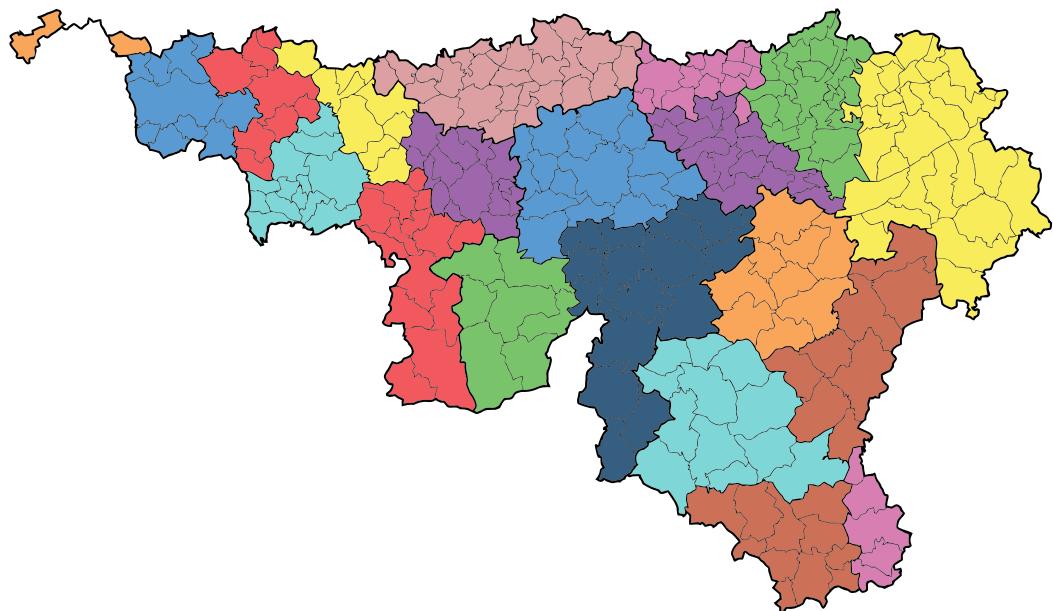
### 7.3 Partitions similaires

Nous avons vu au chapitre précédent que la Wallonie est déjà partitionnée. Il existe en effet un partitionnement de celle-ci en cinq provinces que nous avions illustré à la Figure 6.2 en page 86, ainsi qu'un partitionnement de celle-ci en vingt arrondissements illustré à la Figure 7.7.

Il peut donc être pertinent de s'intéresser à des partitions proches de celles existantes mais qui amélioreraient toutefois significativement les critères étudiés. Différentes façons de procéder peuvent alors être envisagées, nous allons en aborder deux dans les sections qui suivent.

#### 7.3.1 Distance entre partition

Une première méthode pourrait être de définir un nouveau critère dont la fonction d'objectif refléterait une mesure de distance entre le partitionnement considéré et le partitionnement de référence, c'est-à-dire le partitionnement actuel de la Wallonie. On



**Figure 7.7** – Cette figure montre le partitionnement actuel de la Wallonie en vingt arrondissements. Ceux-ci étant : Arlon, Ath, Bastogne, Charleroi, Dinant, Huy, Liège, Marche-en-Famenne, Mons, Mouscron, Namur, Neufchâteau, Nivelles, Philippeville, Soignies, Thuin, Tournai, Verviers, Virton et Waremme.

viserait alors à minimiser ce critère afin d'avoir des solutions proches du partitionnement actuel. De manière générale, une fonction de distance entre deux éléments d'un ensemble  $S$  est définie comme suit :

**Définition 15.** *Distance* Une fonction de distance sur un ensemble  $S$  est une fonction  $d : S \times S \rightarrow \mathbb{R}^+$  respectant les propriétés suivantes pour tout  $x, y, z \in S$  :

- Est symétrique, c'est-à-dire  $d(x, y) = d(y, x)$
- $d(x, y) = 0$  si et seulement si  $x = y$ .
- Respecte l'inégalité triangulaire, c'est-à-dire  $d(x, z) \leq d(x, y) + d(y, z)$

Dans le contexte de ce nouveau critère visant à mesurer la distance entre deux partitionnements, l'ensemble  $S$  est l'ensemble des partitionnements possibles.

Comme l'explique Denœud et al. [39] :

“La comparaison des partitions est un sujet central dans le regroupement, aussi bien pour comparer des algorithmes de partitionnement que pour classer les variables nominales. La littérature abonde en distances (petites valeurs lorsque les partitions sont proches) et en indices (grandes valeurs lorsque les partitions sont proches) définis par de nombreux auteurs pour comparer deux partitions  $P$  et  $Q$  définies sur le même ensemble  $X$ . ”

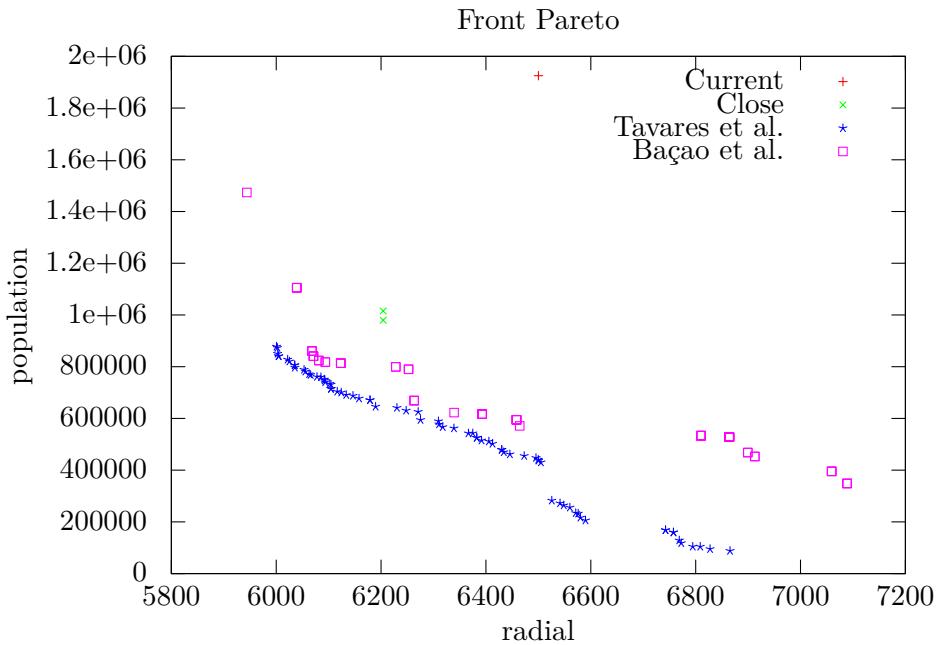
Nous n'entrerons pas dans les détails de ces différentes techniques mais le lecteur intéressé pourra se référer aux ouvrages [39, 54].

### 7.3.2 Recherche locale

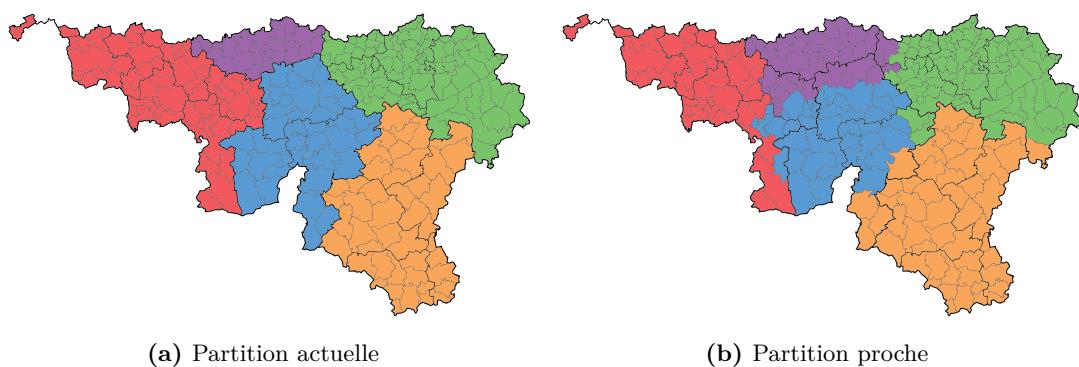
Une autre façon de procéder est d'utiliser la procédure de recherche locale sur le partitionnement actuel afin de générer les solutions voisines. Si l'on considère que la distance entre deux partitions est le nombre d'éléments territoriaux devant être déplacé d'une zone à un autre dans une des deux partitions pour qu'elle soit égale à l'autre partition, les partitionnements générés seront alors distant de 1 du partitionnement existant. En appliquant à nouveau la recherche locale sur ces solutions, il est possible à tout moment de savoir combien d'étapes, au plus, nous sépare du partitionnement actuel. Il est alors trivial de limiter la recherche à une certaine distance maximale.

Le pseudo-code présenté à l'Algorithme 12 montre un programme simple permettant de trouver des solutions proches de la solution actuelle. Celui-ci maintient un ensemble de solutions. A chaque itération, les solutions voisines d'une solution  $Y$  de la population sont ajoutées à cette dernière pour autant qu'elle domine la solution  $Y$ . Le nombre de solutions présentent dans la population explose rapidement. C'est pourquoi, à intervalle régulier, la population est purgée et seules les solutions non dominées sont gardées. Cet algorithme est bien sur une approche moins complète que la première technique que nous avons présenté à la section précédente. Néanmoins, elle permet de très rapidement générer des bons résultats.

La Figure 7.8 est une illustration de la technique que nous venons d'exposer. Cette figure permet de situer en terme de valeurs d'objectif les solutions générées par cette technique par rapport à la partition actuelle de la Wallonie mais aussi par rapport aux solutions trouvées par l'algorithme sans restriction sur la distance par rapport à la partition existante. La Figure 7.9 montre une partition générée via la technique présentée ainsi que la partition actuelle afin de pouvoir les comparer.



**Figure 7.8** – Ce graphique contient de nombreuses séries de données, produites par différentes techniques. La première série ne contient qu'une solution, la partition actuelle de la Wallonie en cinq provinces. La seconde série contient deux solutions distantes d'au plus 36 changements de commune. La troisième série est un front obtenu avec l'algorithme présenté au chapitre précédent en 200 générations. Il est composé de 73 solutions. Enfin, la quatrième série est un front obtenu avec l'algorithme en utilisant la première représentation présentée en Section 7.2.1, toujours en 200 générations. Ce front contient 83 solutions.



**Figure 7.9** – La partition (a) est la partition actuelle de la Wallonie. Les valeurs des deux objectifs ; compactage et homogénéisation du nombre d'habitants sont respectivement 6500 et 1,925,212. La partition (b) est une des deux partitions proches de la partition actuelle que nous avons présentées à la Figure 7.8. Ses valeurs d'objectif sont 6204 et 979,582.

---

**Algorithme 12:** Utilisation de la recherche locale pour trouver des partitionnements similaires à celui existant.

---

```

input :  $Y_c$ 
      MaximalDistance
      ReduceInterval
output : population
      SetDistance( $Y_c$ , 0)
      population  $\leftarrow \{Y_c\}$ 
while population contient des solutions non traitées do
     $t \leftarrow$  une solution non traitée de population
    for  $n \in \text{Neighbours}(t)$  do
      if  $n$  domine  $t$  et GetDistance( $t$ ) < MaximalDistance then
        | SetDistance( $n$ , GetDistance( $t$ ) + 1)
        | population  $\leftarrow$  population  $\cup n$ 
      end
    end
    marquer  $t$  comme traité
    if ReduceInterval solutions ont été traitées depuis la dernière fois then
      | population  $\leftarrow$  NonDominated(population)
    end
  end

```

---

## 7.4 Parallélisation

La parallélisation d’algorithmes génétiques est une pratique très répandue. Ceci s’explique par le fait que cette métaheuristique s’y prête particulièrement bien. En effet, les algorithmes génétiques parallèles sont simple à implémenter et permettent un gain de performance significatif [31, 32, 93, 101].

Cette section ne vise pas à fournir une analyse complète de la parallélisation d’algorithme génétique tant la littérature et les résultats abondent dans ce domaine. L’objectif est plutôt de fournir au lecteur un aperçu de ce que permet la parallélisation et des principales techniques utilisées pour la mettre en place.

De manière générale, la parallélisation d’un algorithme ou d’un programme consiste à faire fonctionner celui-ci sur plusieurs unités de calcul simultanément afin d’en diminuer le temps total de calcul. Le besoin d’utiliser la parallélisation peut s’expliquer par des raisons de coût, plusieurs processeurs reviennent moins cher que un processeur équivalent en puissance ou bien simplement pour atteindre des performances qu’il n’est pas possible d’atteindre sur un processeur unique aujourd’hui.

Afin de mesurer le gain obtenu par la parallélisation, plusieurs notations ont été introduites. On parle donc souvent d’*accélération* et d’*efficacité* [93].

**Définition 16. Accélération** [93] *L’accélération correspond au gain de temps obtenu en exécutant un programme sur  $p$  processeurs par rapport à son exécution sur un processeur unique.*

$$S_p = \frac{T_1}{T_p} \quad (7.1)$$

où  $T_i$  désigne le temps de calcul requis pour l'exécution du programme sur  $i$  processeur.

**Définition 17. Efficacité [93]** L'efficacité d'un programme parallélisé est alors donnée par :

$$E_p = \frac{S_p}{p} \leq 1 \quad (7.2)$$

Celle-ci est inférieurs ou égale à un. En effet, dans le meilleur des cas, chaque processeur s'exécute pendant  $\frac{1}{p}$  du temps que prendrait l'exécution sur un processeur et l'efficacité est donc unitaire. Cependant, ceci est rarement le cas et l'efficacité est plus petite que l'unité. Ceci s'explique par la partie non-parallélisable du programme ou par les coûts additionnels liés à la communication entre les différentes unités de calcul.

Dans le cadre des algorithmes génétiques, la parallélisation se justifie par la demande croissante à pouvoir explorer de grands espaces de recherche avec des fonctions d'objectif coûteuses et l'utilisation de grande population [93].

Le type d'algorithme génétique parallèle prédominant de loin dans la littérature consiste en l'utilisation de plusieurs populations évoluant séparément et échangeant des individus de manière occasionnelle [32]. Nous nous restreindrons donc à ce type d'algorithme.

Comme l'introduit très simplement L. J. Pit [93] :

“En observant la nature, on pourrait dire que le monde des humains se compose d'une grande population. Un autre point de vue consiste à dire qu'il s'agit en fait d'une collection de sous-population qui évoluent indépendamment les unes des autres sur des continents isolées ou dans des régions restreintes. De temps en temps certaines personnes d'une région migrent vers une autre région. Cette migration permet aux sous-populations de partager du matériel génétique. L'idée est que des environnements isolés, ou des îles concurrentes, sont plus efficaces en terme de recherche qu'un environnement plus large dans lequel tous les membres sont maintenus ensemble.”

En appliquant cette idée aux algorithmes génétiques, il est possible de les paralléliser en exécutant l'algorithme sur plusieurs processeurs s'exécutant chacun sur leur propres sous-population et en implémentant un processus de migration permettant à ces processeurs d'échanger leurs meilleures solutions avec les processeur voisins. On parle dans ce cas de l'algorithme génétique distribué.

Plusieurs problèmes doivent alors être considérés [93] :

- Pour chaque processeur, l'ensemble de processeurs avec lesquels celui-ci échange des individus.
- La fréquence des migrations, c'est-à-dire la fréquence à laquelle un processeur échange des individus.
- Le taux de migrations, c'est-à-dire le nombre d'individus échangés lors de chaque migration.
- Les individus sélectionnés pour l'échange.
- Les individus à supprimer après avoir reçu de nouveaux individus.

Nous n'aborderons pas ces différents problèmes dans cette section puisque celle-ci relève plus d'une introduction à la parallélisation comme piste d'extension qu'une étude en profondeur de ce domaine. Toutefois, à titre d'exemple, Bação et al. [21] qui ont proposé le nouvel encodage de solution présenté en Section 7.2 ont eux-mêmes utilisé un

algorithme génétique parallélisé avec une probabilité de migration à chaque itération de 0.001.

Le lecteur intéressé par les différentes approches de ces problèmes ou par les autres techniques de parallélisation d'algorithmes génétiques pourra se référer aux ouvrages suivants [31, 32, 93, 101]. En particulier [31, 32] proposent une étude complète ainsi qu'un résumé des recherches faites dans ce domaine.

## 7.5 Conclusion

Au cours de ce chapitre, nous avons abordé trois extensions à l'algorithme. Comme première extension, nous avons proposé une représentation alternative des solutions du problème ainsi que des procédure de mutation, crossover et recherche locale qui lui sont associé. Nous avons pu comparer les résultats obtenus par les différents encodages et constaté que l'impact sur les solutions non-dominées générée n'était pas négligeable. L'encodage des solutions d'un problème d'optimisation doit donc être réalisée avec soin et constitue un élément primordial lors de l'utilisation d'algorithmes évolutionnistes.

Pour la seconde extension, nous nous sommes intéressé à la génération de partitions proche de la partition existante de la Wallonie mais qui sont toutefois meilleures par rapport aux critères étudiés. Nous avons, dans ce cadre, abordé la notion de distance entre partitions et utilisé la procédure de recherche locale à notre disposition.

Enfin, comme dernière piste d'extension, nous avons abordé la parallélisation, pratique courante dans l'utilisation d'algorithmes évolutionnistes. Nous avons également présenté les différentes questions que pouvait soulever cette technique.

# Chapitre 8

## Conclusion

Au travers de ce mémoire nous avons approfondi la problématique du partitionnement de territoire. Ce sujet complexe revêt une importance capitale dans des domaines variés. Il suscite dès lors un intérêt marqué pour des techniques de résolution appropriées.

Si les algorithmes génétiques ont permis d'obtenir d'excellents résultats dans plusieurs problématiques d'optimisation multicritère [125, 110, 41, 48, 115], s'interroger sur leur pertinence pour le problème de partitionnement de territoire semble naturel.

Dans le cadre du partitionnement de la Région wallonne, nous avons montré dans ce mémoire, en combinant les travaux de Tavares et al.[112] et Baçao [21], qu'il était possible d'obtenir un algorithme suffisamment générique que pour être applicable à une gamme étendue de problèmes de partitionnement de territoire.

En appliquant notre algorithme au partitionnement de la Wallonie en provinces, nous avons pu produire, pour certains ensembles de critères, des alternatives significativement optimisées.

D'autre part, l'application de cette méthode nous a permis de mettre en évidence les interrelations sous-jacentes à certains critères tels que la densité de population et la production agricole.

Aux travers de trois extensions, nous avons montré que des améliorations étaient possibles. En introduisant dans notre algorithme une représentation alternative des solutions proposée par Baçao et al. [21], nous avons mis en évidence l'importance capitale de l'encodage des solutions dans les performances de l'algorithme d'optimisation. En effet, en comparant les différents encodages nous avons fait ressortir les disparités entre leur temps d'exécution et leur vitesse de convergence.

En nous intéressant à la notion de distance entre partitions, nous avons générés de nouveaux partitionnements de la Wallonie, proches du partitionnement actuel, mais offrant toutefois une considérable amélioration des critères étudiés.

Une dernière extension porte sur les pistes d'amélioration de performances que nous amènerait la parallélisation de l'exécution de l'algorithme génétique.

# Bibliographie

- [1] Arcgis. <http://www.esri.com/software/arcgis>. [Online ; accessed 25-May-2013].
- [2] Belgique. <http://www.belgium.be>. [Online ; accessed 1-May-2013].
- [3] Comersis : Cartographie vectorielle. <http://www.cmap.comersis.com/>. [Online ; accessed 6-May-2013].
- [4] Cpdt wallonie. <http://cpdt.wallonie.be/>. [Online ; accessed 6-May-2013].
- [5] Facebook. <http://www.facebook.com>. [Online ; accessed 25-May-2013].
- [6] Facebook is collecting your data. <http://gigaom.com/2012/08/22/facebook-is-collecting-your-data-500-terabytes-a-day/>. [Online ; accessed 25-May-2013].
- [7] Google map api. <https://developers.google.com/maps/>. [Online ; accessed 6-May-2013].
- [8] Grass gis. <http://grass.osgeo.org/>. [Online ; accessed 25-May-2013].
- [9] gvSIG. <http://www.gvsig.org/web/>. [Online ; accessed 25-May-2013].
- [10] Jump gis. <http://jump-pilot.sourceforge.net/>. [Online ; accessed 25-May-2013].
- [11] K-means clustering. [http://home.deib.polimi.it/matteucc/Clustering/tutorial\\_html/kmeans.html](http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html). [Online ; accessed 16-April-2013].
- [12] Open geospatial consortium. <http://www.opengeospatial.org/>. [Online ; accessed 25-May-2013].
- [13] Portail de la wallonie. <http://www.wallonie.be/>. [Online ; accessed 6-May-2013].
- [14] Quantum gis. <http://qgis.org/>. [Online ; accessed 25-May-2013].
- [15] Ruby. <http://www.ruby-lang.org/en/>. [Online ; accessed 2-May-2013].
- [16] Spatialite. <http://www.gaia-gis.it/gaia-sins/>. [Online ; accessed 25-May-2013].
- [17] Vector magic. <http://vectormagic.com>. [Online ; accessed 6-May-2013].
- [18] Wiktionary. <http://fr.wiktionary.org/>. [Online ; accessed 3-May-2013].
- [19] Cs381 discrete structures/discrete mathematics web course material. [http://www.cs.odu.edu/~toida/nerzic/content/web\\_course.html](http://www.cs.odu.edu/~toida/nerzic/content/web_course.html), 2009. [Online ; accessed 2-May-2013].
- [20] Micah Altman. Computational complexity of automated redistricting : Is automation the answer, the. *Rutgers Computer & Tech. LJ*, 23 :81, 1997.

- [21] Fernando Baçao, Victor Lobo, and Marco Painho. Applying genetic algorithms to zone design. *Soft Computing*, 9(5) :341–348, 2005.
- [22] Paul K Bergey, Cliff T Ragsdale, and Mangesh Hoskote. A simulated annealing genetic algorithm for the electrical power districting problem. *Annals of Operations Research*, 121(1-4) :33–55, 2003.
- [23] C.E. Bichot. Introduction au partitionnement de graphe. University Lecture, 2007.
- [24] C.E. Bichot and P. Siarry. *Graph Partitioning*. Iste Series. Wiley, 2011.
- [25] Charles-Edmond Bichot. Introduction au partitionnement de graphe. ENS de Lyon, 2007.
- [26] Burcin Bozkaya, Erhan Erkut, and Gilbert Laporte. A tabu search heuristic and adaptive memory procedure for political districting. *European Journal of Operational Research*, 144(1) :12–26, 2003.
- [27] Paul S Bradley, Usama Fayyad, and Cory Reina. Scaling em (expectation-maximization) clustering to large databases. *Microsoft Research*, 1998.
- [28] RM Brady. Optimization strategies gleaned from biological evolution. *Nature*, 317(6040) :804–806, 1985.
- [29] Michelle H Browdy. Simulated annealing : an improved computer model for political redistricting. *Yale Law & Policy Review*, 8(1) :163–179, 1990.
- [30] P.A. Burrough. *Principles of geographical information systems for land resources assessment*. Monographs on soil and resource surveys. Clarendon Press, 1994.
- [31] Erick Cantú-Paz. A summary of research on parallel genetic algorithms. 1995.
- [32] Erick Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2) :141–171, 1998.
- [33] K.C. Clarke. *Analytical and computer cartography*. Number v. 1 in Prentice Hall series in geographic information science. Prentice Hall, 1995.
- [34] Keith C Clarke. *Getting started with geographic information systems*, volume 3. Prentice Hall Upper Saddle River, NJ, 1997.
- [35] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M Hellerstein, and Caleb Welton. Mad skills : new analysis practices for big data. *Proceedings of the VLDB Endowment*, 2(2) :1481–1492, 2009.
- [36] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, 1998.
- [37] Yves De Smet. Decision engineering. University Lecture (ULB), 2012.
- [38] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization : Nsga-ii. *Lecture notes in computer science*, 1917 :849–858, 2000.
- [39] Lucile Denœud and Alain Guénoche. Comparison of distance indices between partitions. In *Data Science and Classification*, pages 21–28. Springer, 2006.
- [40] Marco Dorigo and Luca Maria Gambardella. Ant colony system : A cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1) :53–66, 1997.
- [41] T.B.C.S.D.U. Dortmund. *Evolutionary Algorithms in Theory and Practice : Evolution Strategies, Evolutionary Programming, Genetic Algorithms : Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, USA, 1995.

- [42] M. Ehrgott. *Multicriteria Optimization*. Lecture notes in economics and mathematical systems. Springer, 2005.
- [43] B.S. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis*. Wiley series in probability and statistics. Wiley, 2011.
- [44] Jacques A Ferland and Gilles Guénette. Decision support system for the school districting problem. *Operations Research*, 38(1) :15–21, 1990.
- [45] FindTheBest. Best smartphones comparison - reviews and ratings — findthebest, 2013. [Online ; accessed 9-February-2013].
- [46] Bernhard Fleischmann and Jannis N Paraschis. Solving a large scale districting problem : a case report. *Computers & Operations Research*, 15(6) :521–533, 1988.
- [47] Flickr. <http://www.flickr.com/>. [Online ; accessed 9-April-2013].
- [48] Carlos M Fonseca and Peter J Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary computation*, 3(1) :1–16, 1995.
- [49] Carlos M Fonseca, Joshua D Knowles, Lothar Thiele, and Eckart Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. In *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 216, 2005.
- [50] Allen D Franklin and Ernest Koenigsberg. Computed school assignments in a large district. *Operations Research*, 21(2) :413–426, 1973.
- [51] Robert S Garfinkel and George L Nemhauser. Optimal political districting by implicit enumeration techniques. *Management Science*, 16(8) :B–495, 1970.
- [52] GIS. <http://wiki.gis.com/>. [Online ; accessed 8-April-2013].
- [53] Google. Google earth. [Online ; accessed 8-April-2013].
- [54] Dan Gusfield. Partition-distance : A problem and class of perfect graphs arising in clustering. *Information Processing Letters*, 82(3) :159–164, 2002.
- [55] Robert Haining, Stephen Wise, and Marcus Blake. Constructing regions for small area analysis : material deprivation and colorectal cancer. *Journal of Public Health*, 16(4) :429–438, 1994.
- [56] Alain Hertz and Dominique de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4) :345–351, 1987.
- [57] H.H. Hoos and T. Stützle. *Stochastic Local Search : Foundations & Applications*. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science, 2004.
- [58] IBM. Big data ibm. <http://www-01.ibm.com/software/data/bigdata/>. [Online ; accessed 9-April-2013].
- [59] IBM. Ibm db2 database software. <http://www-01.ibm.com/software/data/db2/>. [Online ; accessed 25-May-2013].
- [60] Supergeo Technologies Inc. Supergis desktop. [http://www.supergeotek.com/products\\_desktopgis.aspx](http://www.supergeotek.com/products_desktopgis.aspx). [Online ; accessed 25-May-2013].
- [61] Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima. Iterative approach to indicator-based multiobjective optimization. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3967–3974. IEEE, 2007.
- [62] Anil K Jain. Data clustering : 50 years beyond k-means. *Pattern Recognition Letters*, 31(8) :651–666, 2010.

- [63] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering : a review. *ACM computing surveys (CSUR)*, 31(3) :264–323, 1999.
- [64] Tommy R Jensen and Bjarne Toft. *Graph coloring problems*. Wiley-Interscience, 2011.
- [65] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1) :359–392, 1998.
- [66] M. Koksalan, J. Wallenius, and S. Zionts. *Multiple Criteria Decision Making : From Early History to the 21st Century*. World Scientific, 2011.
- [67] Jesper Larsen. Set partitioning and applications. <http://www2.imm.dtu.dk/courses/02735/sppintro.pdf>. University Lecture (Technical University of Denmark).
- [68] Juha K Laurila, Daniel Gatica-Perez, Imad Aad, Jan Blom, Olivier Bornet, Trinh-Minh-Tri Do, Olivier Dousse, Julien Eberle, and Markus Miettinen. The mobile data challenge : Big data for mobile computing research. In *Proceedings of the Workshop on the Nokia Mobile Data Challenge, in Conjunction with the 10th International Conference on Pervasive Computing*, pages 1–8, 2012.
- [69] Eugene L Lawler, Jan Karel Lenstra, AHG Rinnooy Kan, and David B Shmoys. *The traveling salesman problem : a guided tour of combinatorial optimization*, volume 3. Wiley Chichester, 1985.
- [70] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2) :498–516, 1973.
- [71] Steve Lohr. The age of big data. *New York Times*, 11, 2012.
- [72] Paul A Longley, Michael F Goodchild, David J Maguire, and David W Rhind. *Geographic information systems and science*. Wiley, 2005.
- [73] Clifford Lynch. Big data : How do your data grow ? *Nature*, 455(7209) :28–29, 2008.
- [74] Carlos M. Fonseca, Manuel López-Ibáñez, Luís Paquete, and Andreia P. Guerreiro. Computation of the hypervolume indicator. <http://iridia.ulb.ac.be/~manuel/hypervolume>, 2010. [Online ; accessed 6-May-2013].
- [75] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967.
- [76] J.R. McDonnell, R.G. Reynolds, and D.B. Fogel. *Evolutionary Programming IV : Proceedings of the Fourth Annual Conference on Evolutionary Programming March 1-3, 1995, San Diego, California*. A Bradford book. Mit Press, 1995.
- [77] Anuj Mehrotra. Constrained graph partitioning : decomposition, polyhedral structure and algorithms. 1992.
- [78] Anuj Mehrotra, Ellis L Johnson, and George L Nemhauser. An optimization based heuristic for political districting. *Management Science*, 44(8) :1100–1114, 1998.
- [79] P. Merz and B. Freisleben. Genetic local search for the tsp : New results. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 159–164. IEEE, 1997.
- [80] P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation*, 8(1) :61–91, 2000.

- [81] Herve Meunier, E-G Talbi, and Philippe Reininger. A multiobjective genetic algorithm for radio network optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 317–324. IEEE, 2000.
- [82] Microsoft. <http://www.microsoft.com/>. [Online ; accessed 9-April-2013].
- [83] MIKE2. Big data definition. [http://mike2.openmethodology.org/wiki/Big\\_Data\\_Definition](http://mike2.openmethodology.org/wiki/Big_Data_Definition). [Online ; accessed 9-April-2013].
- [84] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7(1) :65–85, 1988.
- [85] Tadahiko Murata, Hisao Ishibuchi, and Hideo Tanaka. Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers & Industrial Engineering*, 30(4) :957–968, 1996.
- [86] D. Nagesh Kumar. Classification of optimization problems. University Lecture (NPTEL).
- [87] T. Nishizeki and M.S. Rahman. *Planar graph drawing*. Lecture notes series on computing. World Scientific Publishing Company Incorporated, 2004.
- [88] Stan Openshaw. A geographical solution to scale and aggregation problems in region-building, partitioning and spatial modelling. *Transactions of the institute of british geographers*, pages 459–472, 1977.
- [89] Oracle. Oracle database. <http://www.oracle.com/us/products/database/overview/index.html>. [Online ; accessed 25-May-2013].
- [90] Carlos Ordóñez and Edward Omiecinski. Frem : fast and robust em clustering for large data sets. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 590–599. ACM, 2002.
- [91] Kyungchul Park, Kyungsik Lee, Sungsoo Park, and Heesang Lee. Telecommunication node clustering with node compatibility and network survivability requirements. *Management Science*, 46(3) :363–374, 2000.
- [92] Fernando Tavares Pereira, José Figueira, Vincent Mousseau, and Bernard Roy. Multiple criteria districting problems, models, algorithms, and applications : The public transportation paris region pricing system. 2004.
- [93] Laurens Jan Pit. Parallel genetic algorithms. *MS (Computer Sci.) Dissertation*, 1995.
- [94] Colin R Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1) :5–13, 1995.
- [95] Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4) :376–384, 1991.
- [96] Alex Reisner. Ruby geocoder. <http://www.rubygeocoder.com/>. [Online ; accessed 6-May-2013].
- [97] H.C. Romesburg. *Cluster Analysis For Researchers*. Lulu Press, 2004.
- [98] S. Rudich and A. Wigderson. *Computational Complexity Theory*. IAS/Park City Mathematics Series. Amer Mathematical Society, 2004.
- [99] James Salvatore. Applying topology to data. [http://www.dyinglovegrape.com/topology\\_data\\_2.html](http://www.dyinglovegrape.com/topology_data_2.html). [Online ; accessed 16-April-2013].
- [100] Henry Sharp. Cardinality of finite topologies. *Journal of Combinatorial Theory*, 5(1) :82–86, 1968.

- [101] Ron Shonkwiler. Parallel genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 199–205. Citeseer, 1993.
- [102] M. Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2012.
- [103] Nidamarthi Srinivas and Kalyanmoy Deb. Muultiobjective optimization using non-dominated sorting in genetic algorithms. *Evolutionary computation*, 2(3) :221–248, 1994.
- [104] Ssolbergj. User ssolbergj. <http://commons.wikimedia.org/wiki/User:Ssolbergj>. [Online ; accessed 1-May-2013].
- [105] J. Star and J.E. Estes. *Geographic information systems : an introduction*. Number pt. 21 in Prentice Hall series in geographic information science. Prentice Hall, 1990.
- [106] Statbel. <http://statbel.fgov.be>. [Online ; accessed 6-May-2013].
- [107] R.E. Steuer. *Multiple criteria optimization : theory, computation, and application*. Wiley series in probability and mathematical statistics : Applied probability and statistics. Wiley, 1986.
- [108] T. Stützle. Local search algorithms for combinatorial problems. *Darmstadt University of Technology PhD Thesis*, 1998.
- [109] Michael W Sweeney. Geographic information systems. *Water environment research*, 71(5) :551–556, 1999.
- [110] E.G. Talbi. *Metaheuristics : From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. Wiley, 2009.
- [111] TED Talks. <http://www.ted.com/>. [Online ; accessed 9-April-2013].
- [112] Fernando Tavares-Pereira, José Rui Figueira, Vincent Mousseau, and Bernard Roy. Multiple criteria districting problems. *Annals of Operations Research*, 154(1) :69–92, 2007.
- [113] N. Ulder, E. Aarts, H.J. Bandelt, P. van Laarhoven, and E. Pesch. Genetic local search algorithms for the traveling salesman problem. *Parallel problem solving from nature*, pages 109–116, 1991.
- [114] Dirk Van Oudheusden, Dirk Cattrysse, and Tsippy Lotan. On the importance of districting and its potential impact on routing. In *World Transport Research : Selected Proceedings of the 8th World Conference on Transport Research*, number Volume 2, 1999.
- [115] David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithms : Analyzing the state-of-the-art. *Evolutionary computation*, 8(2) :125–147, 2000.
- [116] Tobias Wagner, Nicola Beume, and Boris Naujoks. Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In *Evolutionary Multi-Criterion Optimization*, pages 742–756. Springer, 2007.
- [117] Mitch Waldrop et al. Big data : wiomics. *Nature*, 455(7209) :22, 2008.
- [118] Weston.pace. K means example. [http://commons.wikimedia.org/wiki/File:K\\_Means\\_Example\\_Step\\_3.svg](http://commons.wikimedia.org/wiki/File:K_Means_Example_Step_3.svg), 2007. [Online ; accessed 16-April-2013].
- [119] Justin C Williams. Political redistricting : a review. *Papers in Regional Science*, 74(1) :13–40, 1995.
- [120] R. Xu and D. Wunsch. *Clustering*. IEEE Press Series on Computational Intelligence. Wiley, 2008.

- [121] Blaise Aguera y Arcas. Blaise aguera y arcas demos augmented-reality maps. [http://www.ted.com/talks/blaise\\_aguera.html](http://www.ted.com/talks/blaise_aguera.html), 2010. [Online ; accessed 9-April-2013].
- [122] Betul Yagmahan and Mehmet Mutlu Yenisey. Ant colony optimization for multi-objective flow shop scheduling problem. *Computers & Industrial Engineering*, 54(3) :411–420, 2008.
- [123] Chiyuan Zhang. Deciding the number of clusterings. <http://freemind.pluskid.org/machine-learning/deciding-the-number-of-clusterings/>, 11 2012. [Online ; accessed 16-April-2013].
- [124] S. Zionts. Mcdm—if not a roman numeral, then what ? *Interfaces*, 9(4) :94–101, 1979.
- [125] Eckart Zitzler. Evolutionary algorithms for multiobjective optimization. *Methods and Applications PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland*, 1999.
- [126] Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 832–842. Springer, 2004.
- [127] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and V Grunert da Fonseca. Performance assessment of multiobjective optimizers : An analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2) :117–132, 2003.