Méthodes d'optimisation combinatoire en programmation mathématique : Application à la conception des systèmes de verger-maraîcher

Sara Maqrot

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *23/09/2019* **par :**

Sara MAQROT

Méthodes d'optimisation combinatoire en programmation mathématique. Application à la conception des systèmes de verger-maraîcher

---

JURY

Marie-Jo Huguet	Professeure des universités	Présidente
Philippe VISMARA	Professeur des universités	Rapporteur
Van-Dat CUNG	Professeur des universités	Rapporteur
Lakhdar SAIS	Professeur des universités	Examinateur
Marc TCHAMITCHIAN	Directeur de recherche	Membre invité
Gauthier QUESNEL	Chargé de recherche	Co-directeur de thèse
Simon DE GIVRY	Chargé de recherche HDR	Directeur de thèse

École doctorale et spécialité :
MITT : Domaine STIC : Intelligence Artificielle

Unité de Recherche :
Unité de Mathématiques et Informatique Appliquées de Toulouse (MIAT - INRA Toulouse)

Directeur(s) de Thèse :
Simon DE GIVRY, *Gauthier QUESNEL* et *Marc TCHAMITCHIAN*

Rapporteurs :
Philippe VISMARA et *Van-Dat CUNG*

À mes parents, mon mari et ma fille Inès.

Remerciements

Je voudrais tout d'abord remercier grandement mon directeur de thèse Simon DE GIVRY d'avoir dirigé ma thèse avec une forte implication et une grande disponibilité. Je lui remercie également pour ses qualités humaines d'écoute et de compréhension.

Je remercie aussi mes co-directeurs de thèse Gauthier QUESNEL et Marc TCHAMITCHIAN pour leurs aides précieux, leurs conseils pertinents et leurs suggestions toujours avisées.

Mes remerciements vont également aux rapporteurs Philippe VISMARA et Van-Dat CUNG de m'avoir fait l'honneur d'une lecture détaillée et rigoureuse de mon manuscrit de thèse. leurs remarques, commentaires et propositions d'améliorations m'ont permis de prendre du recul sur mon travail.

J'exprime aussi ma profonde reconnaissance à Marie-Jo HUGUET, présidente du jury de thèse, et Lakhdar SAIS pour l'intérêt qu'ils ont porté à mes travaux en examinant mon manuscrit de thèse.

Merci également à Robert FAIVRE et Victor PICHENY pour leur aide sur la partie de l'analyse de sensibilité.

Merci à tout le personnel de l'unité MIAT : le directeur pour sa flexibilité et son bon sens de l'humour, les gestionnaires administratifs (Fabienne, Nathalie et Alain) pour leur grande serviabilité, mes collègues de bureau (Charlotte, Alyssa et Léonard) pour leur soutien aux moments difficiles, et toutes autres personnes dont les permanents, les CDD, les anciens et futurs docteurs ainsi que les stagiaires pour les échanges amicaux et le partage des savoirs scientifiques tout au long de ces formidables années de thèse.

Je tiens à remercier le MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE qui a financé cette thèse en m'accordant un poste de Doctorant Contractuel Chargé d'Enseignement, puis d'Attachée Temporaire d'Enseignement et de la Recherche.

Je tiens également à remercier mes amis et mes collègues à l'UT2J qui m'ont apporté leur soutien moral pendant la phase dure de la rédaction du manuscrit.

Mes remerciements les plus chaleureux vont aussi à ma famille, en particulier à mes parents, toujours présents pour soutenir mes ambitions et m'aider à réaliser mon rêve de décerner le grade de Docteur.

Je remercie également mon cher époux pour son soutien quotidien, ses encouragements et sa patience. Gérer deux projets scientifiques en tant que thésards avec la naissance de notre fille Inès n'était pas facile pour nous.

Sans oublier Ghizlane, qui s'est déplacée plusieurs fois en France pour m'aider à garder ma petite lors de la rédaction du manuscrit. Je lui adresse toute ma gratitude.

Table des matières

Liste des figures

Liste des tableaux

Introduction Générale

Contexte

Aujourd'hui, l'agriculture est confrontée à plusieurs problèmes agronomiques, environnementaux et socio-économiques, comme le changement climatique, l'utilisation massive des ressources (eau, énergie) et la baisse de la productivité agricole. Pour résoudre ces problèmes, de nombreuses études ont été menées proposant des modèles conceptuels de gestion des systèmes de production agricole. Ces études visant à améliorer la productivité favorisent souvent l'émergence des problèmes environnementaux. Par exemple, l'utilisation excessive de produits chimiques dans les systèmes de culture intensive contribue au dérèglement des écosystèmes. Dans ce cadre, les systèmes de vergers-maraîchers associant arbres fruitiers et cultures maraîchères sur la même parcelle est une pratique intéressante qui répond positivement à ces enjeux. Ils combinent des productions à la fois diversifiées et relativement intensifiées, leur permettant de s'insérer en périphérie urbaine. Les interactions complexes entre arbres et cultures conduisent à utiliser moins d'engrais chimiques et pesticides et à mieux gérer la ressource en eau. De plus, la présence d'arbres fruitiers favorise la biodiversité. L'enjeu finalisé de la thèse est le passage à l'échelle de la parcelle avec une discrétisation beaucoup plus fine du sol (de l'ordre du mètre carré). Les interactions entre arbres et cultures vont évoluer dans le temps et dans l'espace (en se développant, l'arbre touche une part plus grande de la parcelle et son impact en termes de production et de moyens mis en œuvre n'intervient que tardivement) rendant le problème beaucoup plus complexe.

Problématique de recherche

Le problème de conception d'un système de verger-maraîcher consiste à définir l'emplacement et le choix des arbres fruitiers ainsi que la rotation des cultures maraîchères sur un horizon temporel de plusieurs années. Ces systèmes agroforestiers reposent sur un ensemble complexe d'interactions modifiant l'utilisation de la lumière, de l'eau et des nutriments. La conception d'un tel système doit donc optimiser l'utilisation de ces ressources en maximisant les interactions positives (facilitation) et en minimisant celles négatives (compétition). Ce problème a été modélisé en un problème de satisfaction de contraintes pondérées [Tchamitchian and Godin, 2014]. La grande taille de ce problème nécessite la mise au point des méthodes d'optimisation efficaces.

Dans cette thèse, nous nous intéressons à une méthode approchée appelée Wedelin ou in-the-middle développée par Dag Wedelin [Wedelin, 1995b] pour résoudre des problèmes de programmation linéaire en nombres entiers. Cette heuristique, basée sur la relaxation lagrangienne, a l'avantage de pouvoir s'appliquer à des problèmes de très grande taille (plusieurs millions de variables et des dizaines de milliers de contraintes pour le problème de planification de rotations d'équipages par exemple) donnant des solutions approchées souvent de bonne qualité. Cette méthode est implémentée dans Carmen system

[Wedelin, 1995b], un outil utilisé par la majorité des entreprises aériennes en Europe pour planifier les rotations d'équipages. Le succès que connaît la méthode in-the-middle dans l'aéronautique pour résoudre ce problème de rotation d'équipage nous motive à l'appliquer dans l'agronomie pour résoudre le problème de planification des rotations de cultures dans le cadre de la conception d'un système de verger maraîcher.

Les travaux de thèse consistent à étudier l'algorithme in-the-middle dans le cadre de la programmation linéaire en nombres entiers, choisir une stratégie de réglage de ses paramètres, développer et évaluer la performance de l'algorithme et tester sa qualité sur la résolution du problème de conception de vergers maraîchers. Il s'agit donc de développer une méthode générique capable de fournir de bonnes solutions en temps limité pour la résolution approchée de systèmes agronomiques complexes de grande taille.

Contributions majeures de la thèse

Les contributions majeures de cette thèse peuvent être divisées en deux catégories :

Développement d'une méthode générique basée sur l'algorithme de Wedelin et réglage de ses paramètres. Nous avons développé un solveur open source, BARYONYX, qui est une version parallèle de l'heuristique de Wedelin (généralisée). Pour régler ses paramètres, nous avons utilisé l'analyse de sensibilité pour identifier les paramètres les plus influents. Une fois trouvés, nous avons fixé les autres et utilisé un algorithme génétique pour régler les plus importants sur un ensemble d'instances d'entraînement. Le jeu de paramètres optimisé peut alors se généraliser pour résoudre d'autres instances de plus grande taille du même type de problème [Maqrot et al., 2018a].

Modélisation du problème de conception de verger-maraîcher et sa formulation mathématique. Dans [Maqrot et al., 2016, Maqrot et al., 2017a], nous avons modélisé le problème de verger-maraîcher comme un problème d'allocation des arbres et des cultures dans les dimensions spatio-temporelles et proposé une formulation quadratique en variables binaires (BQP). Pour améliorer la résolution, nous avons reformulé le problème en un problème linéaire en variables mixtes [Maqrot et al., 2017b] afin d'utiliser la méthode de Décomposition de Benders, qui consiste séparer les variables du problème en deux niveaux de décision (faciles et contrariantes). Ensuite, nous avons proposé une reformulation linéaire en variables binaires pour pouvoir appliquer notre méthode générique basée sur l'heuristique de Wedelin [Maqrot et al., 2018b].

Organisation du manuscrit

La suite du manuscrit est divisé en trois parties.

Partie I est un état de l'art qui se compose de trois chapitres :

- Chapitre 1 présente les notions de base et les concepts utiles d'optimisation combinatoire pour la compréhension de la thèse avec un panorama de méthodes d'optimisation adaptées à la résolution des problèmes d'optimisation discrète en variables 0-1.
- Chapitre 2 définit les problèmes classiques de partitionnement (comme le problème de Set Partitioning) en discutant les méthodes utilisées dans la littérature pour leur résolution.

- Chapitre 3 définit le principe de base de l'analyse de sensibilité et présente une grille de sélection d'une méthode d'analyse de sensibilité. Il présente également un ensemble de méthodes de réglage automatique des paramètres pour un problème d'optimisation.

Partie II sur l'algorithme de Wedelin se compose de deux chapitres :

- Chapitre 4 décrit le principe de base et le fonctionnement de l'algorithme de Wedelin avec les différentes améliorations menées. Il présente les différents paramètres de l'algorithme en discutant la difficulté de leur réglage.
- Chapitre 5 présente le réglage automatique des paramètres du solveur BARYONYX et ses extensions. Il présente également une comparaison de sa performance avec celles des autres logiciels d'optimisation, sur un ensemble d'instances recueillies de plusieurs collections de problèmes.

Partie III sur le problème de conception de verger-maraîcher :

- Chapitre 6 introduit certains concepts agronomiques indispensables à la compréhension du manuscrit.
- Chapitre 7 décrit les choix effectués pour la construction des modèles de conception des systèmes de vergers-maraîchers. Il présente ensuite les modèles mathématiques étudiés pour la reformulation du problème en discutant les résultats obtenus.

Première partie

État de l'art

Chapitre 1

Méthodes d'optimisation combinatoire en programmation mathématique

Sommaire

1.1 Introduction

L'optimisation combinatoire (ou discrète) est une branche très importante en recherche opérationnelle, en mathématiques appliquées et en informatique. Elle comprend un grand nombre de problèmes d'optimisation combinatoire difficiles [Papadimitriou and Steiglitz, 1998] issus d'applications réelles dans différents domaines tels que l'industrie, la finance ou l'armée [Paschos, 2013]. L'objectif de ces problèmes combinatoires consiste à trouver une meilleure solution dans un espace fini et discret de *solutions réalisables*, qui respectent un ensemble de conditions, dites aussi *contraintes*. L'évaluation d'une solution est effectuée à l'aide d'une fonction dite *fonction objectif*. Une meilleure alternative (*solution optimale*) est une solution réalisable qui minimise ou maximise, selon le contexte, la fonction objectif.

Face à la résolution d'un problème d'optimisation, il est important d'identifier à quelle classe de problèmes il appartient. Dans cette thèse, nous nous intéressons particulièrement à la classe des problèmes d'optimisation discrète en variables 0-1. Les algorithmes conçus pour résoudre ce type de problèmes sont nombreux. Ils s'organisent en deux catégories : les algorithmes exactes qui permettent d'obtenir des solutions dont l'optimalité est garantie ; et les algorithmes approchés qui cherchent des solutions de bonne qualité, sans garantie d'optimalité, mais au profit d'un temps de calcul plus réduit.

Dans ce chapitre, nous présentons les notions de base et les concepts utiles d'optimisation combinatoire pour la compréhension de la thèse. Nous illustrons également un ensemble de méthodes d'optimisation adaptées à la résolution des problèmes d'optimisation discrète en variables 0-1. Notons que nous allons considérer, pour les formules mathématiques, des notations matricielles et des notations générales (non matricielles).

1.2 Programmation linéaire en variables 0-1

Un problème de programmation linéaire (PL) en variables 0-1 est un problème d'optimisation où les variables sont binaires et la fonction objectif et les contraintes sont toutes linéaires. Sa formulation mathématique, dans sa forme canonique, est la suivante :

$$\max \quad \sum_{i=1}^{n} c_i x_i \tag{1.1}$$

$$s.c. \quad \sum_{i=1}^{n} a_{ki} x_i \leq b_k, \qquad k = 1, ..., m \tag{1.2}$$

$$x_i \in \{0, 1\} \qquad i = 1, ..., n \tag{1.3}$$

avec $c \in \mathbb{R}^n$ vecteur de coûts, $b \in \mathbb{Z}^m$ vecteur des termes constants des contraintes et $A \in \mathbb{Z}^{n \times m}$ matrice des coefficients des contraintes, où n est le nombre de variables et m est le nombre de contraintes.

Les éléments de l'ensemble $S = \{x \in \{0,1\}^n | Ax \leq b\}$ représentent les solutions réalisables du problème (un vecteur x qui respectent toutes les contraintes (1.2) et (1.3)). Les éléments de l'ensemble S qui maximisent la fonction objectif (1.1) sont les solutions optimales du modèle.

1.3 Programmation linéaire mixte

Un programme linéaire mixte en nombres entiers est un programme linéaire dans lequel les variables sont continues ou discrètes.

$$\max \quad \sum_{i=1}^{n} c_i x_i + \sum_{j=1}^{p} d_j z_j \tag{1.4}$$

$$s.c. \quad \sum_{i=1}^{n} a_{ki} x_i + \sum_{j=1}^{p} a'_{kj} z_j \leq b_k \qquad k = 1, ..., m \tag{1.5}$$

$$x_i \in \{0,1\} \qquad i = 1, ..., n \tag{1.6}$$

$$z_j \in \mathbb{R}_+ \qquad j = 1, ..., p \tag{1.7}$$

avec $c \in \mathbb{R}^n$, $d \in \mathbb{R}^p$, $b \in \mathbb{Z}^m$, $A \in \mathbb{Z}^{n \times m}$ et $A' \in \mathbb{Z}^{p \times m}$, où n est le nombre de variables discrètes, p est le nombre de variables continues et m est le nombre de contraintes. Les deux vecteurs $x \in \{0,1\}^n$ et $z \in \mathbb{R}_+^p$ représentent, respectivement, les variables discrètes et les variables continues.

$S = \{x \in \{0,1\}^n, z \in \mathbb{R}_+^p | Ax + A'z \leq b\}$ est l'ensemble des solutions réalisables.

1.4 Programmation quadratique en variables 0-1

Un problème de programmation quadratique (PQ) est un problème d'optimisation dans lequel on minimise une fonction objectif quadratique, non nécessairement convexe, sur un polyèdre convexe. La forme mathématique la plus générale pour un problème quadratique sous contraintes linéaires en variables 0-1 se présente de la manière suivante :

$$\max \quad \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} q_{ij} x_i x_j \tag{1.8}$$

$$s.c \quad \sum_{i=1}^{n} a_{ki} x_i \leq b_k \qquad k = 1, ..., m \tag{1.9}$$

$$x_i \in \{0,1\} \qquad i = 1, ..., n \tag{1.10}$$

où $Q^{n \times n}$ est une matrice carrée symétrique ($q_{ij} = q_{ji}$).

Rappel : un domaine de solutions admissibles S d'un problème linéaire est dit convexe lorsque, pour tous points x_1 et x_2 de S, le segment $[x_1, x_2]$ est tout entier contenu dans S. Cela se traduit pour un domaine $S \in \mathbb{R}^n$ par la formule suivante :

$$\forall x_1, x_2 \in S, \quad \forall \lambda \in [0,1] \quad \lambda x_1 + (1-\lambda) x_2 \in S \tag{1.11}$$

1.5 Linéarisation en variables 0-1

Une panoplie de méthodes d'optimisation sont conçues pour résoudre des problèmes linéaires (PL). Les appliquer sur un problème quadratique (PQ) nécessite une reformulation linéaire de la fonction objectif en introduisant des variables et des contraintes supplémentaires. Dans la majorité des cas, une reformulation linéaire est effectuée pour résoudre des problèmes non linéaires par des méthodes exactes de programmation linéaire en variables mixtes. Le but est d'améliorer la borne inférieure en renforçant les relaxations utilisées dans

un schéma classique de séparation et d'évaluation (Branch-and-Bound). [Plateau, 2006].

Or, dans nos travaux de thèse nous nous intéressons à la résolution des problèmes quadratiques en variables 0-1 par une heuristique de programmation linéaire en variables 0-1 (voir partie II) [Bastert et al., 2010]. Autrement dit, nous cherchons à reformuler la fonction objectif d'un problème quadratique en un problème linéaire en introduisant des variables 0-1.

La reformulation linéaire la plus utilisée est la linéarisation classique (ou standard) [Fortet, 1959, Fortet, 1960]. Elle consiste à remplacer chaque produit $x_i x_j$ par une variables additionnelle y_{ij} et d'ajouter les contraintes suivantes :

$$y_{ij} \leq x_i \qquad 1 \leq i < j \leq n \tag{1.12}$$
$$y_{ij} \leq x_j \qquad 1 \leq i < j \leq n \tag{1.13}$$
$$y_{ij} \geq x_i + x_j - 1 \qquad 1 < i \leq j \leq n \tag{1.14}$$
$$y_{ij} \geq 0 \qquad 1 \leq i < j \leq n \tag{1.15}$$

Les contraintes (1.12)-(1.15) forcent la variable y_{ij} à la valeur zéro si et seulement si l'une au moins des deux variables x_i ou x_j est égale à zéro ($y_{ij} = 0 \iff x_i = 0 \vee x_j = 0$). De la même façon, elles forcent la variable y_{ij} à la valeur un si et seulement si les deux variables x_i et x_j sont égales à un ($y_{ij} = 1 \iff x_i = 1 \wedge x_j = 1$). La reformulation linéaire obtenue est alors :

$$\max \quad \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} q_{ij} y_{ij} \tag{1.16}$$
$$s.c. \quad \sum_{i=1}^{n} a_{ki} x_i \leq b_k \qquad k = 1, ..., m \tag{1.17}$$
$$(1.12), (1.13), (1.14), (1.15)$$
$$y_{ij} = y_{ji} \qquad 1 \leq i < j \leq n \tag{1.18}$$
$$x_i, x_j \in \{0, 1\} \qquad i = 1, ..., n; j = 1, ..., n \tag{1.19}$$

Cela induit donc $\mathcal{O}(n^2)$ variables et $\mathcal{O}(n^2)$ contraintes.

À l'encontre de notre objectif, reformulation en variables 0-1, il existe d'autres méthodes de linéarisation qui reformulent le problème quadratique en un problème linéaire en variables mixtes (continues et discrètes). Nous citons la méthode RLT (Reformulation Linearization Technique) [Glover, 1975] qui ajoute $\mathcal{O}(n \times m)$ contraintes et la méthode compacte de Glover [Sherali and Adams, 2013] qui introduit $\mathcal{O}(n)$ contraintes.

1.6 Dualité

A tout problème d'optimisation linéaire correspond un autre problème linéaire, impliquant les mêmes données. Le premier problème est appelé *problème primal* et le second est appelé *problème dual*. Le passage du primal au dual est effectué de manière à ce que chaque contrainte du primal devient une variable du dual et chaque variable du primal devient une contrainte du dual. Nous inversons les coefficients de la fonction objectif et des contraintes, et nous prenons la transposée de la matrice des contraintes. Ainsi, les contraintes d'inégalité changent de sens. Le tableau 1.1 récapitule les différentes transformations du problème primal au problème dual.

Problème primal	problème dual
Objectif : maximisation	Objectif : minimisation
k^e contrainte d'inégalité ($\leq$)	k^e variable positive
k^e contrainte d'égalité	k^e variable quelconque
i^e variable positive	i^e contraintes d'inégalité ($\geq$)
i^e variable quelconque	i^e contrainte d'égalité
Coefficients de la fonction objectif : c_i	Termes constants des contraintes : b_i
Termes constants des contraintes : b_k	Coefficients de la fonction objectif : c_k
Matrice des coefficients de contraintes : A	Matrice des coefficients de contraintes : A^T

TABLE 1.1 – Transformation Primal-Dual
(avec les mêmes notations du problème (PL) à la section 1.4)

Pour illustrer cette reformulation, nous associons au problème linéaire primal (PL) (voir la section 1.2) le problème linéaire dual (1.20) suivant :

$$\begin{array}{lll} \min\limits_{\pi} & \sum_{k=1}^{m} b_k \pi_k & \\ s.c. & \sum_{k=1}^{m} a_{ki}\pi_k \geq c_i & i = 1, ..., n \\ & \pi_k \geq 0 & k = 1, ..., m \end{array} \tag{1.20}$$

où π_k sont les variables duales.

Selon les fondamentaux de la programmation linéaire, un problème linéaire est définit dans l'un (et seulement un) des trois cas : problème infaisable, problème non borné ou problème optimal. Pour un problème primal et son dual, seulement quelques combinaisons sont possibles. En effet, le théorème de dualité faible (théorème 1) montre que si le problème primal et son dual sont réalisables, alors ils sont bornés (chacun borne l'autre).

Théorème 1 ***(Dualité faible)*** *Pour un problème linéaire de maximisation, toute solution réalisable w_D du dual est une borne supérieure de n'importe quelle solution réalisable du problème primal w_P*

$$f_P(w_D) \geq f_D(w_P),$$

où f_P et f_D sont respectivement les fonctions objectifs du primal et du dual.

Une version plus forte du théorème de dualité (théorème 2) exclut la possibilité que l'un des problèmes soit infaisable et l'autre optimal (voir la démonstration des deux théorèmes de dualité dans [Thie and Keough, 2011]).

Théorème 2 ***(Dualité forte)*** *Si le problème primal admet une solution réalisable optimale w_P, alors le problème dual admet également une solution optimale w_D et nous avons*

$$f_P(w_P) = f_D(w_D).$$

Primal/Dual	optimal	non-borné	infaisable
optimal	possible (DF)	impossible (df)	impossible (DF)
non-borné	impossible (df)	impossible (df)	possible (df)
infaisable	impossible (DF)	possible (df)	possible

TABLE 1.2 – Lien primal-dual
((DF) signifie une dualité forte et (df) une dualité faible)

Toutes les autres combinaisons sont possibles. Le tableau 1.2 résume les différentes possibilités.

1.7 Relaxation Lagrangienne

La relaxation est une manipulation classique en optimisation combinatoire. Elle est appliquée pour fournir des bornes intéressantes (borne inférieure en minimisation et borne supérieure en maximisation) pour la résolution des problèmes linéaires en nombres entiers. Elle existe en deux types :

1. la relaxation des contraintes et en particulier la *relaxation linéaire*, permet d'élargir l'ensemble des solutions sur un espace plus large et facile à optimiser, en relâchant les contraintes d'intégrité.

$$x \in \{0,1\}^n \Rightarrow x \in [0,1]^n$$

2. Une autre piste consiste à modifier la fonction objectif. Une des techniques les plus utilisées est la *Relaxation Lagrangienne*. L'idée est de supprimer les contraintes difficiles en les intégrant dans la fonction objectif sous forme de combinaisons linéaires, où les coefficients sont des *multiplicateurs de Lagrange* (ou *multiplicateurs Lagrangiens*). Ces derniers pénalisent la fonction objectif si l'une des contraintes intégrées est violée.

Considérons le problème linéaire (P) en variables 0-1 suivant

$$Z : \min \quad \sum_{i=1}^{n} c_i x_i \tag{1.21}$$

$$s.c. \quad \sum_{i=1}^{n} a_{ji} x_i = b_j, \qquad j = 1, ..., m \tag{1.22}$$

$$\sum_{i=1}^{n} h_{li} x_i \leq e_l, \qquad l = 1, ..., k \tag{1.23}$$

$$x_i \in \{0,1\} \qquad i = 1, ..., n \tag{1.24}$$

avec $c \in \mathbb{R}^n$, $b \in \mathbb{Z}^m$, $e \in \mathbb{Z}^k$, $A \in \mathbb{Z}^{n \times m}$ et $H \in \mathbb{Z}^{n \times k}$. Admettons que l'ensemble des contraintes (1.22) est l'ensemble des contraintes difficiles. Soit $\lambda \in \mathbb{R}^m$ un vecteur de multiplicateurs Lagrangiens. Nous définissons la Relaxation Lagrangienne (RL), du problème

(P), associée aux contraintes difficiles (1.22) comme suit :

$$L(\lambda) : \min \quad \sum_{i=1}^{n} c_i x_i + \sum_{j=1}^{m} \lambda_j (\sum_{i=1}^{n} a_{ji} x_i - b_j) \tag{1.25}$$

$$s.c. \qquad \sum_{i=1}^{n} h_{li} x_i \leq e_l, \qquad l = 1, ..., k \tag{1.26}$$

$$x_i \in \{0,1\} \qquad i = 1, ..., n \tag{1.27}$$

Notons que, contrairement à la programmation linéaire, la programmation linéaire en variables binaires n'a pas de dualité forte (voir théorème 2 section 1.6). Cela veut dire que le problème dual Lagrangien (D)

$$L_\lambda = \max_{\lambda \in \mathbb{R}_+^m} L(\lambda) \tag{1.28}$$

et le problème primal initial (P) n'ont pas forcément la même valeur optimale. Selon le théorème de dualité faible (voir théorème 1 section 1.6), au lieu de $L(\lambda) = Z$, nous avons alors $L(\lambda) \leq Z$. Soit x^* une solution optimale au problème primal (P). Nous observons que

$$L(\lambda) \leq \sum_{i=1}^{n} c_i x_i^* + \sum_{j=1}^{m} \lambda_j (\sum_{i=1}^{n} a_{ji} x_i^* - b_j) = Z, \qquad j = 1, ..., m. \tag{1.29}$$

L'égalité dans cette équation provient du fait que $Z = \sum_{i=1}^{n} c_i x_i^*$ et $\sum_{i=1}^{n} a_{ji} x_i^* - b_j = 0$ pour tout $j = 1, ..., m$. Si les contraintes d'égalité (1.22) sont remplacées par des contraintes d'inégalité inférieure $\sum_{i=1}^{n} a_{ji} x_i \leq b_j$ dans (P), le vecteur de multiplicateurs Lagrangiens devient positif $\lambda \in \mathbb{R}_+^m$ et nous avons

$$L(\lambda) \leq \sum_{i=1}^{n} c_i x_i^* + \sum_{j=1}^{m} \lambda_j (\sum_{i=1}^{n} a_{ji} x_i^* - b_j) \leq Z, \qquad j = 1, ..., m, \tag{1.30}$$

où la seconde inégalité se justifie par $Z = \sum_{i=1}^{n} c_i x_i^*$, $\lambda_j \geq 0$ et $\sum_{i=1}^{n} a_{ji} x_i^* \leq b_j$ pour tout $j = 1, ..., m$. Similairement, pour les contraintes d'inégalité supérieure $\sum_{i=1}^{n} a_{ji} x_i \geq b_j$, le vecteur de multiplicateurs Lagrangiens devient négatif $\lambda \in \mathbb{R}_-^m$ pour avoir $L(\lambda) \leq Z$.

En général, trouver un vecteur λ pour $L(\lambda) = Z$ n'est pas garanti. Le fait que $L(\lambda) \leq Z$ nous permet d'utiliser la Relaxation Lagrangienne, au lieu de la relaxation linéaire, pour fournir des bornes inférieures (supérieures en cas de maximisation) au problème primal (P) dans l'algorithme de Branch and Bound (voir section 1.8).

1.7.1 Résolution du problème dual Lagrangien

Pour simplifier, nous supposons que le problème primal est faisable et que l'ensemble $X = \{x \in \{0,1\}^n | \sum_{i=1}^{n} h_{li} x_i \leq e_l, l = 1, .., k\}$ est un ensemble fini, $X = \{x^t, t = 1, ..., T\}$. Cela nous permet d'écrire $L(\lambda)$ de la manière suivante :

$$L(\lambda) : \min \quad \sum_{i=1}^{n} c_i x_i^t + \sum_{j=1}^{m} \lambda_j \times (\sum_{i=1}^{n} a_{ji} x_i^t - b_j), \qquad t = 1, ..., T \tag{1.31}$$

qui est une fonction concave et linéaire par morceaux. La figure 1.1 montre la forme de $L(\lambda)$ pour un exemple de $m = 1$ et $T = 4$.

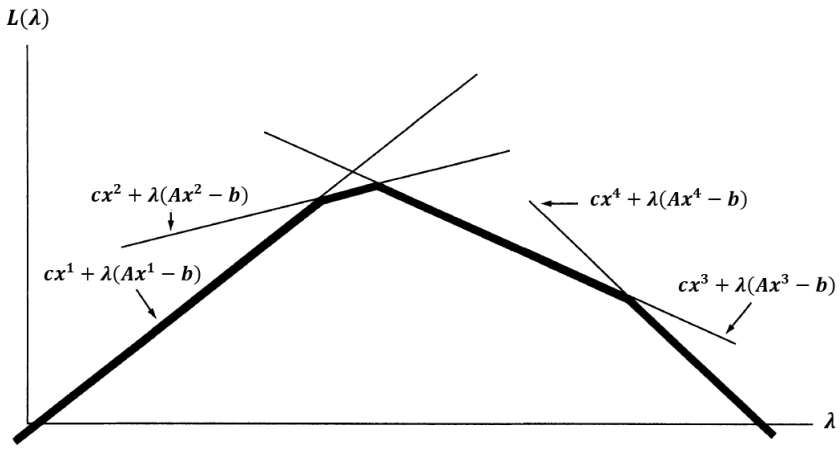


FIGURE 1.1 – La forme de la fonction $L(\lambda)$ pour m=1 et T=4. (Exemple tiré de [Fisher, 2004])

la fonction $L(\lambda)$ n'est pas différentiable partout. En conséquence, nous ne pouvons pas appliquer directement la méthode du gradient (voir section 1.8) pour maximiser les bornes inférieures obtenues, mais plutôt la méthode du sous-gradient (les gradients sont remplacés par des sous-gradients).

Définition 1
Nous appelons $y \in \mathbb{R}^m_+$ un sous-gradient de $L(\lambda)$ au $\overline{\lambda}$ s'il satisfait

$$L(\lambda) \leq L(\overline{\lambda}) + y(\lambda - \overline{\lambda}), \qquad \forall \lambda \in \mathbb{R}^m_+ \tag{1.32}$$

Soit $\lambda^0 \in \mathbb{R}^m_+$ une valeur initiale. La séquence $\{\lambda^k\}$ est générée selon la règle

$$\lambda^{k+1} = \lambda^k + t_k(Ax^k - b) \tag{1.33}$$

où $(Ax^k - b)$ est un sous-gradient au λ^k pour lequel x^k est une solution optimale au problème Lagrangien relaxé (RL) (équations (1.21) -(1.24)) et la valeur t_k représente la taille du pas. D'après [Fisher, 2004], la solution optimale $L(\lambda^k)$ est trouvée lorsque $t_k \to 0$ et $\sum_{i=1}^k t_i \to \infty$. En pratique, la taille du pas est souvent calculée par la formulation suivante :

$$t_k = \frac{\lambda^k(Z^* - L(\lambda^k))}{||Ax^k - b||^2}, \tag{1.34}$$

où $0 < \lambda^k \leq 2$ est un scalaire et Z^* est la borne supérieure de L_λ (voir équation 1.28). La démonstration de cette formule est donnée dans [Held et al., 1974].

Notons que la méthode du sous-gradient n'a aucun moyen pour prouver l'optimum, l'algorithme s'arrête alors lorsqu'il atteint un nombre maximum d'itérations fixé préalablement.

1.7.2 Exemple [Fisher, 1985]

Soit le problème linéaire en variables binaires suivant

$$
\begin{aligned}
Z = \max \quad & 16x_1 + 10x_2 + 4x_4 && (1.35)\\
s.c. \quad & 8x_1 + 2x_2 + x_3 + 4x_4 \leq 10 && (1.36)\\
& x_1 + x_2 \leq 1 && (1.37)\\
& x_3 + x_4 \leq 1 && (1.38)\\
& x \in \{0,1\}^4 && (1.39)
\end{aligned}
$$

En relâchant la contrainte (1.36), nous obtenons le problème Lagrangien relaxé suivant

$$
\begin{aligned}
L(\lambda) = \max \quad & (16-8\lambda)x_1 + (10-2\lambda)x_2 + (0-\lambda)x_3 + (4-4\lambda)x_4 + 10\lambda && (1.40)\\
s.c. \quad & x_1 + x_2 \leq 1\\
& x_3 + x_4 \leq 1\\
& x \in \{0,1\}^4
\end{aligned}
$$

où $\lambda \in \mathbb{R}_+$ est un multiplicateur lagrangien. Trouver la meilleure valeur de λ revient à résoudre le problème dual Lagrangien (le minimum des bornes supérieures)

$$L_\lambda = \min_{\lambda \geq 0} L(\lambda). \tag{1.41}$$

Avant d'appliquer un algorithme de résolution à ce problème (1.41), nous allons tester différentes valeurs de λ. Le tableau 1.3 liste sept valeurs de λ, avec la solution au problème Lagrangien relaxé associé, la borne supérieure $L(\lambda)$ et la valeur Z pour les solutions faisables dans le problème primal.

λ	x_1	x_2	x_3	x_4	$L(\lambda)$	Z
0	1	0	0	1	20	
6	0	0	0	0	60	0
3	0	1	0	0	34	10
2	0	1	0	0	26	10
1	1	0	0	0	18	16
1	1	0	0	1	18	
1	0	1	0	0	18	10
1	0	1	0	1	18	14
1/2	1	0	0	1	19	
3/4	1	0	0	1	18.5	

TABLE 1.3 – Solutions obtenues par la Relaxation Lagrangienne pour différentes valeurs de λ

Selon le tableau 1.3, la meilleure borne supérieure ($L(\lambda) = 18$) est obtenue avec $\lambda = 1$. Ce cas présente quatre alternatives de solutions au problème Lagrangien relaxé : trois solutions faisables ($Z = 10, 14, 16$) pour le problème primal et une solution infaisable. Cette borne ($L(\lambda) = 18$) représente la solution optimale au problème Lagrangien relaxé. Pour le montrer, il suffit d'observer que quelque soit la valeur de x, nous obtenons toujours une fonctions linéaire en λ. La figure 1.2 présente cette famille de fonctions linéaires pour toutes les solutions optimales au problèmes Lagrangien relaxé. Comme l'objectif Lagrangien est une maximisation, la fonction $L(\lambda)$ est alors définie sur l'enveloppe supérieur de ces fonctions linéaires, sous forme d'une fonction linéaire par morceaux (voir Figure 1.2).

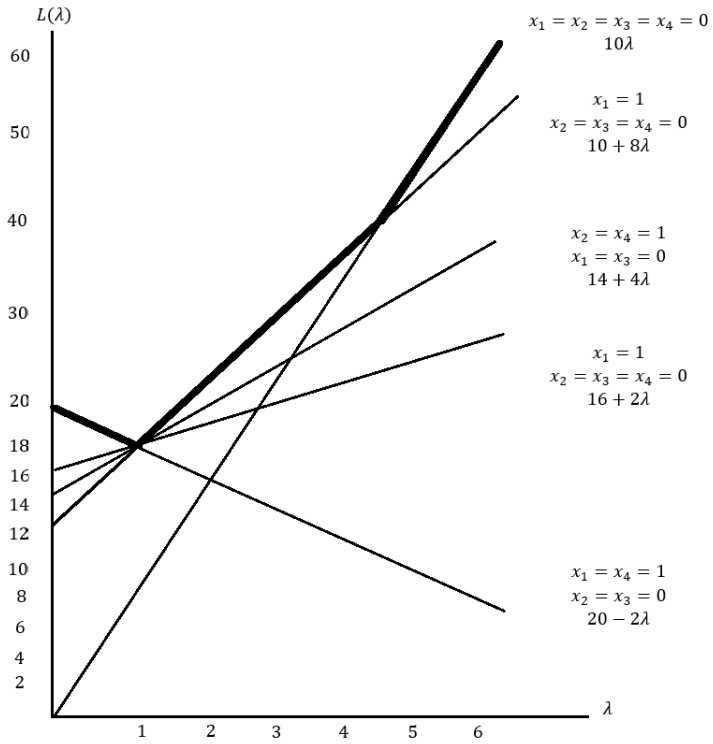


FIGURE 1.2 – La fonction $L(\lambda)$

Comme mentionné auparavant, la fonction $L(\lambda)$ est une fonction convexe et sous-différentiable. Elle n'est pas différentiable aux points où le problème Lagrangien relaxé a de multiples solutions (Par exemple, le point $\lambda = 1$, $L(\lambda) = 18$ sur la figure 1.2). A ces points, la dérivée de $L(\lambda)$ est définie comme suit $8x_1 + 2x_2 + x_3 + 4x_4 - 10$, où x est une solution optimale au problème Lagrangien relaxé. Donc, pour minimiser $L(\lambda)$, i.e. résoudre le problème (1.41), nous appliquons la méthode du sous-gradient (voir la section précédente 1.7.1). Pour résumer, la méthode consiste à trouver une séquence de valeurs de λ en commençant avec une valeur initiale λ^0 et en appliquant la formule suivante

$$\lambda^{k+1} = max\{0, \lambda^k - t_k(b - Ax^k)\}. \tag{1.42}$$

$\lambda^0 = 0$	$\lambda^0 = 0$	$\lambda^0 = 0$
$\lambda^1 = 0 - (-2) = 2$	$\lambda^1 = 0 - (-2) = 2$	$\lambda^1 = 2$
$\lambda^2 = max\{0, 2 - 8\} = 0$	$\lambda^2 = max\{0, 2 - 1/2(8)\} = 0$	$\lambda^2 = max\{0, 2 - 1/3(8)\} = 0$
$\lambda^3 = 0 - (-2) = 2$	$\lambda^3 = 0 - 1/4(-2) = 1/2$	$\lambda^3 = 0 - 1/9(-2) = 2/9$
$\lambda^4 = max\{0, 2 - 8\} = 0$	$\lambda^4 = 1/2 - 1/8(-2) = 3/4$	$\lambda^4 = 2/9 - 1/27(-2) = 0.296$
	$\lambda^5 = 3/4 - 1/16(-2) = 7/8$	$\lambda^5 = 0.296 - 1/81(-2) = 0.321$
	$\lambda^6 = 7/8 - 1/32(-2) = 15/16$	$\lambda^6 = 0.321 - 1 - 243(-2) = 0.329$
		$\lambda^7 = 0.329 - 1/729(-2) = 0.332$

TABLE 1.4 – Les résultats de la méthode du sous-gradient, sur l'exemple, obtenus avec $t_k = 1$ pour la première colonne, $t_k = 1, 1/2, 1/4, 1/8, ...$ pour la deuxième colonne et $t_k = 1, 1/3, 1/9, 1/27, 1/81, ...$ pour la troisième colonne.

où t_k est la taille du pas et x^k la solution optimale du problème Lagrangien relaxé. Le tableau 1.4 présente les résultats de la méthode du sous-gradient sur l'exemple correspondant à trois procédures de réglage de t_k. Dans la première colonne, t_k est fixé à 1 pour toutes les itérations et nous remarquons que les résultats oscillent entre deux valeurs $\lambda = 0$ et $\lambda = 2$. Pour la deuxième colonne, t_k converge vers 0 avec un pas égal à 1/2 et les résultats convergent vers la solution optimale $\lambda = 1$. Dans la troisième colonne, t_k converge également vers 0 mais d'une manière rapide avec un pas égal à 1/3. Dans ce dernier cas, nous remarquons que λ converge vers 1/3 au lieu de converger vers 1. Nous pouvons retenir alors que si la taille du pas t_k converge rapidement vers 0, la méthode du sous-gradient pourra s'éloigner de l'optimum. Une condition nécessaire mais non suffisante pour que $L(\lambda)$ tend vers l'optimum est $\sum_{i=1}^{k} t_i \to \infty$. Cette condition est violée dans le troisième cas ($\sum_{i=1}^{k} t_i \to 2$) où le résultat s'éloigne de l'optimum ($\lambda = 1$).

Comme cité avant, la méthode du sous-gradient ne garantie pas d'avoir l'optimum. Dans cet exemple, nous avons obtenu une solution égale à $Z = 16$ et une borne supérieure égale à $L(\lambda = 1) = 18$. Pour prouver la meilleure solution entière ($Z^* = 16$), il faut continuer la recherche en appliquant la méthode arborescente Branch and Bound (avec des bornes fournies par la relaxation Lagrangienne), qui est coûteuse en terme de temps de calcul pour les problèmes de grande taille.

1.7.3 Relaxation Lagrangienne vs. relaxation linéaire

Pour comparer les bornes obtenues par la relaxation Lagrangienne avec celles fournies par la relaxation linéaire, nous considérons le problème linéaire ((1.35)-(1.39)) montré dans l'exemple précédent. Son problème dual linéaire s'écrit de la manière suivante :

$$\begin{aligned} \min \quad & 10\lambda + v_1 + v_2 + w_1 + w_2 + w_3 + w_4 \\ s.c. \quad & 8\lambda + v_1 + w_1 \geq 16 \\ & 2\lambda + v_1 + w_2 \geq 10 \\ & \lambda + v_2 + w_3 \geq 0 \\ & 4\lambda + v_2 + w_4 \geq 4 \\ & \lambda, v_1, v_2, w_1, ..., w_4 \geq 0 \end{aligned} \tag{1.43}$$

où λ, v_1, v_2 représentent respectivement les variables duales des contraintes (1.36), (1.37), (1.38) et w_j les variables duales des contraintes $x_j \leq 1$ (1.39).

La solution optimale au problème primal est $x_1 = 1$, $x_2 = 0$, $x_3 = 0$, $x_4 = 1/2$ (n'est pas entière) et la solution optimale au problème dual est $\lambda = 1$, $v_1 = 8$, $v_2 = w_1 = ... = w_4 = 0$. Les deux solutions donnent une valeur objectif égale à $Z_{LP} = 18$.

Nous avons donc obtenu, avec la relaxation linéaire ($Z_{LP} = 18$), la même borne supérieure trouvée par la relaxation Lagrangienne. De plus, la variable duale $\lambda = 1$ de la contrainte (1.22) a exactement la même valeur trouvée par le problème dual Lagrangien. Cela est justifié par un ensemble de relations entre les problèmes d'optimisation. Nous avons :

$$\begin{aligned} L_\lambda &= \min\{\max\{cx + \lambda(b - Ax) : Hx \leq e, x \geq 0 \text{ et entier}\}\} \\ &\leq \min\{\max\{cx + \lambda(b - Ax) : Hx \leq e, x \geq 0\}\}. \end{aligned} \tag{1.44}$$

Soit v les variables duales associées aux contraintes $Hx \leq e$. En remplaçant (deux fois) le

problème primal par son dual, nous obtenons :

$$\begin{aligned} L_\lambda &\leq \min\{\min\{\lambda b + ve : \lambda A + vH \geq c, \lambda, v \geq 0\}\} \\ &= \max\{cx : Ax \leq b : Hx \leq e, x \geq 0\} \\ &= Z_{LP} \end{aligned} \tag{1.45}$$

Ces relations montrent que si le problème Lagrangien ne change pas après la suppression des contraintes d'intégrité, alors $L_\lambda = Z_{LP}$. C'est le cas avec l'exemple de [Fisher, 1985], la relaxation Lagrangienne donne toujours des valeurs entières aux variables que ce soit avec ou sans contraintes d'intégrité. Ce qui implique le résultat $L_\lambda = Z_{LP} = 18$.

Nous pouvons donc améliorer la valeur de la borne supérieure en utilisant la relaxation Lagrangienne qui a l'avantage (par rapport à la relaxation linéaire) de donner directement des solutions entières. Dans ce sens, nous présentons une méthode alternative de relaxation Lagrangienne de l'exemple

$$\begin{aligned} L(v_1, v_2) = \max \quad & (16 - v_1)x_1 + (10 - v_1)x_2 + (0 - v_2)x_3 + (4 - v_2)x_4 + v_1 + v_2 \\ s.c. \quad & 8x_1 + 2x_2 + x_3 + 4x_4 \leq 10 \\ & x \in \{0,1\}^4 \end{aligned} \tag{1.46}$$

où nous relaxons les contraintes (1.37) et (1.38) à l'aide des variables duales $v_1 \geq 0$ et $v_2 \geq 0$. Nous obtenons alors le problème classique du sac à dos. Selon les résultats montrés dans [Fisher, 1985], l'application de la méthode du sous-gradient trouve une solution duale ($x_1 = 1, x_2 = x_3 = x_4 = 0$) pour laquelle la borne supérieure ($L(v_1, v_2) = 16$) est égale à la solution du problème primal ($Z^* = 16$).

Cet exemple prouve que, avec un meilleur choix de contraintes à relaxer, la relaxation Lagrangienne peut fournir des bornes meilleures que celles données par la relaxation linéaire. De plus, les solutions qu'elle trouve sont entières (binaires dans notre cas) à l'encontre de la relaxation linéaire qui donnent des solutions fractionnaires.

1.7.4 Applications de la relaxation Lagrangienne

De nombreux problèmes combinatoires ont été résolus de façon exacte ou approchée par l'emploi de la relaxation Lagrangienne, à savoir le problème d'affectation généralisée [Fisher, 2004], le problème du voyageur de commerce [Desrosiers et al., 1988] et le problème de plus court chemin avec contrainte [Handler and Zang, 1980].

Dans la littérature, un grand nombre d'auteurs utilisent des heuristiques Lagrangiennes vu que la relaxation Lagrangienne basique ne garantit pas d'avoir une solution faisable au problème primal. Ces heuristiques sont en majorité utilisées dans une approche de Branch and Bound qui permet de trouver des solutions optimales. Nous citons par exemple [Holmberg and Yuan, 2000] qui proposent une heuristique Lagrangienne intégrée dans un schéma de Branch and Bound pour la résolution du problème de conception de réseau (network design). Or pour des problèmes de grandes tailles, ces méthodes arborescentes deviennent coûteuses en terme de temps de calcul. La solution est donc d'appliquer les heuristiques Lagrangiennes dans un outil de résolution pour fournir des solutions approchées de bonnes qualités en temps raisonnable.

Dans ce sens, [Seki et al., 2010] proposent la méthode de recherche à voisinage pour améliorer les solutions obtenues par la Relaxation Lagrangienne et l'appliquent

sur le problème de planification d'unités de production électrique (Unit Commitment Problem). [Klincewicz and Luss, 1986] décrivent une heuristique basée sur la relaxation Lagrangienne qui utilise la méthode de descente avec une application sur le problème de localisation d'entrepôts avec capacité (Capacitated Facility Location with Single-Source Constraints). [Xie and Turnquist, 2011] proposent une méthode hybride pour les problèmes d'optimisation des réseaux d'évacuation. La méthode combine la relaxation lagrangienne avec la recherche Tabou, la première décompose le problème en sous-problèmes Lagrangiens et la seconde résout ces derniers et adapte les multiplicateurs de Lagrange.

Dans cette thèse, nous nous intéressons particulièrement à la méthode de [Wedelin, 1995b] (voir partie II), qui est une heuristique générale pour les problèmes linéaires en variables bivalentes ayant une matrice de contraintes à coefficients 0/1. L'algorithme est basé sur la relaxation Lagrangienne avec la méthode de descente par coordonnée.

1.8 Méthodes de résolution

Un grand nombre de méthodes de résolution existe en recherche opérationnelle pour l'optimisation combinatoire. Nous distinguons deux grandes familles : les *méthodes exactes* qui garantissent la complétude de la résolution (optimalité) à des temps de calcul parfois prohibitifs ; et les *méthodes approchées* qui peuvent perdre la complétude de résolution mais qui fournissent souvent des solutions approchées de bonne qualité en temps raisonnable.

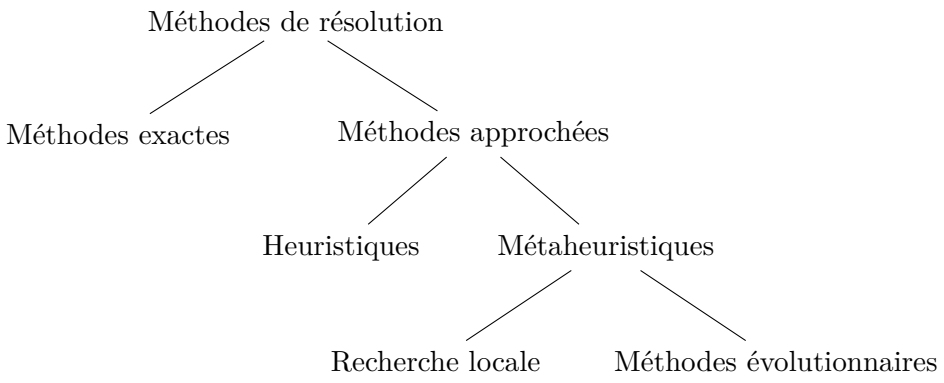


FIGURE 1.3 – Classification des méthodes d'optimisation

1.8.1 Méthodes exactes

Les méthodes exactes sont des méthodes de résolution qui permettent d'obtenir la solution optimale à un problème d'optimisation en parcourant, de manière implicite, toutes les combinaisons possibles.

Dans cette famille, nous citons les méthodes de séparation et évaluation (Branch and Bound) [Land and Doig, 1960]. Leur principe de base consiste à diviser le problème initial en un certain nombre de sous-problèmes pour constituer l'*arbre de recherche*. A chaque itération, un sous-problème est sélectionné 1) pour déterminer l'optimum de l'ensemble des solutions réalisables associées au sous-problème en question, ou 2) pour prouver que cet ensemble ne contient pas de solution optimale. La méthode la plus générale pour prouver

l'optimum consiste à déterminer un minorant (majorant pour une maximisation) des solutions contenues dans l'ensemble, en résolvant une relaxation du sous-problème courant. Si le minorant est supérieur à la valeur de la meilleure solution trouvée, cela affirme que le sous-ensemble ne contient pas l'optimum. Par contre, si les deux bornes supérieure et inférieure sont égales, alors une solution optimale est trouvée.

Plusieurs améliorations de l'algorithme de Branch and Bound ont été proposées. Nous citons la méthode de Branch and Cut [Mitchell, 2011] qui utilise, en plus de la méthode de séparation et évaluation, la méthode des plans sécants. L'objectif de cette dernière consiste à ajouter des contraintes au programme linéaire pour le raffiner et le rapprocher des solutions intégrales. Elle est appliquée au niveau de chaque noeud de l'arbre de recherche lorsque la résolution du problème relaxé donne une solution fractionnaire.

Comme inconvénient majeur, ces méthodes exactes d'énumération sont parfois caractérisées par un temps de calcul prohibitif. Pour réduire le temps d'évaluation d'un noeud et améliorer son efficacité, des méthodes de décomposition sont utilisées, comme la décomposition de Benders [Benders, 1962] et la décomposition de Dantzig-Wolfe [Dantzig and Wolfe, 1960]. Les problèmes décomposés avec ce dernier type utilisent dans leurs résolutions la méthode de génération de colonnes, appelée Branch and Price [Barnhart et al., 1998] lorsqu'elle est combinée avec la méthode de Branch and Bound. L'idée centrale de la méthode de génération de colonnes est de commencer la résolution, d'un problème linéaire de grande taille, avec un sous-ensemble de colonnes de petite taille. Puis de générer, au sein d'un algorithme à plusieurs étapes, les variables (colonnes) qui sont susceptibles d'améliorer la solution courante.

En général, le principe de décomposition est le même pour toutes les méthodes de décomposition : le problème initial est décomposé en un ou plusieurs sous-problèmes faciles, généralement coordonnés par un problème maître.

Nous présentons la décomposition de Benders [Benders, 1962], qui est une méthode itérative alternant entre la résolution d'un problème maître et la résolution d'un sous-problème. Le problème maître est en général plus facile à résoudre que le problème original car il s'agit d'une relaxation. Le sous-problème génère donc des coupes pour renforcer le problème maître jusqu'à ce que ce dernier soit équivalent au problème initial.

Considérons le problème mixte suivant :

$$\min \quad c^T x + d^T z \tag{1.47}$$

$$s.c. \quad Ax + A'z \leq b \tag{1.48}$$

$$x \in \{0,1\}^n \tag{1.49}$$

$$z \in \mathbb{R}^p_+ \tag{1.50}$$

Pour un vecteur fixe $\overline{x} \in \{0,1\}^n$, nous pouvons reformuler le problème de la manière suivante :

$$\min_{\overline{x} \in \{0,1\}^n} \{c^T \overline{x} + \min_{z \geq 0} \{d^T z : A'z \leq b - A\overline{x}\}\} \tag{1.51}$$

La minimisation à l'intérieur de l'équation (1.51) (le sous-problème) est un problème linéaire continu, que nous pouvons dualiser en introduisant un vecteur de variables duales $\pi \in \mathbb{R}^m$ associé à l'ensemble des contraintes $A'z \leq b - A\overline{x}$

$$\max_{\pi \in \mathbb{R}^m} \{\pi^T (b - A\overline{x}) : \pi^T A' \leq d\} \tag{1.52}$$

A partir des deux équations (1.51) et (1.52), nous pouvons déduire la formulation suivante :

$$\min_{\overline{x}\in\{0,1\}^n} \{c^T\overline{x} + \max_{\pi\in\mathbb{R}^m} \{\pi^T(b - A\overline{x}) : \pi^T A' \leq d\}\} \tag{1.53}$$

L'espace des solutions admissibles du sous-problème dual (1.52), i.e., $F = \{\pi | \pi^T A' \leq d\}$ ne dépend pas du choix de $\overline{x}$. Donc, si F est un ensemble non vide, le sous-problème est donc soit non borné ou réalisable quelle que soit la valeur de $\overline{x}$. En effet, si le sous-problème dual (1.52) est non borné, cela implique que le sous-problème primal (1.51) est infaisable (voir section 1.6). Pour éviter les mouvements dans la direction non bornée, nous ajoutons au problème maître une coupe de faisabilité

$$r_q^T(b - A\overline{x}) \leq 0, \quad \forall q \in Q \tag{1.54}$$

où $r_q, q \in Q$, est un rayon extrême non borné, Q est l'ensemble des rayons extrêmes. Or, si le sous-problème dual est réalisable, sa solution est donc un des points extrêmes $\pi_e, e \in E$, avec E est l'ensemble des points extrêmes. Le problème (1.53) devient :

$$\begin{aligned} \min_{\overline{x}\in\{0,1\}^n} \quad & c^T\overline{x} + \max_{e\in E} \{\pi_e^T(b - A\overline{x})\} \\ s.c. \quad & r_q^T(b - A\overline{x}) \leq 0, \quad \forall q \in Q \end{aligned} \tag{1.55}$$

Ainsi, en introduisant une variable continue $\eta \in \mathbb{R}$, la résolution du problème initial (1.47)-(1.50) devient équivalent à résoudre :

$$\min_{x,\eta} \quad c^T x + \eta \tag{1.56}$$

$$s.c. \quad \eta \geq \pi_e^T(b - Ax), \quad \forall e \in E \tag{1.57}$$

$$0 \geq r_q^T(b - Ax), \quad \forall q \in Q \tag{1.58}$$

$$x \in \{0,1\}^n \tag{1.59}$$

L'algorithme de décomposition de Benders est une approche itérative, il n'énumère pas toutes les coupes d'optimalité (1.57) et de faisabilité (1.58). Il résout le problème maître avec un sous-ensemble de coupes afin d'obtenir une valeur pour la variable x, i.e., $\overline{x}$, puis il résout le sous-problème (1.52) avec cette valeur $\overline{x}$. Si le sous-problème est faisable et borné, l'algorithme produit une coupe de type (1.57). Si le sous-problème est non borné, une coupe de type (1.58) est produite. Ensuite, si ces coupes sont violées par la solution courante, l'algorithme les insère directement dans le problème maître (voir Figure 1.4). La procédure se répète jusqu'à l'obtention d'une solution optimale (la solution du problème maître et celle du sous-problème sont égales).

Cet algorithme de décomposition de Benders est implémenté dans le solveur IBM ILOG CPLEX (voir la sous-section 1.8.3), à partir de sa version 12.7.0 [CPLEX, 2015].

Toutefois, malgré les progrès réalisés, les méthodes exactes deviennent inefficaces à mesure que la taille des problèmes devient importante, d'où la nécessité d'appliquer des méthodes approchées.

1.8.2 Méthodes approchées

Les méthodes approchées représentent une alternative pour résoudre les problèmes d'optimisation de grande taille lorsque les méthodes exactes échouent. Elles permettent d'obtenir des solutions de bonne qualité en un temps de calcul réduit. Elles sont fondées principalement sur des heuristiques, qui peuvent être spécifiques à un type de problème, telle

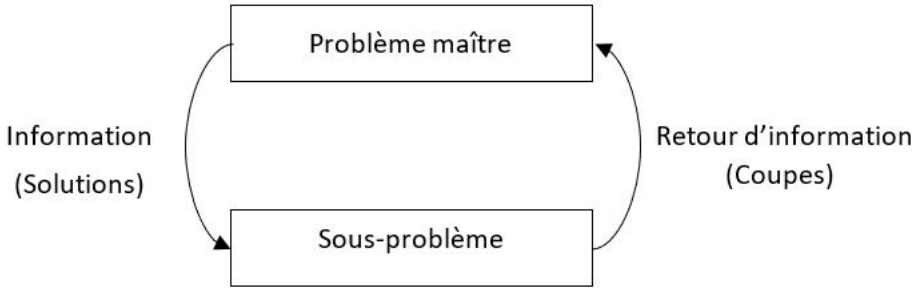


FIGURE 1.4 – Illustration d'une itération de l'algorithme de décomposition de Benders.

que la méthode de Wedelin (voir partie II) ou générales. Cette dernière catégorie appelée également métaheuristique est subdivisée en deux grandes familles : 1) les méthodes de recherche locale [Aarts et al., 2003] à solution unique telle que la méthode de descente, le Recuit Simulé et la recherche Tabou ; et 2) les méthodes évolutionnaires [Coello, 2005] à base de population de solutions comme les algorithmes génétiques (voir figure 1.3).

Méthode de descente (Hill Climbing)

La méthode de descente est une méthode classique de la recherche locale. Son principe général est le suivant : i) débuter avec une solution initiale x dont la valeur objectif est $f(x)$, ii) choisir une solution voisine x' de x telle que $f(x') < f(x)$, puis remplacer x par x' et répéter ii) jusqu'à ce que pour tout voisin x' de x, $f(x) \leq f(x')$. x est alors un optimum local. Pour trouver l'optimum global, une version améliorée de l'algorithme de descente (Iterated Local Search, ILS) consiste à changer la direction de recherche lorsqu'un optimum local est trouvé, en repartant d'une nouvelle solution générée aléatoirement.

Recuit simulé

Le recuit simulé est une métaheuristique inventée par les physiciens Kirkpatrick, Gellat et Vecchi en 1983 [Kirkpatrick et al., 1983]. Elle est inspirée du recuit des métaux en métallurgie dont le principe est d'alterner des cycles de refroidissement lent et de réchauffage. A la base, le refroidissement naturel de certains métaux ne permet pas aux atomes de se placer dans une configuration la plus solide. Lorsque le métal est réchauffé, l'énergie disponible rend aux atomes leur mobilité. Si le métal est refroidi lentement, la mobilité thermique diminue et les atomes cherchent à former de nouvelles liaisons. Physiquement, le mécanisme de minimisation de l'énergie repose sur la distribution de probabilité de Boltzmann $p(E) = \exp(-E/kT)$ où T est la température à l'équilibre thermique, k est une constante de Boltzmann et E représente l'état d'énergie. L'objectif final de ce processus est de parvenir, d'un état méta-stable correspondant à un minimum local d'énergie, à l'état stable de plus basse énergie. La probabilité qu'un système physique passe d'un niveau d'énergie E_1 à un niveau E_2 est donnée par la fonction $p = \exp[-(E_1 - E_2)/kT]$, qui est toujours inférieure à 1 si $E_1 \geq E_2$.
Pour résumer, la méthode du recuit simulé considère un processus d'exploitation du voisinage, qui permet de se diriger vers une solution voisine de moins bonne qualité avec une probabilité non nulle pour éviter les optima locaux. L'algorithme commence avec 1) une

température initiale $T = T_0$ qui décroît tout au long de l'algorithme pour tendre vers 0 et 2) une probabilité d'acceptation (des solutions moins bonnes) élevée. Plus la température T décroît, plus la probabilité p diminue. La difficulté du recuit simulé réside dans le choix de ses paramètres : la température initiale et la loi de décroissance de la température. Cette méthode peut nécessiter un critère d'arrêt lorsque la probabilité d'acceptation devient très faible ou lorsque la fonction d'évaluation cesse d'évoluer.

Recherche Tabou

La recherche Tabou est une méthode heuristique présentée par Glover [Glover, 1990]. Son principe de base consiste à poursuivre la recherche, même lorsqu'un optimum local est trouvé, en utilisant une notion de mémoire. Pour éviter les mouvements cycliques entre un optimum local et son voisin, la méthode considère une liste taboue contenant des mouvements ou des solutions qui sont temporairement interdits. La taille de cette liste, généralement gérée en FIFO (First In First Out), est un paramètre ajustable de l'heuristique. Également, cette stratégie nécessite de définir un critère d'arrêt (nombre d'itérations, temps de calcul, solution prouvée optimale, ...).

Algorithmes génétiques

Les algorithmes génétiques [Coello, 2005] sont des algorithmes d'optimisation fondés sur les mécanismes de la génétique. Ils simulent le processus d'évolution d'une population, où chaque individu représente une solution x. Le degré d'adaptation d'un individu à l'environnement est exprimé par une fonction de fitness $f(x)$. Le fonctionnement est simple. Nous partons d'une population de N individus créés aléatoirement. Chaque individu de la population est évalué par la fonction objectif. Ensuite, une stratégie de sélection choisit des individus pour former une population parent. Sur les individus de cette dernière, un opérateur de croisement est appliqué pour obtenir une population enfant, puis un opérateur de mutation est appliqué sur les individus de la population enfant. La nouvelle population obtenue par le choix de N individus parmi les populations parent et enfant est appelée génération suivante. De cette manière, nous produisons une population plus riche en individus qui sont mieux adaptés aux contraintes du problème. La figure 1.5 illustre le schéma général d'un algorithme génétique, où chaque itération représente une génération.

1.8.3 Logiciels d'optimisation

De nombreux solveurs ont été développés pour la résolution des problèmes combinatoires. Ces logiciels peuvent être libres à la disposition de la communauté scientifique ou commerciaux. Dans la première catégorie, nous présentons le solveur exact : TOULBAR2. Quant à la seconde catégorie, nous présentons le solveur IBM ILOG CPLEX basé sur les méthodes exactes et le solveur LOCALSOLVER basé sur la recherche locale, qui sont disponibles gratuitement pour l'usage académique.

TOULBAR2[1] est une plate-forme C++ d'optimisation combinatoire. Les contraintes et la fonction objectif sont exprimées sous forme de fonctions locales avec des variables discrètes et des coûts positifs dans $[0, \infty]$. TOULBAR2 cherche une affectation de variables qui minimise la somme de toutes les fonctions. Le logiciel repose sur l'algorithme de Branch and bound en utilisant la consistance d'arc. Il intègre également une méthode de recherche

[1] http://www7.inra.fr/mia/T/toulbar2/

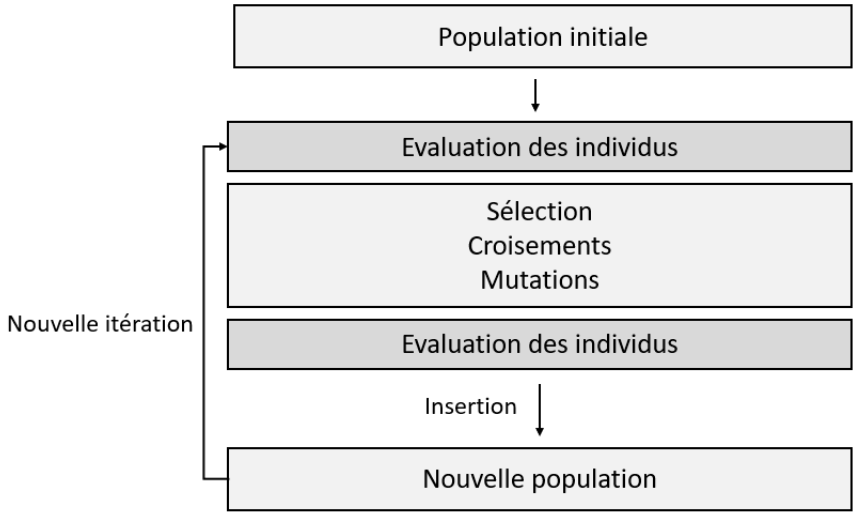


FIGURE 1.5 – Schéma général d'un algorithme génétique.

à voisinage [Ouali et al., 2017] et une méthode de recherche locale de type random walk [Neveu et al., 2004].

IBM ILOG CPLEX[2] est l'un des solveurs les plus utilisés dans la résolution des problèmes d'optimisation en programmation linéaire, mixte et quadratique. Il est commercialisé par la société ILOG depuis la version 6.0. Le logiciel repose sur des méthodes exactes et plus particulièrement sur les méthodes arborescentes. Il permet, à partir de sa version 12.7, d'effectuer une décomposition de Benders de manière (semi-)automatique pour la résolution des problèmes mixtes.

LOCALSOLVER[3] est un logiciel initialement développé pour la programmation 0-1 généralisée (non linéaire) [Benoist et al., 2011]. Il se base généralement sur des algorithmes de recherche locale. Par défaut, le logiciel utilise la méthode de descente comme stratégie de recherche. La méthode de recuit simulé est également disponible en spécifiant certaines options dans la ligne de commande. Les mouvements dans l'espace de recherche sont choisis d'une manière aléatoire avec une distribution non uniforme. Une version améliorée (version 4.0) permet d'attaquer des problèmes plus généraux avec des variables de décision mixtes, c'est-à-dire combinatoires et/ou continues.

1.9 Conclusion du chapitre

Dans ce premier chapitre, nous avons présenté les notions de base et les concepts d'optimisation combinatoire nécessaires à la compréhension de la thèse. Nous avons limité la description des concepts à la programmation en variables bivalentes. Nous avons commencé par définir la programmation linéaire, mixte et quadratique en variables 0-1. Puis nous avons montré comment linéariser une fonction objectif quadratique en variables 0-1, afin de résoudre des problèmes non-linéaires avec des méthodes de programmation linéaire.

La définition des fondamentaux telle que la dualité et la relaxation des contraintes nous a permis d'aborder les méthodes de résolution, que nous avons défini en deux grandes ca-

[2] https://www.ibm.com/fr-fr/products/ilog-cplex-optimization-studio
[3] http://www.localsolver.com

tégories : les méthodes exactes et les méthodes approchées.

Dans cette thèse, nous nous intéressons particulièrement à une heuristique spécifique, dite heuristique de Wedelin (voir partie II). Cette méthode résout des problèmes de programmation linéaire en variables 0-1 avec une forme mathématique spécifique, tel que le problème de Set Partitioning, que nous présentons dans le chapitre suivant.

Chapitre 2

Problèmes classiques de partitionnement

Sommaire

2.1 Introduction

Dans la littérature, de nombreux problèmes sont reformulés sous forme des problèmes de partitionnement : problème de Set Packing, de Set Covering et de Set Partioning. Les problèmes de planification (e.g., tournées de véhicules, ordonnancement et localisation) prennent souvent la structure d'un Set Covering lorsque l'objectif est de définir une couverture d'ensemble. Par exemple, assurer que chaque client est desservi par un véhicule, une personne ou un lieu. Toutefois, lorsque le client est desservi exactement une seule fois, comme dans le problème de la planification des rotations d'équipages où chaque vol est affecté à un seul équipage, le problème prend la forme d'un Set Partitioning. De même, les problèmes de planification qui consistent à satisfaire au maximum les demandes tout en évitant les conflits prennent souvent la forme d'un problème de Set Packing.

Dans ce chapitre, nous rappelons les définitions de ces trois problèmes (Set Packing, Set Covering et Set Paritioning) avec leurs formulations mathématiques. Nous présentons ensuite une version généralisée qui combine leurs objectifs, avec un cas particulier : problème des n reines pondérées. Puis nous discutons les méthodes de résolution qui sont largement utilisées dans la littérature pour résoudre cette famille de problèmes.

2.2 Problèmes de partitionnement

L'objectif commun des problèmes de partitionnement est de trouver une famille optimale des sous-ensembles d'éléments parmi un ensemble de base.

Considérons une matrice $A = (a_{ij}) \in \{0,1\}^{m \times n}$. Soit $I = \{1, ..., m\}$ l'ensemble des lignes et $J = \{1, ..., n\}$ l'ensemble des colonnes de la matrice A .

$$a_{ij} = \begin{cases} 1, & si \text{ la colonne } j \text{ couvre la ligne } i \\ 0, & sinon \end{cases}$$

Soit c_j le coût (et p_j le profit) de la colonne $j \in J$. Nous associons à chaque colonne j une variable binaire x_j telle que

$$x_j = \begin{cases} 1, & si \text{ la colonne } j \text{ est sélectionnée} \\ 0, & sinon \end{cases}$$

2.2.1 Problème de Set Packing (SSP)

Le premier problème que nous présentons est le problème de Set Packing, qui est un problème NP-difficile [Crescenzi and Kann, 1997, Cygan, 2013]. Son objectif est de déterminer un couplage de sous-ensembles disjoints, de profit maximal, à partir d'un ensemble donné. Un exemple d'application est celui de la mise en œuvre d'une planification des visites de touristes dans une zone géographique avec un maximum de satisfaction [Avella et al., 2006].

Le problème de Set Packing est formulé comme un programme linéaire en variables binaires.

$$\max \quad \sum_{j \in J} p_j x_j \tag{2.1}$$

$$s.c. \quad \sum_{j \in J} a_{ij} x_j \leq 1, \quad \forall i \in I \tag{2.2}$$

$$x_j \in \{0,1\} \quad \forall j \in J \tag{2.3}$$

L'objectif (2.1) maximise le profit total des colonnes sélectionnées. Les contraintes (2.2) imposent que les sous-ensembles soient disjoints et les contraintes (2.3) indiquent que les variables x_j soient binaires, i.e., une colonne j peut être sélectionnée (1) ou non (0).

2.2.2 Problème de Set Covering (SCP)

Le problème de Set Covering, aussi appelé problème de couverture d'ensemble est un problème NP-difficile [Crescenzi and Kann, 1997]. Son objectif est de déterminer une couverture de coût minimal. La littérature comprend un grand nombre de problèmes pratiques qui sont formulés comme un problème de couverture d'ensemble. Nous citons par exemple le problème de réaménagement de l'équipage ferroviaire qui a été reformulé comme un problème de Set Covering de grande taille [Huisman, 2007].

La formulation mathématique du problème de Set Covering est la suivante :

$$\min \quad \sum_{j \in J} c_j x_j \tag{2.4}$$

$$s.c. \quad \sum_{j \in J} a_{ij} x_j \geq 1, \quad \forall i \in I \tag{2.5}$$

$$x_j \in \{0,1\} \quad \forall j \in J \tag{2.6}$$

Les contraintes (2.5) assurent que les sous-ensembles couvrent tous les éléments de l'ensemble de base, i.e., chaque ligne i est couverte au moins une fois.

2.2.3 Problème de Set Partitioning (SPP)

Un problème lié au problèmes de Set Packing et de Set Covering est le problème de Set Partitioning, appelé également problème de partitionnement d'ensemble. Il consiste à déterminer un sous-ensemble, de coût minimal, qui couvre tous les éléments de l'ensemble de base, à condition que chaque élément est exactement couvert une seule fois. La formulation mathématique du problème de Set Partitioning sous forme d'un programme linéaire en variables bivalentes est la suivante :

$$\min \quad \sum_{j \in J} c_j x_j \tag{2.7}$$

$$s.c. \quad \sum_{j \in J} a_{ij} x_j = 1, \quad \forall i \in I \tag{2.8}$$

$$x_j \in \{0,1\} \quad \forall j \in J \tag{2.9}$$

Les contraintes (2.8) assurent que chaque ligne i soit couverte exactement par une seule colonne j.

En dépit de la simplicité de ce modèle mathématique, le problème de Set Partitioning est un problème NP-difficile [Rasmussen, 2011, Groiez, 2013]. Il représente l'un des problèmes les plus étudiés dans la littérature avec différentes applications pratiques. Le problème le plus connu par une reformulation en problème de Set Partitioning est celui

de la planification d'horaires d'équipages. Son objectif est de déterminer les équipages nécessaires pour assurer un certain nombre de vols d'avions, nous mentionnons par exemple [Hoffman and Padberg, 1993, Desaulniers et al., 1997].

2.2.4 Problème de Set Partitioning étendu

En examinant les trois ensembles de contraintes (2.2), (2.5) et (2.8), nous pouvons remarquer qu'un problème de Set Partitioning est aussi un problème de Set Covering et de Set Packing, mais l'inverse n'est pas nécessairement vrai. En effet, plusieurs auteurs ont démontré la transformation du problème de Set partioning en un problème de Set Covering ou en un problème de Set Packing [Balas and Padberg, 1976, Darby-Dowman and Mitra, 1983, Vemuganti, 1998]. Ces équivalences, cependant, ne s'appliquent que lorsque le problème de partitionnement possède une solution admissible, i.e. une couverture exacte. Malheureusement en pratique, lorsqu'un problème est formulé sous forme d'un problème de Set Partitioning, ce dernier est souvent sans solution. Une des raisons est la réduction du nombre de variables par les heuristiques utilisées pour obtenir un problème réduit en termes de taille et de temps de calcul. Face à cette situation du problème de Set Partitioning sans solution admissible (pas de couverture exacte), [Darby-Dowman and Mitra, 1983] ont développé un modèle étendu qui combine les trois problèmes : Set packing, Set Covering et Set Partitioning. L'idée de base est de permettre une relaxation de couverture des contraintes de partitionnement d'ensemble ; sous-couverture pour obtenir un Set Packing ou sur-couverture pour avoir un Set Covering. Donc, lorsqu'une couverture exacte n'existe pas, nous pouvons essayer de résoudre soit le problème de Set Packing ou le problème de Set Covering, selon ce qui convient le mieux à l'application.

La formulation étendue du problème de Set Partitioning, montrée ci-dessous, définit Set Packing, Set Partitioning et Set Covering comme des cas particuliers et englobe également un quatrième modèle qui autorise, en même temps, une sous-couverture et une sur-couverture. Le modèle est défini comme suit :

$$\min \quad \sum_{j\in J} c_j x_j + \sum_{i\in I} w_i^u u_i + \sum_{i\in I} w_i^o o_i \tag{2.10}$$

$$s.c. \quad \sum_{j\in J} a_{ij} x_j + u_i - o_i = 1, \qquad \forall i \in I \tag{2.11}$$

$$x_j \in \{0,1\} \qquad \forall j \in J \tag{2.12}$$

$$u_i \in \{0,1\} \qquad \forall i \in I \tag{2.13}$$

$$o_i \geq 0, \quad o_i \in \mathbb{N} \qquad \forall i \in I \tag{2.14}$$

avec $w_i^u \geq 0$ la pénalité associée à la sous-couverture de la $i^{ème}$ ligne et $w_i^o \geq 0$ la pénalité associée à la sur-couverture de la $i^{ème}$ ligne, où au moins une valeur des deux pénalités (w_i^u, w_i^o) est strictement positive. Les variables u_i et o_i sont définies comme suit :

$$u_i = \begin{cases} 1, & si \text{ la ligne } i \text{ n'est pas couverte} \\ 0, & sinon \end{cases}$$

$$o_i = \text{ le nombre de fois où la ligne } i \text{ est sur-couverte}$$

- **Set Packing :** Si la pénalité $w_i^o, i \in I$ est définie sur un nombre positif suffisamment grand (supérieur à $\sum_{j\in J} c_j$) et la pénalité $w_i^u, i \in I$ est mise à zéro, alors le problème de Set Partitioning étendu (avec $c_j = -p_j, j \in J$) et le problème de Set Packing ont la même solution optimale.

- **Set Covering :** Si la pénalité $w_i^u, i \in I$ est définie sur un nombre positif suffisamment grand et la pénalité $w_i^o, i \in I$ est mise à zéro, alors le problème de Set Partitioning étendu et le problème de Set Covering ont la même solution optimale.

- **Set Partitioning :** Si à la fois, les deux pénalités $w_i^o, i \in I$ et $w_i^u, i \in I$ sont définies sur des nombres positifs suffisamment grands, alors le problème de Set Partitioning et celui de Set Partitioning étendu ont la même solution optimale.

Un exemple d'application du modèle étendu de Set Partitioning est celui de [Darby-Dowman and Mitra, 1983], appliqué sur le problème de planification des horaires d'équipages. Son bjectif est de trouver l'ensemble des tâches les moins coûteuses de manière à ce que chaque voyage soit exactement couvert par un seul équipage.

2.2.5 Cas particulier : problème des n-reines pondérées

Le problème des n-reines pondérées est un cas particulier du problème de Set Partitioning étendu, où $o_i = 0$ et $u_i = 0$ dans les équations (2.10)- (2.11). C'est un problème classique d'optimisation combinatoire. Son objectif est de placer n reines dans un échiquier de taille $n \times n$ de manière à avoir une reine par rangée, colonne, et au plus une reine par diagonale ascendante et diagonale descendante. La figure 2.1 illustre un exemple de 8 reines ($n = 8$) non menacées. Chaque case dans l'échiquier est associée à un coût. Une solution optimale est donc de trouver une combinaison de reines avec un coût minimal. Cela se traduit par la formulation suivante :

$$\min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.15}$$

$$s.c. \quad \sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j = 1..n \tag{2.16}$$

$$\sum_{j=1..n} x_{ij} = 1, \qquad \forall i = 1..n \tag{2.17}$$

$$\sum_{j=1}^{n} \sum_{j=1, i+j=k}^{n} x_{ij} \leq 1, \quad \forall k = 2..2n \tag{2.18}$$

$$\sum_{j=1}^{n} \sum_{j=1, i-j=k}^{n} x_{ij} \leq 1, \quad \forall k = 1-n..n-1 \tag{2.19}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j = 1..n \tag{2.20}$$

Dans la littérature, plusieurs méthodes ont été proposées pour résoudre le problème des n reines (2.16)-(2.19), qui est un problème NP-complet [Gent et al., 2017]. Par exemple, [Kosters, 2013] propose 336 références, dont les plus récentes datent de 2018 [Grigoryan, 2018, Jha et al., 2018]. La majorité des méthodes proposées reposent sur la construction de solution, comme la méthode de backtracking (retour sur trace) qui testent systématiquement l'ensemble des affectations potentielles du problème [Bozinovski and Bozinovski, 2004]. L'inconvénient majeur de ces méthodes de construction est le temps de calcul vu que le nombre de solutions augmente de façon exponentielle avec l'augmentation du nombre de reines [Sosic and Jun Gu, 1994]. Par conséquent, trouver une affectation de reines (2.16)-(2.19) avec un coût minimal (2.15) devient un problème plus difficile. Il s'agit du problème des n reines pondérées qui est moins étudié dans littérature.

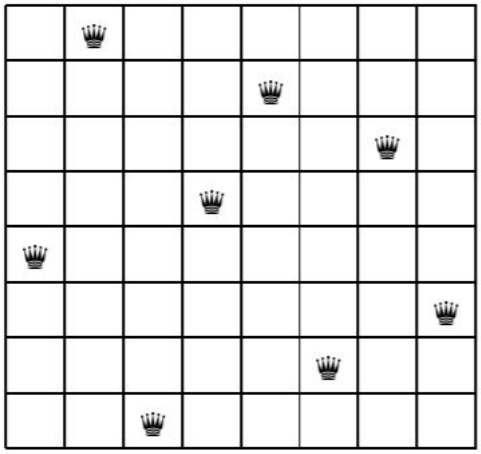


FIGURE 2.1 – Illustration d'une combinaison de 8 reines non menacées.

2.3 Méthodes de résolution des problèmes de partitionnement

Reformuler un problème sous forme d'un problème de Set Packing (SSP), de Set Covering (SCP) ou de Set Partitioning (SPP) est une technique classique et populaire, qui s'est avérée efficace pour résoudre beaucoup de problèmes combinatoires tel que les problèmes de planification des tournées de véhicules (Vehicle Routing and Scheduling Problem) [Agarwal et al., 1989, Bramel and Simchi-Levi, 1997, Kelly and Xu, 1999, Avella et al., 2004, Baldacci et al., 2008] et les problèmes de planification des horaires d'équipages (Crew Scheduling Problem) [Rubin, 1973, Smith and Wren, 1988, Hoffman and Padberg, 1993, Wedelin, 1995a, Mingozzi et al., 1999, Medard and Sawhney, 2007, Froger et al., 2015]. Ces reformulations sont largement abordées dans la littérature.

En ce sens, de nombreux algorithmes ont été développés pour résoudre les problèmes SSP, SCP et SPP. Les méthodes exactes arborescentes sont les plus utilisées : [Desaulniers et al., 1997] proposent de résoudre le problème de Crew Scheduling par la décomposition de Dantzig-Wolfe intégrée dans le schéma de Branch and Bound, où le problème maître est un SPP. [Borndörfer, 1998, Hoffman and Padberg, 1993] adoptent la procédure exacte de Branch and Cut. [Avella et al., 2006] utilisent la même procédure avec la méthode de génération de colonnes et de lignes pour réduire le nombre de variables. Également, plusieurs approches basées sur la génération de colonnes ont été suggérées. Par exemple, [Avella et al., 2004] pour un Set Packing, [Huisman, 2007] pour un Set Covering et [Agarwal et al., 1989] pour un Set Partitioning. De plus, d'autres méthodes ont été proposées pour améliorer la résolution de la génération de colonnes. Nous mentionnons par exemple [Elhallaoui et al., 2005] qui proposent une méthode d'agrégation dynamique de contraintes. Cette dernière permet de réduire le nombre de contraintes de partitionnement (dans le problème maître) qui génèrent une dégénérescence élevée. Pour le même objectif, traiter la dégénérescence, [Zaghrouti et al., 2014] propose une méthode de Simplexe intégrale basée sur une décomposition, qui a été ensuite appliquée au problème de la planification des horaires d'équipages [ROSAT, 2016].

Malheureusement, ces algorithmes exactes sont limités à des instances avec quelques centaines de lignes et quelques milliers de colonnes. Traiter donc des instances de grandes tailles nécessite d'appliquer une méthode approchée. En effet, plusieurs tra-

vaux dans la littérature ont été proposés pour résoudre ces problèmes NP-difficile. Une première piste consiste à appliquer des heuristiques à population de solutions : [Beasley and Chu, 1996, Chu and Beasley, 1998] proposent une version améliorée de l'algorithme génétique avec de nouveaux opérateurs de croisement et de mutation pour des problèmes de Set Covering et de Set Partitioning, et [Gandibleux et al., 2004] proposent l'algorithme de colonies de fourmis pour le problème de Set Packing. Ces méthodes fournissent des solutions de bonnes qualités mais au détriment du temps de calcul qui est très significatif [Chu and Beasley, 1998]. La seconde piste de résolution consiste à utiliser des heuristiques à solution unique. Nous citons la recherche tabou [Kelly and Xu, 1999, Yaghini et al., 2015], le recuit simulé [Brusco et al., 1999], la méthode GRASP (Greedy Randomized Adaptive Search Procedure) [Delorme et al., 2004], la recherche à voisinage [Umetani, 2017] et la méthode de descente par coordonnée [Wedelin, 1995a, Bastert et al., 2010].

Dans cette thèse, nous nous intéressons particulièrement à l'heuristique de Wedelin [Wedelin, 1995a], basée sur la relaxation Lagrangienne. A la différence de la méthode du sous-gradient souvent utilisée avec la relaxation Lagrangienne, l'heuristique de Wedelin utilise la méthode de descente par coordonnée pour résoudre le problème dual Lagrangien. Elle a été appliquée avec succès par de grandes compagnies aériennes (via le système Carmen) pour la planification des horaires d'équipages. Cette méthode est initialement développé pour résoudre des problèmes de type Set Partitionning. Elle a été ensuite réétudiée par [Bastert et al., 2010] qui ont proposé différentes extensions et généralisations (voir Partie II du manuscrit).

2.4 Conclusion du chapitre

Dans ce chapitre introductif, nous avons discuté différentes méthodes appliquées pour la résolution des problèmes de partitionnement. Cet état de l'art nous permet de choisir des méthodes de différentes catégories qui servent de comparaison pour évaluer la performance de notre version de l'algorithme de Wedelin. Par exemple, une heuristique à population de solution (algorithme génétique) [Beasley, 1990b], une heuristique à solution unique (4-flip recherche à voisinage) [Umetani and Yagiura, 2007], et une méthode exacte (branch and cut) [Borndörfer, 1998].

Chapitre 3

Réglage automatique des paramètres des méthodes d'optimisation

Sommaire

3.1 Introduction

Une des principales difficultés de l'utilisation des méthodes d'optimisation est le réglage des paramètres. Il est important, pour chaque problème posé, de trouver un jeu de paramètres qui conduit à des performances optimales. Choisir, manuellement, les meilleures valeurs demande beaucoup d'expérimentations. Ce qui est compliqué et long à faire. En effet, plusieurs méthodes existent permettant une configuration automatique des valeurs des paramètres. [Eiben et al., 1999] classe ces méthodes en deux catégories selon la manière d'utilisation, avant l'exécution de l'algorithme d'optimisation (réglage des paramètres) et pendant son exécution (contrôle des paramètres).

Les méthodes de contrôle consistent à auto-régler les valeurs des paramètres pendant la résolution et à contrôler dynamiquement ces valeurs au fur et à mesure de la recherche de solution. Ces techniques sont largement appliquées pour auto-régler les paramètres des algorithmes évolutionnaires [Hamadi, 2013], tels que le taux de croisement [Yang and Abbass, 2003], le taux de mutation [Smith and Fogarty, 1996] et la taille de population [Eiben et al., 2004] pour les algorithmes génétiques. D'après [Hinterding et al., 1997], ces méthodes peuvent encore être sous-divisées, selon le degré d'autonomie des stratégies appliquées, en deux branches : déterministes et adaptatives. Les méthodes déterministes se basent sur des règles déterministes, qui ne changent pas durant l'exécution de l'algorithme. Leur objectif est de calculer des valeurs approximatives des paramètres puis de les ajuster selon le problème traité [Barbulescu et al., 2000]. Les méthodes adaptatives utilisent des informations sur l'état actuel de la recherche pour modifier les valeurs des paramètres. L'adaptation est effectuée de manière à changer la fonction objectif, en augmentant ou en diminuant les coefficients de pénalité pour violation des contraintes, d'une génération à l'autre. Ces méthodes de contrôle sont développées dans un cadre automatisé pour le réglage des paramètres d'un problème (une instance) spécifique. Si l'objectif est de résoudre différentes instances, ces méthodes peuvent être coûteuses en terme de temps de calcul, compte tenu du nombre de réglage des paramètres (un réglage pour chaque instance).

Il serait donc intéressant d'utiliser les méthodes de réglage des paramètres. Leur principe de base est plus simple, que celui des méthodes de contrôle, dans le sens que les paramètres ne changent pas de valeurs au cours de la résolution. Ils sont ajustés avant l'exécution de l'algorithme, et restent inchangés pendant son exécution. Le réglage obtenu par un apprentissage sur un sous-ensemble d'instances pourrait être utile pour la résolution de la totalité des instances d'un problème posé. Différentes stratégies de réglage des paramètres, pour une méthode d'optimisation, ont été examinées dans la littérature. Le choix d'application d'une stratégie dépend du type des paramètres qui peuvent être discrets, continus ou catégoriques. Il dépend également de la nature du problème à optimiser. Un problème déterministe est un problème où les sorties sont liées aux entrées par une fonction déterministe (en utilisant les mêmes entrées, nous obtenons toujours les mêmes sorties). Dans un problème stochastique, certaines sorties sont liées aux entrées par une fonction aléatoire (les sorties peuvent varier avec l'utilisation des mêmes entrées).

Une analyse de sensibilité peut faciliter le réglage des paramètres en se concentrant sur les paramètres sensibles pour un problème traité. Cela a pour objectif de réduire le temps et l'effort qui peuvent être gaspillés en réglant des paramètres peu sensibles. Dans la littérature, il y a peu de travaux sur l'utilisation de ces deux techniques, l'analyse de sensibilité pour réduire l'espace de recherche des paramètres et le réglage automatique de leurs valeurs. Dans cette optique, [Kim et al., 2006] effectue une analyse de sensibilité sur les paramètres dynamiques du modèle de glace de mer, en utilisant la différencia-

tion automatique. Des informations sur le gradient fournies par cette dernière sont utilisées dans un algorithme d'optimisation de paramètres, basé sur la méthode quasi-Newton [Zhu et al., 1997]. Un autre travail combinant l'analyse de sensibilité et le réglage automatique des paramètres est celui de [Teodoro et al., 2016]. Il s'agit d'une étude de paramètres pour le problème de segmentation d'images. Leur approche consiste à identifier les paramètres les moins influents et réduire l'espace de recherche des paramètres (100 points au lieu des trillions de points). Une application sur l'analyse d'images du cancer de cerveau a montré des résultats de segmentation de bonne qualité en comparaison avec des résultats obtenus avec les paramètres par défaut.

Dans les deux travaux cités plus hauts, l'étude des paramètres est effectuée pour une seule instance. L'originalité de notre travail consiste à définir un jeu de paramètres universel, d'un algorithme d'optimisation, pour un ensemble d'instances d'un problème posé. L'idée de base est de i) choisir un sous-ensemble d'instances, ii) étudier la sensibilité des paramètres pour fixer les valeurs des paramètres non sensibles, iii) régler automatiquement les valeurs des paramètres sensibles, et iv) appliquer ce jeu de paramètres sur la totalité d'instances du problème traité.

Dans ce chapitre, nous définissons le principe de base de l'analyse de sensibilité. Ensuite, nous présentons un ensemble de méthodes avec une grille de sélection d'une méthode d'analyse de sensibilité. Puis nous présentons un panorama de méthodes de réglage automatique des paramètres pour un problème d'optimisation.

3.2 Analyse de sensibilité

Soit un modèle mathématique, composé d'un ensemble de variables d'entrée $X = (x_1, ..., x_p)$, d'une fonction déterministe f et d'une variable de sortie Y.

$$\begin{aligned} f :& \mathbb{R}^p \to \mathbb{R} \\ & X \mapsto Y = f(X) \end{aligned} \tag{3.1}$$

Une analyse de sensibilité permet d'étudier comment la variation des entrées X du modèle engendre la variabilité de la sortie Y.

Chaque méthode d'analyse de sensibilité définit des indices de sensibilité mesurant la sensibilité du modèle aux variables d'entrée. Une valeur élevée d'indice indique que la variable d'entrée a une forte influence sur la sortie du modèle et, inversement, un indice faible indique que la variable d'entrée a un effet faible sur la variable de sortie. Calculer des indices de sensibilité pour toutes les variables d'entrée permet donc de hiérarchiser (et de quantifier) leur influence sur la sortie du modèle.

Les méthodes d'analyse de sensibilité peuvent être classées de deux manières : 1) les méthodes qui qualifient (hiérarchisent) ou quantifient (avec un ordre de grandeur des écarts) l'influence des variables d'entrée, et 2) les méthodes dont le résultat de l'analyse de sensibilité est locale (l'influence d'une variable d'entrée est obtenue en fixant les autres variables) ou globale (l'espace de variation de toutes les variables est exploré conjointement). La plupart de ces méthodes se basent sur quatre étapes :

1. La première étape consiste à identifier les variables d'entrée et leurs domaines de définition.

2. La deuxième étape a pour but de générer un plan d'expérience à partir des valeurs définies à l'étape 1. Cela est effectué par planification expérimentale, tirage aléatoire et/ou simulation numérique. L'échantillonnage se présente sous forme de N séries de valeurs de variables d'entrée. Définir le nombre N est un choix crucial, une valeur trop petite donnera une analyse imprécise alors que le choix d'une valeur trop grande impliquera un temps de calcul long à l'étape 3.

3. La troisième étape consiste à calculer les N valeurs de la variable de sortie Y du modèle. Chaque sortie correspond à une série de valeurs des variables d'entrée générée à l'étape 2.

4. La quatrième et la dernière étape consiste à calculer les indices de sensibilité.

Nous présentons dans ce chapitre trois types de méthodes : la méthode One-At-a-Time (OAT) qui est une méthode de sensibilité locale, la méthode de Morris qui est une méthode de sensibilité globale qualitative et la méthode de Sobol qui est une méthode de sensibilité globale quantitative. Nous présentons ensuite une grille proposant des critères pour sélectionner une méthode d'analyse de sensibilité.

3.2.1 Méthode de sensibilité locale : méthode One-At-a-Time (OAT)

L'analyse de sensibilité locale étudie la variabilité de la sortie du modèle en ne faisant varier qu'une seule variable d'entrée, les autres restant fixées à une valeur nominale. Son but est d'étudier comment de petites perturbations autour d'une valeur des variables d'entrées peuvent influer la sortie du modèle. La méthode d'analyse de sensibilité locale la plus classique est l'approche One-At-a-Time (OAT). Elle consiste à calculer des indices de sensibilité définis par les dérivées partielles,

$$S_i = \frac{\partial Y}{\partial x_{i|X=X^0}}, \tag{3.2}$$

S_i exprime l'effet, sur la valeur de la variable de sortie Y, de perturber la variable d'entrée x_i autour d'un point $X^0 = \{x_1^0, ..., x_p^0\}$. Cet indice ne représente pas une grandeur normée vu que les variables d'entrées ne sont pas toujours exprimées par les mêmes unités. Par conséquent, la comparaison de plusieurs indices de sensibilité devient impossible. Il convient alors de normer les indices de sensibilité par,

$$S_i = \frac{x_i^0}{Y^0} \frac{\partial Y}{\partial x_{i|X=X^0}}, \tag{3.3}$$

où Y^0 est la valeur de la variable de sortie du modèle obtenue pour le vecteur X^0 comme variables d'entrée, $Y^0 = f(X^0)$.

Comme mentionné auparavant, l'analyse de sensibilité locale donne une information locale autour du point nominal X^0. Elle ne permet pas alors de réaliser une mesure de sensibilité globale (l'influence des paramètres d'entrée sur l'ensemble de leurs domaines de variabilité). Pour pallier à cette limite, une nouvelle catégorie de méthodes s'est développée. Il s'agit des méthodes globales, qui s'intéressent à l'ensemble du domaine de variation des variables d'entrée.

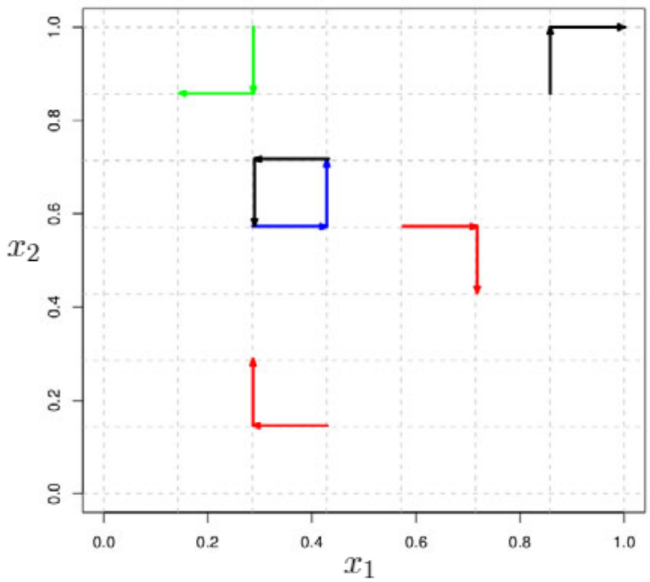


FIGURE 3.1 – Exemple de trajectoires de Morris pour deux variables d'entrée continues définies sur $[0,1]^2$, avec une discrétisation en $Q=8$ niveaux. Un total de $r=6$ trajectoires passant par les noeuds d'une grille régulière superposée sur le carré. Le pas de déplacement est $\delta=1/7$. La direction de la flèche indique le sens du déplacement [Faivre et al., 2013].

3.2.2 Méthode de sensibilité globale qualitative : méthode de Morris

La méthode de Morris [Morris, 1991] est une méthode d'analyse de sensibilité qualitative qu'on appelle méthode de criblage (screening en anglais). Elle permet d'identifier les variables d'entrée influentes et de les classer (sans quantifier) selon leurs effets sur la sortie du modèle : i) effets négligeables, ii) effets linéaires et sans interactions avec les autres variables d'entrée, et iii) effets non linéaires et/ou avec interactions. Elle fait partie des méthodes d'analyse de sensibilité globale. Elle représente une généralisation de la méthode d'analyse de sensibilité locale OAT sur l'ensemble du domaine de variabilité de chaque variable d'entrée.

La méthode consiste à discrétiser la gamme de variation de chaque variable d'entrée en Q niveaux $\{0, \frac{1}{Q-1}, \frac{2}{Q-1}, ..., 1\}$. Le croisement de ces niveaux définit un ensemble de Q^p noeuds. L'échantillonnage de la méthode constitue une suite de r trajectoires aléatoires passant chacune par $p+1$ noeuds, de manière à ce que chaque variable d'entrée ne varie qu'une seule fois par trajectoire. Chaque trajectoire démarre avec un noeud de départ choisi aléatoirement. La longueur du pas δ entre deux noeuds successifs de la trajectoire est identique dans toutes les directions et proportionnelle à $\frac{1}{Q-1}$. La figure 3.1 présente un exemple de trajectoires pour deux variables d'entrée, x_1 et x_2.

Pour garantir l'uniformité de la distribution des points échantillonnés, Morris propose de choisir un niveau de discrétisation Q pair et une longueur de pas δ égale à $\frac{Q}{2(Q-1)}$ [Faivre et al., 2013]. Cela donne 4/7 et non pas 1/7 dans l'exemple de la figure 3.1.

Définition et calcul des indices

L'approche de Morris repose sur l'analyse des variations de la sortie du modèle entre deux points des trajectoires. Soit $d_{i,j}(Y)$ la variation de la sortie Y sur la trajectoire i (i=1,...,r) relative à la variation $\epsilon\delta$ (avec $\epsilon=\pm 1$) de la variable d'entrée x_j. Cette variation, appelée

effet élémentaire de la variable d'entrée x_j, est définie par

$$d_{i,j}(Y) = \epsilon \frac{Y(x_1^i, ..., (x_j^i + \epsilon\delta), ..., x_p^i) - Y(x_1^i, ..., x_j^i, ..., x_p^i)}{\delta} \tag{3.4}$$

Les r trajectoires permettent de calculer r effets élémentaires pour chaque variable d'entrée. Au total, $N = r(p+1)$ simulations ou expériences réalisées. Trois indices peuvent être calculés pour chaque variable d'entrée x_j, l'indice de sensibilité, μ_j mesurant la moyenne des effets élémentaires, μ_j^* mesurant la moyenne des valeurs absolues des effets élémentaires et l'indice des interactions et des effets non linéaires σ_j mesurant la variance des effets élémentaires sur les r trajectoires.

$$\mu_j = \frac{1}{r}\sum_{i=1}^{r} d_{i,j}(Y) \tag{3.5}$$

$$\mu_j^* = \frac{1}{r}\sum_{i=1}^{r} |d_{i,j}(Y)| \tag{3.6}$$

$$\sigma_j = \sqrt{\frac{1}{r-1}\sum_{i=1}^{r} (d_{i,j}(Y) - \mu_j)^2} \tag{3.7}$$

Les résultats de la méthode de Morris sont synthétisés dans un graphique, appelé graphique de Morris, combinant les résultats de chaque variable d'entrée selon les coordonnées (μ_j^*, σ_j). Une analyse du graphique permet alors d'interpréter les résultats de cette étude. Ainsi,

- σ_j et μ_j^* proches de l'origine $(0,0)$: l'entrée x_j a des effets négligeables (les points X_2, X_4, X_9, X_{11} et X_{12} sur la figure 3.2),
- $\sigma_j \ll \mu_j^*$ et μ_j^* est élevé : l'entrée x_j influe sut la sortie du modèle Y (les points X_3, X_5, X_6 et X_{10} sur la figure 3.2),
- σ_j et μ_j^* sont du même ordre : x_j est fortement influente avec des effets non linéaires et/ou des interactions (les points X_7, X_8, X_1 sur la figure 3.2).

Plus la variance σ_j est élevée par rapport à la moyenne μ_j, plus les effets de la valeur d'entrée x_j dépendent de l'entrée elle-même (effet non-linéaire) ou des autres entrées (interactions). La méthode de Morris ne permet pas de distinguer ces deux cas (interactions et non linéarité). De plus, le classement qu'elle fournit n'est pas quantifiable. Pour lever ces limitations, la méthode de Sobol permet d'obtenir plus d'informations sur le modèle étudié grâce à ses indices de sensibilité.

3.2.3 Méthode de sensibilité globale quantitative : méthode de Sobol

La méthode de Sobol [Sobol, 1993] est une méthode d'analyse de sensibilité globale quantitative. Elle permet de mesurer, de manière précise, l'effet d'une variable d'entrée d'un modèle sur la variance de sa sortie. Cette variance est décomposée sur une somme de termes :

$$\mathbb{V}ar(Y) = V_1 + ... + V_p + V_{\{1,2\}} + ... + V_{\{p-1,p\}} + ... + ... + V_{\{1,...,p\}} \tag{3.8}$$

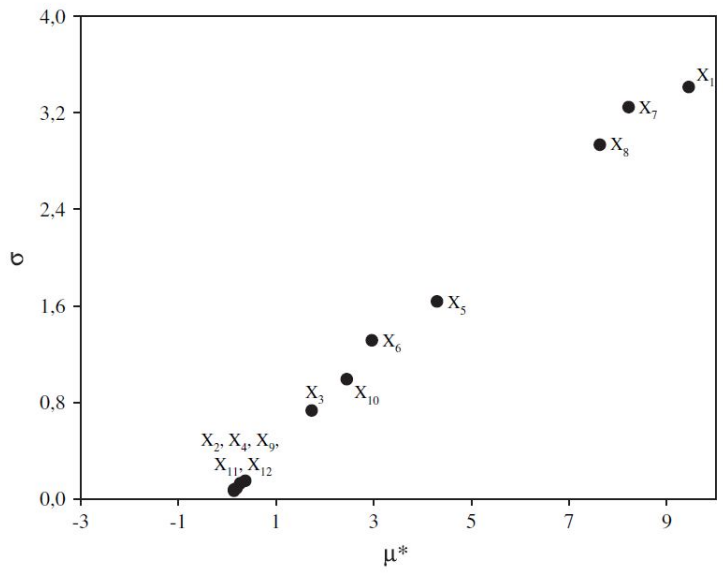


FIGURE 3.2 – Exemple du graphique de Morris obtenu avec 130 simulations pour 12 variables d'entrée [Campolongo et al., 2007].

où chaque terme V_i représente la variabilité de la sortie Y due à la variable d'entrée x_i et chaque terme V dont l'indice est un ensemble S désigne la variabilité de la sortie induite par l'interaction de plusieurs variables $x_i, i \in S$.

L'indice de sensibilité principal de la variable x_i est défini comme suit

$$S_i = \frac{\mathbb{V}ar[\mathbb{E}(Y|x_i)]}{\mathbb{V}ar(Y)} \tag{3.9}$$

Cet indice S_i mesure la variance de l'espérance de Y conditionnellement à la variable d'entrée x_i. La somme des p indices de sensibilité associés à chaque variable d'entrée x_i du modèle est égale à 1. Les indices de valeur proche de 1 correspondent aux variables d'entrées les plus influentes sur la sortie.

Toutefois, lorsque le nombre total des variables d'entrée est élevé, le nombre des interactions devient important. Donc, il est utile en pratique de regrouper les indices de sensibilité pour faciliter leur analyse. Cela est effectué à l'aide de l'indice de sensibilité total qui consiste à regrouper les indices qui sont associés à une même variable d'entrée.

L'indice de sensibilité total ST_i de la variable d'entrée x_i est défini par

$$ST_i = \frac{\mathbb{E}[\mathbb{V}ar(Y|x_j, j \neq i)]}{\mathbb{V}ar(Y)} = 1 - \frac{\mathbb{V}ar(\mathbb{E}(Y|x_i))}{\mathbb{V}ar(Y)} \tag{3.10}$$

Cet indice permet de mesurer la variance moyenne de Y quand toutes les variables d'entrée, autres que x_i, sont fixées. En d'autres termes, il mesure l'effet principal de x_i plus l'effet des interactions de x_i avec les autres variables d'entrée. La figure 3.3 présente un exemple de trois variables d'entrée avec sept indices de sensibilité : trois indices pour les effets principaux, trois indices pour l'interaction entre deux entrées et un indice pour l'interaction entre trois entrées. La figure 3.3 montre, comme exemple, les indices composant l'indice de sensibilité total de la variable d'entrée A [Faivre et al., 2013].

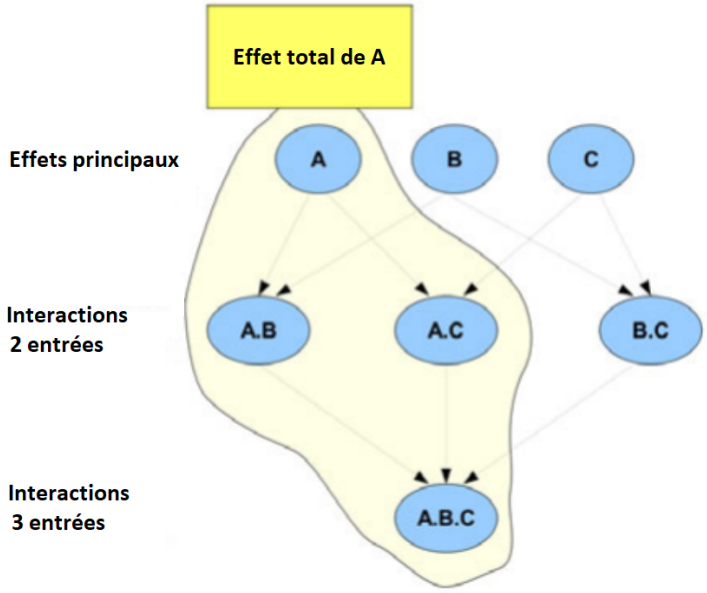


FIGURE 3.3 – Exemple illustratif de trois variables d'entrées (A, B et C), qui présente les indices composant l'indice de sensibilité total de la variable d'entrée A. [Faivre et al., 2013]

Méthodes d'estimation des indices de Sobol

Pour estimer les indices de Sobol, différentes méthodes basées sur des échantillons Monte Carlo ont été développées. La méthode classique la plus explorée dans la littérature est la méthode de Sobol [Sobol, 1993, Sobol, 2001]. Son inconvénient majeur est le nombre élevé de simulations pour réaliser une estimation précise des indices, de l'ordre typiquement d'une dizaine de milliers par entrée [Faivre et al., 2013].

Pour réduire le nombre d'évaluations, une autre génération de méthodes a été développée comme la méthode FAST (Fourier Amplitude Sensitivity Test) [Cukier et al., 1975], inspirée de la décomposition de Fourier en théorie du signal. Cette méthode permet de calculer les indices principaux (indépendamment les uns des autres) d'un modèle à partir d'un même échantillon de simulations. L'échantillonnage est défini sous forme de trajectoires discrètes parcourant le domaine de variation des différentes entrées avec des fréquences différentes. L'estimation des indices est moins coûteuse que celle de la méthode de Sobol, mais cela concerne uniquement les indices principaux, vu que la méthode FAST ne permet pas de calculer l'indice de sensibilité total. Des améliorations ont été apportées pour le calcul des indices totaux menant à la méthode FAST étendu [Saltelli et al., 1999] et à la méthode FAST-RBD [Tarantola et al., 2006] qui restent coûteuses et moins robustes lorsque le nombre des entrées augmente [Tissot and Prieur, 2012].

Une autre manière pour réduire le nombre de simulations élevé est la construction d'un métamodèle, appelé également modèle de surface de réponse [Jones, 2001]. Son objectif est de créer une inférence du modèle par un modèle simplifié en étudiant un nombre restreint d'entrées. Les indices de sensibilité sont calculés pour le métamodèle et non pour le modèle d'origine. Pour connaître la part non estimé du modèle, [Storlie B. et al., 2009] propose une méthode de bootstrap qui permet d'estimer l'erreur sur chaque indice de sensibilité. Il existe une panoplie de méthodes de construction d'un métamodèle, dont les plus utilisées sont les méthodes de régression [Storlie B. et al., 2009, Simpson et al., 2001, Villa-Vialaneix et al., 2012]. Pour plus de détail, nous invitons le lecteur à lire le livre [Faivre et al., 2013].

3.2.4 Grille de sélection d'une méthode d'analyse de sensibilité

Mettre en oeuvre une analyse de sensibilité nécessite de spécifier clairement les objectifs de l'analyse du modèle, de définir les variables de sortie sur lesquelles l'analyse va porter et de sélectionner les variables d'entrée tout en définissant leurs natures (discrète ou continue) et leurs domaines de variation. Cette définition est essentielle pour fixer le cadre de l'étude et choisir la méthode adéquate qui répond aux mieux aux objectifs du modèle, qui peuvent être : 1) identification et hiérarchisation des entrées les plus influentes, 2) détermination des entrées non influentes (les rendre constantes), ou 3) calibration des entrées par rapport à certaines informations disponibles [Iooss and Lemaître, 2015].

Toutefois, le nombre et la diversité des méthodes d'analyse de sensibilité existantes dans la littérature rend parfois le choix difficile. Nous nous intéressons ici essentiellement aux méthodes d'analyse de sensibilité globale qui prennent explicitement en compte l'ensemble du domaine de variation des entrées. Pour choisir parmi ces méthodes, [Faivre et al., 2013] propose un guide synthétique de sélection d'une méthode d'analyse de sensibilité globale. Ce guide présente deux grilles de sélection selon le niveau de connaissance du modèle : une utilisation boîte noire ou une utilisation boîte blanche . Un modèle est dit boîte noire s'il vérifie les trois propriétés suivantes : le domaine est connu, il est possible de connaître la valeur de sortie du modèle en tout point du domaine grâce à une simulation, et que cette information est la seule disponible pour ce point choisi [Kargupta, 1995, Kleijnen, 2005]. Un modèle boîte noire est un modèle auquel nous n'avons pas accès à son processus interne, contrairement au modèle boîte blanche où l'utilisateur a une connaissance interne du modèle.

En général, le choix d'une méthode d'analyse de sensibilité est dicté par des contraintes de temps qui limitent le nombre d'expériences réalisables. La nature des variables d'entrée (discrète ou continue) du modèle, leur domaine de variation et l'intensité de variation de ces domaines limitent également le nombre de simulations. En conséquence, choisir une méthode d'analyse de sensibilité revient alors à trouver un meilleur compromis entre une exploration intensive du domaine de variation des entrées et un temps de simulation raisonnable.

A partir de ces critères, deux grilles de choix d'une méthode d'analyse de sensibilité sont construites : une pour un modèle boîte noire (Figure 3.4) et l'autre pour un modèle boîte blanche (Figure 3.5), toutes deux présentées ci-dessous.

Grille pour un modèle boîte noire

La première grille présente un ensemble de méthodes dont le choix dépend du nombre de simulations, du nombre d'entrées et de leur nature (continues ou discrètes).

Si toutes les entrées du modèle sont discrètes, les méthodes utilisables sont les plans d'expériences usuels couplés avec une analyse de variance. Un plan d'expériences factoriel complet consiste à évaluer le modèle pour toutes les combinaisons des niveaux des entrées. Ce plan devient impraticable au delà d'une dizaines d'entrées vu le nombre important des simulations requises. Un plan fractionnaire, comme le plan One-At-a-Time (OAT) (voir la sous-section 3.2.1), permet d'estimer d'une manière non biaisée les effets principaux de chaque entrée en supposant que les effets des interactions sont nuls. Cela limite la préci-

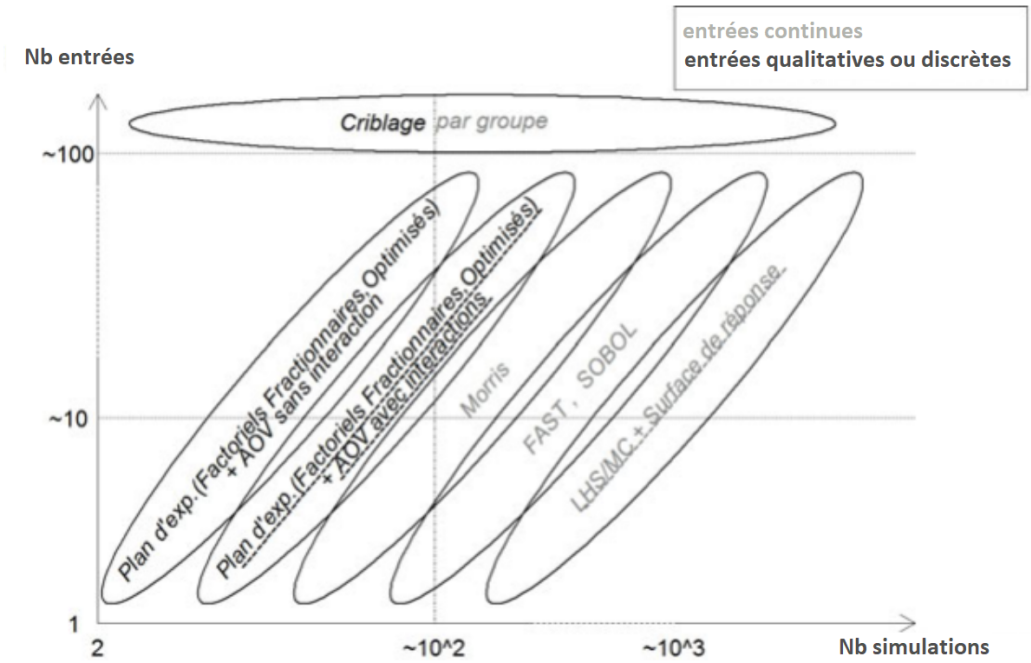


FIGURE 3.4 – Grille de sélection d'une méthode d'analyse de sensibilité. Les nombres des entrées du modèle en ordonnées et les nombres de simulations en abscisses apparentent à des choix pour un modèle dont le temps de calcul d'une simulation est de l'ordre de 10 minutes et la durée maximale de l'analyse est d'une journée [Faivre et al., 2013].

sion d'estimation de la sensibilité des entrées. Une manière plus raisonnable qu'un simple plan d'expériences consiste à coupler ce dernier avec une analyse de variance (ou AOV pour Analysis Of Variance), qui permet de distinguer et quantifier les effets principaux et interactions d'ordre plus ou moins élevé. Pour plus de détail sur les plans d'expériences et l'analyse de variance, nous invitons le lecteur à se référer à [Faivre et al., 2013].

Si toutes les entrées du modèle sont continues, une première approche à appliquer est la méthode de Morris (voir la sous-section 3.2.2), qui permet d'explorer, d'une manière régulière, le domaine de variation de chaque entrée pour estimer les indices de sensibilité principaux et d'identifier les possibles interactions entre entrées sans les quantifier. La méthode de Morris peut être appliquée dans toutes les configurations du nombre d'entrées, en précisant le pas de découpage du domaine de variation des entrées, ce qui définit le nombre de simulations. Pour plus de précision, nous pouvons appliquer des méthodes plus gourmandes en nombre de simulations qui permettent une exploration fine des domaines de variation. Il s'agit des méthodes basées sur la décomposition de variance comme les méthodes de Sobol ou de FAST voire de surfaces de réponse (métamodèles), présentées dans la sous-section 3.2.3. pour un modèle avec un grand nombre d'entrées (plus de 100 entrées), nous pouvons appliquer l'approche de criblage par groupe. Son objectif consiste à regrouper les entrées qui ont une influence similaire sur la sortie du modèle. Chaque groupe constitué est traité comme une entrée unique. L'analyse de sensibilité passe donc d'une étude de p entrées à une étude de p' entrées, $p' \ll p$. Cette stratégie repose sur l'hypothèse que toutes les entrées d'un même groupe n'interagissent pas entre elles.

Si le modèle dispose d'un mélange d'entrées continues et d'entrées discrètes, il est souvent nécessaire de considérer l'ensemble des entrées comme discrètes en discrétisant a priori le domaine de variation des entrées continues, et en appliquant les méthodes pour entrées discrètes [Faivre et al., 2013].

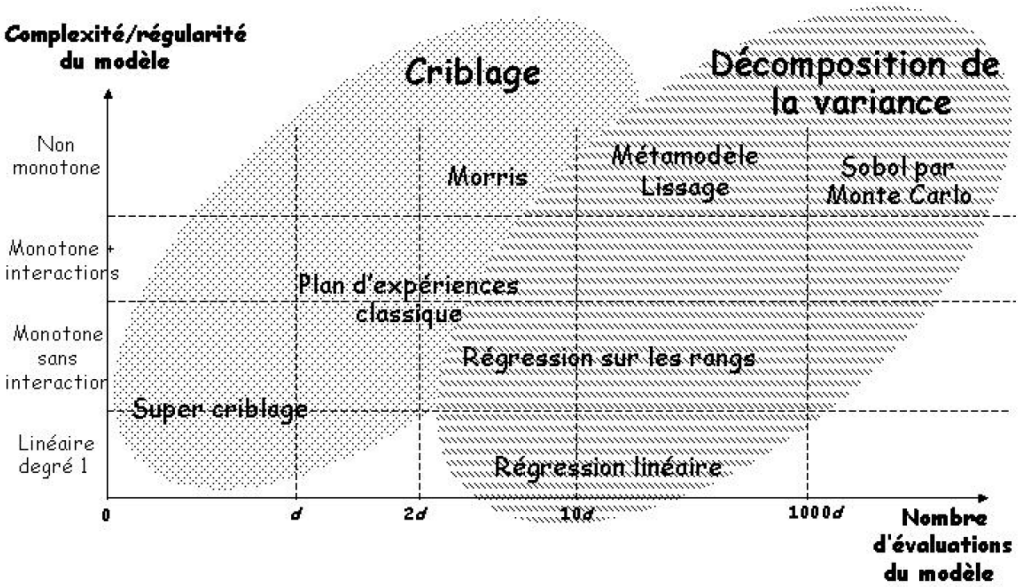


FIGURE 3.5 – Cartographie des méthodes d'analyse de sensibilité classées selon la complexité du modèle et le nombre d'évaluations (proportionnel aux nombre d'entrées) du modèle [Iooss, 2011].

Grille pour un modèle boîte blanche

La seconde grille, illustrée sur la figure 3.5, présente une synthèse des principales méthodes d'analyse de sensibilité classées en deux types : les méthodes de criblage (ou screening) permettant d'identifier les entrées non influentes pour les rendre constantes, et les méthodes de décomposition de la variance hiérarchisant les entrées du modèle tout en quantifiant leur influence sur les sorties. Ces méthodes sont partitionnées, sur la figure, en régions en fonction de leurs hypothèses sur la complexité et la régularité du modèle (fonction linéaire avec ou sans interactions, ou non linéaire) et en fonction du nombre de simulations qui est proportionnel au nombre d'entrées du modèle.

En effet, le choix d'une méthode est effectué de manière à ce que cette dernière soit la plus simple adaptée au problème posé en fonction de l'objectif de l'analyse, du nombre de simulations possibles et de la connaissance des propriétés internes du modèle. Or, plus le modèle est complexe (nombreuses entrées et interactions), plus il est difficile d'appréhender ses propriétés (linéarité, continuité, monotonie). Donc, la première grille (Figure 3.4) est le support adapté pour choisir une première méthode d'analyse de sensibilité, sachant que la complexité du modèle engendre souvent un temps de simulation long.

Pour résumer, une étude d'analyse de sensibilité permet de quantifier l'influence des entrées d'un modèle sur sa sortie et de hiérarchiser ces entrées selon leur degré d'influence. Elle permet également de définir les entrées qui n'ont pas d'influence sur la sortie pour les rendre constantes. Nous avons présenté deux grilles de sélection d'une méthode d'analyse de sensibilité selon la connaissance interne du modèle : une grille pour un modèle boîte noire et une grille pour un modèle boîte blanche. Le choix d'une méthode dépend de l'objectif de l'étude et de différents critères. Néanmoins, l'analyse de sensibilité ne permet pas de trouver de bonnes valeurs pour les entrées du modèle. D'où la nécessité d'appliquer une méthode de réglage automatique des valeurs des entrées. Dans cette thèse, nous nous intéressons particulièrement au réglage automatique des méthodes d'optimisation consi-

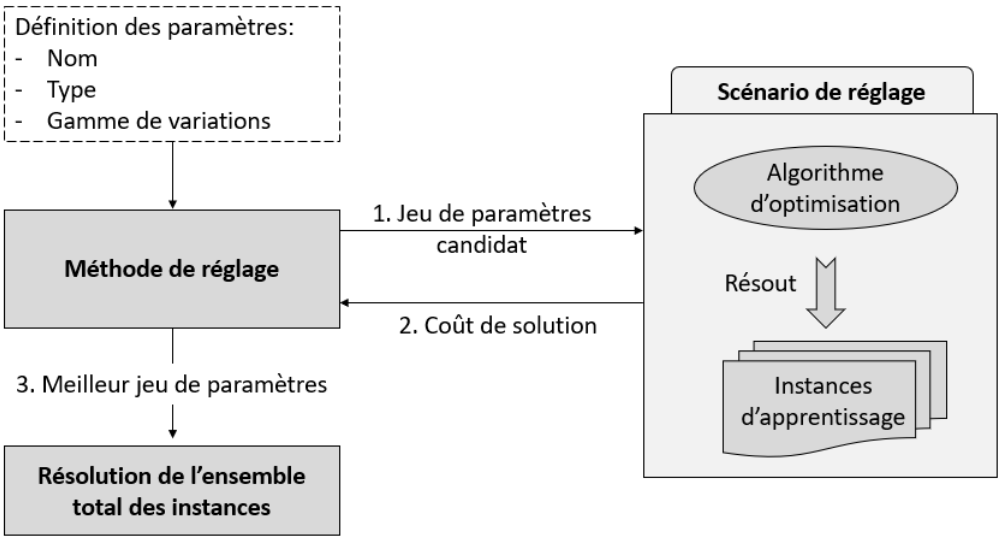


FIGURE 3.6 – Un scénario de réglage comprend un algorithme d'optimisation et un ensemble d'instances d'un problème quelconque. Une procédure de réglage exécute l'algorithme à régler sur une partie ou sur la totalité d'instances, reçoit des informations sur les performances de ces exécutions pour définir le meilleur jeu de paramètres.

dérées comme des modèles boîtes noires, où les sorties (objectif de la méthode) sont liées aux entrées (paramètres de la méthode) par une fonction objectif.

3.3 Réglage automatique des paramètres

Le réglage des paramètres d'un algorithme d'optimisation est généralement difficile, mais nécessaire pour atteindre des performances optimales. Dans la littérature, plusieurs études utilisent des valeurs de paramètres suite à des expérimentations. Mais cela demande beaucoup de temps et ne garantit pas d'avoir un meilleur jeu de paramètres. De nombreuses autres études choisissent des valeurs de paramètres rapportées, comme efficaces, dans d'autres travaux. Cela se fait souvent sans confirmation expérimentale sur leur efficacité dans le contexte actuel. Trouver un jeu de paramètres en maximisant la qualité de solution et en minimisant le temps de réglage, a poussé les chercheurs à développer des méthodes de réglage automatique. Nous nous intéressons aux méthodes dont le réglage se fait sur un ensemble d'instances d'apprentissage. En général, leur principe de base consiste en quatre étapes : 1) exécuter l'algorithme d'optimisation avec des jeux de paramètres candidats sur un sous-ensemble d'instances, 2) recevoir des informations sur les performances de ces exécutions, 3) utiliser ces informations pour définir le meilleur jeu de paramètres, et 4) appliquer ce jeu de paramètres pour résoudre la totalité des instances (voir Figure 3.6, inspirée de [Hutter, 2009]).

Une grande variété de stratégies de réglage des paramètres a été explorée dans la littérature. Nous présentons dans cette section un ensemble de stratégies génériques que nous divisons en deux catégories. La première concerne les stratégies basées sur des méthodes statistiques comme l'optimisation séquentielle à base de modèles, les tests statistiques et les plans d'expériences. La seconde catégorie comprend des stratégies basées sur des heuristiques qui peuvent être des méthodes de recherche locale ou des méthodes évolutionnaires.

3.3.1 Stratégies basées sur des méthodes statistiques

Optimisation séquentielle à base de modèles

L'optimisation séquentielle à base de modèles (SMBO, pour Sequential Model-Based Optimization), également appelée modèle de surface de réponse [Jones, 2001] que nous avons cité dans la section de l'analyse de sensibilité, est un paradigme adapté aux situations où le nombre de paramètres est grand. Pour éviter une évaluation coûteuse de la totalité des paramètres, SMBO utilise des informations obtenues à partir des évaluations précédentes pour construire un espace de configurations, qu'elle utilise pour définir une prochaine configuration. Souvent les problèmes basés sur les SMBO sont considérés comme des problèmes d'optimisation boîte noire, où la fonction objectif est inconnue. SPOT (Sequential Parameter Optimization Toolbox) [Bartz-Beielstein, 2010] est l'un des outils basés sur les SMBO. SPOT est un paquet R[1] qui sert de boîte à outils pour le réglage des paramètres. Il se base sur l'optimisation séquentielle de paramètres (Sequential Parameter Optimization (SPO)) [Bartz-Beielstein et al., 2005] qui regroupe différentes techniques classiques de régressions et d'analyse de la variance. Les principaux inconvénients de SPO et ses variantes (comme SPO$^+$ [Hutter, 2009]) sont leur limitation aux paramètres continus et à l'optimisation de performance des algorithmes stochastiques.

Tests statistiques : les algorithmes de Racing

Une autre approche de réglage automatique des paramètres est basée sur les algorithmes de Racing (course), comme la technique Hoeffding Races introduite par [Maron and Moore, 1994]. Son principe de base est simple. Chaque configuration de paramètres est évaluée de manière itérative tout au long du processus de réglage. Une configuration est éliminée si la solution fournie est moins bonne que la meilleure solution trouvée à un stade donné de la recherche. La comparaison des configurations candidates est effectuée à l'aide de tests statistiques. Le nombre des instances d'apprentissage varie au long de la configuration afin de concentrer le temps d'évaluation sur les configurations les plus prometteuses. Dans ce sens, [Birattari et al., 2002] proposent une procédure appelée F-Race pour le réglage des paramètres des algorithmes stochastiques. Cette technique utilise le test de Friedman [Friedman, 1937]. Un inconvénient majeur de cette technique se présente à la première étape de la procédure de réglage, où toutes les configurations doivent être évaluées. En conséquence, avec un grand nombre de paramètres, l'ensemble des configurations initiales devient très large, ce qui entraîne souvent un temps de réglage prohibitif. Pour remédier à cette situation, deux variantes de F-Race ont été introduites : Sampling F-Race (F-Race d'échantillonnage) et I/F-Race (F-Race itérative) [Balaprakash et al., 2007]. Sampling F-Race utilise un processus d'échantillonnage pour déterminer la configuration de paramètres initiale. I/F-Race se base sur un processus itératif où à chaque itération, des configurations sont échantillonnées à partir d'un modèle probabiliste. Cette catégorie d'approches basées sur des tests statistiques est adaptée à la configuration des algorithmes stochastiques. Nous avons employé le terme configuration de paramètres au lieu du terme jeu de paramètres vu que ces techniques peuvent régler non seulement des paramètres numériques, mais également des paramètres catégoriques comme le choix d'une heuristique.

[1] Logiciel de statistiques gratuit et à code source ouvert (open source).

Plans d'expériences

Un plan d'expériences (que nous avons cité dans la section précédente) est une méthode d'expérimentation permettant d'organiser au mieux les essais qui accompagnent une recherche scientifique. Dans notre cas, la recherche concerne un jeu de paramètres avec une meilleure performance. Dans cette catégorie de méthodes, [Coy et al., 2001] présente une approche qui emploie le plan factoriel fractionnaire[2]. La méthode fonctionne en quatre étapes : 1) Choisir un sous-ensemble d'instances d'apprentissage, 2) Définir, selon des expérimentations, le point de départ de chaque paramètre, ses bornes et le pas de variations, 3) Choisir le meilleur jeu de paramètres pour chaque instance en utilisant un plan factoriel fractionnaire et la descente de gradient et 4) moyenner les valeurs trouvées pour chaque paramètre pour obtenir le meilleur jeu de paramètres. Cette quatrième étape de calcul de la moyenne limite l'utilisation de la méthode pour le réglage des algorithmes uniquement avec des paramètres numériques. Une approche similaire basée sur un plan factoriel fractionnaire est implémenté dans le système CALIBRA [Adenso-Diaz and Laguna, 2006]. CALIBRA utilise le plan factoriel fractionnaire de Taguchi en combinaison avec de multiples exécutions de la recherche locale pour affiner la zone de recherche. Malheureusement, CALIBRA est limité à cinq paramètres, qui doivent être tous numériques ou ordinaux.

3.3.2 Stratégies basées sur des heuristiques

Recherche locale : ParamILS

ParamILS est un algorithme de configuration automatique de paramètres [Hutter et al., 2007, Hutter, 2009], basé sur la recherche locale itérative ou ILS pour Iterated Local Search (voir chapitre 1). En pratique, ParamILS commence par une configuration initiale (typiquement la configuration par défaut de l'algorithme cible) ainsi que d'autres configurations qui sont choisies uniformément de manière aléatoire dans l'espace de configuration donné. Ces configurations sont évaluées sur un ensemble d'instances d'apprentissage. La meilleure configuration évaluée est sélectionnée comme point de départ pour le processus de recherche locale itérée, où la relation de voisinage consiste à modifier un seul paramètre à la fois. Pour sortir des minima locaux, ParamILS utilise un mécanisme de diversification qui consiste à réinitialiser la recherche avec une probabilité (donnée par défaut égale à 0.01 selon [Hutter et al., 2007]) à chaque itération. La recherche continue jusqu'à ce que le changement de paramètres n'apporte aucune amélioration à la solution obtenue. A la fin du réglage, la meilleure configuration est renvoyée puis testée sur un ensemble séparé des instances du même problème posé.

La version de base de ParamILS, BasicILS, évalue chaque configuration de paramètres sur les mêmes instances d'apprentissage en utilisant les mêmes graines aléatoires. Cette approche permet de trouver de bonnes configurations de paramètres lorsque l'ensemble des instances d'apprentissage est petit ou très hétérogène [Hutter, 2009]. Un des inconvénients de BasicILS est le choix du nombre d'instances. Configurer des paramètres en utilisant un petit ensemble d'apprentissage peut conduire à de bonnes performances, alors qu'avec un ensemble d'apprentissage énorme, la configuration devient très lente. FocusedILS est une variante de ParamILS qui résout ce problème en utilisant un ensemble d'instances de plus en plus grand pour concentrer le temps de configuration sur les configurations les plus prometteuses. Notons que cette idée est la même utilisée dans les algorithmes de Racing. La différence est que FocusedILS détermine les configura-

[2] Un plan factoriel fractionnaire utilise un sous-ensemble d'un plan factoriel complet qui mesure toutes les combinaisons de niveaux de facteurs.

tions prometteuses de manière heuristique plutôt que d'utiliser des tests statistiques. Il existe d'autres variantes de ParamILs. Nous citons Adaptive Capping qui permet de réduire, dans certaines conditions, le temps consacré à l'évaluation des configurations moins prometteuses. Une autre variante, MOParamILS, étend ParamILS pour configurer automatiquement des algorithmes d'optimisation multi-critères [Blot and Hoos, 2016]. Notons que ParamILs, avec toutes ses variantes, est un logiciel de configuration de paramètres de type catégoriques. Donc régler des paramètres numériques nécessite, a priori, une discrétisation du domaine de paramétrage.

Méthode évolutionnaire : GENOUD

Nous décrivons une méthode de réglage des paramètres basée sur les algorithmes génétiques (voir chapitre 1) et une méthode de quasi-Newton (méthode de Broyden-Fletcher-Goldfarb-Shanno (BFGS) [Zhu et al., 1997]). Il s'agit de GENOUD (GENetic Optimization Using Derivatives) [Sekhon and Mebane, 1998, Mebane Jr et al., 2011], un paquet de R, logiciel de statistiques, qui permet de régler les paramètres des problèmes d'optimisation difficiles, rgenoud. Ces difficultés surviennent souvent du fait que la fonction objectif est non linéaire et par conséquent non globalement concave, ayant de multiples optima locaux. Dans les problèmes d'optimisation, la fonction objectif est souvent concave dans le voisinage d'un optimum local. Ce voisinage représente une zone dans l'espace des paramètres. En dehors de cette zone, la fonction objectif est peut être irrégulière. Dans ce cas, une méthode basée entièrement sur les dérivés est pratiquement inutilisable.

Appliquer BFGS avec les algorithmes génétiques permet d'utiliser des informations sur les dérivées dans le voisinage des optima locaux. Elle est appliquée sur la meilleure solution produite par les opérateurs (e.g croisement) à chaque génération. Cela permet de converger rapidement vers l'optimum global si cette solution est dans le voisinage correcte. Or, une utilisation excessive de BFGS peut empêcher cette convergence. Avant de passer à une prochaine génération ou s'arrêter, rgenoud calcule également le gradient de cette meilleure solution afin de vérifier la convergence. De manière générale, le calcul du gradient et l'optimisation BFGS ne sont utilisés que dans un contexte continu. Ils ne sont pas applicables lorsque les paramètres de la fonction à optimiser sont discrets.

3.3.3 Récapitulatif des méthodes

Dans l'ensemble, les stratégies de configuration de paramètres telles que celles présentées dans cette section, jouent un rôle crucial dans le développement, l'évaluation et l'utilisation des méthodes d'optimisation pour résoudre des problèmes difficiles. Le défi pourrait provenir de la complexité du calcul (admissibilité, optimalité) ou des contraintes de ressources (temps de configuration, espace mémoire). En conséquence, une meilleure stratégie de configuration est celle qui réalise un meilleur compromis entre la qualité de solution et le temps de calcul. Le choix dépend également du type des paramètres à régler. Certaines stratégies sont conçues pour régler uniquement des paramètres numériques, soit discrets, continus ou les deux. D'autres peuvent également régler des paramètres catégoriques, comme le choix d'une heuristique, du pré-traitement ou de l'ordre d'exécution des contraintes. Ainsi, la nature de l'algorithme cible, que se soit déterministe ou stochastique, est également un critère essentiel de choix de la stratégie de réglage. Le tableau 3.1 ci-dessous récapitule, les stratégies discutées dans cette section, en précisant le type de paramètres qu'elles peuvent régler, ainsi qu'un aperçu sur leurs performances et limites.

Stratégies	Paramètres	Performances / Limitations
SMBO : SPO	~~discrets~~ continus ~~catégoriques~~	SPO (et ses variantes) est une stratégie de réglage des paramètres continus uniquement pour les algorithmes stochastiques. Le temps d'exécution de SPO est à l'ordre de $\mathcal{O}(p^3)$ en fonction du nombre de configurations de paramètres. D'après [Hutter, 2009], SPO est moins performante que CALIBRA et PARAMILS.
Tests statistiques : F-Race	discrets continus catégoriques	Avec un grand nombre de paramètres, l'ensemble de configurations initiales dans la stratégie F-Race devient très large. Ce qui entraîne un temps de calcul très long. Cela a été traité par les deux versions I/F-Race et Sampling F-Race qui ont été appliquées pour le réglage de différents algorithmes stochastiques de recherche locale [Balaprakash et al., 2007]. L'application la plus complexe de ces stratégies concerne 12 paramètres dont certains dépendent conditionnellement des valeurs des autres [Birattari et al., 2010].
Plan d'expériences : CALIBRA	discrets continus ~~catégoriques~~	L'outil CALIBRA s'applique sur un champ restreint de méthodes, il est limité au réglage des algorithmes avec cinq paramètres numériques. Cette limitation provient de l'utilisation des plans d'expériences qui entraînent un nombre très large de jeux de paramètres.
Recherche Locale : PARAMILS	discrets ~~continus~~ catégoriques	Il existe dans la littérature de nombreuses applications de l'algorithme PARAMILS et ses variantes. [Hutter et al., 2010] étudie l'application d'une procédure de configuration de paramètres à différents solveurs de la programation mixte en nombres entiers (Mixed Integer Programming (MIP)). Dans le cas du solveur CPLEX (version 12.1), FOCUSEDILS a été utilisé pour configurer 76 paramètres sur un ensemble d'instances avec l'objectif de minimiser le gap d'optimalité. Cette approche a accéléré l'exécution de CPLEX (par un facteur de 52) et a amélioré sa performance par rapport à CPLEX avec les paramètres par défaut. La majorité des algorithmes réglés par PARAMILS ont des paramètres continus, qui exige leur discrétisation. Toutefois, trouver des discrétisassions raisonnables, dont l'utilisation n'entraîne pas de perte majeure de performance, peut s'avérer difficile. Dans ce cas, plusieurs exécutions de la procédure de configuration peuvent être utilisées de manière itérative, afin d'affiner les domaines des paramètres continus.

Algorithme évolutionnaire : RGENOUD	discrets continus ~~catégoriques~~	La stratégie RGENOUD a été utilisée par un grand nombre d'utilisateurs et de développeurs. Plus de douze paquets R se basent actuellement sur RGENOUD [Mebane Jr et al., 2011], comme le paquet BOOLEAN (modèles booléens de réponse binaire) [Morgan, 2014] et le paquet JOP (optimisation simultanée de réponses multiples) [Kuhnt and Rudak, 2011]. Parmi ses inconvénients, RGENOUD demande un grand espace mémoire et passe beaucoup de temps au réglage lorsque le nombre de paramètres est grand. Un autre inconvénient est qu'il ne mixe pas un réglage des paramètres discrets et continus. Si l'utilisateur veut utiliser les deux, il est suggéré de choisir le type discret et de reformuler la fonction objectif pour avoir un domaine de valeurs continues. Par exemple, pour un paramètre continu définit sur $[0, 1.0]$, nous choisissons un type discret définit sur $[0, 100]$ et nous le divisons par 100 dans la fonction objectif pour obtenir une grille de recherche de 0.0 à 1.0 avec un pas de 0.1.

3.4 Conclusion du chapitre

Il existe de nombreuses méthodes d'analyse de sensibilité et de réglage automatique des paramètres. Ce chapitre a exposé une liste de méthodes qui n'est pas exhaustive, mais qui regroupe différentes classes de méthodes.

Pour l'analyse de sensibilité, nous avons présenté deux catégories de méthodes. Les méthodes de sensibilité locale (comme la méthode One-at-a-Time) qui analysent le comportement d'un algorithme d'optimisation cible autour d'une valeur de référence (un jeu de paramètres fixe). Les méthodes de sensibilité globale qui permettent d'analyser la variabilité de l'algorithme induite par la variation de ses paramètres sur l'ensemble de leurs domaines de définition. Nous avons subdivisé cette dernière catégorie en deux sous-ensembles. Le premier concerne les méthodes de sensibilité globale qualitative (comme la méthode de Morris) dont l'objectif est d'identifier les paramètres les plus influents et de les classer selon leurs effets sur les résultats de l'algorithme. Le second concerne les méthodes de sensibilité globale quantitative (comme la méthode de Sobol) qui permettent de quantifier l'effet du changement de la valeur d'un paramètre sur le résultat de l'algorithme. Ensuite, nous avons illustré une démarche de sélection d'une méthode d'analyse de sensibilité globale adaptée à l'objectif de l'analyse, aux moyens disponibles et aux caractéristiques de l'algorithme à analyser.

Pour le réglage automatique des paramètres, nous avons discuté deux catégories de stratégies. Les stratégies basées sur des méthodes statistiques et celles basées sur des heuristiques. Dans la première catégorie, l'optimisation séquentielle à base de modèles (SMBO) construit, à partir de l'algorithme cible, un métamodèle qu'elle utilise pour identifier le meilleur jeu de paramètres ; les algorithmes de Racing évaluent de manière itérative les configurations de paramètres sur un ensemble d'instances d'apprentissage et utilisent des

tests d'hypothèses statistiques pour éliminer les configurations les moins performantes ; et les plans d'expériences organisent à l'avance une série de jeux de paramètres pour les évaluer et définir le meilleur jeu. Dans la seconde catégorie, ParamILS utilise la recherche locale itérative pour rechercher dans des espaces vastes de configurations de paramètres ; et rgenoud utilise l'algorithme génétique pour diversifier la recherche (trouver un optimum global) et une méthode de quasi-Newton pour intensifier la recherche au voisinage d'un optimum.

Pour conclure, une meilleure méthode de réglage automatique de paramètres n'existe pas. Selon les besoins du chercheur (temps d'exécution, qualité de solution, espace mémoire,...), une méthode pourrait être plus intéressante qu'une autre. L'essentiel c'est de choisir une méthode qui respecte le type de paramètres (discret, continu ou catégorique), la nature de l'algorithme cible (discret ou stochastique) et l'objectif de réglage (sur une seule instance ou un ensemble d'instances d'apprentissage).

Deuxième partie

Algorithme de Wedelin et réglage automatique de ses paramètres

Chapitre 4

Algorithme de Wedelin

Sommaire

4.1 Introduction

Nous adoptons dans cette thèse une méthode approchée, nommée algorithme de Wedelin ou également algorithme In-the-middle. La méthode est conçue pour la résolution des problèmes linéaires en variables 0-1 [Wedelin, 1995a]. Elle se base généralement sur la relaxation Lagrangienne qui a l'avantage, par rapport à la relaxation linéaire, de fournir directement des solutions entières (voir chapitre 1).

L'algorithme de Wedelin a été largement discuté par ses principaux auteurs. [Wedelin, 1995b] décrit le principe de base de l'algorithme et son application dans le système Carmen, pour la planification des rotations d'équipages pour les compagnies aériennes. [Alefragis et al., 2000] présentent une parallélisation évolutive de l'algorithme d'origine utilisé dans le système Carmen. [Grohe and Wedelin, 2008] introduisent à partir de In-the-middle, un algorithme similaire pour les problèmes de somme de fonctions de coûts qu'ils appellent propagation de coûts. Cette formulation a été examinée plus en détail par [Wedelin, 2013]. D'autres chercheurs ont évalué et amélioré l'algorithme de Wedelin. [Mason, 2001] décrit une variante de l'algorithme de Wedelin et l'applique sur le problème de planification du personnel, en montrant des résultats meilleurs que ceux fournis par le solveur commercial de programmation linéaire mixte CPLEX. De manière similaire, [Ernst et al., 2006] utilise In-the-middle avec les mêmes modifications rapportées par Mason pour la résolution du problème de répartition des tâches, avec des résultats favorables par rapport à CPLEX. [Bastert et al., 2010] présentent de nombreuses extensions et généralisations de l'algorithme de Wedelin avec différentes améliorations. Ils ont évalué la performance de leur variante sur un ensemble d'instances de différentes sources, où les résultats étaient favorables par rapport à FICO XPRESS, un autre logiciel d'optimisation commercial.

Dans ce chapitre, nous décrivons le principe de base et le fonctionnement de l'algorithme de Wedelin avec les différentes améliorations menées par d'autres chercheurs, plus particulièrement les versions de [Mason, 2001] et [Bastert et al., 2010]. Nous présentons également les différents paramètres de l'algorithme en discutant la difficulté de leur réglage. La description de notre version de l'algorithme de Wedelin et son implémentation seront abordées dans le chapitre suivant.

4.2 Algorithme de Wedelin

4.2.1 Problème

La version basique de l'algorithme de Wedelin a été conçue pour résoudre des problèmes linéaires en variables 0-1 de la forme suivante :

$$\begin{aligned} \min \quad & cx \\ s.c. \quad & Ax = b \\ & x \in \{0,1\}^n \end{aligned} \tag{4.1}$$

où $c \in \mathbb{R}^n$ est un vecteur ligne de coûts, $b \in \mathbb{N}^m$ un vecteur colonne des termes constants des contraintes et $A \in \{0, \pm 1\}^{m \times n}$ une matrice de coefficients des contraintes. Pour simplifier la description de l'algorithme, nous considérons seulement des contraintes avec des coefficients 0 et 1. Soit "contrainte i" l'équation spécifiée par la i^{ème} ligne de $Ax = b$, et "colonne j" la j^{ème} colonne de A associée à la variable x_j. Soit $J = \{1, 2, ..., n\}$ l'ensemble des indices de colonnes, et $J(i) = \{j : a_{ij} = 1\} \subseteq J$ les indices de colonnes (variables) contribuant dans

la contrainte i. De manière similaire, soit $I = \{1, 2, ..., m\}$ l'ensemble des indices de lignes, et $I(j) = \{i : a_{ij} = 1\} \subseteq I$ les indices des contraintes contenant la variable x_j.

Avec ces restrictions sur la matrice A et le vecteur b, le problème (4.1) est NP-difficile. Parmi les problèmes NP-difficiles ayant ces restrictions, nous citons le problème de Set Partioning décrit dans le chapitre 2.

4.2.2 Idée de base : Relaxation Lagrangienne

La majorité des méthodes de programmation linéaires en variables 0-1 sont basées sur la relaxation linéaire qui consiste à relâcher les contraintes d'intégrité ($x \in \{0,1\}^n \Rightarrow x \in [0,1]^n$), et par conséquent de fournir des valeurs fractionnaires. Donc, pour produire des solutions entières, elle est toujours associée à une stratégie de branchement comme la méthode de Branch and Bound (voir chapitre 1). Toutefois, ces méthodes sont limitées par la taille et le type des problèmes à résoudre. Elles deviennent inefficaces à mesure que la taille du problème devient importante.

L'algorithme de Wedelin se base sur une autre approche. Il ne considère pas les solutions fournies par la relaxation linéaire, comme une étape intermédiaire, mais essaye de résoudre directement le problème linéaire (4.1), en considérant sa relaxation Lagrangienne,

$$\begin{aligned} \min_x \quad & cx - \pi(Ax - b) \\ s.c. \quad & x \in \{0,1\}^n \end{aligned} \tag{4.2}$$

où les éléments de $\pi = \{\pi_1, \pi_2, ..., \pi_m\} \in \mathbb{R}^m$ représentent les multiplicateurs de Lagrange, également appelés les variables duales.

La résolution du problème (4.2) est triviale lorsque les éléments de π sont fixés à $\hat{\pi}$,

$$\begin{aligned} \hat{\pi}b + \quad \min_x \quad & (c - \hat{\pi}A)x \\ s.c. \quad & x \in \{0,1\}^n \end{aligned} \tag{4.3}$$

La solution $\hat{x}_j$ est construite en se basant sur le signe du coût réduit $(c_j - \hat{\pi}a_j)$, avec a_j la colonne associée dans la matrice A.

$$\hat{x}_j = \begin{cases} 1 & si \quad (c_j - \hat{\pi}a_j) < 0, \\ 0/1 & si \quad (c_j - \hat{\pi}a_j) = 0, \\ 0 & si \quad (c_j - \hat{\pi}a_j) > 0. \end{cases} \tag{4.4}$$

Si le coût réduit $(c - \hat{\pi}A)$ est non nul, la solution $\hat{x}$ est entière et unique. Cependant, si un ou quelques éléments de $(c - \hat{\pi}A)$ sont nuls, il sera généralement difficile de trouver une solution entière réalisable pour le problème non relaxé (4.1). Ainsi, le but de l'algorithme est de trouver un $\hat{\pi}$ pour lequel tous les éléments sont non nuls, et que la solution entière unique du problème relaxé (4.2) soit réalisable pour le problème primal (4.1).

Nous remarquons que la modification de la valeur d'une seule composante $\hat{\pi}_i$ de $\hat{\pi}$ modifie les valeurs (et les signes) des coûts réduits $(c - \hat{\pi}A)$, et par conséquent affecte la solution $\hat{x}$. L'idée de l'algorithme est donc de considérer itérativement une seule contrainte i, en mettant à jour l'élément $\hat{\pi}_i$ de manière à ce que la solution $\hat{x}$ satisfait la contrainte i du problème primal (4.1). Il existe toujours un intervalle de sélection de $\hat{\pi}_i$. L'algorithme de Wedelin choisit la valeur de $\hat{\pi}_i$ au milieu de cet intervalle, d'où sa nomination

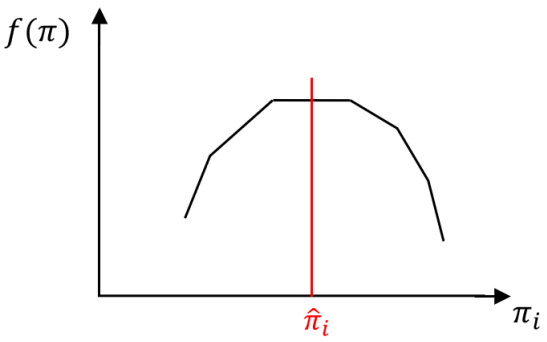


FIGURE 4.1 – $f(\pi)$ est une fonction concave et linéaire par morceaux

algorithme In-the-middle. Cela correspond à la recherche dual de descente par coordonnée.

Comme décrit dans le chapitre 1, un problème linéaire (4.1) et sa relaxation Lagrangienne (4.2) sont liés à un problème dual

$$\max_{\pi \in \mathbb{R}^m} f(\pi) = \max_{\pi \in \mathbb{R}^m} \{\pi b + \min_{x \in \{0,1\}^n} (c - \pi A)x\} \tag{4.5}$$

Sa solution est une borne duale au problème primal (4.1). Donc, la meilleure borne possible à ce dernier est la solution optimale $\hat{\pi}$ du problème dual Lagrangien (4.5).

$$\hat{\pi} = \arg\max_{\pi \in \mathbb{R}^m} f(\pi) \tag{4.6}$$

Toutefois, π est un vecteur réel. Trouver donc le maximum de $f(\pi)$ est un problème d'optimisation continue. Or $f(\pi)$ n'est pas différentiable, elle est concave et linéaire par morceaux (ensemble de fonctions linéaires) ; un morceau pour chaque instanciation de x. En conséquence, trouver $\hat{\pi}$ par un calcul de gradient n'est pas possible. La solution est de calculer un sous-gradient pour chaque morceau de la fonction $f(\pi)$. Cela veut dire de calculer la dérivée partielle pour chaque élément π_i du vecteur π,

$$\frac{\partial f}{\partial \pi_i} = b_i - a_i \hat{x} \tag{4.7}$$

où $\hat{x}$ est la solution du problème Lagrangien relaxé (4.2) pour ce vecteur π. Rappelons que la recherche par coordonnée, π_i, est orientée de manière à satisfaire la contrainte $a_i\hat{x} = b_i$, ce qui implique $\frac{\partial f}{\partial \pi_i} = 0$ pour ce point. Comme $f(\pi)$ est concave, nous pouvons conclure que ce point est son maximum le long de π_i. De plus, si tous les éléments de $(c - \pi A)$ sont non nuls, le maximum de $f(\pi)$ se produit sur un plateau, et en n dimensions dans un polytope de volume positif. D'après [Wedelin, 1995a], l'algorithme de recherche par coordonnée choisit la valeur de $\hat{\pi}_i$ au milieu de ce plateau, selon l'axe considéré (voir Figure 4.1).

Exemple [Wedelin, 1995a] : Soit (P) le problème primal suivant :

$$\begin{aligned} \min \quad & 6x_1 + 9x_2 + 2x_3 + 3x_4 + 5x_5 + 2x_6 + 5x_7 \\ s.c. \quad & x_1 + x_2 + x_3 + x_4 + x_5 = 2 && (4.8) \\ & x_1 + x_2 + x_6 + x_7 = 3 && (4.9) \\ & x = (x_1, ..., x_7)^T \in \{0,1\}^7 \end{aligned}$$

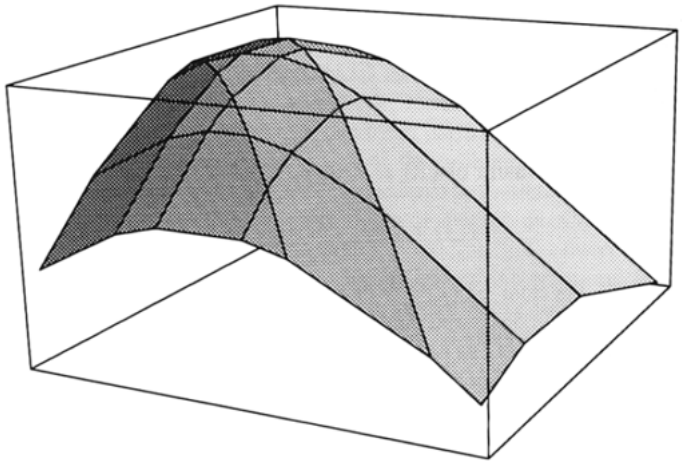


FIGURE 4.2 – Représentation de la fonction $f(\pi_1, \pi_2)$

Nous avons deux contraintes dans le problème primal (P). Donc, sa relaxation Lagrangienne est exprimée en fonction de deux variables duales (multiplicateurs Lagrangiens), π_1 et π_2.

$$f(\pi_1, \pi_2) = \min_x [(6x_1 + 9x_2 + 2x_3 + 3x_4 + 5x_5 + 2x_6 + 5x_7) + \pi_1(x_1 + ... + x_5) + \pi_2(x_1 + ... + x_7)]$$

Son dual (D) associé est le suivant

$$\max_{\pi_1, \pi_2} f(\pi_1, \pi_2)$$

Il consiste à calculer le maximum de la fonction $f(\pi_1, \pi_2)$, permettant de satisfaire les deux contraintes du problème (P). Cela correspond au plus haut plateau du polytope sur la figure 4.2 représentant la fonction $f(\pi_1, \pi_2)$.

La figure 4.3 illustre une vue aérienne de la fonction $f(\pi)$, $\pi = (\pi_1, \pi_2)$. Les lignes représentent les coûts réduits nuls $(c - \pi A) = 0$ et les zones correspondent aux valeurs Ax. Chaque zone est représentée par deux valeurs séparées par une virgule, une valeur par contrainte. Pour satisfaire les deux contraintes du problème (P), l'algorithme procède par des mouvements horizontaux et verticaux pour arriver à la zone 2,3, i.e, des mouvements horizontaux vers les zones où le premier indice est égal à 2 (satisfaire la première contrainte (4.8)), et des mouvements verticaux vers les zones où le second indice est égal à 3 (satisfaire la seconde contrainte (4.9)).

Effectuer une recherche itérative répétée sur des directions fixes d'une fonction non continue, peut amener la recherche sur une arête où aucune des directions de coordonnées ne donne une augmentation de $f(\pi)$, alors qu'une autre direction peut le faire. Cela se produit lorsque le vecteur $(c - \pi A)$ est encombré de zéros. L'algorithme de Wedelin utilise donc une heuristique itérative pour éviter les coûts réduits nuls $\bar{c}$ et pour accélérer la convergence vers une solution admissible.

4.2.3 Étapes de l'algorithme

L'algorithme de Wedelin comprend deux parties. La première est une recherche duale de descente par coordonnée, son objectif est de trouver un meilleur vecteur de multiplicateurs Lagrangiens. La seconde est une approche heuristique qui consiste à modifier les éléments

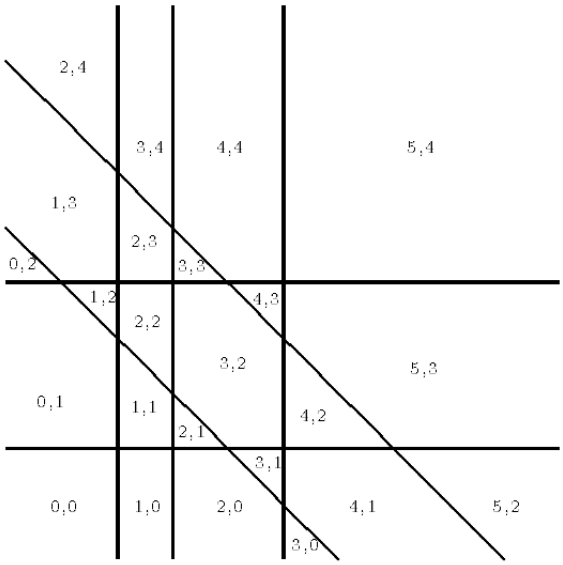


FIGURE 4.3 – Vue aérienne de l'espace dual du problème (P). Les lignes représentent les coûts réduits nuls $\bar{c} = (c - \pi A) = 0$ et les zones correspondent aux valeurs Ax (deux valeurs séparées par une virgule pour les deux contraintes 4.8 et 4.9).

du vecteur de coûts réduits $(c - \pi A)$ afin de trouver une solution entière réalisable au problème primal (4.1), i.e., qui satisfait l'ensemble des contraintes.

Recherche duale de descente par coordonnée

L'algorithme de Wedelin applique une recherche duale de descente par coordonnée. Il parcourt tous les éléments du vecteur π de multiplicateurs Lagrangiens (variables duales). L'algorithme choisit itérativement une valeur $\hat{\pi}_i$ pour chaque élément π_i, de manière à trouver une solution $\hat{x}$ au problème relaxé (4.3), qui satisfait la contrainte i de l'ensemble $A\hat{x} = b$.

Comme le calcul est répété plusieurs fois pour chaque élément $\hat{\pi}_i$, il est nécessaire de mettre à jour la valeur courante du vecteur des coûts réduits $(c - \hat{\pi} A)$. En effet, pour chaque élément $j \in J(i) = \{j : a_{ij} = 1\} \subseteq J$ de la contrainte i , nous avons

$$r_j = c_j - \sum_{i \in I(j)} a_{ij} \hat{\pi}_i \tag{4.10}$$

Nous trions les éléments du vecteur r (où chaque élément est exprimé par l'équation (4.10)) dans un ordre croissant : $r_{[1]}, ..., r_{[|J(i)|]}$. Soit $r_{[b_i]}$ le $b_i^{\text{ème}}$ petit élément, et $r_{[b_{i+1}]}$ le $(b_i+1)^{\text{ème}}$ petit élément. Les valeurs $r_{[b_i]}$ et $r_{[b_{i+1}]}$ représentent les valeurs critiques de la contrainte i. La nouvelle valeur de la variable duale $\hat{\pi}_i$ est choisie comme suit

$$\hat{\pi}_i = \hat{\pi}_i + \frac{1}{2}(r_{[b_i]} + r_{[b_{i+1}]}) \tag{4.11}$$

Notons que $r_{[b_i]} \leq \hat{\pi}_i \leq r_{[b_{i+1}]}$ est l'intervalle de maximisation de la fonction $f(\pi)$ (voir l'équation (4.5)) tout au long de la coordonnée π_i. Nous choisissons donc la valeur de $\hat{\pi}_i$ au milieu de cet intervalle.

La mise à jour de la valeur de $\hat{\pi}$ engendre également une mise à jour de la solution courante $\hat{x}$ (voir l'équation (4.4)).

Exemple : Soit c un vecteur de coûts, b un vecteur de termes constants des contraintes et A une matrice de coefficients des contraintes. Soit $\hat{\pi} = 0$ et soit $\hat{x} = 0$ une solution initiale.

$$c = \begin{pmatrix} \mathbf{3} & 8 & 6 & 5 & \mathbf{8} & 2 & 3 & 2 & \mathbf{6} & 3 & 10 & 8 & \mathbf{5} & 8 & 10 & 5 \end{pmatrix}$$

$$b^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Pour la cinquième contrainte $a_{5,j}$, $j = 1, ..., n$:

Les coefficients non nuls sont : $a_{1,1}$, $a_{1,5}$, $a_{1,9}$ et $a_{1,13}$.
Les coûts correspondants sont respectivement : $c_1 = 3$, $c_5 = 8$, $c_9 = 6$ et $c_{13} = 5$.
En utilisant l'équation (4.10), nous avons :

$$r = \begin{pmatrix} 3 & 8 & 6 & 5 \end{pmatrix}$$

Nous trions les coûts réduits : $r_{[1]} = 3 < r_{[2]} = 5 < r_{[3]} = 6 < r_{[4]} = 8$
Le 1er petit élément : $r_{[1]} = 3$
Le 2ème petit élément : $r_{[2]} = 5$
La valeur de la cinquième variable duale (multiplicateur Lagrangien) $\hat{\pi}_5$ est :

$$\hat{\pi}_5 = \hat{\pi}_5 + \frac{1}{2}(3+5) = 4$$

Le vecteur des coûts réduits est alors :

$$c - \hat{\pi}A = \begin{pmatrix} \mathbf{-1} & 8 & 6 & 5 & \mathbf{4} & 2 & 3 & 2 & \mathbf{2} & 3 & 10 & 8 & \mathbf{1} & 8 & 10 & 5 \end{pmatrix}$$

La solution courante (selon l'équation (4.4)),

$$\hat{x}^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

Pour cet exemple, l'algorithme de Wedelin a trouvé une solution courante qui satisfait une contrainte. En suivant la même procédure, In-the-middle parcourt toutes les contraintes jusqu'à ce qu'il trouve une solution admissible. Toutefois et dans la majorité des cas, le vecteur des coûts réduits $(c - \pi A)$ converge vers une solution non entière qui contient des éléments de coûts nuls. Pour éviter cette situation, l'algorithme de Wedelin adopte une approche heuristique qui permet de converger rapidement vers une solution entière approchée.

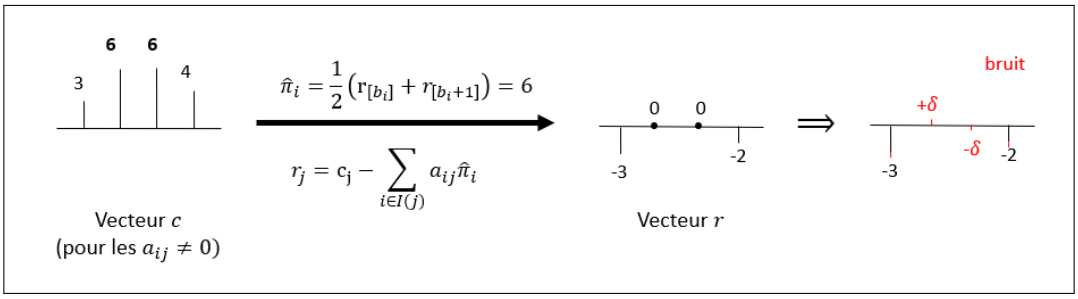


FIGURE 4.4 – Illustration du bruit ajouté aux coûts réduits.

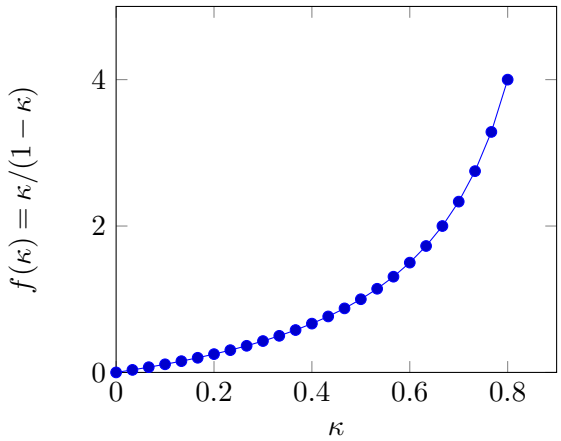


FIGURE 4.5 – Fonction $f(\kappa) = \kappa/(1-\kappa)$, utilisée dans l'expressions d'approximation des multiplicateurs Lagrangiens (équation (4.12)).

Heuristique de Wedelin

L'heuristique de Wedelin consiste à ajouter un bruit aux coûts réduits, afin d'aider l'algorithme de descente par coordonnée à trouver une solution entière. La magnitude de distorsion est déterminée par un paramètre κ, qui détermine le degré d'approximation. Lorsque $\kappa = 0$, l'algorithme fonctionne sans approximation, et quand $\kappa = 1$, l'approximation est maximale. A cet égard, l'utilisation de ce paramètre $0 \leq \kappa \leq 1$ est similaire à l'utilisation du paramètre de température dans le recuit simulé (voir chapitre 1).

La recherche duale de descente par coordonnée est modifiée de la manière suivante. Au lieu d'avoir une seule valeur $\hat{\pi}_i$ par contrainte, l'algorithme associe pour chaque contrainte i, deux valeurs différentes, $(\hat{\pi}_i + \delta)$ et $(\hat{\pi}_i - \delta)$, avec

$$\delta = \frac{\kappa}{1-\kappa}(r_{[b_{i+1}]} - r_{[b_i]}) + l \tag{4.12}$$

L'idée est de remplacer $\hat{\pi}_i$ par $(\hat{\pi}_i - \delta)$ pour les variables qui obtiennent des coûts réduits négatifs après une mise à jour de $\hat{\pi}_i$ (en utilisant l'équation (4.11)). De manière similaire, $(\hat{\pi}_i + \delta)$ remplace $\hat{\pi}_i$ pour les variables qui obtiennent des coûts réduits positifs.

L'expression $\kappa/(1-\kappa)$ est un choix arbitraire effectué par Wedelin pour maintenir l'amplitude de décalage dans l'intervalle $[0,1]$ et pour permettre une convergence rapide (Figure 4.5). Le paramètre $l \geq 0$ est une perturbation aléatoire de petite valeur, ajoutée aux coûts réduits pour assurer des valeurs non nulles tout au long du calcul (Figure 4.4).

L'algorithme 1 est un pseudo-code qui résume les étapes principales de l'heuristique de Wedelin. En principe, il commence la résolution avec une solution initiale $\hat{x}$, qu'il

Algorithm 1: Algorithme de base de Wedelin[Wedelin, 1995a]

Input : $A \in \{0,1\}^{m \times n}$, $b \in \mathbb{N}^m$, $c \in \mathbb{R}^n$
Output: $\hat{x} \in \{0,1\}$ avec $A\hat{x} = b$ et $c\hat{x}$ minimisé ou message *"Pas de solution"*

Soit $\hat{\pi} \in \mathbb{R}^m$, $\hat{\pi} \leftarrow 0$, $l \geq 0$, $\kappa \in [0,1[$
// Construire une solution initiale selon l'équation (4.4)
for $j \in J$ **do**
  if $c_j \leq 0$ **then** $\hat{x}_j \leftarrow 1$
  else $\hat{x}_j \leftarrow 0$
while Les éléments de c ne changent plus de signe **do**
  for $i \in I$ **do**
    // Calculer les coûts réduits (4.10)
    for $j \in J(i)$ **do**
      $r_j \leftarrow c_j - \sum_{i \in I(j)} a_{ij}\hat{\pi}_i$
    // Trier les coûts réduits dans un ordre croissant
    $r_{[1]} < r_{[2]} < \ldots < r_{[|J(i)|]}$
    // Mettre à jour les multiplicateurs Lagrangiens (4.11), (4.12)
    $\hat{\pi}_i \leftarrow \hat{\pi}_i + \frac{1}{2}(r_{[b_{i+1}]} + r_{[b_i]})$
    $\delta \leftarrow \frac{\kappa}{1-\kappa}(r_{[b_{i+1}]} - r_{[b_i]}) + l$
    for $j \in J(i)$ **do**
      if $r_j \leq r_{[b_i]}$ **then** $\hat{\pi}'_i \leftarrow \hat{\pi}_i - \delta$
      else if $r_j \geq r_{[b_{i+1}]}$ **then** $\hat{\pi}_i \leftarrow \hat{\pi}'_i + \delta$
      $c_j \leftarrow r_j + \sum_{i \in I(j)} a_{ij}\hat{\pi}'_i$
    // Mettre à jour le vecteur de solution
    for $j \in J$ **do**
      if $c_j \leq 0$ **then** $\hat{x}_j \leftarrow 1$
      else $\hat{x}_j \leftarrow 0$
  // Vérifier si la solution est admissible
  if $A\hat{x} = b$ **then**
    return $\hat{x}$
  Ajuster κ et l
return " Pas de solution"

construit à partir du vecteur de coûts c (selon l'équation (4.4)). Il parcourt ensuite toutes les contraintes du problème, en mettant à jour les multiplicateurs Lagrangiens, les coûts réduits et le vecteur de solution. A la fin de chaque itération, l'algorithme met à jour le paramètre d'approximation κ en augmentant sa valeur d'une itération à l'autre (la stratégie de mise à jour de κ ainsi que celles des autres paramètres seront discutées dans la section 4.4). Il s'arrête à la première solution admissible trouvée. Toutefois, si l'algorithme ne trouve aucune solution et que le vecteur des coûts réduits ne change plus de signe, il s'arrête pour afficher "Pas de solution".

4.3 Amélioration de l'algorithme de Wedelin

Une amélioration de l'heuristique de Wedelin a été présentée par [Mason, 2001]. Elle consiste à utiliser, dans le calcul des coûts réduits, l'historique des modifications menées lors des itérations précédentes.

Chaque variable x_j est associée à un vecteur $p_j = (p_{ij}), i = 1, ..., m$, où p_{ij} est la valeur de préférence utilisée pour modifier le coût r_j. Cette valeur p_{ij} spécifie le bonus ajoutée à la fonction objectif par la colonne j lors de la satisfaction de la contrainte i. La matrice $P = (p_1 p_2 ... p_n) = (p_{ij}), i = 1, ..., m, j = 1, ..., n$ englobe toutes les valeurs de préférence. Le coût réduit modifié par la préférence p_{ij} est donné par

$$r_j = c_j - \sum_{i \in I(j)} a_{ij} \hat{\pi}_i - \sum_{i \in I(j)} a_{ij} p_{ij} \tag{4.13}$$

La recherche duale est donc modifiée pour incorporer les valeurs de préférence P. Supposons que nous avons des valeurs non nulles de préférences P et des variables duales $\hat{\pi}$ à l'itération courante. Pour mettre à jour une contrainte i, la première étape de l'algorithme de Wedelin est de supprimer les préférences p_{ij} associées à cette contrainte. [Mason, 2001] utilise une approche alternative qui préserve une fraction des préférences en mettant $p_{ij} \leftarrow \theta . p_{ij}$, $j \in J(i)$, $\theta \in [0, 1]$. Utiliser $\theta > 0$ encourage la nouvelle solution à maintenir certaines similitudes avec la solution précédente. Autrement dit, l'algorithme modifie les préférences de manière à préserver la satisfaction de la contrainte i lors du traitement de la contrainte $i+1$. Il définit des préférences $p_{ij} > 0$ pour les variables $x_j = 1$ de la contrainte i, diminuant ainsi les coûts correspondants à ces variables. De manière similaire, il définit des préférences $p_{ij} < 0$ pour les variables $x_j = 0$, ce qui augmente les coûts correspondants à ces variables. Cela ce traduit par l'équation suivante :

$$p_{i[j]} = \begin{cases} -\delta & si \quad j \in [1], ..., [b_i] \\ +\delta & si \quad j \in [b_i + 1], ..., [|J(i)|] \end{cases} \tag{4.14}$$

où $[1], ..., [|J(i)|]$ est l'ordre des variables x_j appartenant à la contrainte i, classées selon l'ordre croissant des coûts réduits : $r_{[1]} < r_{[2]} < ... < r_{[|J(i)|]}$. Dans cette équation (4.14), la préférence p_{ij} est entièrement remplacée, et donc l'algorithme fait un seul pas en arrière en considérant une seule itération. Une autre approche proposée par [Bastert et al., 2010] consiste à ajouter les termes $\pm\delta$ aux préférences p_{ij} au lieu de remplacer leurs valeurs.

$$p_{i[j]} = p_{i[j]} \pm \delta \tag{4.15}$$

Cela permet de maintenir l'historique complet dans la matrice P, avec des valeurs de préférences à partir de la i^{ème} itération définie par θ^i. [Bastert et al., 2010] appelle la première méthode de [Mason, 2001] "remplacement de préférences" et la seconde méthode "accumulation de préférences".

Comme la notion de préférence préserve l'historique de la satisfaction des contraintes, [Mason, 2001] et [Bastert et al., 2010] proposent de traiter uniquement les contraintes violées au lieu de mettre à jour toutes les contraintes du problème. Ils proposent également de fixer un nombre maximal d'itérations, car pour beaucoup de problèmes de programmation linéaire, trouver une solution admissible est un problème NP-difficile. De plus, ils expriment le vecteur de solution d'une manière différente. Au lieu d'utiliser directement l'équation (4.4), ils utilisent le classement des coûts réduits, en mettant les variables $x_{[1]},...,x_{[b_k]}$ à 1 et les variables $x_{[b_k+1]},...,x_{[|J(i)|]}$ à 0, sachant que $r_{[b_k]}$ et $r_{[b_k+1]}$ sont respectivement le premier et le deuxième petit élément du vecteur des coûts réduits. L'algorithme 2 résume les étapes de la version améliorée de l'algorithme de Wedelin.

4.4 Difficulté du choix des paramètres

La performance de l'algorithme de Wedelin dépend du choix de ses principaux paramètres $\kappa \in [0,1[$, $l \geq 0$ et $\theta \in [0,1]$.

[Wedelin, 1995a] cite qu'il est difficile de connaître à l'avance la valeur du paramètre κ, puisqu'elle dépend de l'instance à résoudre, i.e., chaque problème à un point de convergence différent. Mais il confirme que la qualité de la solution est meilleure lorsque la valeur de κ est faible à la convergence. Il interprète son algorithme de trois manières selon la valeur de κ : comme une méthode exacte de la programmation linéaire pour $\kappa = 0$, un algorithme de programmation dynamique non-sérial pour $\kappa = \frac{1}{3}$, et un algorithme glouton pour $\kappa = 1$. En conséquence, il propose une stratégie de réglage qui consiste à commencer avec une petite valeur de κ pour donner une indication sur le point de convergence puis de l'augmenter lentement à chaque itération pour affiner la recherche dans cette zone critique. Malheureusement, il n'a fourni aucun détail sur la manière de contrôler ce paramètre. [Bastert et al., 2010] propose donc une fonction linéaire simple pour le contrôler,

$$\kappa = \kappa_{min} + \max\{0, i - w\} * \kappa_{step} \tag{4.16}$$

où i est le nombre des itérations effectuées et w le nombre des itérations où la valeur de κ n'a pas changé. Généralement, nous mettons κ_{min} à 0 et nous choisissons un paramètre additionnel κ_{max} qui définit la valeur maximale permise de κ. Avec cette méthode, nous avons la même taille du pas k_{step} jusqu'à la convergence. En effet, une petite valeur de k_{step} entraînera un temps d'exécutions très long, alors qu'avec un pas plus important, l'algorithme peut dépasser le point de convergence, et par conséquent il ne trouvera aucune solution. Pour contourner ce problème en appliquant l'idée de [Wedelin, 1995a], [Bastert et al., 2010] exprime le pas d'approximation en fonction du nombre des contraintes violées, vu que ce dernier diminue de manière significative à la convergence. Cela est effectué en utilisant, à chaque itération, la fonction suivante :

$$\kappa_{new} \leftarrow \begin{cases} \kappa_{min} & si \quad i \leq w \\ \kappa_{old} + \kappa_{step}(\frac{|R|}{m})^{\alpha} & sinon \end{cases} \tag{4.17}$$

où R est l'ensemble des contraintes violées et m est le nombre total de contraintes. α est un nouveau paramètre nommé paramètre adaptatif, qui lie la taille du pas au nombre actuel des contraintes violées. Utiliser $\alpha = 0$ revient à utiliser la fonction linéaire (4.16) ci-dessous où la taille du pas est fixe, alors qu'avec $\alpha > 0$ la taille du pas diminue au fil de la recherche. Toutefois, cette méthode dépend du problème à résoudre. Par exemple, soit 10^{-1} l'ordre de la fraction des contraintes violées pour certains problèmes et 10^{-3} pour

Algorithm 2: Algorithme de Wedelin [Mason, 2001, Bastert et al., 2010]

Input : $A \in \{0,1\}^{m \times n}$, $b \in \mathbb{N}^m$, $c \in \mathbb{R}^n$
Output: $\hat{x} \in \{0,1\}$ avec $A\hat{x} = b$ et $c\hat{x}$ minimisé ou message *"Pas de solution"*

Soit $\hat{\pi} \in \mathbb{R}^m$, $\hat{\pi} \leftarrow 0$, $l \geq 0$, $\kappa \in [0,1[$
// Construire une solution initiale selon l'équation (4.4)
for $j \in J$ **do**
  if $c_j \leq 0$ **then** $\hat{x}_j \leftarrow 1$
  else $\hat{x}_j \leftarrow 0$
while Nombre maximal d'itérations non atteint **do**
  $R \leftarrow \{i | \sum_{j \in J} a_{ij}\hat{x}_j \neq b_i\}$ // L'ensemble des contraintes violées
  for $i \in R$ **do**
    // Réduire l'influence des préférences locales
    for $j \in J(i)$ **do**
      $p_{ij} \leftarrow \theta . p_{ij}$
    // Calculer les coûts réduits (4.13)
    for $j \in J(i)$ **do**
      $r_j \leftarrow c_j - \sum_{i \in I(j)} a_{ij}\hat{\pi}_i - \sum_{i \in I(j)} a_{ij}p_{ij}$
    // Trier les coûts réduits dans un ordre croissant
    $r_{[1]} < r_{[2]} < \ldots < r_{[|J(i)|]}$
    // Mettre à jour les multiplicateurs Lagrangiens (4.11), (4.12)
    $\hat{\pi}_i \leftarrow \hat{\pi}_i + \frac{1}{2}(r_{[b_i+1]} + r_{[b_i]})$
    $\delta \leftarrow \frac{\kappa}{1-\kappa}(r_{[b_i+1]} - r_{[b_i]}) + l$
    // Mettre à jour les variables et les préférences
    for $j = 1, \ldots, b_i$ **do**
      $\hat{x}_{[j]} \leftarrow 1$
      $p_{i[j]} \leftarrow +\delta$ /* Alternativement: $p_{i[j]} \leftarrow p_{i[j]} + \delta$ */
    for $j = (b_i + 1), \ldots, |J(i)|$ **do**
      $\hat{x}_{[j]} \leftarrow 0$
      $p_{i[j]} \leftarrow -\delta$ /* Alternativement: $p_{i[j]} \leftarrow p_{i[j]} - \delta$ */
  // Vérifier si la solution est admissible
  if $A\hat{x} = b$ **then**
    return $\hat{x}$
  Ajuster κ, l et θ
return "Pas de solution"

d'autres problèmes. Mettant par exemple $\alpha = 1$ pourrait être une bonne valeur pour le premier cas. Mais pour le second cas, la taille du pas de κ serait trop petite pour obtenir une solution dans un temps raisonnable. Choisir la valeur de α dépend donc du problème à résoudre.

Pour le paramètre l, [Wedelin, 1995a] indique qu'une grande valeur de l accélère la convergence, mais il n'a précisé aucune information sur l'existence d'une relation directe avec la qualité de la solution. [Bastert et al., 2010] remarque qu'il existe une forte dépendance entre ce paramètre et le vecteur de coûts c. Comme l est ajouté aux coûts réduits r qui dépendent des coûts c, une valeur très grande de l ($l \gg \max(c_j), j \in J$) détruit complètement les résultats alors qu'une valeur très petite ($l \ll \max(c_j), j \in J$) n'a presque aucun impact sur l'heuristique. Pour éliminer cette dépendance, Bastert propose de normaliser les coûts c à l'entrée de l'algorithme pour avoir des valeurs entre 0 et 1 (norme l^∞). Cela est effectué en divisant chaque élément c_j par le plus grand élément du vecteur des coûts $max(|c_j|), j \in J$. De cette manière, la valeur choisie de l peut s'appliquer sur différentes fonctions objectif. Bastert cite également d'autres types de normalisation, comme la norme l^1 (diviser les coûts par $\sum_{j \in J} |c_j|$) et la norme l^2 (diviser les coûts par $\sum_{j \in J} (c_j)^2$)

Concernant le paramètre θ qui contrôle l'historique de préférences, utiliser $\theta = 0$ comme dans la version de Wedelin élimine l'historique des itérations précédentes dans le calcul courant. Alors que $\theta > 0$ prend en considération les décisions antérieures. Généralement, il est difficile de choisir le bon réglage puisque le choix du paramètre dépend de l'implémentation de l'algorithme et de l'instance à résoudre. Pour trouver le meilleur jeu de paramètres, il semble intéressant d'appliquer une méthode de réglage automatique de paramètres (voir chapitre 3). Nous présentons le réglage automatique des paramètres de l'algorithme de Wedelin dans le chapitre suivant.

4.5 Généralisation de l'algorithme de Wedelin

La version de base de l'algorithme de Wedelin [Wedelin, 1995a] et la version améliorée de [Mason, 2001] permettent de résoudre uniquement des problèmes linéaires avec des contraintes d'égalité, des variables binaires et des coefficients en 0/1, comme le problème de Set Partitiong (voir chapitre 2). Pour résoudre des problèmes avec des contraintes plus générales, nous pouvons les transformer en cette forme basique, mais cela peut augmenter la taille du problème d'une manière exponentielle. En conséquence, un traitement implicite (dans l'algorithme) de la généralisation des contraintes peut améliorer la performance de l'algorithme et la qualité des solutions. Dans cette optique, [Bastert et al., 2010] propose une généralisation de l'algorithme de Wedelin pour traiter des contraintes d'inégalité avec des variables entières et des coefficients entiers positifs ou négatifs.

4.5.1 Inégalités

Nous considérons les contraintes d'inégalité de la forme suivante

$$\underline{b}_i \leq \sum_{j \in J} x_j \leq \overline{b}_i, \tag{4.18}$$

où $\underline{b}_i$ et $\overline{b}_i$ représentent, respectivement, les bornes inférieure et supérieure de la contrainte i. Introduire une contrainte d'égalité est effectué en mettant $\underline{b}_i = \overline{b}_i = b_i$.

De manière générale, la transformation d'une contrainte d'inégalité en contrainte d'égalité est effectuée en ajoutant des variables d'écart de coût nul. En effet, la contrainte d'inégalité (4.18) est reformulée comme suit :

$$\sum_{j \in J} x_j + e_i = \bar{b}_i, \tag{4.19}$$

où $0 \leq e_i \leq \bar{b}_i - \underline{b}_i$ est une variable d'écart de coût nul. Cependant, au lieu d'ajouter ces variables d'écart explicitement au problème, comme dans [Wedelin, 1995a], [Bastert et al., 2010] propose de traiter ces inégalités d'une manière implicite en modifiant les étapes de mise à jour des contraintes.

Exemple : Soit la contrainte d'inégalité suivante :

$$2 \leq x_1 + x_2 + x_3 + x_4 \leq 4$$

Soit r le vecteur de coûts réduits r_j correspondant aux variables x_j, $j = 1, ..., 4$ de cette contrainte

$$r = (-1,2 \quad -5,2 \quad -3,1 \quad 2)$$

Après le classement des coûts réduits (ligne 12 de l'algorithme 2),

$$r_2 = -5,2 \quad < \quad r_3 = -3,1 \quad < \quad r_1 = -1,2 \quad < \quad r_4 = 2,$$

l'algorithme procède à la satisfaction de la borne inférieure en mettant les ($\underline{b}_i = 2$) premières variables à 1, ici :

$$\hat{x}_2 = 1, \quad \hat{x}_3 = 1$$

Ensuite, il procède à la satisfaction de la borne supérieure, en mettant

$$\hat{x}_j = \begin{cases} 1 & si \quad r_j < 0, \\ 0 & si \quad r_j > 0. \end{cases}$$

Cela correspond à

$$\hat{x}_1 = 1, \quad \hat{x}_4 = 0$$

4.5.2 Coefficients en $-1/0/1$

Contrairement à [Wedelin, 2013] qui mentionne sans détail que l'algorithme In-the-middle peut résoudre des problèmes avec des coefficient -1 en plus des coefficients 0 et 1, [Bastert et al., 2010] décrit sa méthode en détaillant toutes les étapes de l'algorithme.

Cela est résumé dans l'algorithme 3 qui décrit les différentes étapes de traitement d'une contrainte d'inégalité avec des coefficients en $-1/0/1$. Après le calcul des coûts réduits (lignes 1-11 de l'algorithme 2), l'algorithme repère les coefficients négatifs. Il inverse ensuite leurs signes ainsi que ceux des préférences et des coûts réduits correspondants. Puis il ajuste les bornes inférieure et supérieure de la contrainte traitée. Par exemple, pour une contrainte i, nous avons :

$$\underline{b}_i \leq x_1 + x_2 - x_3 - x_4 \leq \bar{b}_i \quad \Rightarrow \quad \underline{b}_i + 2 \leq x_1 + x_2 + x'_3 + x'_4 \leq \bar{b}_i + 2 \quad \text{(i.e., } x'_3 = 1 - x_3 \text{ et } x'_4 = 1 - x_4\text{)}$$

Avec ces étapes, le problème devient un problème avec des coefficients en 0/1. Nous pouvons donc le résoudre en utilisant l'algorithme 2 à partir de l'étape du calcul des coûts réduits (ligne 13 de l'algorithme 2). A la fin du traitement de la contrainte, nous corrigeons les bornes et le signe des coefficients, des préférences et des variables négatifs.

Algorithm 3: Traitement d'une contrainte d'inégalité avec des coefficients en $-1/0/1$ [Bastert et al., 2010]. (Complément de l'algorithme 2)

```
   // Exécuter l'algorithme 2 sur la contrainte i jusqu'au calcul des coûts réduits
1  Algorithm2 (lignes 1-11)
   // Trouver les variables avec des coefficients négatifs
2  C ← {j : a_ij < 0}
   // Inverser le signe des coefficients négatifs, des préférences et des coûts
      réduits correspondants
3  for j ∈ C do
4      a_ij ← −a_ij;   p_ij ← −p_ij;   r_j ← −r_j
   // Ajuster les bornes supérieure et inférieure de la contrainte i
5  b̲_i ← b̲_i + Σ_{j∈C} a_ij;   b̄_i ← b̄_i + Σ_{j∈C} a_ij
   // Exécuter l'algorithme 2 sur la contrainte i sans coefficients négatifs
6  Algorithm2 (lignes 12-20)
   // Corriger les bornes
7  b̲_i ← b̲_i − Σ_{j∈C} a_ij;   b̄_i ← b̄_i − Σ_{j∈C} a_ij
   // Corriger le signe des coefficients, des préférences et des variables à
      coefficients négatifs
8  for j ∈ C do
9      a_ij ← −a_ij;   p_ij ← −p_ij;   x̂_j ← 1 − x̂_j
```

Pour un cas général, [Bastert et al., 2010] aborde également le traitement des variables entières et des coefficients dans $\mathbb{N}^+$. Son idée est de considérer chaque variable entière x_j, avec une borne supérieure u_j et un coût c_j, comme un ensemble de u_j variables binaires. Il propose un traitement local de ces variables et des coefficients généraux lors de la mise à jour des contraintes. Pour plus de détail, nous invitons le lecteur à se référer à l'article [Bastert et al., 2010].

4.6 Conclusion du chapitre

Dans ce chapitre, nous avons présenté l'heuristique de base de Wedelin et son extension pour traiter des problèmes avec une structure plus générale. Dans cette thèse, nous nous intéressons aux problèmes avec des égalités ou des inégalités et des coefficients en $-1/0/1$ comme les problèmes de Set Packing, Set Covering et Set Partitioning (voir chapitre 2). C'est pour cette raison que nous n'avons détaillé que ces deux extensions. Nous présentons dans le chapitre suivant nos améliorations apportées sur l'algorithme de Wedelin et le réglage automatique de ses paramètres.

Chapitre 5

Réglage automatique de l'algorithme de Wedelin et ses extensions

Sommaire

5.1 Introduction

Intéressés par la résolution des problèmes de partitionnement (voir chapitre 2), nous avons implémenté en C++ la version généralisée de l'algorithme de Wedelin (voir algorithme 3, chapitre 4). Le solveur s'appelle BARYONYX[1] [Maqrot et al., 2018a, Maqrot et al., 2018b]. Il a deux modes d'exécutions : un mode de résolution et un mode d'optimisation. Dans le mode de résolution, BARYONYX s'exécute une seule fois avec un objectif de satisfaire toutes les contraintes du problème, il s'arrête à la première solution admissible trouvée. Dans le mode d'optimisation, BARYONYX fonctionne avec des exécutions répétées et simultanées sur différents processeurs. Son objectif est d'améliorer la solution trouvée à chaque exécution et d'indiquer la meilleure lorsqu'il atteint le temps limite.

Dans ce chapitre, nous présentons les extensions de l'algorithme de Wedelin implémentées dans le solveur BARYONYX, comme l'ordre de traitement des contraintes et le choix de la solution initiale. Nous présentons également une synthèse de ses paramètres et leur réglage automatique. A la fin de ce chapitre, nous évaluons la performance de BARYONYX en le comparant à d'autres solveurs de programmation linéaire, sur un ensemble d'instances recueillies de plusieurs collections de problèmes.

5.2 Extensions de l'algorithme

5.2.1 Ordre des contraintes et des variables

Il est bien connu, en optimisation, que le choix de l'ordre des contraintes influe souvent sur l'efficacité d'un algorithme. Inspirés du travail de [Bastert et al., 2010], nous avons ajouté la possibilité de trier les contraintes à deux niveaux : au pré-traitement (avant la résolution) et lors du traitement des contraintes violées (pendant la résolution).

Pré-traitement

L'objectif du pré-traitement est de simplifier l'ordre de traitement des contraintes, réduire la taille du problème et éliminer les redondances (variables et contraintes par substitution). En effet, BARYONYX trie les variables selon leur ordre de déclaration dans la fonction objectif puis dans la description des contraintes. Par exemple, pour un problème avec quatre variables (x_1, x_2, x_3 et x_4),

$$\begin{array}{ll} \min & 3x_4 \\ s.c. & x_1 + x_4 = 1 \\ & x_1 + x_2 + x_3 + x_4 = 1 \\ & x_1, x_2, x_3, x_4 \in \{0,1\}, \end{array}$$

BARYONYX utilise l'ordre suivant : $x_4 < x_1 < x_2 < x_3$. Cet ordre est utilisé pour trier les variables à l'intérieur des contraintes, permettant d'améliorer la mémoire cache en la rendant plus rapide. Concernant les contraintes, l'utilisateur peut fixer l'ordre souhaité (égalité / inégalité inférieure / inégalité supérieure) selon le type du problème à résoudre. En général, les contraintes d'égalité sont les plus contraignantes (moins de possibilités pour les satisfaire). Cela est montré par le tableau 5.1 qui présente un exemple sur le nombre de solutions possibles pour satisfaire les trois types de contraintes. Le tableau montre que les inégalités ont plus de possibilités de satisfaction par rapport aux égalités, avec un avantage

[1] https ://github.com/quesnel/baryonyx

Contrainte	Nombre de possibilités pour satisfaire la contrainte
$x_1+x_2+x_3=1$	3 possibilités : $(x_1=1,x_2=0,x_3=0)$ $(x_1=0,x_2=1,x_3=0)$ $(x_1=0,x_2=0,x_3=1)$
$x_1+x_2+x_3\leq 1$	4 possibilités : $(x_1=0,x_2=0,x_3=0)$ $(x_1=1,x_2=0,x_3=0)$ $(x_1=0,x_2=1,x_3=0)$ $(x_1=0,x_2=0,x_3=1)$
$x_1+x_2+x_3\geq 1$	7 possibilités : $(x_1=1,x_2=0,x_3=0)$ $(x_1=0,x_2=1,x_3=0)$ $(x_1=0,x_2=0,x_3=1)$ $(x_1=1,x_2=1,x_3=0)$ $(x_1=0,x_2=1,x_3=1)$ $(x_1=1,x_2=0,x_3=1)$ $(x_1=1,x_2=1,x_3=1)$

TABLE 5.1 – Nombre de possibilités pour satisfaire la contrainte $x_1+x_2+x_3=(\leq ou \geq)1$.

pour les contraintes d'inégalité supérieure. De manière générale, nous avons n possibilités pour satisfaire une contrainte $x_1+...+x_n=1$, $(n+1)$ possibilités pour une contrainte $x_1+...+x_n\leq 1$ et (2^n-1) possibilités pour une contrainte $x_1+...+x_n\geq 1$. En conséquence, classer les contraintes par catégories peut aider l'algorithme dans la résolution.

Pour résumer, les options implémentées dans BARYONYX permettant un pré-traitement de l'ordre de lecture des variables et des contraintes sont :

1. *none* : ordre des contraintes et des variables suivant le fichier d'entrée.
2. *memory* : ordre des contraintes suivant le fichier d'entrée, avec tri des variables dans les contraintes.
3. *lge* : tri des contraintes selon leur type ($\leq$ puis $\geq$ puis $=$), avec tri des variables.
4. *leg* : tri des contraintes selon leur type ($\leq$ puis $=$ puis $\geq$), avec tri des variables.
5. *gle* : tri des contraintes selon leur type ($\geq$ puis $\leq$ puis $=$), avec tri des variables.
6. *gel* : tri des contraintes selon leur type ($\geq$ puis $=$ puis $\leq$), avec tri des variables.
7. *elg* : tri des contraintes selon leur type ($=$ puis $\leq$ puis $\geq$), avec tri des variables.
8. *egl* : tri des contraintes selon leur type ($=$ puis $\geq$ puis $\leq$), avec tri des variables.

Pour tenter de montrer l'utilité du pré-traitement, nous avons comparé les différentes options. Nous avons choisi 30 instances du problème des 10-reines pondérées présenté dans le chapitre 2, vu qu'il contient des égalités et des inégalités (uniquement inférieures). Le tableau 5.2 indique, pour chaque option du pré-traitement, la solution trouvée en exécutant BARYONYX en mode optimisation. En comparant l'option *none* (sans tri des variables) avec l'option *memory* (avec tri des variables), nous remarquons que BARYONYX avec *memory* améliore 3 solutions sur 30 trouvées avec l'option *none*. Ce qui montre l'utilité du tri des variables dans l'amélioration des solutions.

Instances	Optimum	*none*	*memory*	*le*	*el*
10-queens-problem-0	293	∞	∞	**300**	∞
10-queens-problem-1	222	∞	∞	∞	∞
10-queens-problem-2	308	343	343	**309**	343
10-queens-problem-3	187	222	**214**	245	**214**
10-queens-problem-4	246	**246**	**246**	**246**	**246**
10-queens-problem-5	206	**206**	**206**	**206**	**206**
10-queens-problem-6	261	∞	∞	∞	∞
10-queens-problem-7	278	∞	∞	**307**	∞
10-queens-problem-8	295	∞	∞	∞	∞
10-queens-problem-9	273	356	**328**	356	**328**
10-queens-problem-10	260	∞	∞	∞	∞
10-queens-problem-11	270	**270**	**270**	**270**	**270**
10-queens-problem-12	221	276	**263**	276	**263**
10-queens-problem-13	363	**363**	**363**	**363**	**363**
10-queens-problem-14	167	**167**	**167**	**167**	**167**
10-queens-problem-15	207	**207**	**207**	256	**207**
10-queens-problem-16	239	**239**	**239**	**239**	**239**
10-queens-problem-17	292	375	375	**321**	375
10-queens-problem-18	134	**134**	**134**	**134**	**134**
10-queens-problem-19	277	**277**	**277**	**277**	**277**
10-queens-problem-20	241	**244**	**244**	**244**	**244**
10-queens-problem-21	241	298	298	**241**	298
10-queens-problem-22	266	**266**	**266**	335	**266**
10-queens-problem-23	225	**259**	**259**	**259**	**259**
10-queens-problem-24	191	**191**	**191**	**191**	**191**
10-queens-problem-25	230	**230**	**230**	**230**	**230**
10-queens-problem-26	292	**292**	**292**	**292**	**292**
10-queens-problem-27	235	**247**	**247**	**247**	**247**
10-queens-problem-28	249	**249**	**249**	338	**249**
10-queens-problem-29	227	∞	∞	∞	∞
Meilleures solutions		16	**19**	18	**19**
Infaisabilités		7	7	**5**	7
Moyenne (μ)	245	259	**257**	266	**257**
Écart-type (σ)	45	61	59	**57**	59
Coef. ($C_v = \frac{100\sigma}{\mu}$)	18%	24%	23%	**21%**	23%

TABLE 5.2 – Résultats de comparaison des différentes options du pré-traitement pour la résolution de 30 instances du problème des 10-reines pondérées, générées avec des valeurs de poids aléatoires.[baryonyx library, 2017]
Coût de la meilleure solution satisfaisant la totalité des contraintes, obtenu par BARYONYX après 60 secondes d'exécution en mode optimisation en utilisant un seul processeur.
∞ : problème infaisable (aucune solution trouvée en 60 secondes).
Optimum : solutions optimales obtenues par CPLEX V12.8.

.

Concernant les contraintes, nous n'avons que deux types : égalités (e) et inégalités inférieures (l). Par conséquent, les options *leg*, *lge* et *gle* représentent le même ordre : l puis e. Idem pour les options *gel*, *egl* et *elg* dont l'ordre est e puis l. De plus, pour chaque instance, les contraintes sont classées par ordre d'égalité puis d'inégalité dans le fichier d'entrée. Nous allons ainsi comparer deux classes d'options : (*leg*, *lge* et *gle*) noté ordre *le* avec (*memory*, *gel*, *egl* et *elg*) noté ordre *el*. D'après les résultats du tableau 5.2, nous avons 18 meilleures solutions et 5 cas d'infaisabilités pour les options d'ordre *le*, contre 19 meilleures solutions et 7 cas d'infaisabilités pour les options d'ordre *el*. Ce dernier ordre a en moyenne la meilleure solution (257 contre 266 pour l'ordre *le*) mais avec une valeur d'écart-type plus grande (59 contre 57). Pour classer les deux catégories d'options du pré-traitement sur ce jeu d'instances, nous avons utilisé le coefficient de variation C_v, qui permet de comparer la dispersion des valeurs (dans différentes grandeurs) autour de la moyenne.

$$C_v = \frac{\sigma}{\mu} \times 100\% \tag{5.1}$$

où σ est l'écart-type et μ représente la moyenne. En se basant sur cette mesure, nous pouvons conclure que l'ordre *le* (inégalité inférieure puis égalité) est le meilleure pour résoudre ce jeux d'instances du problème des 10-reines pondérées.

Suite à ces expérimentations, nous avons fixé, par défaut, l'option *memory* qui permet de trier les variables sans modifier l'ordre de lecture des contraintes. Ce choix est également basé sur l'existence d'une autre extension, permettant de classer les contraintes violées lors de la résolution du problème (sous-section suivante). En conséquence, choisir l'ordre de tri des contraintes violées doit prendre en considération l'ordre des contraintes en pré-traitement. Par exemple, choisir un ordre aléatoire pour les contraintes violées annule l'effet de classement des contraintes en pré-traitement.

Tri des contraintes violées

Trier les contraintes violées dans l'algorithme de Wedelin consiste à choisir un ordre de traitement des éléments de l'ensemble R dans l'algorithme 2. Nous avons sept options : 1) garder l'ordre des contraintes comme spécifié dans le fichier d'entrée, 2) inverser leur ordre, 3) utiliser un ordre aléatoire ou trier les contraintes selon un 4) ordre croissant ou 5) décroissant de leur infaisabilité. Cette dernière est mesurée par l'équation suivante :

$$\frac{1}{\max_j |a_{ij}|} \begin{cases} 0 & si \quad l_i \leq \sum_j a_{ij}\hat{x}_j \leq u_i \\ l_j - \sum_j a_{ij}\hat{x}_j & si \quad \sum_j a_{ij}\hat{x}_j \leq l_i \\ \sum_j a_{ij}\hat{x}_j - u_j & si \quad u_i \leq \sum_j a_{ij}\hat{x}_j \end{cases} \tag{5.2}$$

où $l_i \leq \sum_j a_{ij}x_j \leq u_i$ est la contrainte à satisfaire et $\hat{x}$ la solution courante.

En plus de ces cinq options citées dans [Bastert et al., 2010], nous avons ajouté deux options pour trier les contraintes violées selon leurs valeurs associées des multiplicateurs Lagrangiens : 6) un ordre croissant ou 7) un ordre décroissant.

Les stratégies de tri des contraintes violées implémentées dans BARYONYX sont ainsi :

1. *none* : ordre des contraintes dans le fichier d'entrée.
2. *reversing* : ordre inverse des contraintes dans le fichier d'entrée.
3. *random* : ordre aléatoire des contraintes
4. *infeasibility-incr* : ordre croissant selon la mesure de l'infaisabilité.

5. *infeasibility-decr* : ordre décroissant selon la mesure de l'infaisabilité.
6. *lagrangian-incr* : ordre croissant selon les valeurs des multiplicateurs lagrangiens.
7. *lagrangian-decr* : ordre décroissant selon les valeurs des multiplicateurs lagrangiens.

Le tableau 5.3 présente les résultats expérimentaux de la résolution d'un ensemble d'instances recueillies de plusieurs problèmes : problème des n-reines pondérées, problème de Set Partioning et problème de Set Covering (voir chapitre 2). Dans ces expérimentations, nous avons exécuté BARYONYX en mode optimisation en utilisant 20 processeurs et un temps limite de 60 secondes.

Avec la stratégie *lagrangian-decr*, BARYONYX trouve 8 meilleures solutions sur 16, contre 7 avec *lagrangian-incr*, 6 avec *infeasibility-decr* ou *random*, et 5 avec *none*, *reversing* ou *infeasibility-incr*. Pour analyser plus précisément ces résultats, nous avons classé les stratégies pour chaque instance suivant les valeurs de solutions (de la meilleure à la pire). Puis nous avons utilisé la moyenne des rangs pour faire un classement total (dernière ligne du tableau 5.3). Les stratégies de classement par rapport aux valeurs de multiplicateurs Lagrangiens (rang(*lagrangian-decr*)=1, rang(*lagrangian-incr*)=2) sont en tête du classement, suivies des stratégies de classement par rapport à la mesure d'infaisabilité (rang(*lagrangian-decr*)=2, rang(*lagrangian-incr*)=3), puis la stratégie de tri aléatoire (rang(*random*)=4), et en dernier lieu les stratégies de tri suivant l'ordre de lecture ou l'ordre inverse des contraintes dans le fichier d'entrée (rang(*none*)=5, rang(*reversing*)=6).

Nous pouvons expliquer ce classement par le fait que chaque contrainte est liée à un multiplicateur Lagrangien. Les valeurs de tous les multiplicateurs Lagrangiens permettent de calculer une borne duale au problème primal. La meilleure borne possible est la solution optimale au problème dual (maximum, si l'objectif du problème primal est une minimisation) (voir chapitre 4). En conséquence, classer les contraintes selon les valeurs des multiplicateurs Lagrangiens revient à les classer en fonction de leur contribution dans le calcul de la borne duale. Le choix de l'ordre de tri des contraintes dépend de l'objectif d'optimisation du problème primal. En d'autres termes, choisir un ordre décroissant en cas de minimisation, oriente l'algorithme à satisfaire en premier, les contraintes contribuant le plus dans la maximisation de la borne duale. Similairement, choisir un ordre croissant en cas de maximisation revient à satisfaire en premier les contraintes contribuant le plus dans la minimisation de la borne duale.

Pour les stratégies de tri des contraintes selon l'infaisabilité, nous constatons que malgré leurs classements avantageux, elles ont presque le même nombre de meilleures solutions (*infeasibility-decr* (6 solutions), *infeasibility-incr* (5 solutions)) que les stratégies *random* (6 solutions), *none* (5 solutions) et *reversing* (5 solutions). Cela s'explique du fait que lors du calcul de l'infaisabilité, nous pouvons avoir une même valeur pour plusieurs contraintes et pour les classer, nous utilisons un tri aléatoire.

En se basant sur ces résultats, nous avons décidé de fixer la stratégie *random* (ordre aléatoire) par défaut dans BARYONYX. Ce choix vient du fait que cette stratégie ne dépend pas de l'objectif du problème. De plus, elle garantit d'avoir un ordre différent à chaque itération (changement de direction dans l'espace de recherche). Cela permet d'éviter les cycles. En d'autres termes, éviter d'avoir le même ordre de traitement des contraintes (se bloquer dans une même zone de recherche), ce qui n'aide pas à trouver une solution ou à l'améliorer.

Instances	Solutions	Stratégies de tri des contraintes violées						
		none	*reversing*	*random*	*infeasibility-incr*	*infeasibility-decr*	*lagrangian-incr*	*lagragian-decr*
10queens[1]	25	**25**	**25**	**25**	**25**	**25**	**25**	**25**
30queens[1]	101	**103**	104	**103**	**103**	**103**	**103**	**103**
40queens[1]	154	**165**	**165**	**165**	**165**	**165**	**165**	**165**
50queens[1]	173	204	217	**191**	215	204	204	194
go19[2]	84	**101**	**101**	**101**	117	117	117	117
macrophage[2]	374	789	797	791	783	770	**767**	777
tanglegram2[2]	443	1186	1262	1212	986	1019	**866**	1078
toll-like[2]	610	1314	1362	1298	1309	1309	1293	**1283**
sppaa03[3]	49649	49923	49884	49923	49873	49883	50029	**49850**
sppaa06[3]	27040	27395	27389	27382	27363	27385	27299	**27256**
sppnw20[3]	10898	17577	17577	17577	**17553**	17556	17556	18054
sppkl02[3]	219	229	230	229	229	230	229	**228**
t0415[4]	5339422	**5346798**	**5346798**	**5346798**	**5346798**	**5346798**	**5346798**	**5346798**
t1716[4]	157442	188683	192438	188683	196558	**184458**	195352	187853
v0419[4]	2590326	2590376	**2590361**	2590376	2590376	2590376	**2590361**	2590376
v1618[4]	1153871	1160584	1161073	1160555	1159659	**1159511**	1159789	1159785
Nb. meilleures solutions		5	5	6	5	6	7	**8**
Rang		5	6	4	3	2	2	**1**

TABLE 5.3 – Exemple de résolution (valeurs de solutions trouvées après 60s d'exécution en mode optimisation en utilisant 20 processeurs) avec les différentes stratégies de classement des contraintes sur un ensembles de problèmes :
[1] : Problème des n-reines pondérées [CFLib, 2018]
[2] : Problème de Set Covering [MIPLIB, 2010]
[3] : Problème de Set Partitioning [Beasley, 1990a]
[4] : Problème de tournées de véhicules [Borndörfer, 1998].
Solutions : coût des meilleures solutions connues, calculées avec CPLEX.
Gras : meilleure valeur trouvée parmi les sept stratégies.
Rang : calculé par rapport à la moyenne des rangs de toutes les instances.

Les autres stratégies restent comme options que l'utilisateur peut choisir pour résoudre des instances avec un ordre de contraintes spécifiques.

5.2.2 (Ré)initialisation de solution

Comme l'algorithme de Wedelin s'exécute plusieurs fois dans BARYONYX, nous avons décidé de changer à chaque itération la politique de (ré)initialisation de solution. Cela permet de bien explorer les zones de recherche et d'éviter de se bloquer dans un optimum local (principe des métaheuristiques). Nous avons trois manières pour construire une solution initiale :

1. *bastert* : en fonction du vecteur des coûts (lignes 2-4 de l'algorithme 2, chapitre 4),
2. *random* : aléatoire, en utilisant un paramètre $p \in [0,1]$ de la loi de Bernoulli. Ce paramètre est une probabilité pour tirer la valeur 1 dans le vecteur solution (prendre la valeur 1 avec une probabilité p et la valeur 0 avec une probabilité $1-p$).
3. *best* : en utilisant la meilleure solution trouvée aux itérations précédentes. La première exécution s'effectue avec la politique *bastert*.

De plus, nous avons ajouté trois politiques permettant d'alterner les trois stratégies *bastert*, *random* et *best* pendant la résolution (une politique à chaque exécution de BARYONYX) :

5. *bastert-cycle* : commencer par *bastert* puis alterner *random* > *best* > *bastert*,
6. *random-cycle* : commencer par *random* puis alterner *best* > *bastert* > *random*,
7. *best-cycle* : commencer par la politique *bastert*. Si l'algorithme ne trouve pas de solution, il applique la politique *random*, sinon il utilise la meilleure solution trouvée comme solution initiale à la prochaine itération. Ensuite, s'il améliore la solution trouvée, il garde la politique *Best*, sinon il compte trois exécutions avant de basculer vers la politique *bastert*. Puis il refait les mêmes étapes, en essayant d'améliorer la meilleure solution trouvée. La figure 5.1 résume les différentes étapes de la politique *best-cycle*.

Afin de diversifier plus la recherche de solution, nous avons utilisé le paramètre p de la loi de Bernouli. L'objectif est de définir le pourcentage du vecteur solution sur lequel s'applique une des politiques. Autrement dit, (ré)initialiser les valeurs de quelques variables du vecteur solution. Ces variables sont choisies aléatoirement. Par exemple, pour un vecteur de 10 variables,

$$\hat{x} = \begin{pmatrix} \hat{x}_1 & \hat{x}_2 & \hat{x}_3 & \hat{x}_4 & \hat{x}_5 & \hat{x}_6 & \hat{x}_7 & \hat{x}_8 & \hat{x}_9 & \hat{x}_{10} \end{pmatrix},$$

fixer $p = 0,5$, (ré)initialise aléatoirement 5 variables du vecteur ($\hat{x}_1$, $\hat{x}_2$, $\hat{x}_4$, $\hat{x}_7$ et $\hat{x}_{10}$ par exemple) avec la politique choisie. Les valeurs des autres variables sont définies avec la politique *random*.

Le tableau 5.4 présente les résultats de comparaison des six politiques de (ré)initialisation (*bastert*, *random*, *best*, *bastert-cycle*, *random-cycle* et *best-cycle*) appliquées sur 50% du vecteur solution ($p = 0,5$), plus la politique *bastert* sur la totalité du vecteur solution ($p = 1$). Cette dernière représente la version basique de l'algorithme de Wedelin (voir algorithme 2). Dans ces expérimentations, nous avons choisi le même ensemble d'instances présenté dans la sous-section précédente 5.2.1 sur le tri des contraintes

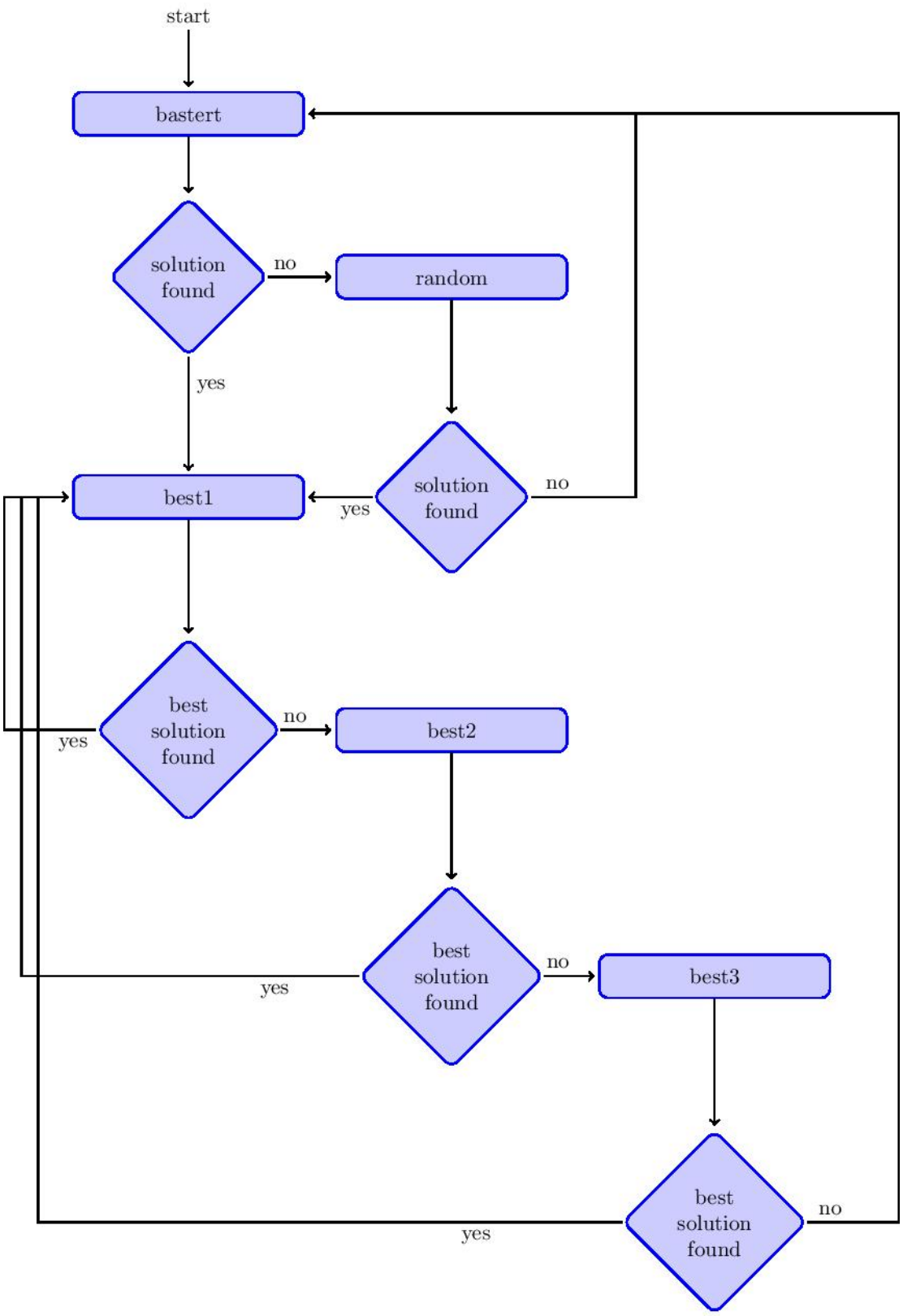


FIGURE 5.1 – L'algorithme *best-cycle* est conçu à la fois pour améliorer la diversification et pour rechercher une meilleure solution lorsqu'une solution est trouvée. Chaque case représente une exécution complète de l'algorithme de Wedelin. Le solveur s'arrête lorsqu'il atteint le temps limite.

violées. Nous avons exécuté BARYONYX en mode optimisation en utilisant 20 processeurs et un temps limite de 60 secondes.

En comparant les solutions fournies par la version basique de *bastert* ($p = 1$) avec les six politiques de (ré)initialisation sur 50% du vecteur solution, nous remarquons que la politique *best* est la meilleure. Elle trouve 9 meilleures solutions sur 16, contre 7 pour la politique *best-cyle*, 6 pour la version basique *bastert* ($p = 1$), 5 pour *random* et 3 pour *bastert* ($p = 0,5$), *bastert-cycle* et *random-cycle*.

En regardant le classement par rapport à la moyenne des rangs de toutes les instances, nous constatons que la politique *best-cycle* est en tête du classement, suivie de la politique *best*, puis de *bastert* ($p = 1$) et *bastert-cycle*, et en dernier lieu de *random* et *random-cycle*.

En effet, construire une solution initiale en fonction de la meilleure solution trouvée aux itérations précédentes (politique *best*) aide l'algorithme à intensifier et affiner la recherche autour d'un optimum local (exploitation du voisinage). En outre, construite la solution initiale d'une manière aléatoire (politique *random*) permet de diversifier les directions dans l'espace de recherche. La politique *bastert* avec une probabilité p adopte également ce principe de diversification, tout en gardant la politique de base de l'algorithme. Pour $p = 0,5$, l'algorithme construit 50% de la solution en fonction du vecteur des coûts et 50% restante d'une manière aléatoire. Idem pour la politique *best* ($p = 0,5$), 50% du vecteur solution est construit avec la meilleure solution trouvée précédemment et 50% restante d'une manière aléatoire. D'autre part, cycler les trois politiques *bastert random* et *best* permet d'alterner entre la diversification, l'intensification et l'utilisation de la même solution. Cependant, pour les deux politiques *bastert-cycle* et *random-cycle*, l'algorithme est incapable de choisir la meilleure politique à chaque exécution vu que l'ordre est fixé par défaut. Au contraire, la politique *best-cycle* effectue un choix en fonction de la solution trouvée, i.e., elle applique *best* pour affiner la recherche autour d'une solution trouvée (optimum local), et change de politique (*random* ou *bastert*) pour diversifier la recherche en cherchant un optimum global.

En somme, nous avons décidé de fixer la politique *best-cycle* par défaut avec une probabilité de $p = 0,5$. Ce choix est effectué pour 1) diversifier la recherche de solution tout en évitant d'aller dans la même direction de recherche (politique *bastert* avec $p = 1$), 2) éviter de stagner dans un optimum local (politique *best* avec $p = 1$), 3) éviter de s'éloigner de l'optimum en générant une solution entièrement aléatoire (politique *random*) et 4) éviter un mauvais choix de politique à chaque exécution (politiques *bastert-cycle* et *random-cycle*).

Instances	Solutions	Politiques de (ré)initialisation du vecteur solution						
		bastert ($p = 1$)	*bastert* ($p = 0,5$)	*random* ($p = 0,5$)	*best* ($p = 0,5$)	*bastert-cycle* ($p = 0,5$)	*random-cycle* ($p = 0,5$)	*best-cycle* ($p = 0,5$)
10queens[1]	25	**25**	**25**	**25**	**25**	**25**	**25**	**25**
30queens[1]	101	**103**	104	104	**103**	108	104	**103**
40queens[1]	154	165	165	**162**	165	165	165	**162**
50queens[1]	173	**191**	207	194	194	201	197	194
go19[2]	84	**101**	125	124	**101**	122	127	125
macrophage[2]	374	785	838	827	**784**	828	835	815
tanglegram2[2]	443	1080	2039	1970	**974**	1936	2000	1526
toll-like[2]	610	1340	1388	1395	**1323**	1385	1378	1368
sppaa03[3]	49649	49923	49838	49813	49921	49804	49840	**49791**
sppaa06[3]	27040	27423	27123	27157	27382	27150	27172	**27065**
sppnw20[3]	10898	17577	**17478**	**17478**	17577	**17478**	**17478**	**17478**
sppkl02[3]	219	229	229	229	**228**	229	229	229
t0415[4]	5339420	**5346798**	**5346798**	**5346798**	**5346798**	**5346798**	**5346798**	**5346798**
t1716[4]	157442	188683	188683	**184171**	188683	188683	188683	188683
v0419[4]	2590330	**2590376**	2595148	2594381	**2590376**	2593184	2598305	2593328
v1618[4]	1153870	∞	∞	∞	∞	∞	∞	∞
Nb. meilleures solutions		6	3	5	**9**	3	3	7
Moyenne des rangs		3.250	4.625	3.875	2.937	3.812	4.625	**2.875**
Classement		3	6	5	2	4	6	**1**

TABLE 5.4 – Exemple de résolution (valeurs de solutions trouvées après 60s d'exécution en mode optimisation en utilisant 20 processeurs) avec les différentes politiques de (ré)initialisation de solution, appliquées sur 50% du vecteur solution ($p = 0,5$), sur un ensembles de problèmes :
[1] : Problème des n-reines pondérées [CFLib, 2018]
[2] : Problème de Set Covering de la librairie [MIPLIB, 2010]
[3] : Problème de Set Partitioning [Beasley, 1990a]
[4] : Problème de tournées de véhicules [Borndörfer, 1998]
Solutions : coût de meilleures solutions connues, calculées avec CPLEX.
Gras : meilleure valeur trouvée parmi les six politiques.
Classement effectué par rapport à la moyenne des rangs de toutes les instances.

5.2.3 Dépendance du paramètre l avec les coûts réduits

Comme présenté dans le chapitre 4, le paramètre l est une perturbation aléatoire de petite valeur, ajoutée aux coûts réduits pour assurer des valeurs non nulles tout au long du calcul. Malheureusement, ce paramètre a une forte dépendance avec les valeurs des coûts réduits. Une grande valeur de l détruit complètement les résultats alors qu'une petite valeur n'a presque aucun impact. Une des solutions est de normaliser les coûts à l'entrée de l'algorithme pour n'avoir que des valeurs entre 0 et 1. Toutefois, choisir la bonne valeur (proche de 0 ou de 1) dépend du problème à résoudre. Pour cela, nous avons adopté une stratégie de réglage automatique qui privilégie l'effet mémoire.

Normalisation des coûts

Pour éviter la forte dépendance du paramètre l avec les valeurs des coûts, nous avons trois types de normalisation, où nous divisons (une seule fois au début de l'algorithme) chaque élément $c_j, j \in J$ du vecteur des coûts par une valeur :

1. l^1 : diviser par $\sum_{j \in J} |c_j|$
2. l^2 : diviser par $\sum_{j \in J} (c_j)^2$
3. l^∞ : diviser par $max(|c_j|)$.

Dans la version améliorée de [Mason, 2001, Bastert et al., 2010], l'algorithme de Wedelin trie les éléments du vecteur des coûts réduits dans un ordre croissant (algorithme 2). Or, ce vecteur peut avoir des valeurs similaires. Pour résoudre ce cas, nous avons ajouté une quatrième option,

4. l^{random},

où nous modifions la normalisation l^∞. L'idée est d'ajouter une petite valeur aléatoire aux coûts réduits pour différentier les valeurs similaires. Par exemple, pour un vecteur des coûts réduits r,

$$r = \begin{bmatrix} 1 & 5 & 2 & 5 & 2 & 2 & 8 \end{bmatrix}.$$

trier ses éléments tout en ayant des valeurs différentes revient à écrire :

$$r^{trie} = \begin{bmatrix} 1 & 2+ & 2++ & 2+++ & 5+ & 5+++ & 8 \end{bmatrix},$$

où le symbole $+$ représente une petite valeur aléatoire qui tend vers zéro, avec $+ < ++ < +++$.

Réglage automatique du paramètre l

Comme présenté précédemment, la valeur du paramètre l dépend des valeurs des coûts réduits. Choisir donc une valeur par défaut est très difficile, i.e., elle peut donner de bons résultats ou les détruire complètement. En effet, nous avons réfléchi à un réglage automatique en fonction des coûts réduits. Pour un vecteur de coûts réduits r, nous considérons la valeur du plus petit élément non nul.

$$v = \min_{j, r_j \neq 0} (|r_j|) \tag{5.3}$$

Ensuite, nous calculons la valeur de l en fonction de la valeur du paramètre θ, chargé de contrôler l'historique de la satisfaction des contraintes :

$$l = v(1 - \theta). \tag{5.4}$$

Choix d'une stratégie de normalisation

Pour choisir une stratégie de normalisation par défaut, nous avons comparé les différentes méthodes (*none* (sans normalisation), l^1, l^2, l^∞ et l^{random}) avec différents réglages du paramètre l : un réglage avec une valeur fixée (10^{-2} et 10^{-3}) et le réglage automatique par défaut. Le tableau 5.5 indique les résultats de comparaison sur le même ensemble d'instances utilisé dans les sous-sections précédentes. Nous avons exécuté BARYONYX en mode optimisation avec 20 processeurs et un temps limite de 60 secondes.

Nous avons considéré deux critères de comparaison. Pour chaque option de normalisation, nous avons calculé le nombre de meilleures solutions sur l'ensemble des instances. Puis nous les avons classées en se basant sur la valeur de la moyenne des rangs de toutes les instances.

En analysant les résultats, nous remarquons que les méthodes de normalisation l^1 et l^2 sont généralement les moins bonnes quel que soit le réglage du paramètre l, i.e, elles donnent moins de meilleures solutions et sont en dernier du classement par rapport aux autres stratégies (*none*, l^∞ et l^{random}). Le classement de ces dernières varie selon le réglage du paramètre l. Nous constatons que le réglage $l = 10^{-2}$ donne des solutions de moins bonne qualité en comparant au réglage automatique et au réglage $l = 10^{-3}$. En outre, la meilleure stratégie est l^∞. Elle est en tête du classement avec le réglage automatique et en deuxième lieu avec le réglage $l = 10^{-3}$. La stratégie l^{random} est également au premier rang du classement avec moins de meilleures solutions (3 face à 5) que la stratégie l^∞. Or, elle est moins bonne avec le réglage automatique (au 8ème rang face au 1er pour l^∞).

D'après cette analyse, nous avons choisi de mettre la méthode de normalisation l^∞ par défaut dans BARYONYX, avec un réglage automatique du paramètre l. Pour avoir plus de performance en terme de qualité de solutions, nous allons régler le paramètre l en combinaison avec les autres paramètres du solveur. L'objectif est de définir un jeu de paramètres optimisé pour une famille d'instances. Nous présentons dans la section suivante un récapitulatif des paramètres de BARYONYX avec le protocole de réglage adopté.

Instances	Options de normalisation des coûts réduits														
	none	l^1	l^2	l^∞	l^{random}	*none*	l^1	l^2	l^∞	l^{random}	*none*	l^1	l^2	l^∞	l^{random}
	réglage automatique de l					$l = 10^{-2}$					$l = 10^{-3}$				
10queens[1]	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**
30queens[1]	105	105	105	105	104	**103**	125	127	**103**	109	**103**	110	129	107	107
40queeens[1]	164	**162**	165	**162**	164	165	249	293	165	165	164	212	288	164	**162**
50queens[1]	194	194	207	201	204	**190**	435	408	194	204	**190**	394	408	197	204
go19[2]	124	124	126	123	126	132	117	**115**	131	133	132	125	126	130	130
macrophage[2]	818	830	829	816	805	853	779	**773**	861	845	817	801	782	831	792
tanglegram2[2]	**1728**	1797	1853	1777	1901	2015	1952	2113	2092	1772	1964	2066	2078	2107	2031
toll-like[2]	1389	1380	1388	1370	1427	1447	1249	**1232**	1452	1433	1398	1318	1308	1405	1424
sppaa03[3]	49786	49808	49808	49839	49839	49724	65004	51047	49823	49871	49758	65017	65835	**49649**	49650
sppaa06[3]	27124	27132	27079	27146	27155	27065	36708	36720	27099	27077	27063	36918	37473	**27043**	27045
sppnw20[3]	17478	17478	17478	17232	17478	**16812**	21822	21822	17478	**16812**	**16812**	19740	21822	**16812**	**16812**
sppkl02[3]	228	229	229	229	262	**219**	239	239	**219**	220	**219**	242	239	**219**	220
t0415[4]	5346798	5346798	5346798	**5339422**	5346798	5346798	5346798	5346798	5346798	5346798	5346798	5346798	5346798	5346798	5404140
t1716[4]	188683	188683	188683	184171	191019	188093	520428	525681	**165191**	172806	186953	511395	525671	173042	181353
v0419[4]	2596123	2594248	2594953	**2593697**	2593898	2598379	2801600	2800749	2599574	2600916	2600168	2802061	2800024	2593825	2597762
v1618[4]	1163988	1164566	1164580	1165315	1165806	**1161657**	1526212	1527602	1173164	1173848	1162790	1529819	1530320	1163377	1162232
Nb. meilleures	2	2	1	4	1	**6**	1	4	4	2	5	1	1	5	3
Moy. rangs	4,94118	5,17647	5,64706	**4,70588**	6,23529	5,47059	7,11765	7,64706	6,58824	6,41176	4,88235	7,88235	8,29412	4,82353	**4,70588**
Classement	4	5	7	**1**	8	6	11	12	10	9	3	13	14	2	**1**

TABLE 5.5 – Exemple de résolution (valeurs de solutions trouvées après 60s d'exécution en mode optimisation en utilisant 20 processeurs) avec les différentes options de normalisation des coûts.
[1] : Problème des n-reines pondérées [CFLib, 2018]
[2] : Problème de Set Covering de la librairie [MIPLIB, 2010]
[3] : Problème de Set Partitioning [Beasley, 1990a]
[4] : Problème de tournées de véhicules [Borndörfer, 1998]
Gras : meilleure valeur trouvée parmi les quinze options.
Classement effectué par rapport à la moyenne des rangs de toutes les instances.

5.2.4 Traitement des cas particuliers

Nous avons décrit dans le chapitre 4 les étapes de l'algorithme de Wedelin (voir l'algorithme 2) pour la résolution d'un problème linéaire en variable 0/1 (cas de minimisation). Pour rappel, l'algorithme parcourt toutes les contraintes du problème en appliquant une recherche duale de descente par coordonnées. Autrement dit, il parcourt tous les éléments du vecteur $\hat{\pi}$ de multiplicateurs Lagrangiens (variables duales). Pour satisfaire une contrainte i : $\sum_j a_{ij}x_j = b_i$, l'algorithme met à jour le multiplicateur Lagrangien $\hat{\pi}_i$. Pour le faire, il trie les coûts réduits (calculés par l'équation 4.13) dans un ordre croissant. Puis, il calcule $\hat{\pi}_i$ en fonction du $b_i^{\text{ème}}$ et $(b_i+1)^{\text{ème}}$ petits éléments selon l'équation suivante :

$$\hat{\pi}_i = \hat{\pi}_i + \frac{1}{2}(r_{[b_i]} + r_{[b_i+1]}). \tag{5.5}$$

Par exemple, pour une contrainte

$$x_1 + x_2 + x_3 + x_4 = 2$$

avec un vecteur trié de coûts réduits

$$r = \begin{pmatrix} -4 & -3 & 1 & 4 \end{pmatrix},$$

la valeur du multiplicateur Lagrangien est la suivante

$$\hat{\pi}_i = \frac{1}{2}(-4-3) = -3,5$$

Or, pour une contrainte d'égalité avec un second membre nul ($b_i = 0$), comme

$$-x_1 - x_2 + x_3 + x_4 = 0,$$

l'utilisation de l'équation (5.5) est impossible, car le $(b_i = 0)^{\text{ème}}$ petit élément du vecteur des coûts réduits n'existe pas. Idem pour une contrainte d'égalité où le nombre de variables est égale à la valeur du second membre, comme

$$x_1 + x_2 + x_3 + x_4 = 4,$$

le $(b_i+1) = 5^{\text{ème}}$ élément du vecteur des coûts réduits n'existe pas.

Nous avons donc modifié l'équation (5.5) pour traiter ces cas particuliers.

Contrainte d'égalité avec un second membre nul

Nous considérons la plus petite valeur des coûts réduits triés r_j correspondant aux variables de la contrainte

$$\hat{\pi} = \hat{\pi}_i + \frac{\min(r_j)}{2} \tag{5.6}$$

Contrainte d'égalité avec un second membre égal au nombre de variables

Nous considérons la plus grande valeur des coûts réduits triés r_j et sa valeur double

$$\hat{\pi} = \hat{\pi}_i + \frac{3}{2}\max(r_j) \tag{5.7}$$

5.3 Réglage automatique des paramètres

5.3.1 Synthèse des paramètres

Le tableau 5.6 récapitule les différents paramètres de BARYONYX, classés en deux catégories : les paramètres numériques continus et les paramètres catégoriques de type chaîne de caractères. Ce dernier type représente les extensions de l'algorithme décrites dans ce chapitre (voir la sous-section 5.2). Suite à des expérimentations, nous avons fixé ces paramètres (choix par défaut dans BARYONYX) :

- *preprocessing* = *memory*
- *constraint-order* = *random*
- *init-policy* = *best-cycle*
- *norm* = l^{∞}
- $p = 0,5$

Concernant les paramètres numériques, nous avons remarqué, en pratique, que leurs valeurs dépendent du problème à résoudre. Les régler manuellement pour chaque problème demande beaucoup de temps et ne garantit pas d'avoir la meilleure configuration. En conséquence, une des solutions est d'utiliser une méthode de réglage automatique. De plus, appliquer une méthode d'analyse de sensibilité avant le réglage permet de réduire le temps de ce dernier en se focalisant sur les paramètres les plus importants.

5.3.2 Choix des méthodes de réglage

Dans le chapitre 3, nous avons présenté un panorama de méthodes d'analyse de sensibilité, que nous avons classé en deux catégories :

- Des méthodes de sensibilité locale dont l'objectif est d'étudier la variabilité d'un seul paramètre en fixant les autres.
- Des méthodes de sensibilité globale qui s'intéresse à l'ensemble des domaines de variation des paramètres, qui qualifient (comme la méthode de Morris) ou quantifient (comme la méthode de Sobol) l'influence des paramètres d'une méthode d'optimisation sur le résultat fourni.

Choisir une méthode adéquate nécessite de spécifier clairement les objectifs de l'analyse. En effet, nous cherchons à trouver un jeu de paramètres qui conduit à des performances optimales pour un ensemble d'instances donné d'un problème posé, avec un contrat de temps fixe. Donc, nous nous intéressons nécessairement aux méthodes d'analyse de sensibilité globale.

Dans cette optique, nous avons présenté dans le chapitre 3, deux grilles de sélection d'une méthode d'analyse de sensibilité globale. La sélection dépend du niveau de connaissance du modèle (boîte noire (Figure 3.4, chapitre 3) ou boîte blanche (Figure 3.5)), la nature de ses paramètres (discret ou continu) et leurs domaines de variation. Nous considérons BARYONYX comme un modèle boîte noire déterministe.

Paramètre	Domaine	Type	Description
t	$[0,+\infty[$	Continus	- Temps maximal (en secondes) pour arrêter l'algorithme
κ_{min}	$[0,\kappa_{max}[$		- Valeur minimal du degré d'approximation κ (équation (4.12))
κ_{max}	$[\kappa_{min},1]$		- Valeur maximal de κ
κ_{step}	$[0,(\kappa_{max}-\kappa_{min})]$		- Valeur de mise à jour de κ
θ	$[0,1]$		- Paramètre qui contrôle l'historique
l	$[0,+\infty[$		- Valeur de la perturbation aléatoire des coûts réduits
α	$[0,+\infty]$		- Paramètre adaptatif de κ (au nombre actuel des contraintes violées)
p	$[0,1]$		- Paramètre de la loi de bernoulli $p \in [0,1]$ utilisé pour choisir le pourcentage du vecteur solution initial ou la probabilité de trier les uns dans la solution pour la politique *init-policy=random*
i	$[0,+\infty[$	Discrets	- Nombre des itérations maximal pour arrêter l'algorithme
w	$[0,i[$		- Nombre des itérations avant la mise à jour de κ
preprocessing	*none* ***memory*** $\leq$ / $\geq$ / $=$ (6 combinaisons)	Catégoriques	- Ordre de lecture des contraintes et des variables du problème
constraint-order	*none* *reversing* ***random*** *infeasibility-incr* (ou *-decr*) *lagrangian-incr* (ou *-decr*)		- Ordre de traitement des contraintes violées
init-policy	*bastert* *random* *best* *bastert-cycle* *random-cycle* ***best-cycle***		- Politique de (ré)initialisation du vecteur solution
norm	l^1 l^2 $\boldsymbol{l^\infty}$ l^{random}		- Choix de la fonction de normalisation des coûts

TABLE 5.6 – Paramètres de BARYONYX,
avec leurs types, domaines de définition et descriptions. les paramètres en **gras** sont configurés par défaut.

De ce fait, nous considérons la première grille de sélection (Figure 3.4) dont le choix d'une méthode dépend du nombre et de la nature des paramètres. Si tous les paramètres sont discrets, les méthodes utilisables sont les plans d'expériences. Si ils sont continus, la première approche à appliquer est la méthode de Morris. Pour plus de précision, la grille propose des méthodes gourmandes en nombre de simulations comme la méthode de Sobol. En cas d'un nombre important de paramètres, la méthode recommandée est l'approche de criblage par groupes.

D'après le tableau 5.6, BARYONYX a dix paramètres numériques à régler : huit continus et deux discrets. Comme les paramètres sont majoritairement continus, nous avons choisi la méthode de Morris en ne considérant que des valeurs continues (nous prenons l'arrondi pour les deux paramètres discrets). Ce choix répond à notre objectif qui consiste à classer les paramètres par rapport à leur influence sur le résultat afin de régler uniquement les paramètres importants.

Concernant le réglage automatique, nous avons présenté dans le chapitre 3 différents types de méthodes : 1) Optimisation séquentielle à base de modèles (SMBO) pour les situations où le nombre de paramètres est grand, 2) les méthodes de Racing basées sur des tests statistiques qui sont souvent utilisés pour choisir un algorithme de résolution pour une application quelconque, 3) les plans d'expériences qui sont limités au réglage d'un nombre restreint de paramètres (par exemple, 5 paramètres pour l'outil CALIBRA), 4) la recherche locale comme la méthode paramILS qui est limitée au réglage des paramètres catégoriques (elle nécessite de discrétiser à priori les domaines de variation dans le cas des paramètres numériques), et 4) les méthodes évolutionnaires comme le paquet RGENOUD qui est plus adapté au réglage des paramètres continus. Ayant une dizaine de paramètres numériques, nous avons choisi d'utiliser le paquet RGENOUD dans un contexte continu.

5.3.3 Protocole de réglage des paramètres

Nous cherchons à définir un jeu de paramètres universel pour une famille d'instances d'un problème donné. Le protocole consiste à :

1. **Choisir un ensemble d'apprentissage**
   Nous sélectionnons 20% d'instances de l'ensemble total. Pour cela, nous choisissons une instance sur cinq classées par ordre alphabétique. Ce choix est effectué pour sélectionner un ensemble d'apprentissage qui représente, si possible toutes les sous-familles d'instances.

2. **Calculer la qualité d'un jeu de paramètre**
   La qualité d'un jeu de paramètre $(x_1, \ldots, x_K)$ est calculée comme suit. Premièrement, nous exécutons, d'une manière parallèle, notre solveur BARYONYX sur chaque ensemble d'apprentissage, en utilisant le réglage de paramètres de WEDELIN GOOD modifié[2] (voir Table 5.9). Chaque exécution utilise Γ' processeurs[3] en parallèle pendant un temps limite. Nous collectons les meilleures solutions l_e^{init} et les pires solutions u_e^{init} trouvées par Algorithm 1 à chaque instance e. Si aucune solution n'est trouvée, nous supprimons cette instance de l'ensemble d'apprentissage. Dans cette

[2]nous fixons $\alpha = 1$, $p = 0.5$ avec la politique de (ré)initialisation de solution best-cycle, et un ordre aléatoire des contraintes violées à chaque itération.

[3]Dans nos expérimentations, nous utilisons un petit nombre de processeurs $\Gamma' = 3$ dans la phase de réglage de paramètres par rapport à la phase de résolution ($\Gamma = 30$).

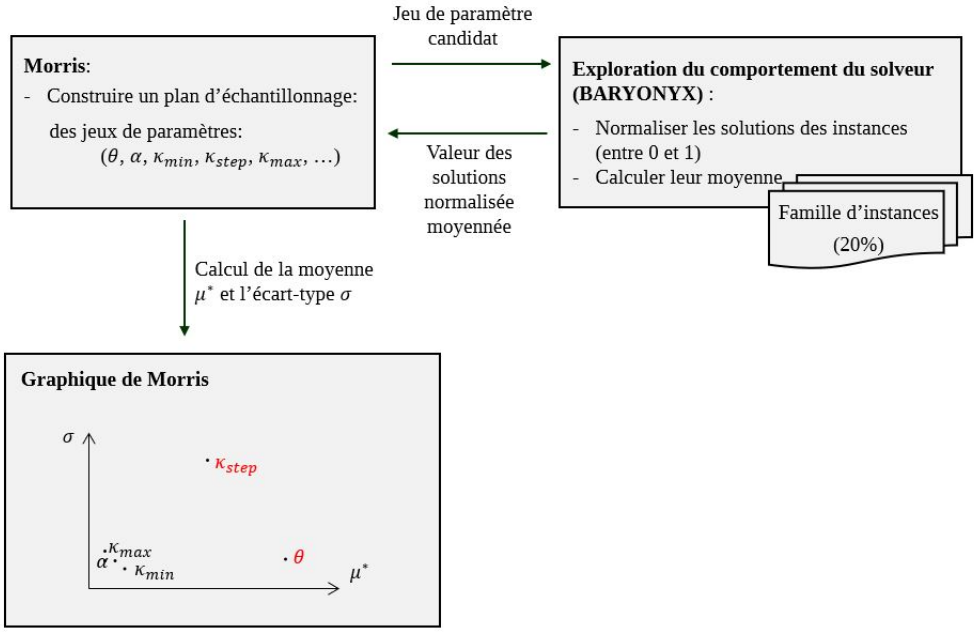


FIGURE 5.2 – Protocole appliqué pour l'analyse de sensibilité des paramètres de BARYONYX, en utilisant la méthode de Morris.

phase d'apprentissage, la qualité normalisée d'un jeu de paramètres est obtenue en calculant la moyenne (sur les V instances valides d'apprentissage) de la distance relative par rapport aux meilleures solutions initiales :

$$y(x_1, \ldots, x_K) = 1 - \frac{1}{V} \sum_{e=1}^{V} \frac{l_e - l_e^{init}}{10u_e^{init} - l_e^{init}} \tag{5.8}$$

Si aucune solution n'est trouvée pour une instance quelconque, nous supposons que $l_e = 10u_e^{init}$. Lorsque $y > 1$, cela veut dire que nous avons trouvé un jeu de paramètres meilleur que celui de WEDELIN GOOD.

3. **Appliquer une analyse de sensibilité**
Nous avons choisi la méthode de Morris. Cette dernière génère un plan d'échantillonnage, i.e., un ensemble de jeux de paramètres candidats. Ce plan est définit en fonction du nombre de paramètres à régler N, du niveau de discrétisation de leurs domaines de définition Q, du nombre de trajectoires r, et de la longueur du pas de déplacement δ entre deux points d'une trajectoire (voir chapitre 3 pour plus de détails). Le solveur à régler, BARYONYX, s'exécute avec chaque jeu de paramètres candidat pour l'ensemble des instances d'apprentissage. Les valeurs des solutions obtenues sont normalisées (selon l'équation 5.8). Ensuite, leur moyenne est renvoyée à Morris. Ce dernier analyse les résultats et élabore un graphique qui classe les paramètres selon leur impact sur la variation de la solution. La figure 5.2 résume les différentes étapes du protocole appliqué.

4. **Régler automatiquement les paramètres les plus importants**
Pour régler les paramètres les plus importants définis par Morris, nous avons choisi le paquet RGENOUD, que nous avons appliqué sur le même ensemble d'apprentissage. RGENOUD se base sur les algorithmes génétiques. Au début, il construit une population initiale, où chaque individu représente un jeu de paramètres. Ensuite, BARYONYX

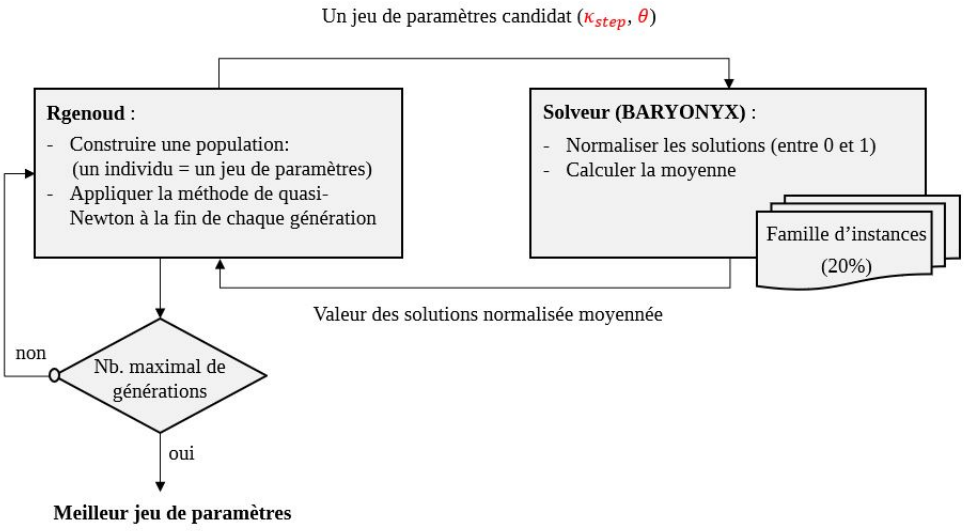


FIGURE 5.3 – Protocole appliqué pour le réglage automatique des paramètres de BARYONYX, en utilisant le paquet RGENOUD

évalue chaque jeu de paramètres candidat sur les instances d'apprentissage. Il normalise les solutions trouvées (des valeurs entre 0 et 1) et calcule leur moyenne. Pour créer une nouvelle génération, RGENOUD sélectionne des individus de la population initiale. Puis, il utilise des opérateurs de croisement et de mutation pour construire une population enfant (voir les méthodes approchées, chapitre 1). À la fin de chaque génération, une méthode de quasi-Newton BFGS (voir chapitre 3) est appliquée sur la meilleure solution produite. Cette méthode utilise des informations sur les dérivées dans le voisinage des optima locaux. Elle permet une convergence rapide vers l'optimum global si cette solution est dans le voisinage correct.
Pour arrêter la recherche du jeu de paramètre optimisé, nous avons choisi le nombre de générations comme critère d'arrêt (fixé à 10 générations dans nos expérimentations). La figure 5.3 résume les étapes du protocole de réglage adopté.

5. **Appliquer le jeu de paramètres optimisé sur l'ensemble des instances**
Résoudre la totalité des instances avec le meilleur jeu de paramètres trouvé par RGENOUD.

5.4 Expérimentations

5.4.1 Instances

Nous avons conduit nos expérimentations sur un ensemble d'instances, de différentes tailles, recueillies de plusieurs problèmes d'optimisation combinatoire : problème des n-reines pondérées, problème de Set Partioning, problème de Set Packing et problème de Set Covering (voir chapitre 2).

Instances nqueens [CFLib, 2018]

Les instances nqueens représentent le problème des n-reines pondérées. Son objectif est de placer les n reines dans un échiquier de taille $n \times n$, de manière à avoir une reine par

rangée, colonne, et au plus une reine par diagonale ascendante et diagonale descendante. Chaque case est associée à un coût. Nous cherchons donc une combinaison avec un coût minimal. Nous avons crées ces instances par un générateur automatique[4] avec des coûts choisis aléatoirement entre 1 et n. Nous avons 8 instances avec des tailles différentes : de 8 à 1000 reines.

Instances MIPLIB [MIPLIB, 2010]

MIPLIB est une bibliothèque électronique qui contient des instances de différents types de problèmes, que se soit avec des variables entières ou mixtes. Intéressés par les problèmes linéaires avec des variables bivalentes et des coefficients de contraintes en $-1/0/1$, nous avons choisi uniquement des instances appartenant à cette catégorie de problèmes[5]. A savoir les problèmes de Set Packing, de Set Covering et de Set Partitioning. Au total, nous avons 29 instances : 19 instances faciles qui peuvent être résolues en une heure en utilisant un solveur commercial, 2 instances de difficulté moyenne et 8 instances difficiles dont les solutions optimales sont inconnues.

Instances SPP et SCP [Beasley, 1990a]

L'ensemble de test comprend 55 instances du problème de Set Partioning (SPP)[6] et 30 instances du problème de Set Covering (SCP)[7]. Ces problèmes sont tirés de la bibliothèque [Beasley, 1990a] et décrits dans l'article [Beasley, 1990b]. Les problèmes SPP, inspirés de la planification des équipages aériens, ont été fournies par K.L. Hoffman and D. Levine. Un sous-sous ensemble de ces problèmes a été résolu initialement dans [Hoffman and Padberg, 1993]. Concernant les problèmes SCP, les instances utilisées dans nos tests proviennent de [J.E.Beasley, 1987].

Instances Telebus [Borndörfer, 1998]

Le problème de Telebus[8] consiste à programmer des tournées de véhicules pour des personnes handicapées. L'objectif est de fournir un service ponctuel avec un minimum de coûts, en respectant un ensemble de contraintes, comme la capacité des véhicules et les pauses obligatoires. Le problème a été modélisé en utilisant l'approche de partitionnement, qui contient deux étapes. La première étape "clustering" identifie tous les circuits de bus possibles permettant de transporter plusieurs personnes à la fois. L'objectif est de sélectionner un ensemble de trajets de sorte que la distance de déplacement du véhicule soit minimale. Dans la seconde étape "chaining", les trajets sélectionnés sont enchaînés pour générer des circuits de bus respectant la totalité des contraintes avec l'objectif de minimiser la distance totale parcourue par les véhicules. Ces deux étapes représentent des problèmes de Set Partitioning. Pour tester la résolution de ce problème de Telebus, [Borndörfer, 1998] proposent 14 problèmes de "clustering" pour deux périodes différentes : (v0415-v0421) pour avril 15–22-1996 et (v1616-v1622) pour septembre 16–22-1999. Les 5 premières instances de chaque ensemble correspondent aux jours de la semaine du lundi

[4] https://forgemia.inra.fr/thomas.schiex/cost-function-library/tree/master/random/wqueens/

[5] http://miplib2010.zib.de/miplib2010-BP.php

[6] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/sppinfo.html

[7] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html

[8] http://www.zib.de/opt-long_projects/TrafficLogistics/Telebus/index.en.html

au vendredi, alors que les deux derniers correspondent au week-end. Les résultats de "clusterning" ont été utilisés pour construire les 14 instances de "chaining" correspondantes : (t0415–t0421) pour le mois d'avril et (t1716–t1721) pour le mois de septembre.

Instances CSPLib022 [Suniel, 1999]

CSPLib est une bibliothèque électronique regroupant un ensembles de problèmes classés par catégories. Nous avons choisi une famille d'instances de la catégorie des problèmes d'ordonnancement. Il s'agit du problème prob022[9] contenant 12 instances du problème de planification des chauffeurs de Bus, modélisé sous forme du problème de Set Partitioning. Les instances proviennent de trois compagnies de bus différentes : *Reading* (r1 à r5a), *CentreWest Ealing area* (c1, c1a et c2) et *the former London Transport* (t1 et t2).

Instances UAI Eval [Hurley et al., 2016]

Nous avons 113 instances de modèles graphiques probabilistes ou déterministes convertis en un problème de Set Partitioning. La plupart ont une formulation objectif non linéaire (souvent quadratique). Ces modèles proviennent de plusieurs domaines de recherche, à savoir la segmentation d'images et la conception de protéines. [Hurley et al., 2016] présentent ces instances en indiquant leurs sources de téléchargement. Elles sont regroupées dans la librairie evalgm[10]. Ces 113 instances représentent une sélection d'instances difficiles, pour la plupart non résolues [Ouali et al., 2017].

Instances VCS [Zaghrouti et al., 2014]

Nous avons deux instances VCS du problème de planification générées aléatoirement. VCS1200 est une instance de taille moyenne du problème de planification des chauffeurs ($\sim$1200 contraintes $\times$ 130000 variables), et VCS1600 une instance de grande taille du problème de la planification des tournées de véhicules ($\sim$1600 contraintes $\times$ 500000 variables). Les solutions optimales de ces problèmes sont connues, trouvées par GENCOL[11]. D'après [Zaghrouti et al., 2014], CPLEX ne parvient pas à trouver une solution admissible pour l'instance la plus large au bout de 10 heures d'exécution.

5.4.2 Protocole expérimental du réglage des paramètres

Pour résoudre la collection des problèmes (présentée ci-dessus) par notre solveur BARYONYX, nous procédons au réglage automatique de ses paramètres, en se basant sur le protocole décrit dans la sous-section 5.3.3. Le tableau 5.7 présente les paramètres à régler ($N = 9$) avec leurs domaines de variation (voir tableau 5.6 pour une description complète).

Premièrement, nous avons constitué un ensemble d'apprentissage, en sélectionnant 20% d'instances de chaque famille de problèmes. Ensuite, nous avons appliqué la méthode de Morris pour étudier la sensibilité des paramètres à régler. En d'autres termes, analyser l'effet de variation des valeurs des paramètres sur la variabilité du résultat.

[9] http://www.csplib.org/Problems/prob022

[10] http://genoweb.toulouse.inra.fr/~degivry/evalgm/

[11] GENCOL est un logiciel commercial développé par le centre de recherche GERAD. Il appartient à la société AD OPT, une division de KRONOS.

Paramètre	Gamme de variation			
i	[	10^2	, 10^5	]
w	[	0	, 100	]
θ	[	0	, 1	]
l	[	10^{-3}	, 10^{-1}	]
κ_{min}	[	0	, 5×10^{-1}	]
κ_{step}	[	10^{-4}	, 10^{-2}	]
κ_{max}	[	6×10^{-1}	, 1	]
α	[	0	, 2	]
p	[	10^{-4}	, $1 - 10^{-4}$	]

TABLE 5.7 – Paramètres de BARYONYX à régler, avec leurs domaines d'analyse.

Analyse de sensibilité : application de la méthode de Morris

L'analyse est réalisée avec une discrétisation de la gamme de variation de chaque paramètre en $Q = 10$ niveaux, avec une longueur de pas égale à $\delta = 5$. Le croisement de ces niveaux définit un ensemble de ($Q^N = 10^9$) nœuds. Le plan d'échantillonnage est constitué d'une suite de $r = 50$ trajectoires aléatoires, passant chacune par ($N + 1 = 10$) nœuds (chaque paramètre ne varie qu'une seule fois par trajectoire). L'analyse est donc effectuée avec $50 \times (9 + 1) = 500$ nœuds. Autrement dit, 500 jeux de paramètres candidats pour explorer le comportement du solveur BARYONYX.

La mise en œuvre est effectuée sous le logiciel R en utilisant le paquet Morris [Pujol et al., 2012]. Pour chaque jeu de paramètre candidat (nœud d'une trajectoire), BARYONYX s'exécute pendant 60 secondes en mode optimisation en utilisant 3 processeurs pour la résolution parallèle de l'ensemble d'apprentissage (3 processeurs pour chaque instance). En conséquence, le temps écoulé pour effectuer l'analyse est de $500 \times 60 = 30000$ secondes.

En se basant sur les résultats induits par l'utilisation de l'ensemble des jeux de paramètres, Morris calcule les indices de sensibilité : la moyenne des valeurs absolues des effets élémentaires μ^* et leur variance σ sur les r trajectoires. Nous synthétisons ces résultats dans le graphique de Morris, qui combine μ^* et σ pour l'ensemble des paramètres.

Interprétation des résultats de l'analyse de sensibilité

Nous avons obtenu presque le même graphique des indices de Morris pour les familles d'instances nqueens, MIPLIB, SPP, Telebus, CSPLib022 et UAI Eval, avec le même classement des paramètres. La figure 5.4 représente, comme exemple, le graphique des indices de Morris pour la famille d'instances CSPLib022. La figure montre trois classes de paramètres :

1. i, p, w, α et κ_{max} :
   Des paramètres avec des valeurs (μ^*,σ) égales ou très proches de l'origine (0,0). Ces paramètres ont un effet négligeable sur la résolution de ces familles d'instances. Par exemple, choisir une valeur entre 6.10^{-1} et 1 pour le paramètre κ_{max} ne modifierait pas la résolution, car la solution est toujours trouvée avant d'atteindre $\kappa_{max} = 6.10^{-1}$.

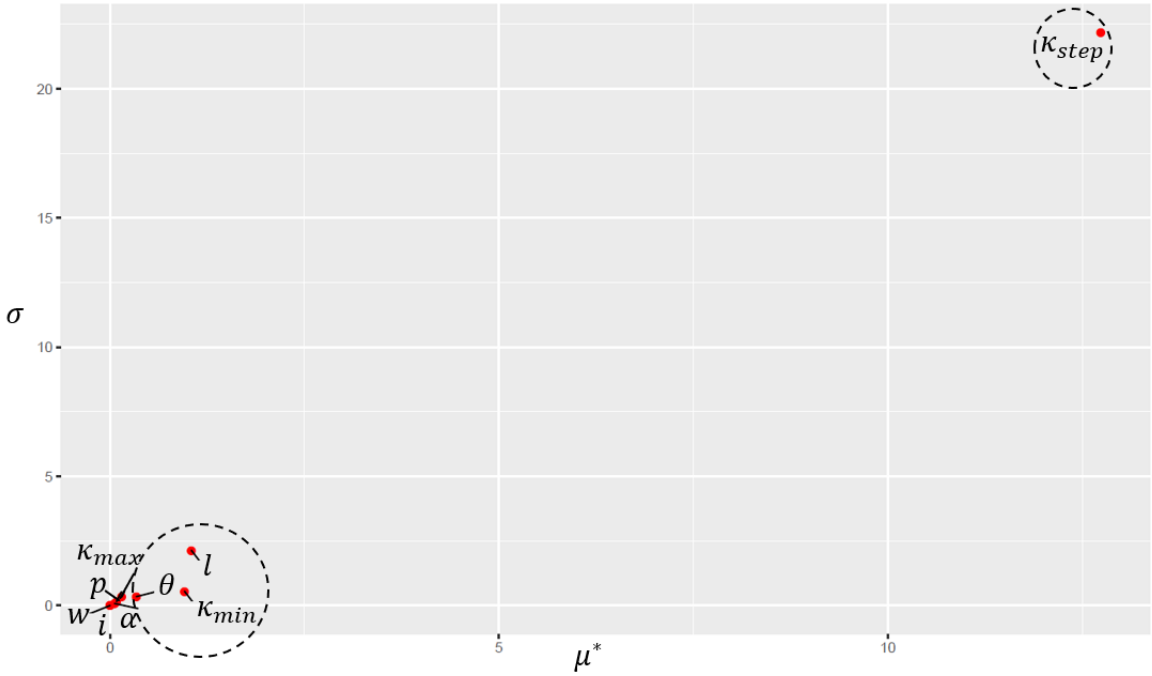


FIGURE 5.4 – Graphique des indices de Morris (la moyenne des valeurs absolues des effets élémentaires μ^* et leur variance σ) pour la famille d'instances CSPLib022.

2. l, κ_{min} et θ :
   Des paramètres avec des valeurs (μ^*,σ) différentes de l'origine (0,0), mais qui ne sont pas très élevées. Cela veut dire que dans les domaines de variation de ces paramètres, il existe peu de valeurs qui peuvent modifier le résultat fourni par BARYONYX. Par exemple pour le paramètre κ_{min} qui consiste à définir la valeur initiale d'approximation. Si nous commençons avec une petite valeur, l'algorithme met beaucoup de temps pour trouver une première ou même une bonne solution. Si nous choisissons une grande valeur, nous risquons de limiter notre espace de recherche en éliminant la zone qui contient la solution. Avec deux valeurs de κ_{min} entre ces deux dernières, le résultat de résolution pourrait être le même. Également pour le paramètres l (bruit aléatoire ajouté aux coûts réduits) et θ (chargé de contrôler l'historique de la satisfaction des contraintes), à partir d'une valeur ou avant cette valeur, le résultat de résolution de ces familles d'instances pourrait être le même.

3. κ_{step} :
   Un paramètre avec des valeurs très élevées de μ^* et σ qui sont du même ordre. Ce paramètre κ_{step} joue un rôle très important dans la résolution de ces familles d'instances. Autrement dit, la performance de BARYONYX est très sensible à la variation de ce paramètre, varier sa valeur peut changer complètement le résultat de résolution.

La seule famille d'instances où nous avons obtenu un classement de paramètres différent est celle de SCP [Beasley, 1990b]. La figure 5.5 présente le graphique des indices de Morris correspondant. De manière similaire à l'exemple précédent, ce graphique montre trois catégories de paramètres :

1. i, θ, w, α et κ_{max} :
   Des paramètres avec des valeurs nulles : $(\mu^*,\sigma) = (0,0)$, qui n'ont pas d'effet sur la résolution de cette famille d'instances.

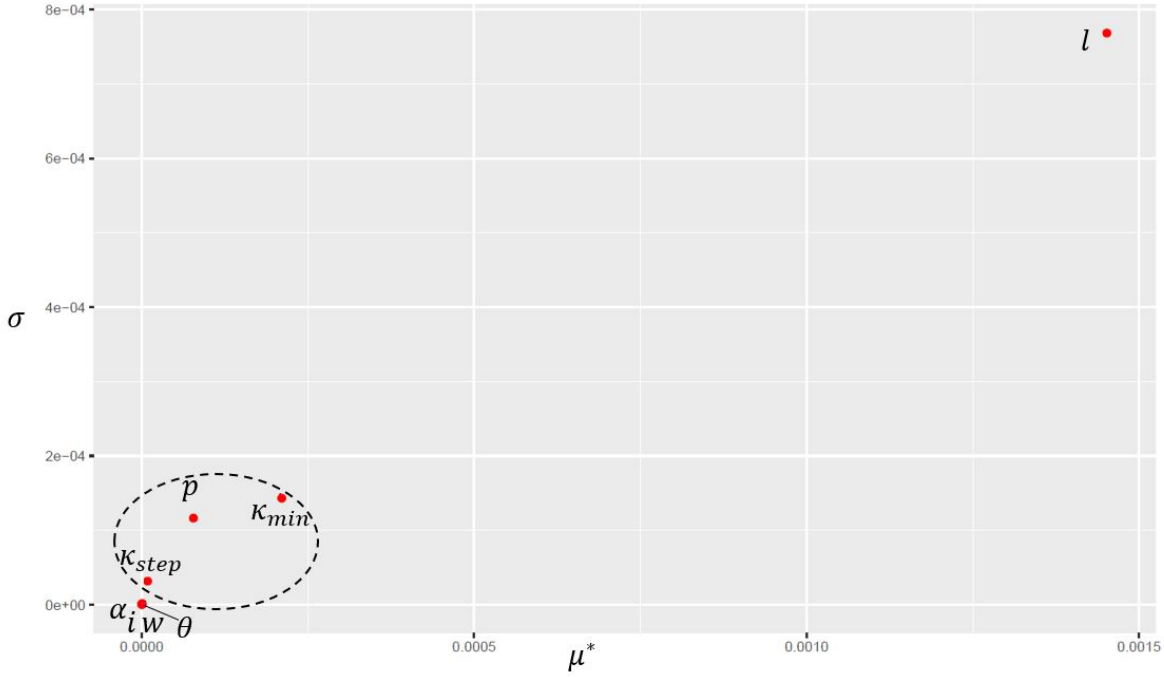


FIGURE 5.5 – Graphique des indices de Morris (la moyenne des valeurs absolues des effets élémentaires μ^* et leur variance σ) pour la famille d'instances SCP.

2. κ_{min}, p et κ_{step} :
   Des paramètres avec des valeurs (μ^*,σ) différentes de l'origine (0,0), mais qui ne sont pas très élevées.

3. l :
   Un paramètre avec des valeurs très élevées de μ^* et σ qui sont du même ordre.

Cette famille d'instances SCP représente le problème de Set Covering, où les contraintes ont beaucoup plus de possibilités de satisfaction (inégalités supérieures) par rapport à celles du problème de Set Partitioning (égalités) ou du problème des n-reines pondérées (inégalités inférieures). Le tableau 5.1 présente un exemple sur le nombre de possibilités pour satisfaire les trois types de contraintes.

En conséquence, le classement des paramètres par rapport à leurs sensibilités (Figure 5.5) n'est pas le même que pour les autres familles d'instances (Figure 5.4). En effet, comme les problèmes SCP ont un nombre important de solutions admissibles, le paramètre κ_{step} qui définit le degré d'approximation n'a pas beaucoup d'importance, car quel que soit sa valeur, nous trouvons souvent une solution admissible. En revanche, la variation du paramètre p (qui fixe le pourcentage de (ré)initialisation du vecteur solution) a un effet sur le résultat de la résolution, vu qu'il permet de diversifier la direction de recherche et donc de trouver une solution différente. Concernant le paramètre θ, nous remarquons qu'il n'a pas d'effet sur ce jeu d'instances. Nous pouvons expliquer cela du fait que l'algorithme n'a pas besoin de l'historique des itérations précédentes sur la satisfaction des contraintes, vu le nombre important de possibilités de satisfaction.

Concernant les instances VCS qui sont de grandes tailles, trouver une solution admissible est déjà un problème difficile. Par conséquent, 60 secondes d'exécution de BARYONYX est insuffisante pour résoudre ces problèmes. Au lieu de modifier les conditions, nous avons

Famille d'instances	Paramètres Importants	Paramètres négligeables
nqueens MIPLIB SPP Telebus CSPLib022 UAI Eval VCS	κ_{step} l κ_{min} θ	i w α κ_{max} p
SCP	l κ_{min} p κ_{step}	i w α κ_{max} θ

TABLE 5.8 – Résultat de l'analyse de sensibilité (méthode de Morris) : classement des paramètres de BARYONYX selon leur effet sur la variabilité des solutions.

considéré le même classement des paramètres trouvé pour les ensembles d'instances de type SPP, vu que les VCS sont également des problèmes de Set Partitioning (SPP).

Pour résumer, le tableau 5.8 présente, pour toutes les familles d'instances, les paramètres jugés les plus importants par Morris, ainsi que les paramètres avec effet négligeable sur la variabilité des solutions.

Choix des valeurs des paramètres avec effet négligeable

Pour fixer les valeurs des paramètres avec effet négligeable sur la variabilité de la résolution, nous nous sommes basés sur les valeurs des paramètres fournis par [Bastert et al., 2010]. Ces derniers proposent deux jeux de paramètres selon l'objectif cherché (voir Tableau 5.9) :

- Réglage WEDELIN FAST qui est un compromis entre la qualité et la rapidité de résolution.
- Réglage WEDELIN GOOD dont l'objectif est la qualité des solutions avec un temps de résolution raisonnable.

Basés sur les deux réglages WEDELIN FAST et WEDELIN GOOD, nous avons fixé un réglage par défaut des paramètres de BARYONYX, que nous avons validé par des expérimentations. La troisième colonne du tableau 5.9 présente les valeurs choisies : $\kappa_{min} = 0$, $\kappa_{step} = 1 \times 10^{-3}$ et $\kappa_{max} = 6 \times 10^{-1}$ pour éviter de dépasser le point de convergence (éliminer, de la recherche, une zone contenant la solution), $\theta = 5.10^{-1}$ (valeur intermédiaire entre les deux réglages FAST et GOOD) pour utiliser l'historique de la satisfaction des contraintes, $w = 20$ pour tenter de résoudre le problème sans approximation, $\alpha = 1$ pour exprimer la taille du pas d'approximation κ en fonction du nombre des contraintes violées (affiner la recherche lors de la convergence, voir section 4.4). Pour le paramètre l, nous avons choisi le réglage automatique que nous avons présenté dans la sous-section 5.2.3. Concernant le

Paramètre	Réglage		
	WEDELIN FAST	WEDELIN GOOD	BARYONYX par défaut
κ_{min}	0		0
κ_{step}	1×10^{-3}	2×10^{-4}	1×10^{-3}
κ_{max}	6×10^{-1}		6×10^{-1}
θ	4×10^{-1}	6×10^{-1}	5×10^{-1}
l	1×10^{-2}		auto. (section 5.2.3)
w	20		20
α	0		1
p	0		5×10^{-1}
i	100,000		100,000
preprocessing	*none*		*memory*
constraint-order	*none*		*random*
init-policy	*bastert*		*best-cycle*
norm	l^{∞}		l^{∞}

TABLE 5.9 – Réglage des paramètres de l'algorithme de Wedelin : deux réglages WEDELIN FAST et WEDELIN GOOD proposés par [Bastert et al., 2010], plus notre réglage par défaut de BARYONYX.

paramètre i qui définit le nombre des itérations, [Bastert et al., 2010] ne proposent aucune valeur. En conséquence, nous l'avons fixé à une valeur par défaut égale à 100.000.

Raglage automatique : application de la méthode de GENOUD

Nous avons effectué le réglage automatique des paramètres importants pour chaque famille d'instances. Nous avons utilisé le paquet RGENOUD [Mebane and Sekhon, 2012] sous le logiciel R. Le réglage est réalisé avec un nombre maximal de générations égal à 10. A chaque génération, nous avons une population de 100 individus. Donc, RGENOUD propose approximativement 1000 individus (1000 jeux de paramètres).

Pour chaque jeu de paramètres candidat, BARYONYX résout en parallèle les instances d'apprentissage (20%) d'une famille quelconque. Il s'exécute pendant 60 secondes en mode optimisation en utilisant 3 processeurs. Par conséquent, le temps écoulé pour effectuer l'analyse est de $1000 \times 60 = 60000$ secondes.

À la fin de chaque génération, RGENOUD affiche pour tout paramètre réglé : la meilleure valeur trouvée, la moyenne de toutes les valeurs et leur variance. Le tableau 5.10 résume pour chaque famille d'instances, le meilleur réglage trouvé par GENOUD au bout de 10 générations.

Comparaison du réglage optimisé de RGENOUD avec des réglages statiques

Pour évaluer l'utilité de notre approche de réglage automatique des paramètres, nous avons comparé les résultats de résolution de plusieurs familles d'instances (queens, MIPLIB, SPP, Telebus, CSPLib022, UAI Eval et VCS), en utilisant BARYONYX avec des réglages statiques (BARYONYX par défaut, BARYONYX FAST et BARYONYX GOOD) et avec

Famille d'instances	Réglage BARYONYX^{rgn}			
	κ_{min}	κ_{step}	l	θ
nqueens	$3,37 \times 10^{-2}$	$7,45 \times 10^{-3}$	$9,04 \times 10^{-3}$	$7,89 \times 10^{-1}$
MIPLIB	$0,00$	$1,99 \times 10^{-4}$	$1,35 \times 10^{-3}$	$5,53 \times 10^{-1}$
SPP et VCS	$0,00$	$1,88 \times 10^{-4}$	$3,19 \times 10^{-3}$	$2,85 \times 10^{-1}$
Telebus	$1,05 \times 10^{-1}$	$4,10 \times 10^{-4}$	$1,13 \times 10^{-2}$	$3,54 \times 10^{-1}$
CSPLib022	$0,00$	$2,00 \times 10^{-4}$	$1,00 \times 10^{-2}$	$6,00 \times 10^{-1}$
UAI Eval	$0,00$	$2,00 \times 10^{-4}$	$1,13 \times 10^{-2}$	$3,72 \times 10^{-1}$
	κ_{min}	κ_{step}	l	p
SCP	$6,49 \times 10^{-2}$	$1,98 \times 10^{-1}$	$2,23 \times 10^{-3}$	$2,69 \times 10^{-3}$

TABLE 5.10 – Meilleurs jeux de paramètres trouvés par RGENOUD pour les familles d'instances nqueens, MIPLIB, SPP, VCS, Telebus, CSPLib022, UAI Eval et SCP.

un réglage optimisé par RGENOUD sur chaque famille d'instances (BARYONYX^{rgn}). Nous avons utilisé 30 processeurs pour la résolution de ces problèmes, avec un temps limite de 60 secondes pour tous les ensembles sauf pour l'ensemble VCS où nous avons fixé 1800 secondes (des instances de grande taille).

Pour analyser et comparer les résultats, nous avons calculé pour chaque instance la distance relative à la meilleure solution connue

$$\frac{\text{solution obtenue} - \text{meilleure solution connue}}{|\text{meilleure solution connue}|} \times 100\% \tag{5.9}$$

Ensuite, nous avons calculé la moyenne des distances relatives pour toutes les instances d'une même famille (les non résolues ne sont pas considérées) (voir tableau 5.11).

D'après le tableau 5.11, le réglage BARYONYX^{rgn} trouvé par RGENOUD est meilleur par rapport aux réglages statiques (BARYONYX par défaut, BARYONYX FAST et BARYONYX GOOD) pour toutes les familles d'instances, à l'exception de l'ensemble MIPLIB. Rappelons que ce réglage a été paramétré sur 20% des instances de chaque famille et a été utilisé pour résoudre la totalité des instances. Les résultats présentés dans ce tableau montrent que ce réglage optimisé se généralise bien aux instances qui ne sont pas dans son jeu d'apprentissage (i.e. les instances de test). En revanche, il est moins bien pour l'ensemble des instances MIPLIB. Cela s'explique du fait que cette famille regroupe différents problèmes qui ne sont pas du même type (Set Packing, Set Covering, Set Partitioning et d'autres problèmes comme le problème de sac à dos). Or pour les autres familles, chacune représente des instances du même type de problème. Pour l'ensemble UAI Eval, nous remarquons que les solutions du réglage automatique sont relativement moins bien par rapport aux réglages statiques. Cela est expliqué par le nombre des instances résolues. Pour illustrer, la valeur 388,50% du réglage automatique est moyennée sur l'ensemble de 101/113 instances alors que la valeur 15,63% du réglage par défaut est calculée sur 82/113 instances. En comparant les résultats pour le même nombre d'instances (82/103), nous trouvons 8,03 % pour BARYONYX, 19,29 % pour WEDELIN FAST, 18,24 % pour WEDELIN GOOD et 17,08 % pour BARYONYX^{rgn}. Donc, le réglage optimisé de RGENOUD résout plus d'instances UAI Eval que les réglages statiques bien que les solutions soient de moins bonne qualité.

F. d'instances (temps limite)	Type de réglage des paramètres			
	BARYONYX	WEDELIN FAST	WEDELIN GOOD	BARYONYX^{rgn}
nqueens (60s)	*4,33 %* [6/8]	*8,11 %* [7/8]	5,44 % **[8/8]**	**0,61 %** **[8/8]**
MIPLIB (60s)	*768,10 %* **[21/29]**	*236,78 %* **[21/29]**	***236,74 %*** **[21/29]**	*771,09 %* **[21/29]**
SPP (60s)	0,06 % **[55/55]**	2,26 % **[55/55]**	3,41 % **[55/55]**	**0,01 %** **[55/55]**
Telebus (60s)	4,16 % **[28/28]**	7,89 % **[28/28]**	2,72 % **[28/28]**	**0,44 %** **[28/28]**
CSPLib022 (60s)	6,87 % **[12/12]**	**0,00 %** **[12/12]**	**0,00 %** **[12/12]**	**0,00 %** **[12/12]**
UAI Eval (60s)	***15,63 %*** [82/113]	*374,76 %* [97/113]	*370,30 %* [98/113]	*388,50 %* **[101/113]**
VCS (1800s)	1,83 % **[2/2]**	- [0/2]	- [0/2]	**0,07 %** **[2/2]**
SCP (60s)	5865,92 % **[30/30]**	64,11 % **[30/30]**	64,11 % **[30/30]**	**26,85 %** **[30/30]**
				Réglage manuel (SCPFAST) (tableau 5.12) **2,42 %**

TABLE 5.11 – Résultats de résolution de 7 familles de problèmes par BARYONYX en mode optimisation (30 processeurs et un temps limite de 60 secondes), avec un réglage statique (par défaut, FAST et GOOD) et un réglage optimisé sur chaque ensemble de problèmes (BARYONYX^{rgn}).

Pour chaque famille d'instances :
- 1ère ligne : valeur moyennée de la distance relative à la meilleure solution connue, calculée uniquement pour les instances résolues,
- 2ème ligne : nombre des solutions obtenues sur le nombre total des instances.
gras : les meilleurs résultats.
italique : une ou plusieurs instances ne sont pas résolues (leurs solutions ne sont pas comptabilisées).

Paramètre	Réglage manuel des instances SCP (SCPFAST)
κ_{min}	$\kappa_{min} = 0$
κ_{step}	$\kappa_{step} = 10^{-3}$
κ_{max}	6×10^{-1}
θ	4×10^{-1}
l	10^{-2}
w	20
α	0
p	999×10^{-3}
i	10
preprocessing	*memory*
constraint-order	*none*
init-policy	*bastert*
norm	l^{∞}

TABLE 5.12 – Réglage manuel des paramètres de BARYONYX pour l'ensemble des instances SCP.

Concernant les instances SCP de type Set Covering, nous remarquons que BARYONYX^{rgn} est meilleur que les autres réglages (26% face à 64,11 % pour les réglages de WEDELIN et 5865,92 % pour le réglage par défaut). La qualité moins bonne de ces résultats est expliquée par la nature du problème Set Covering qui se caractérise par des contraintes d'inégalité supérieure, où le nombre de solutions admissibles est très grand (voir tableau 5.1). Donc, l'algorithme a besoin de peu d'itérations par exécution pour trouver une solution. Or, lors du réglage automatique par RGENOUD, nous avons fixé une gamme de variation de $[10^2, 10^5]$. Pour confirmer cette remarque, nous avons testé un réglage manuel de 10 itérations avec une stratégie de ré-initialisation presque complète du vecteur de solution (voir tableau 5.12). Ce dernier (nommé SCPFAST) a amélioré les résultats de résolution (2,42 % au lieu de 26,85%), montrant une meilleure diversification.

En conclusion, le réglage automatique effectué par RGENOUD sur un sous-ensemble restreint d'apprentissage se généralise bien pour résoudre presque la totalité des instances du même ensemble. Ce réglage fournit des solutions de bonne qualité en temps raisonnable (dans notre cas, moins de 60 secondes), à condition qu'il soit paramétré (optimisé en choisissant bien la gamme de variation des paramètres) et appliqué sur des instances du même type de problème.

5.4.3 Comparaison de BARYONYX avec d'autres logiciels d'optimisation

Pour évaluer la performance de BARYONYX, nous l'avons comparé avec d'autres logiciels d'optimisation, à savoir IBM ILOG CPLEX V12.8, LOCALSOLVER V8.0 et TOULBAR2 V1.0.1 (voir section 1.8.3, chapitre 1). Les premiers exécutent des fichiers de format *lp*, le deuxième de format `lsp` et le dernier de format *wcsp*. Malheureusement, nous ne disposons pas des fichiers de format *lsp* et *wcsp* pour toutes les instances de test. Dans le tableau 5.13, nous présentons les instances non disponibles par N/A (Not Available).

Le tableau 5.13 récapitule les résultats de résolution de sept familles d'instances en utilisant les quatre logiciels d'optimisation mentionnés ci-dessus. Tous les calculs sont effectués en mode parallèle. Chaque solveur utilise 30 processeurs (4GB de mémoire par

Instances (temps limite)	Logiciels d'optimisation			
	BARYONYX^{rng}	CPLEX	LOCALSOLVER	TOULBAR2
nqueens (60s)	**0,61 %** **[8/8]**	453,26 % **[8/8]**	36,34 % **[8/8]**	*9,87 %* [6/8]
MIPLIB (60s)	*771,09 %* [21/29]	***1,70%*** **[24/29]**	N/A	N/A
SPP (60s)	0,01 % **[55/55]**	**0,00 %** **[55/55]**	*8,75 %* [51/55]	N/A
Telebus (60s)	**0,44 %** **[28/28]**	2,95 % **[28/28]**	*35,82 %* [18/28]	N/A
CSPLib022 (60s)	**0,00 %** **[12/12]**	**0,00 %** **[12/12]**	*1,54 %* [10/12]	N/A
UAI Eval (60s)	*388,50 %* [101/113]	*636,92 %* [106/113]	*9,00 %* **[108/113]**	***0,19 %*** [105/113]
VCS (1800s)	**0,07 %** **[2/2]**	*3,05 %* [1/2]	- [0/2]	N/A
SCP (60s)	26,85 % **[30/30]**	**0,00 %** **[30/30]**	0,74 % **[30/30]**	N/A

TABLE 5.13 – Résultats de résolution parallèle (30 processeurs) de 7 familles d'instances en utilisant 4 logiciels d'optimisation (BARYONYX, LOCALSOLVER, CPLEX et TOULBAR2).

Pour chaque famille d'instances :
- 1ère ligne : valeur moyennée de la distance relative à la meilleure solution connue, calculée uniquement pour les instances résolues,
- 2ème ligne : nombre des solutions obtenues sur le nombre total des instances.
gras : les meilleurs résultats.
italique : une ou plusieurs instances ne sont pas résolues (leurs solutions ne sont pas comptabilisées).
N/A : instance (format *lsp* ou *wcsp*) non disponible.

processeur) et un temps limite de 60 secondes, sauf pour les instances volumineuses de VCS où nous avons fixé un temps CPU de 1800 secondes. Pour chaque famille d'instances, la première ligne représente les valeurs moyennées des distances relatives aux meilleures solutions connues et la seconde ligne indique le nombre des instances résolues sur le nombre total. Notons que les taux (valeurs moyennées) sont calculés uniquement pour les instances résolues.

A partir de ces résultats (tableau 5.13), nous observons que pour les instances nqueens, BARYONYX trouve de meilleures solutions que les autres logiciels (CPLEX, LOCALSOLVER et TOULBAR2), qui trouvent des difficultés pour résoudre les instances de grande taille (500 reines et 1000 reines). BARYONYX est donc le mieux placé pour résoudre le problème des N reines pondérées.

Concernant la famille d'instances MIPLIB qui regroupe un ensemble de problèmes (Set Partitioning, Set Packing, Set Covering et d'autres), nous remarquons que les deux solveurs utilisés (BARYONYX et CPLEX) n'ont pas pu résoudre la totalité des instances. Rappelons que le choix des instances a été effectué de manière à respecter la forme des problèmes traités par BARYONYX. Dans cette sélection, nous avons des problèmes infaisables (optimum inconnu ou pas de solution). Malheureusement, le réglage automatique des paramètres de BARYONYX a été effectué sur un ensemble d'apprentissage contenant ces instances infaisables. Ce qui explique les résultats moins bons de BARYONYX.

Pour les instances SPP, Telebus et CSPLib022, nous observons que BARYONYX atteint de bonnes performances, qui sont meilleures que CPLEX pour Telebus, proches pour SPP et égales pour CSPLib022. En revanche, LOCALSOLVER ne résout pas la totalité des instances, il a besoin de plus de temps pour résoudre et améliorer les solutions trouvées. Avec 600 secondes, LOCALSOLVER trouve des solutions optimales (0,00 %) pour toutes les instances CSPLib022, des solutions de bonne qualité (1,01 %) pour toutes les instances SPP, mais il n'arrive toujours pas à résoudre la totalité des instances Telebus (7 problèmes non résolus sur 28). Pour analyser davantage les résultats, nous avons comparé notre logiciel BARYONYX avec d'autres méthodes présentées dans la littérature et utilisées pour résoudre ces instances, à savoir la méthode de Branch and Cut (voir chapitre 1) [Borndörfer, 1998] et l'algorithme 4-flip neighborhood qui est une méthode de recherche locale cherchant à réduire l'espace de recherche [Umetani, 2017]. Le tableau 5.14 présente, pour chaque sous famille d'instances, les distances relatives aux meilleures solutions connues avec le nombre des instances résolues par les six méthodes (BARYONYX, CPLEX, LOCALSOLVER, Branch and Cut [Borndörfer, 1998] et 4-flip neighborhood [Umetani, 2017]). D'après le tableau 5.14, BARYONYX est généralement le meilleur pour résoudre les instances Telebus, il fournit des solutions de bonne qualité surtout pour les instances t qui représentent une difficulté pour les autres solveurs. Pour les instances SPP, BARYONYX atteint des performances proches de CPLEX et [Borndörfer, 1998] et meilleures que LOCALSOLVER et [Umetani, 2017]. Concernant les instances CSPLib022, nous avons également comparé avec des résultats présentés dans la littérature. Il s'agit de [Curtis et al., 2000] où 8/12 instances CSPLib022 sont résolues en utilisant le logiciel GENET de résolution des problèmes CSP binaires. Le tableau 5.15 montre que ce dernier présente des solutions de qualité moins bonne en comparant à LOCALSOLVER et à BARYONYX et CPLEX qui trouvent les optima.

Pour les instances UAI Eval, nous remarquons que TOULBAR2 trouve des solutions de bonne qualité (0,19 % face à 9,00 % pour LOCALSOLVER, 388,50 % pour BARYONYX et 636,92 % pour CPLEX). Toutefois, le nombre des infaisabilité est légèrement plus grand (8) par rapport à CPLEX (7) et LOCALSOLVER (5). Notons que les performances de TOULBAR2 dans la résolution des problèmes de type UAI Eval ont été montrées dans le travail de [Hurley et al., 2016], où TOULBAR2 a été comparé à plusieurs logiciels d'optimisation, y compris CPLEX. Pour BARYONYX, nous pouvons dire qu'il est moins bien pour cette catégorie d'instances (12 infaisabilités). Cependant, il fournit des solutions de bonne qualité en comparant à CPLEX.

Pour les instances VCS, le tableau 5.13 montre que BARYONYX résout les deux instances (vcs1200 et la plus large vcs1600) au bout de 1800 secondes, LOCALSOLVER ne parvient pas à les résoudre et CPLEX ne trouve pas de solution pour la plus large instance (vcs1600). Cela est confirmé par [Zaghrouti et al., 2014] qui mentionne que la résolution de cette instance représente une difficulté pour CPLEX même au bout de 10 heures d'exécution avec un seul CPU. Comme mentionné avant, Zaghrouti indique que les solutions optimales de ces deux instances sont connues, mais sans préciser leurs valeurs.

Instances	BARYONYX (60s)	CPLEX (60s)	LocalSolver (60s)	Borndorfer[1] (7200s)	Umetani[2] (600s)
Telebus					
v0415-0421(7)	0,30 % (7)	0,00 % (7)	0,04 % (7)	0,00 % (7)	0,00 % (7)
v1616-1622(7)	1,43 % (7)	0,00 % (7)	6,21 % (7)	0,01 % (7)	0,09 % (7)
t0415-0421(7)	0,03 % (7)	0,61 % (7)	– (0)	1,88 % (7)	0,95 % (6)
t1716-1722(7)	0,00 % (7)	11,19 % (7)	150,25 % (4)	9,70 % (7)	10,71 % (7)
Moy. (28)	0,44 % (28)	2,95 % (28)	35,82 % (18)	2,90 % (28)	3,01 % (27)
SPP					
aa (6)	0,08 % (6)	0,00 % (6)	23,02 % (3)	0,00 % (6)	1,64 % (6)
us (4)	0,02 % (4)	0,00 % (4)	0,00 % (3)	0,00 % (4)	0,04 % (4)
nw (43)	0,00 % (43)	0,00 % (43)	8,53 % (43)	0,00 % (43)	N/A
kl (2)	0,00 % (2)	0,00 % (2)	0,00 % (2)	0,00 % (2)	N/A
Moy. (55)	0,01 % (55)	0,00 % (55)	8,54 % (51)	0,00 % (55)	

Table 5.14 – Résultats de calcul des instances Telebus et SPP.
Les taux représentent la distance relative à la meilleure solution connue et les valeurs entre parenthèses indiquent le nombre des instances résolues.
[1] : [Borndörfer, 1998], [2] : [Umetani, 2017]

Instances	BARYONYX	CPLEX	LocalSolver	[Curtis et al., 2000]
CSPLib022 (8/12)	0,00 %	0,00 %	2,20 %	26,74 %

Table 5.15 – Résultats de calcul (distance relative à la meilleure solution connue) des instances CSPLib022 (temps CPU $\leq$ 60s).

Instances	BARYONYX		CPLEX	LocalSolver	Beasley[1]	Umetani[2]	
	SCPFast	BARYONYX^{rgn}				1/	2/
SCP							
4.1-4.10	1,64 %	22,61 %	0,00 %	0,00 %	0,06 %	1,69 %	0,04 %
5.1-5.10	2,60 %	28,62 %	0,00 %	0,00 %	0,18 %	3,30 %	0,08 %
6.1-6.5	2,89 %	20,95 %	0,00 %	0,00 %	0,56 %	8,60 %	0,55 %
A.1-1.5	3,14 %	37,69 %	0,00 %	4,44 %	0,82 %	8,12 %	0,75 %
Moy.	2,42 %	26,85 %	0,00 %	0,74 %	0,31 %	5,43 %	0,35 %

Table 5.16 – Résultats de calcul (distance relative à la meilleure solution connue) des instances SCP (temps CPU < 60s).
[1] : [Beasley, 1990b], [2] : [Umetani and Yagiura, 2007]
1/ Algorithme glouton, 2/ heuristique basée sur la relaxation Lagrangienne

Concernant les résultats de calcul des instances SCP (tableau 5.13), nous observons que BARYONYX résout la totalité des instances avec une qualité moins bonne (26,85 % de la meilleure solution connue) que LOCALSOLVER (0,76 %) et CPLEX (0,00 %) qui résout à l'optimalité. Pour une analyse plus globale, nous avons comparé ces résultats avec ceux présentés dans [Beasley, 1990b] utilisant une heuristique basée sur la relaxation Lagrangienne, et avec les résultats de [Umetani and Yagiura, 2007] qui compare plusieurs algorithmes. Nous avons donc choisi de comparer avec deux algorithmes (1/ un algorithme de construction glouton et 2/ une heuristique basée sur la relaxation Lagrangienne) donnant les meilleurs résultats dans son article. Notons que ces algorithmes sont spécialisés dans la résolution du problème de Set Covering. Le tableau 5.16 présente les résultats de comparaison de ces méthodes avec les résultats des deux versions de BARYONYX : SCPFAST (jeu de paramètres trouvés manuellement) et BARYONYX^{rng} (jeu de paramètre optimisé par RGENOUD). Ce tableau montre que les résultats de BARYONYX (avec les paramètres de SCPFAST) sont meilleurs que ceux fournis par les algorithmes gloutons mais moins bien en comparant aux autres solveurs.

5.5 Conclusion du chapitre

Dans ce chapitre, nous avons présenté les extensions de l'algorithme de Wedelin implémenté dans notre solveur BARYONYX, à savoir l'ordre de traitement des contraintes et des variables, les politiques de (ré)initialisation de solution, les méthodes de normalisation des coûts et le traitement des cas particuliers. Basés sur des expérimentations, nous avons fixé des stratégies (considérées comme des paramètres catégoriques) pour une implémentation de BARYONYX par défaut. Concernant les paramètres numériques, nous avons procédé à une analyse de sensibilité pour choisir les paramètres les plus importants, que nous avons réglé automatiquement en utilisant la méthode de GENOUD. Ce réglage automatique a été effectué par familles de problèmes, optimisé sur un sous-ensemble d'apprentissage et généralisé sur la totalité des problèmes. Pour évaluer son utilité, nous l'avons comparé à des réglages statiques. D'après les résultats, ce réglage automatique fournit des solutions de bonne qualité en temps raisonnable pour le même type de problème, à condition de bien choisir la gamme de variation des paramètres.

Pour évaluer la performance de BARYONYX, nous l'avons comparé à d'autres logiciels d'optimisation (CPLEX, LOCALSOLVER et TOULBAR2) et à des méthodes présentées dans la littérature. Les résultats montrent qu'aucune méthode ne domine systématiquement les résultats. La meilleure méthode dépend de chaque famille de problèmes. De façon générale, BARYONYX atteint en moyenne des performances meilleures que les autres solveurs pour la résolution du problème des n reines pondérées et pour le problème de Set Partitioning. Malheureusement, il trouve des difficultés pour réaliser des solutions de bonne qualité pour les autres types de problèmes, à savoir le problème de Set Covering. Nous proposons donc de refaire le réglage automatique des paramètres de BARYONYX en adaptant leurs gammes de variation pour le problème Set Covering. Par exemple, nous proposons de réduire le nombre des itérations par exécution, vu que ce problème possède une multitude de solutions admissibles et donc il a besoin de peu d'itérations par exécution, ce qui permet d'explorer plus de solutions dans l'espace de recherche.

Troisième partie

Application : Conception des systèmes de verger-maraîcher

Chapitre 6

Concepts généraux sur la conception des systèmes de vergers-maraîchers

Sommaire

6.1 Introduction

Aujourd'hui, l'agriculture est confrontée à plusieurs problèmes agronomiques, environnementaux et socio-économiques, comme le changement climatique, l'utilisation massive des ressources (eau, énergie) et la nécessité de nourrir une population grandissante. Malheureusement, beaucoup de stratégies visant à améliorer la productivité favorisent l'émergence des problèmes environnementaux. Pour illustrer, l'utilisation massive d'intrants chimiques dans les systèmes de culture intensifs contribue au dérèglement des écosystèmes. L'agroécologie se présente aujourd'hui comme voie de transformation possible pour l'agriculture, en s'appuyant sur des fonctions écologiques pour remplacer les intrants de synthèse. Dans ce cadre, l'agroforesterie et les systèmes de vergers-maraîchers associant arbres fruitiers et cultures maraîchères sur la même parcelle semble une pratique intéressante pour répondre à ces enjeux. En effet, Cette association permet de produire d'une manière intensive et écologique avec une meilleure exploitation des ressources naturelles disponibles. De plus, elle offre une biodiversité permettant les régulations naturelles des maladies et des ravageurs.

Le verger-maraîcher était une pratique courante traditionnelle en zone méditerranéenne en Europe jusqu'au milieu du XXème siècle [HERZOG, 1998, Coulon et al., 2000, Eichhorn et al., 2006a]. Elle a été délaissée progressivement en raison de la spécialisation, de l'intensification et de la mécanisation (coûts de main d'oeuvre) des parcelles agricoles pour répondre à de forts besoins productifs [Dupraz and Liagre, 2006]. Aujourd'hui, la rentabilité économique n'est plus le seul critère d'évaluation des systèmes agricoles, les impacts environnementaux liés à l'utilisation massive d'engrais chimiques et de pesticides de synthèse ont conduit à la recherche de nouvelles voies menant à une agriculture plus durable, tant dans les régions tempérées que tropicales [Malézieux, 2012]. Dans cette optique, le concept de verger-maraîcher est remis à l'étude pour optimiser les bienfaits des interactions naturelles entre arbres et cultures (voir Figure 6.1).

Pour dynamiser cette pratique d'agroforesterie en Europe et notamment en France, des projets de recherche expérimentale sur les systèmes innovants de verger-maraîcher ont été mis en place. Nous citons le projet CASDAR SMART[1] 2014-2016 (prolongé jusqu'à 2017) qui vise à développer des connaissances autour des associations agroforestières entre arbres fruitiers et cultures maraîchères pour répondre à une forte demande technique de jeunes agriculteurs, proposer des modèles agricoles, optimiser les interactions bénéfiques entre espèces cultivées et améliorer la performance globale de l'agriculture. Nous citons également le projet de la Durette[2], une ferme pilote à l'initiative du GRAB (Groupe de Recherche en Agriculture Biologique) en ceinture verte d'Avignon dans le sud-est de la France, où des systèmes innovants de verger-maraîcher ont été implantés.

Un projet de système innovant de verger-maraîcher débute généralement par une phase de conception préliminaire à son implémentation. Cette phase est à la fois complexe et déterminante : complexe car il faut prendre en compte de nombreuses interactions biologiques, pas toujours très connues, mais avec des objectifs et des contraintes économiques ou sociales (charge de travail par exemple), déterminante car certains choix, notamment d'implantation des arbres, verront leurs conséquences se développer tout au long de la durée de vie de ce système. L'objectif de cette thèse est donc de proposer des modèles de conception des systèmes de verger-maraîcher qui tirent bénéfice des associations d'arbres et de cultures maraîchères en optimisant les interactions qui les composent et en tenant

[1] http://www.agroforesterie.fr/SMART/smart-agroforesterie-maraichage-le-projet.php
[2] http://www.grab.fr/durette

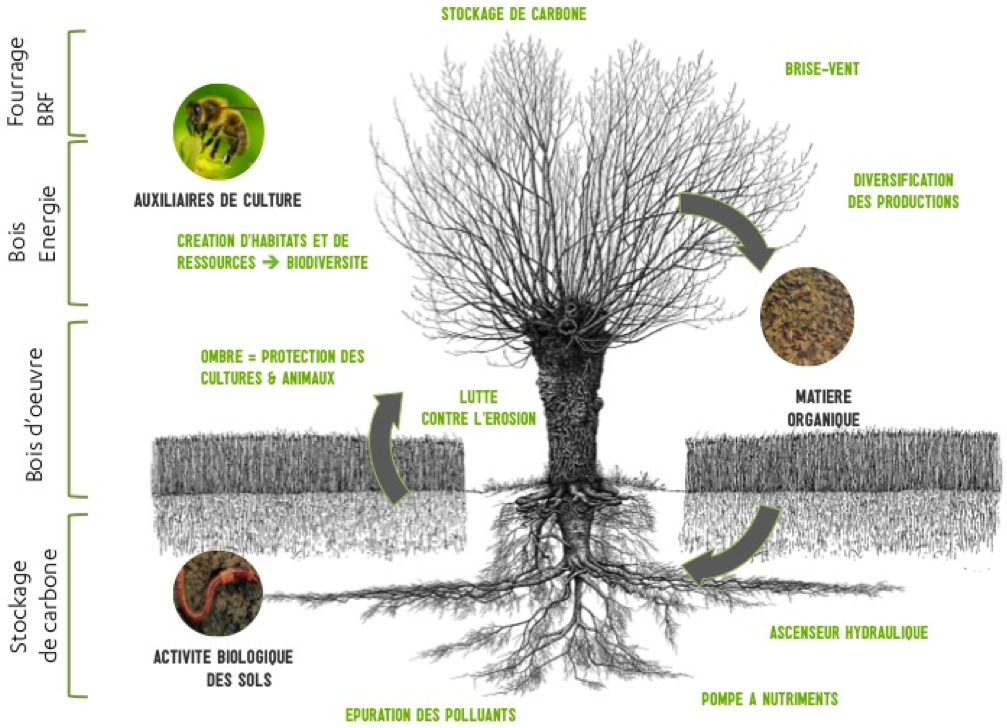


FIGURE 6.1 – Apports de l'arbre en milieu agricole.
Source : http://www.agroforesterie.fr/definition-agroforesterie.php

FIGURE 6.2 – Ferme pilote de la Durette (Photo prise lors de notre visite en novembre 2015).

compte de leurs évolutions dans l'espace et dans le temps.

Dans ce chapitre, nous présentons les concepts généraux nécessaires pour la conception d'un système de verger-maraîcher. Dans un premier temps, nous présentons les propriétés des systèmes agroforestiers, à savoir les interactions aériennes et souterraines entre arbres et cultures, la biodiversité, l'organisation du travail, la rotation des cultures et le rendement de production. Dans un second temps, nous discutons les méthodes de recherche opérationnelle utilisées pour la conception des systèmes agricoles.

6.2 Propriétés des systèmes agroforestiers

Comme cité auparavant, l'agroforesterie est une pratique agricole associant arbres, cultures et/ou animaux sur une même parcelle agricole[3]. Les cultures peuvent être pérennes ou annuelles. Comme nous nous intéressons particulièrement aux vergers-maraîchers, nous réservons le terme d'agroforesterie à l'agrosylviculture qui est une pratique associant sur la même parcelle des arbres (pérennes) et des cultures (annuelles dans notre cas).

6.2.1 Interactions écologiques entre arbres et cultures

L'objectif de l'agroforesterie est d'exploiter des processus de facilitation (au sens écologique) entre les deux types de productions (arbres et cultures). Cela se réalise par une meilleure utilisation des ressources disponibles (e.g. eau et nutriments). En effet, deux espèces voisines peuvent s'influencer mutuellement soit positivement (facilitation, mutualisme), soit négativement (compétition, inhibition) ou les deux simultanément [Bruno et al., 2003] (Figure 6.3). Ces interactions écologiques peuvent avoir des impacts majeurs sur le comportement global du système, mais il est souvent difficile de déterminer les influences relatives à chaque interaction [Hunter and Aarssen, 1988]. Donc, pour tirer le maximum d'avantages (durabilité écologique) de cette association agroforestière, il est nécessaire d'optimiser ces interactions de manière à ce que les interactions positives l'emportent sur les interactions négatives.

Une interaction écologique fait référence à l'impact majeur d'une espèce sur une autre ou sur le même type d'espèce [Batish et al., 2007]. En général, il existe trois types d'interactions : neutre (0), positive (+) et négative (-). Le tableau 6.1 résume les différents types d'interactions selon le signe caractérisant l'effet d'une espèce sur le taux de croissance d'une autre. Si l'interaction est positive, il s'agit d'une complémentarité entre espèces. Par contre, si l'interaction est négative, elle devient compétitive. Ces interactions peuvent évoluer au cours du temps, en passant d'un système complémentaire à un système compétitif ou l'inverse. Généralement, en agroforesterie, les arbres ont une influence majeure sur les cultures. Ils déterminent l'ampleur des interactions grâce à l'évolution continue de leurs systèmes racinaires et à la variation de leur potentiel d'interception du rayonnement solaire. Dans cette optique, les interactions entre arbres et cultures dans un système agroforestier peuvent être classées à deux échelles : des interactions aériennes et des interactions souterraines (Figure 6.4).

[3]Définition de l'Association Française de l'Agroforesterie (AFAF)

Interaction	Effet[1]		Description	Exemple en agroforesterie
	Espèce 1	Espèce 2		
Inhibition	-	0	Une des espèces a un effet inhibiteur sur l'autre, sans être affectée par l'association.	Allélopathie (e.g. la tomate émet des substances volatiles qui inhibe la croissance de la vigne [Bruno et al., 2001]).
Facilitation	+	0	Une des espèces associées tire bénéfice de l'association.	Certaines plantes peuvent bénéficier de la proximité d'espèces attirant les pollinisateurs (e.g. Épervière orangée attire les pollinisateurs au profit de l'espèce jaune) [Tjomson, 1978].
Compétition	-	-	Les deux espèces ont une influence négative réciproque sur l'utilisation commune d'une ressource.	Compétition pour l'eau en période estivale dans les zones tempérées [Jose et al., 2004].
Mutualisme	+	+	Les deux espèces tirent bénéfice de l'association.	La plupart des exemples concernent des associations animal-animal, animal-champignon, plante-champignon ou plante-animal [Hunter and Aarssen, 1988]. Nous citons l'exemple des espèces Mycorhizes et Rhizobium dans les légumineuses [Jose et al., 2004].
Neutralisme	0	0	Aucune espèce n'affecte l'autre.	Les interactions neutres sont très rares et ne se produisent que si les niches sont très éloignées [Batish et al., 2007].
Prédation et parasitisme	+	-	Une espèce tire bénéfice de l'association au détriment de l'autre.	Les cultures associées peuvent attirer plus de prédateurs et de parasites que les monocultures, réduisant ainsi la densité des organismes nuisibles par la prédation et le parasitisme.

TABLE 6.1 – Interactions écologiques entre deux espèces (de cultures) associées dans le même endroit et en même temps, présentées par le signe caractérisant l'effet d'une espèce sur le taux de croissance d'une autre.
[1] : (-) effet négatif, (0) effet neutre et (+) effet positif.
Source : [Jose et al., 2004] modifiée.

FIGURE 6.3 – Les interactions entre plantes peuvent être simultanément positives et négatives [Hunter and Aarssen, 1988].

Interactions aériennes

Microclimat (ombre, lumière, température et effet brise-vent) et biodiversité.

La présence des arbres dans les systèmes agroforesteries contribue à la modification des conditions microclimatiques. Par exemple, la diminution du rayonnement solaire sous les arbres protège les cultures sciaphiles (qui aiment l'ombre) et aide à réduire le stress thermique [Dupraz and Liagre, 2011], en particulier pendant les saisons chaudes. Une étude menée au nord de la Floride a montré que l'ombre a un effet réducteur (limite le rendement) sur certaines cultures de légumes au printemps, mais offre de meilleurs rendements en été [Valli et al., 1965]. Cet auteur a montré que le haricot d'Espagne cultivé dans des parcelles ombragées (au printemps) présente un rendement plus faible que dans les parcelles ensoleillées, mais observe un résultat inverse en été. Au contraire, une culture de concombre présente une réponse homogène et positive à l'ombrage (amélioration de la qualité et du rendement) dans les deux saisons de printemps et d'été.

Toutefois, un niveau d'ombre élevé des arbres peut réduire la photosynthèse des cultures associées. Autrement dit, l'ombre modifie la composition et surtout l'intensité du rayonnement photosynthétiquement actif (RPA) reçu par les cultures et par conséquent modifie la production de biomasse [Ong et al., 1991]. Une étude sur la culture de poivron a montré que les niveaux d'ombre modérés (30% et 47%) sont les plus favorables à la croissance des poivrons, un niveau élevé peut réduire le nombre et le poids des feuilles de la plante [Diaz-Perez, 2013]. Ce niveau varie au cours de l'année et de chaque journée. Les jours ensoleillés de l'été présentent une variation journalière marquée de l'interception de la lumière par les arbres. Cette variation est considérablement réduite dans les autres saisons d'une part car les arbres perdent leurs feuilles [Palmer, 1997].

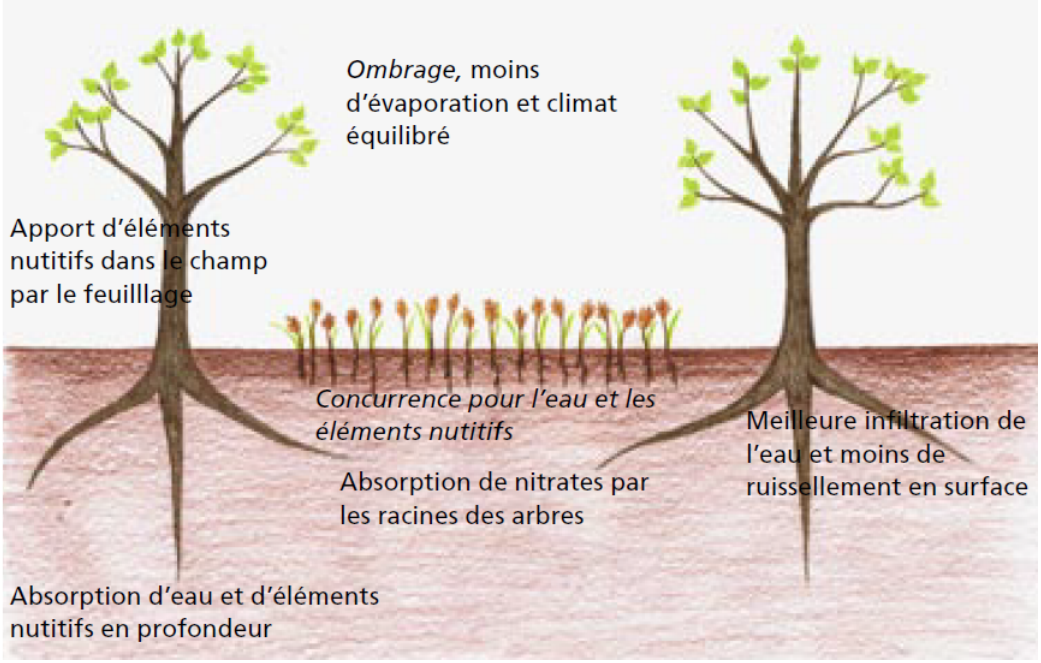


FIGURE 6.4 – Influences positives et négatives des arbres sur les cultures. Les impacts négatifs sont écrits en italique. [Kaeser et al., 2010]

Les arbres servent également de brise-vent, ils permettent de ralentir le mouvement de l'air et de réduire le stress d'évaporation [Davis and Norman, 1988]. Cependant, à cause d'une humidité et d'une hygrométrie accrues, ils peuvent favoriser le développement des maladies et des ravageurs (notamment mollusques) [Griffiths et al., 1998]. Inversement, ils peuvent abriter une faune (oiseaux, insectes) capable de freiner la propagation de certaines épidémies et l'invasion de parasites [Dupraz and Liagre, 2011]. Dans leur revue, [Simon et al., 2010] présentent un ensemble d'études sur le rôle de la diversité des cultures dans l'amélioration de la biodiversité et la lutte contre les ravageurs. Ces associations peuvent être principalement positives, nulles mais aussi négatives dans certains cas. Cela révèle la difficulté d'identifier les associations de cultures bénéfiques de celle nuisibles.

Interactions souterraines

Eau et éléments nutritifs : compétition ou facilitation ?

Dans les systèmes agroforestiers, les arbres jouent un rôle primordial dans la structure du sol. Ils ont globalement des systèmes racinaires profonds qui impactent les conditions subies par les racines des cultures associées, notamment l'exploitation des ressources en eau et en nutriments. Généralement, la compétition pour ces ressources se produit lorsque les systèmes racinaires des espèces associées occupent les mêmes couches du sol [van Noordwijk et al., 1996]. D'après les observations de [Lehmann et al., 1998, Jose et al., 2000a] dans les zones tempérées et les régions tropicales, la concentration de la densité des racines d'arbres se trouve dans les couches superficielles du sol, qui représentent la zone de compétition avec les cultures.

Dans leur revue, [Jose et al., 2004] expliquent que malgré la difficulté de déterminer quel genre de compétition souterraine (l'eau ou nutriments) limite la production, la compétition la plus fréquente dans la zone tempérée est celle pour l'eau en période estivale.

Cette compétition prépondérante peut engendrer une concurrence pour les éléments nutritifs. Suite à leur expérimentations sur l'association noyer noir - maïs, [Jose et al., 2000b] trouvent que la disponibilité de l'eau était un facteur de compétition pour les éléments nutritifs. Car la concurrence pour l'eau due aux racines des arbres était responsable de la réduction de la biomasse racinaire du maïs, entraînant ainsi une diminution de l'efficacité d'utilisation des nutriments.

L'effet positif des arbres dans les interactions souterraines est aussi important. Grâce à leur durée de vie plus longue, les systèmes racinaires des arbres se développent latéralement et verticalement d'une manière hétérogène en cherchant les zones riches en eau et en nutriments [van Noordwijk et al., 1996]. Cela est confirmé par [Atkinson, 1983] qui a observé une grande proportion de racines longues des fruitiers dans les zones non irriguées par rapport aux zones irriguées. En effet, une association arbres-cultures peut favoriser l'utilisation des ressources. Par exemple, grâce à la redistribution hydraulique dans les racines des arbres, l'eau dans les sols profonds et humides est transportée vers les sols secs de surface, fournissant ainsi plus d'humidité à la végétation pendant les périodes sèches [Burgess et al., 1998]. De plus, l'eau résiduelle dans le sol après la récolte peut être récupérée par les arbres [Rao et al., 1993], réduisant ainsi leurs besoins en irrigation. Dans leur livre, [Dupraz and Liagre, 2011] expliquent que les cultures d'hiver ont un rôle très important dans l'exploration des racines. Elles obligent les arbres à passer leurs racines sous la zone racinée par les cultures (voir Figure 6.5). Cela permet une exploitation plus complète est optimale des ressources. De manière similaire à l'eau, les arbres peuvent améliorer l'efficacité de la fertilisation, en récupérant le surplus dans le sol avant son érosion [Nissen et al., 1999] et en acquérant les ressources non utilisables par les cultures [Cannell et al., 1996]. Dans les premières années de leur plantation, les arbres fruitiers ont besoin de plus de nutriments pour s'établir, contrairement aux arbres de bois d'oeuvre qui sont plus résilients et qui peuvent décaler leur absorption de nutriments [Eichhorn et al., 2006b]. Il s'agit donc d'une période où les arbres sont plus en compétition avec les cultures associées ayant les mêmes besoins en ressources.

6.2.2 Productivité des systèmes agroforestiers

Les principaux avantages productifs des systèmes agroforestiers sont liés à une meilleure utilisation de l'espace et des ressources naturelles. La productivité d'un tel système dépend du type des arbres (voir Figure 6.6) et de leurs rythmes de croissances qui sont différents de ceux des cultures. Généralement, l'espace associé aux cultures dans la parcelle est diminué avec la croissance des arbres, ce qui engendre une diminution de leur rendement. Les systèmes agroforestiers sont donc conçus pour optimiser l'utilisation des ressources dans le temps (rythmes de croissance des arbres et cultures) et dans l'espace (e.g. évolution des racines et de l'ombre) en maximisant les interactions positives et en minimisant celles négatives.

Il est possible d'obtenir des rendements plus élevés dans les systèmes agroforestiers (association) que dans les assolements agriculture-forêt (séparation). La comparaison se fait en calculant la surface équivalente assolée (SEA), ou LER (Land Equivalent Ratio), illustrée en Figure 6.7. Un LER supérieur à 1 indique que l'association agroforestière est plus productive que l'assolement [Dupraz and Liagre, 2011]. Suite à des expérimentations sur l'association de poiriers et des radis en Angleterre, [Newman, 1986] a mesuré des LER variant de 1,50 à 2,01. Cela signifie qu'avec un LER de 1,50, une parcelle de 100 hectares en agroforesterie produira autant de produits qu'une parcelle de 150 hectares en assolement. En d'autres termes, les parcelles agroforestières peuvent être beaucoup plus productives

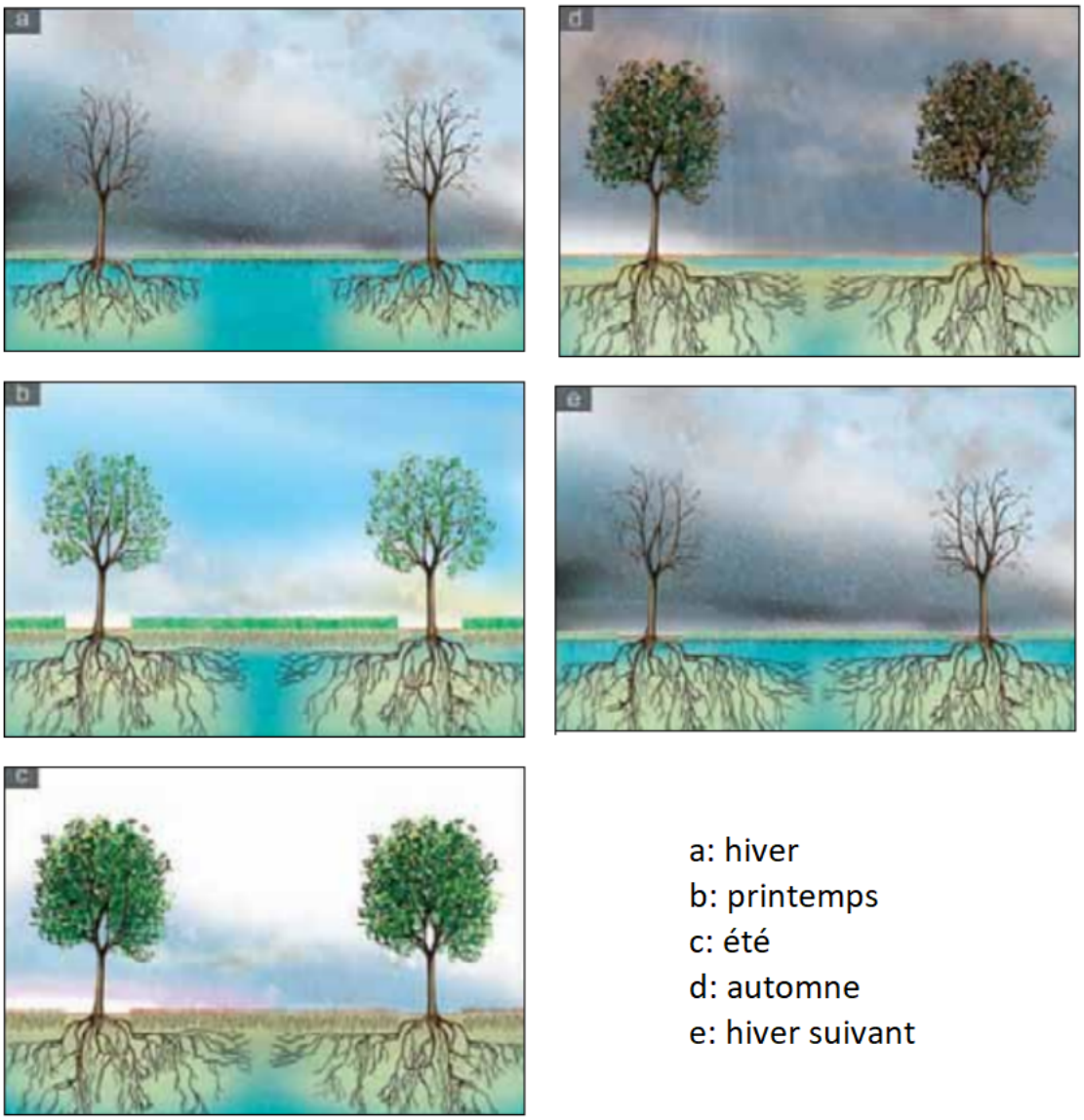


FIGURE 6.5 – Dynamique de l'eau du sol et de la colonisation racinaire dans une parcelle agroforestière associant une culture d'hiver et des arbres [Dupraz and Liagre, 2011].

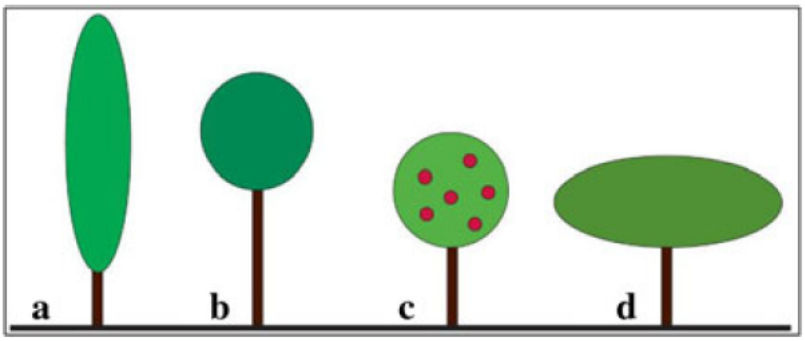


FIGURE 6.6 – Illustration schématique des différentes tailles et formes des arbres
a : arbre brise-vent
b : arbre taillé pour produire un bois de haute qualité
c : arbre avec une tige courte pour produire des fruits et du bois de chauffage
d : arbre avec une cime élargie pour fournir de l'ombre aux animaux (mais a un impact négatif sur le rendement des cultures)
Source : [Nerlich et al., 2013]

que les assolements séparés.

Toutefois, le rendement d'un système agroforestier peut être réduit avec le temps en fonction de la croissance des arbres et de leur densité. Dans cette optique, [Dupraz and Liagre, 2011] distingue trois types de stratégies agroforestières :

- Agroforesterie stable : des parcelles avec une densité d'arbres faible (entre 20 et 50 arbres par hectare). Les cultures peuvent coexister en permanence avec les arbres et sur toute la surface.
- Agroforesterie évolutive : des parcelles avec une plus forte densité (entre 50 et 200 arbres par hectare). La surface cultivée diminue au cours du temps avec la croissance des arbres.
- Agroforesterie éphémère : des parcelles de très forte densité (au delà de 200 arbres par hectare), où les cultures ne sont possibles que pendant les premières années.

Pour garantir une bonne productivité des cultures, Il faut bien choisir une des stratégies de densité mentionnées ci-dessous et décider de positionner les arbres d'une manière dispersée ou en alignement. Il faut également choisir de planter les cultures sous les arbres ou de laisser des distances non cultivées autour de leurs troncs. Ces choix se font en fonction de la nature de production (e.g. maraîchage, cultures annuelles, production du bois, protection des animaux,...), des espèces choisies et de la mécanisation adoptée.

6.2.3 Organisation du travail

Dans le cas d'une production de bois d'oeuvre, les arbres ne doivent être ébranchés (interventions de taille) que durant les quinze premières années, les travaux d'entretien diminuent considérablement par la suite. En revanche, la charge de travail des arbres fruitiers ne diminue pas au fil du temps, vu que les arbres devant être en permanence entretenus et les fruits récoltés [Kaeser et al., 2010]. Les temps de travaux et leur répartition sur l'année varient d'une espèce fruitière à l'autre, voire des variétés choisies d'une même espèce. Par exemple pour les pommes, la variété Gala se récolte en août, alors que les variétés Golden

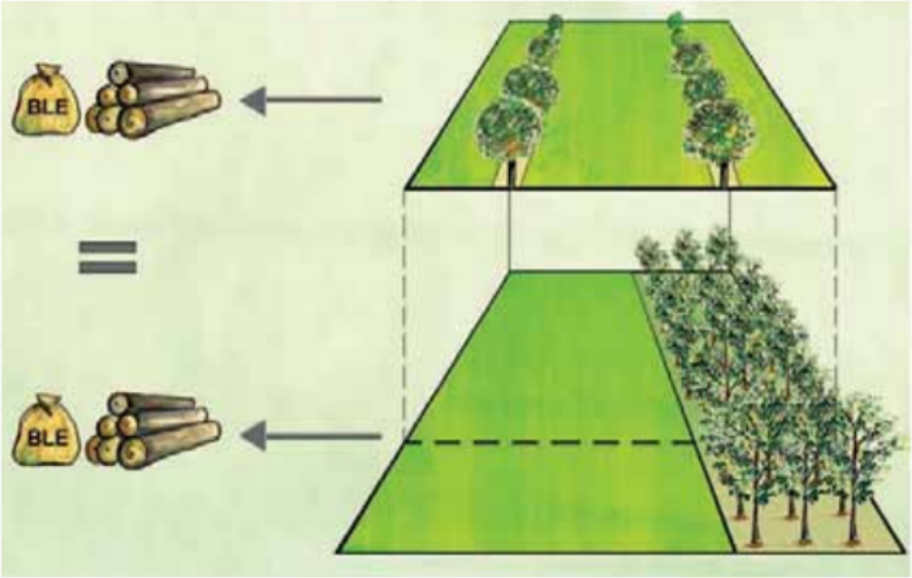


FIGURE 6.7 – La surface équivalente assolée est la surface nécessaire pour obtenir un assolement agriculture-forêt la même production qu'un hectare de système agroforestier [Dupraz and Liagre, 2011]

.

ou Granny se récoltent en octobre [Chambre d'agriculture, 2016].

De la même manière, les durées d'interventions des cultures annuelles ou saisonnières sont très variables d'une exploitation et d'une culture à l'autre. Ces durées varient en fonction de l'organisation de la parcelle (regroupement ou dispersion des cultures) et des ressources disponibles (la mécanisation adoptée, la main d'oeuvre et le rythme de chacun). Pour aider les agriculteurs, les chambres régionales d'agriculture[4] publient des références des temps de travaux, données à titre indicatif, en plus des fiches proposées par les centre agricoles. Nous citons par exemple la fiche [Argouarc'H, 2005] pour les cultures maraîchères de plein champ et sous abri (ces références sont toutefois acquises sur des cultures, pas dans des systèmes mixtes).

Dans les systèmes agroforestiers, associer des arbres et des cultures avec des rythmes de croissance différents impacte l'organisation des travaux, en augmentant les temps de travaux et en diminuant les possibilités de mécanisation. Pour concevoir un système agroforestier, il faut réorganiser la répartition des travaux (préparation de la parcelle, plantation, entretien des cultures, récolte, etc) sur l'année en fonction des cultures choisies et de leur arrangement spatial.

6.2.4 Rotation des cultures

Après le positionnement des arbres (discuté dans la sous-section précédente) lors de la conception d'un système agroforestier, il convient de gérer la disposition des cultures diversifiées, qui ont des besoins différents en éléments nutritifs. Ainsi, laisser une culture au même endroit plusieurs années peut favoriser les adventices [Doucet et al., 1999], appauvrir le sol et par conséquent affaiblir les rendements. Par exemple, une production fréquente des pommes de terre peut accroître l'érosion des sols [Edwards et al., 1998].

[4]https ://occitanie.chambre-agriculture.fr/

Une solution consiste à pratiquer une succession ou rotation de cultures avec des périodes de jachère, où le sol repose après la production. Selon [Leteinturier et al., 2006], les deux termes font référence à l'ordre de succession des cultures sur la même parcelle. La *rotation* se caractérise par un cycle, défini par une liste de cultures ordonnée qui se reproduit identiquement au cours du temps. Tandis que la *séquence* de culture se limite à l'ordre d'apparition des cultures dans la séquence durant une période déterminée. Ainsi, cette dernière peut être le développement partiel ou total d'un cycle de rotation. Cet intérêt pour le concept de rotation des cultures peut s'expliquer par différents avantages : maintenir la fertilité du sol, assurer une production de qualité et diminuer l'utilisation des résidus de pesticides, en donnant la priorité à l'utilisation des mécanismes de régulation naturels [Boller et al., 1997, Leteinturier et al., 2006].

Pour atteindre ces objectifs, il faut remplir certaines exigences. Dans ce sens, [Castellazzi et al., 2008] définissent quatre grands types de contraintes ou de règles déterminant la rotation des cultures. La première règle consiste à définir le délai de retour d'une même culture sur une même parcelle. Cela permet d'empêcher l'accumulation des organismes nuisibles comme les nématodes (vers parasites) [Jones and Perry, 1978]. Par exemple, les pois secs doit être cultivés un an sur sept, les pommes de terre un an sur quatre et le maïs un an sur deux [Boller et al., 1997]. La deuxième règle concerne les avantages ou risques de succession d'une culture après l'autre. Par exemple, l'apport en azote, la disponibilité de matière organique ou de l'eau dans le sol, et la présence des parasites, des maladies ou des mauvaises herbes [Berzsenyi et al., 2000]. La troisième règle concerne les cycles intra-annuels liés aux variations climatiques. Généralement, une culture doit être récoltée avant de semer la culture suivante. Or, les variations climatiques peuvent retarder la récolte d'une culture, ce qui représente une contrainte à la plantation d'une autre. [Castellazzi et al., 2008] cite un exemple sur la contrainte de plantation, au Royaume-Uni, d'une culture céréalière semée à l'automne après une culture de betterave sucrière semée au printemps, car cette dernière est récoltée entre octobre et janvier. La quatrième et dernière règle consiste à définir les proportions globales des cultures sur une parcelle. Cela permet d'organiser la répartition du travail en tenant compte de la capacité disponible en termes de machines et de main d'oeuvre.

Basé sur ces règles, [Castellazzi et al., 2008] présentent quatre classes de rotations, illustrées dans la Figure 6.8. La rotation (a) est définie par une séquence de cultures caractérisée par une taille fixe et un ordre prédéfini répétitif. La rotation (b) est également cyclique de taille fixe mais représente une flexibilité sur le choix de cultures. La rotation (c) est aussi flexible, elle se distingue des autres classes par la variation de la taille des séquences répétées. La rotation (d) est moins structurée avec une grande flexibilité, i.e., une culture peut être suivie par tout autre type sauf par elle-même. En général, ces trois classes de rotation flexible représentent une combinaison de plusieurs rotations.

La durée des cycles de séquences dépend de la nature des cultures. Les cultures maraîchères ont des temps de rotation courts de quelques mois. Par exemple, la durée d'occupation de l'oignon est limitée à deux saisons maximum : printemps - été. Le semis de l'oignon se fait généralement au printemps et la récolte à n'importe quel stade de sa croissance. Le melon se sème au printemps mais se récolte uniquement en été. La laitue est une culture qui se produit toute l'année. Selon les variétés et la saison, elle sera récoltée 30 à 45 jours après la transplantation [Fortier, 2016]. Cela donne la possibilité de faire se succéder plusieurs cycles sur la même parcelle au cours d'une même année.

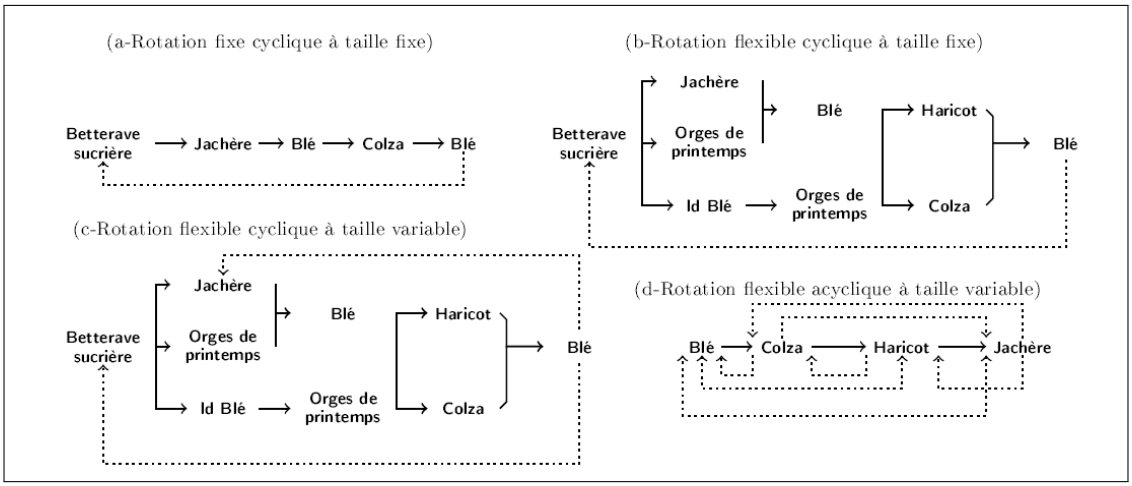


FIGURE 6.8 – Exemple de classes de rotations, extrait de [Castellazzi et al., 2008] et traduit par [Akplogan, 2013]

6.3 Méthodes de recherche opérationnelle pour la planification des cultures

La planification des cultures est un exercice crucial pour la conception d'un système agricole rentable. Toutefois, il n'est pas facile de savoir quelles cultures planter, en quelle quantité et à quel moment, tout en respectant les propriétés physiques (utilisation de l'eau) et chimiques (utilisation des engrais et pesticides) du sol. Dans ce sens, de nombreuses études sur la planification des cultures ont été menées pour aider les agriculteurs dans leurs décisions. Nous citons la revue de [Dury et al., 2012] qui fait référence à plus de 120 études sur les modèles de planification et de rotation des cultures. Ces modèles sont basés sur des objectifs différents. Le critère économique est le plus étudié dans la littérature, i.e., la planification des cultures est souvent effectuée de manière à maximiser les rendements de productions [Heady, 1948, Itoh et al., 2003, Galán-Martín et al., 2015]. De plus, les préoccupations d'aujourd'hui sur la durabilité des systèmes de cultures ont conduit les chercheurs à cibler explicitement des objectifs autres que la rentabilité, des objectifs environnementaux ou écologiques. Par exemple, minimiser l'utilisation des pesticides, réduire l'érosion du sol et améliorer la fertilité physique et biologique du sol [Dogliotti et al., 2005].

Outre les objectifs qui déterminent la sélection d'un plan de cultures, les modèles diffèrent également par le type de contraintes, qui limitent les décisions relatives aux plans de cultures. Ces contraintes sont souvent liées à la gestion des ressources limitées. Par exemple, la gestion des ressources disponibles en eau d'irrigation avec un objectif de maximiser les revenus [Gupta et al., 2000, Tsakiris and Spiliotis, 2006, Niu et al., 2016]. Autres ressources sont également incorporées dans les modèles de planification de cultures, comme la main d'oeuvre et le niveau de mécanisation [Dogliotti et al., 2005].

Un certain nombre de techniques explorées dans la littérature, pour la planification des cultures, se distinguent selon les objectif visés et les contraintes considérées. La programmation mathématique est la plus utilisée [Glen, 1987]. En particulier la programmation linéaire (PL) depuis le modèle simplifié de [Heady, 1954], qui vise à maximiser les rendements de production en tenant compte de la disponibilité des ressources en main d'oeuvre. Ensuite, des modèles plus réalistes que ce dernier ont été considérés comme une base des systèmes de planification des cultures. Nous citons le modèle "Purdue Top Farmer Cropping Model 'B'", destiné à être utilisé directement par les agriculteurs pour la

planification de la production des exploitations agricoles [McCarl et al., 1977]. Nous mentionnons également le modèle ROTAT de [Dogliotti et al., 2005]. Pour les fermes urbaines, nous citons l'outil de [Brulard et al., 2016] à base de programmes mathématiques d'aide à la conception de fermes maraîchères diversifiées. Toutefois, l'application de la programmation linéaire n'est pas toujours simple, certains problèmes rencontrent des difficultés de formulation mathématique (objectif et contraintes) ou d'interprétation des résultats [Nevo et al., 1994]. Pour réduire ces limites, la programmation linéaire a été progressivement étendue à d'autres techniques. Par exemple, plusieurs chercheurs ont suggéré d'intégrer l'incertitude dans leurs modèles. Nous citons [Itoh et al., 2003] qui ont proposé un modèle de programmation linéaire avec des valeurs incertaines (coefficients stochastiques) pour étudier l'influence du climat sur la planification des cultures. Nous mentionnons également [Sethi et al., 2006] qui ont développé un modèle de programmation linéaire pour optimiser l'allocation des cultures, tout en considérant les besoins en eau d'irrigation modélisés comme des variables stochastiques.

Motivés par plusieurs facteurs économiques, environnementaux ou organisationnels, les chercheurs ont développé des modèles d'optimisation multi-critères afin de regrouper divers objectifs lors de la planification des cultures [Gupta et al., 2000, Sarker and Quaddus, 2002, Tsakiris and Spiliotis, 2006]. Cette approche a été utilisée dans d'autres cadres autres que la programmation linéaire. [Sarker and Ray, 2009], par exemple, proposent deux formulations multi-critères du problème de planification des cultures (version linéaire et non linéaire), qu'ils ont résolues par une version améliorée de l'algorithme évolutionnaire.

L'utilisation des métaheuristiques, telles que l'algorithme évolutionnaire, vient du fait que le problème de planification des cultures est un problème NP-difficile (i.e., difficulté de vérifier la solution en temps polynomial) [Chetty and Adewumi, 2014]. L'objectif donc des méthodes approchées est de trouver des solutions de bonne qualité en temps raisonnable. Dans ce sens, [Chetty and Adewumi, 2014] examinent l'efficacité de l'utilisation des métaheuristiques à populations. Il s'agit des algorithmes d'intelligence en essaim comparés à un algorithme génétique sur une étude de cas à l'Afrique du Sud, où les premiers ont fourni les meilleurs résultats.

Les chercheurs ont suggéré d'autres approches pour résoudre le problème de planification de cultures, formalisé comme un problème de rotation ou de succession de cultures. Par exemple, [Haneveld and Stegeman, 2005] utilisent les réseaux de flots pour modéliser les séquences de cultures non permises, afin de les utiliser comme des contraintes dans un modèle standard de programmation linéaire. De même, [Detlefsen and Jensen, 2007] proposent une modélisation par les réseaux de flots, en tirant parti de leur structure claire et visuelle, pour utiliser des algorithmes spéciaux pour la résolution des problèmes de réseaux. Face à ces approches, [Filippi et al., 2017] développent un modèle de programmation linéaire mixte pour l'allocation de cultures, avec un objectif de maximiser le profit total, tout en tenant compte de la variabilité des prix et des rendements. [Alfandari et al., 2011] proposent également un modèle de programmation mixte, qui vise à minimiser la surface nécessaire pour couvrir les besoins des agriculteurs, plutôt que de maximiser les revenus. L'aspect combinatoire du modèle provient du fait de cultiver une parcelle ou de la laisser en jachère, alors que les variables réelles concernent les contraintes de demande. Ce problème a été prouvé NP-dfficile suite à une réduction au problème de Set Covering (voir chapitre 2). Après l'avoir démontré, [Alfandari et al., 2015] présentent une amélioration du modèle en proposant une formulation compact binaire de la programmation linéaire basée sur les graphes de séquences de cultures, puis une formulation étendue en utilisant la décomposition de Dantzig-Wolfe (voir chapitre

1). Ils présentent ensuite une résolution par l'algorithme de Branch-and-Price-and-Cut qui a montré son efficacité pour la formulation étendue. Dans la même optique, [Santos et al., 2015] proposent la même approche de Branch-and-Price-and-Cut pour résoudre le problème de rotation de cultures maraîchères avec l'objectif de minimiser la surface totale utilisée. Les résultats obtenus ont montré l'efficacité de la méthode suite à des expérimentations de calcul sur un ensembles d'instances basées sur des données réelles.

Des méthodes d'intelligence artificielle sont également utilisées dans ce cadre. Nous citons par exemple [Akplogan et al., 2013] qui proposent une résolution du problème d'allocation de cultures par satisfaction de contraintes pondérées. Les simulations à base d'agents sont également utilisées. Nous mentionnons [Taillandier et al., 2012] qui présente une application d'une architecture multi-agents basée sur le modèle de la rationalité d'un agent intelligent (Belief-Desire-Intention).

A notre connaissance, aucune des nombreuses études menées sur la planification de cultures n'a examiné l'allocation des cultures annuelles et des cultures pérennes dans une même approche de modélisation. Nous citons par exemple [Brulard, 2018] qui a proposé un outil à base de programmes mathématiques d'aide à la conception de systèmes de production maraîchers urbains. L'originalité donc de notre étude consiste à la modélisation de conception d'un système agroforestier urbain (verger-maraîcher), qui associe des arbres fruitiers (cultures pérennes) et des cultures maraîchères (cultures annuelles) sur une même parcelle.

6.4 Conclusion du chapitre

Dans ce chapitre, nous avons présenté les concepts généraux sur les systèmes agroforestiers, notamment les systèmes de verger-maraîcher où nous associons des arbres fruitiers et des cultures maraîchères sur la même parcelle. En premier lieu, nous avons défini les interactions écologiques entre les cultures pérennes et les cultures annuelles, à savoir les interactions aériennes et les interactions souterraines. En deuxième lieu, nous avons présenté la productivité dans les systèmes agroforestiers, qui dépend de plusieurs critères comme la densité des arbres et l'emplacement des cultures selon les espèces choisies. Ensuite, nous avons parlé de l'organisation du travail qui est impactée par les rythmes de croissance différents des arbres et des cultures et qui doit être considérée lors de la conception d'un système agroforestier. Puis, nous avons défini la rotation ou la succession des cultures sur la même parcelle et leur rôle important dans la préservation de la qualité du sol. En dernier lieu, nous avons présenté un panorama de méthodes de recherche opérationnelle, utilisées pour modéliser et résoudre des problèmes de planification et de rotation de cultures, en citant d'autres méthodes de l'intelligence artificielle et de la simulation à base d'agents.

Comme évoqué précédemment, aucune de ces nombreuses études menées sur la planification des cultures n'a présenté un modèle de conception spatio-temporelle d'un système agroforestier. La majorité des études cherche à démonter l'intérêt de l'agroforesterie sans présenter une approche de modélisation. D'où l'originalité de notre modèle de conception des systèmes de verger-maraîcher, que nous présentons dans le chapitre suivant.

Chapitre 7

Modélisation et résolution du problème de verger-maraîcher

Sommaire

7.1 Introduction

Comme nous l'avons présenté au chapitre précédent, les systèmes agroforestiers reposent sur un ensemble complexe d'interactions arbres - cultures modifiant l'utilisation de la lumière, de l'eau et des nutriments. Donc, concevoir un tel système nécessite d'optimiser l'utilisation de ces ressources, en maximisant les interactions positives (facilitation) et en minimisant celles négatives (compétition). En effet, de nombreuses études ont été menées pour montrer l'intérêt de l'agroforesterie ou pour proposer des modèles de planification et de rotation de cultures. Mais à notre connaissance, aucune étude n'a modélisé l'allocation spatio-temporelle des arbres fruitiers et des cultures maraîchères dans les systèmes agroforestiers (vergers-maraîchers). En conséquence, nous avons construit un premier prototype d'un système de verger-maraîcher basé sur la satisfaction de contraintes pondérées (WCSP Weighted Constraint Satisfaction Problem). Mais la résolution a été limitée à des systèmes de petite taille [Tchamitchian and Godin, 2014]. Dans ce chapitre, nous présentons des modèles plus larges basés sur la programmation mathématique.

Ce chapitre est divisé en trois sections. Dans la section 7.2, nous décrivons les choix effectués pour la construction des modèles de conception des systèmes de vergers-maraîchers. A savoir, les cultures choisies et leurs caractéristiques, les interactions aériennes et souterraines considérées, les rendements attendus et les règles d'organisations adoptées. Nous présentons ensuite dans la section 7.3 quatre formulations mathématiques du problème : quadratique, décomposée, mixte et linéaire. Puis dans la section 7.4 nous discutons les résultats obtenus.

7.2 Modélisation du problème

Nous définissons le problème de conception d'un système de verger-maraîcher comme un problème d'allocation de cultures. Son objectif est de positionner, sur la même parcelle, des arbres fruitiers et des cultures maraîchères sur un horizon temporel fini. Nous avons représenté la parcelle sous forme d'un carré découpé en $l \times l$ cellules, toutes de même taille (1m^2 à 4m^2) Chaque cellule représente une unité spatiale attribuée à une espèce cultivée (un arbre fruitier, une culture maraîchère ou de l'engrais vert), ou au sol nu lorsque l'unité spatiale ne porte aucune espèce cultivée.

Concernant le temps, nous avons considéré un horizon temporel de trois périodes, selon les phases de croissance d'un arbre : période P1 correspond à très jeune arbre, juste après la plantation, P2 à un jeune arbre qui développe rapidement son système racinaire et sa structure aérienne et P3 à un arbre mature. Chaque période représente un an de rotation (quatre saisons), sauf pour P1 que nous représentons par une seule saison, vu que nous considérons la terre en jachère à cette période. Donc, nous avons neuf pas de temps et par conséquent neuf grilles de $l \times l$ unités spatiales, comme schématisé dans la figure 7.1.

7.2.1 Choix et caractéristiques des arbres fruitiers

Pour rappel, notre modèle consiste à étudier l'impact des arbres sur les cultures maraîchères et d'optimiser les interactions aériennes et souterraines entre eux. Choisir plusieurs espèces fruitières ne fait que compliquer le modèle, en multipliant le nombre de variables et de contraintes. Pour simplifier le modèle, nous avons donc choisi un seul type d'arbre fruitier, un pommier, que nous considérons comme un arbre générique dans le modèle.

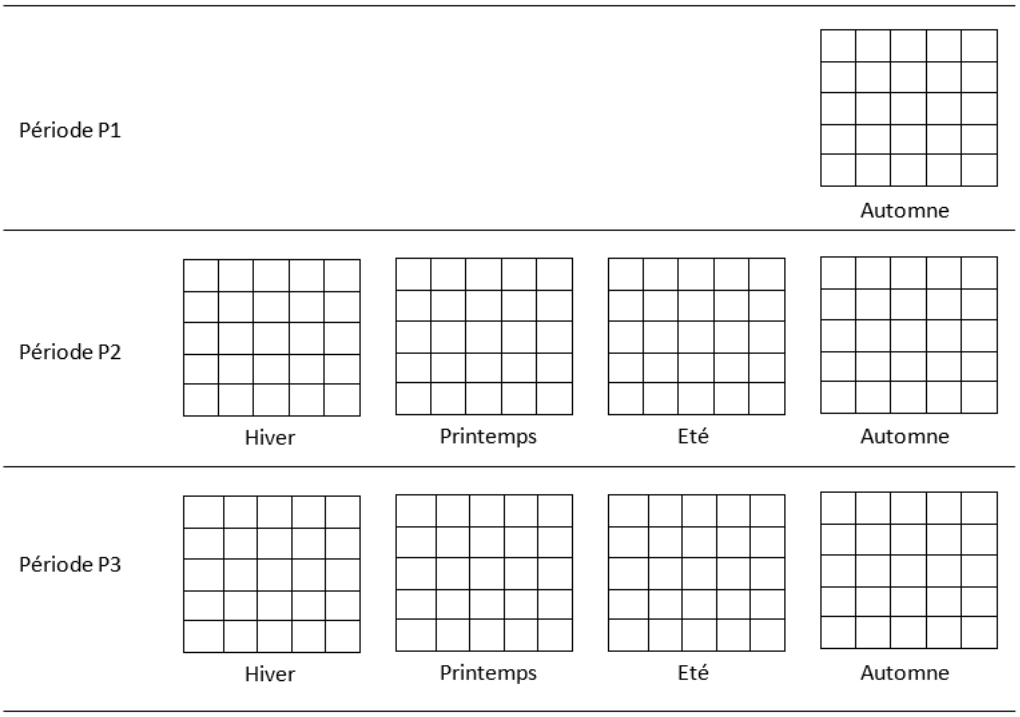


FIGURE 7.1 – Représentation spatiale et temporelle d'une parcelle de taille 5×5 dans le modèle de conception.

Espacement minimal entre arbres

Le choix d'espacement entre les arbres au moment de la plantation influe sur leur évolution. De ce fait, il faut définir un espacement permettant, aux arbres, une croissance sans conflit et une meilleure répartition lumineuse. Cet espacement minimal est plus faible que celui recommandés dans les systèmes de vergers purs, afin d'assurer un bon fonctionnement des cultures maraîchères associées, qui peuvent être impactées positivement ou négativement par la présence des arbres. La figure 7.2 illustre l'espacement minimal considéré dans notre modèle.

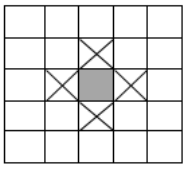


FIGURE 7.2 – Représentation de l'espacement minimal entre arbres. La cellule grise représente un arbre et les cellules croisées représentent les zones interdites à la plantation des autres arbres.

Interception du rayonnement solaire

Naturellement, le feuillage des arbres se développe au printemps, se stabilise en été, puis tombe en automne et recommence le printemps suivant. Ainsi, lors des saisons printemps et été, où les arbres ont des feuilles, les cultures plantées à proximité peuvent subir une réduction du rayonnement solaire disponible. Cette zone impactée est généralement liée

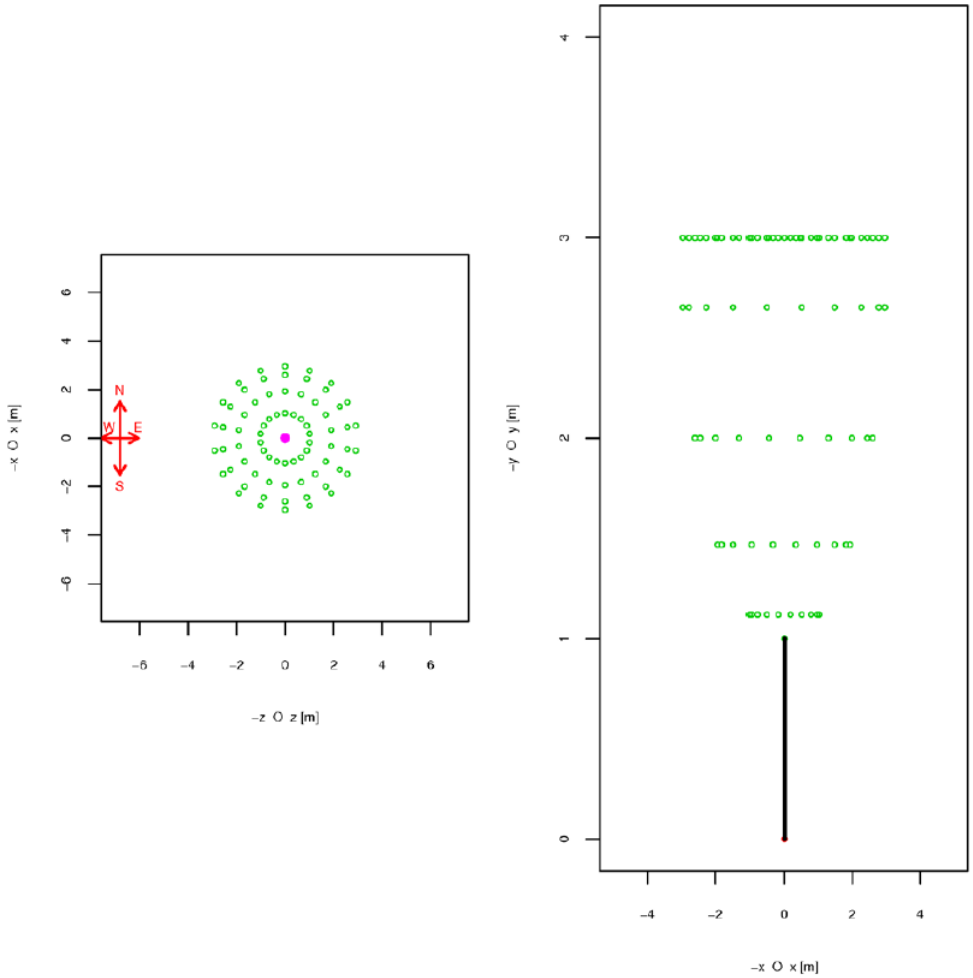


FIGURE 7.3 – Morphologie générique choisie pour la modélisation d'un pommier à maturité. Source : outil de simulation développé par l'unité PSH (Plantes et Systèmes de cultures Horticoles) INRA Avignon

à la morphologie de l'arbre, à savoir la hauteur, la largeur et la densité foliaire de houppier. Elle est également liée à la période de l'année où l'inclinaison des rayons solaires varie.

La figure 7.3 présente la morphologie générique choisie pour la modélisation d'un pommier à maturité. Pour définir le périmètre d'interception du rayonnement solaire autour de ce fruitier, nous nous sommes basés sur les résultats d'un logiciel de simulation qui prend en considération tous les critères cités ci-dessus comme la saison, la morphologie de l'arbre, et sa densité foliaire. Les résultats de cette simulation sont illustrés dans la figure 7.4 qui représente les niveaux d'interception par des couleurs différentes, durant les saisons de printemps et d'été. La couleur rouge est utilisée pour une interception supérieure à 50%, où ne nous pouvons pas planter des cultures. La couleur jaune pour une interception inférieure à 20% qui n'a pas d'effet sur les cultures. Enfin, la couleur orange pour une interception entre 20% et 50% qui représente l'ombre potentielle. Nous avons traduit ces résultats de simulation pour représenter les niveaux d'interception du rayonnement solaire sous les arbres sur la parcelle carrée de notre modèle de conception (Figure 7.5). Cela nous a permis de modéliser l'ombre potentielle autour d'un arbre par une zone fixe orientée vers le nord (Figure 7.4).

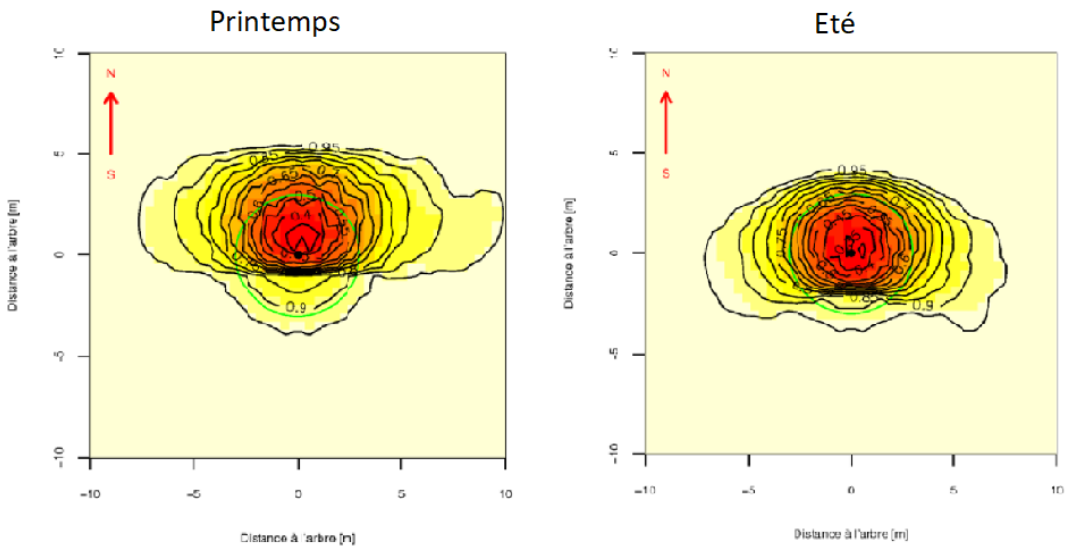


FIGURE 7.4 – Résultats de simulation d'interception du rayonnement solaire par les arbres, avec une densité foliaire de 2,5. Les couleurs indiquent les niveaux d'interception autour de l'arbre : plus la couleur est foncée, plus le rayonnement intercepté est important. Source : outil de simulation PSH - INRA Avignon

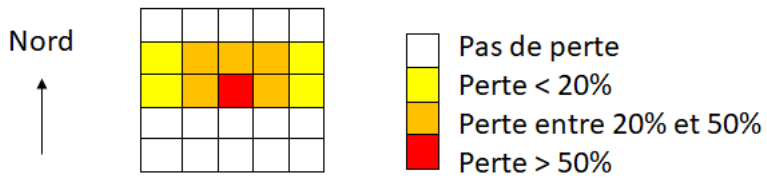


FIGURE 7.5 – Modélisation des niveaux d'interception du rayonnement solaire par les arbres.

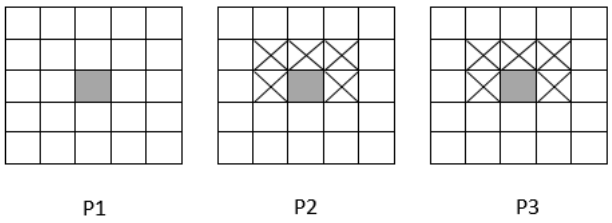


FIGURE 7.6 – Unités spatiales affectées par la perte de rayonnement solaire (cellules croisées) par un arbre (cellule grise), durant les trois périodes P1, P2 et P3.

Systèmes racinaires des fruitiers

Les systèmes racinaires des arbres se différencient des cultures annuelles par leur pérennité. Le développement des racines varie d'une année à l'autre avec une production plus élevée les premières année de l'arbre [Atkinson, 1983]. Malheureusement, il existe très peu d'informations sur le développement des racines des fruitiers lors des premières années après la plantation. Pour modéliser donc le système racinaire des arbres fruitiers, nous nous sommes basés sur une simulation 3D d'architecture racinaire d'un pêcher greffé sur un prunier [Vercambre et al., 2003]. Le modèle est réalisé, à l'INRA d'Avignon, sur la base de quatre ans d'observation dans un verger expérimental avec une irrigation minimale et une densité de plantation faible (70 arbres/ha). De sorte que les arbres soient largement espacés pour éviter les interactions souterraines entre eux. En regardant la figure 7.7, qui présente les résultats de simulation, nous remarquons que le système racinaire des arbres colonise un grand volume du sol. Certaines racines descendent en profondeur, progressivement, d'environ 1m par an. Tandis que d'autres s'étendent latéralement à toutes les profondeurs atteintes avec un maximum de 3m. Cette observation nous a conduit à élaborer le modèle schématisé dans la figure 7.8.

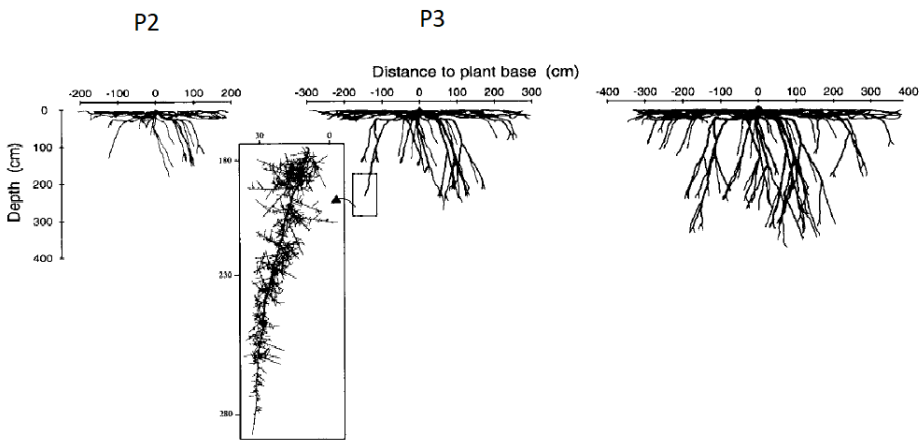


FIGURE 7.7 – Simulation 3D du système racinaire d'un pêcher greffé sur prunier, à 2, 3 et 4 ans après la plantation.

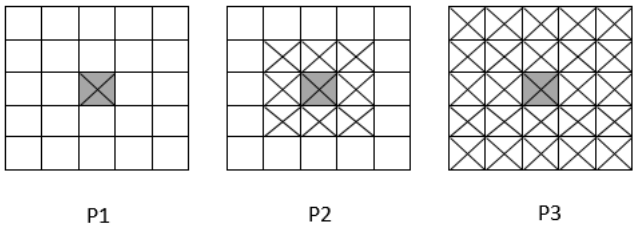


FIGURE 7.8 – Croissance des racines (cellule croisées) d'un arbre fruitier (cellule grise) en périodes P1, P2 et P3.

7.2.2 Choix et caractéristiques des cultures maraîchères

Quant aux cultures maraîchères, nous avons sélectionné un ensemble de cultures avec des dates et des durées de plantation différentes, illustrées sur le tableau 7.1. Nous avons choisi des cultures de cycle court (une saison), à savoir le melon planté en été et trois variétés de salade, chacune sur une saison (printemps, automne et été). De plus, nous avons choisi des cultures avec des cycles longs qui s'étalent sur deux saisons : tomate et oignon au printemps-été et carotte en été-automne. De plus, nous avons ajouté deux types d'engrais verts à intercaler avec ces cultures maraîchères pour préserver la qualité du sol.

Hiver	Printemps	Été	Automne
	Salade 1		
		Salade 2	
			Salade 3
	Tomate		
		melon	
		Carotte	
	Oignon		
			Engrais vert 1
Engrais vert 2			Engrais vert 2

TABLE 7.1 – Dates et durées de plantation des cultures maraîchères en nombre de saisons.

Sensibilité des cultures à l'ombre

Pour analyser l'impact de l'ombre des arbres sur les cultures maraîchères, nous nous sommes basés sur les connaissances des agriculteurs locaux. Basé sur ces informations collectées, le tableau 7.2 présente le degré de sensibilité des espèces choisies à l'interception du rayonnement solaire. Cela est présenté par une évaluation qualitative en fonction de l'espèce et la saison pendant laquelle elle est cultivée. Le symbole (0) indique un effet neutre, (-) un effet négatif et (+) un effet positif. Le nombre des symboles utilisés (++ ou +++) indique le degré de sensibilité à l'ombre potentiel des arbres.

Cultures	Printemps	Eté
Salade	0	+++
Tomate	0	++
Oignon	-	0
Melon		-
Carotte		0

TABLE 7.2 – Sensibilité des cultures maraîchères choisies à la perte de rayonnement solaire

Systèmes racinaires des cultures maraîchères

Le développement des systèmes racinaires varie d'une espèce à l'autre, vu qu'elles ont des besoins différents en eau et en nutriments. Tiréé de [Weaver and Bruner, 1927], la figure 7.9 illustre les systèmes racinaires des espèces choisies. Nous constatons que l'oignon a des racines peu ramifiées qui se développent dans les couches superficielles autour de la plante (1 pied = 30cm). De même, le melon a des racines de faible profondeur (2 pieds)

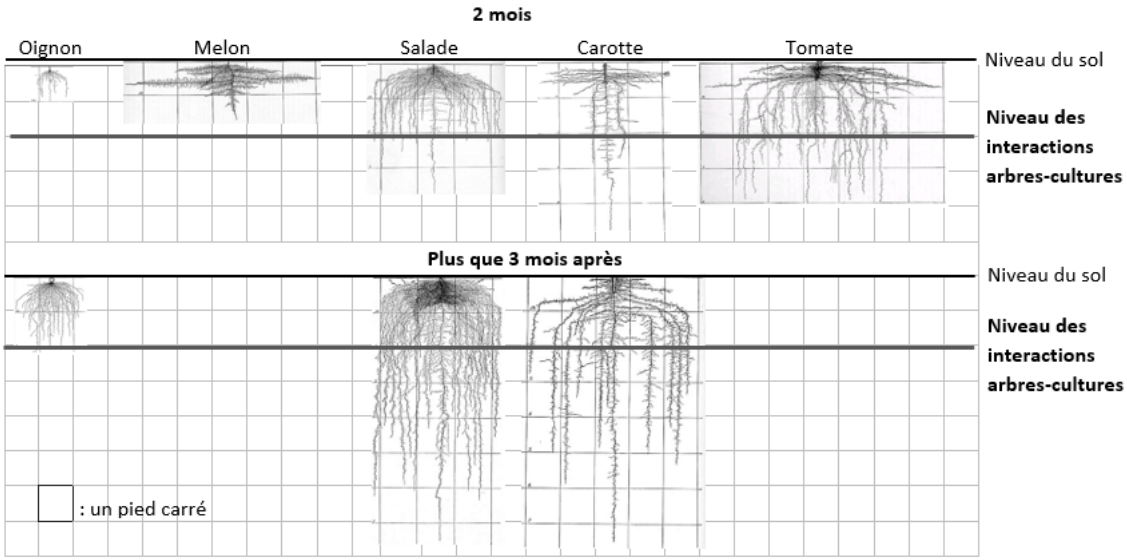


FIGURE 7.9 – Systèmes racinaires d'espèces maraîchères

mais qui s'étendent bien horizontalement. La salade a des racines latérales bien ramifiées avec une racine pivotante profonde (4 pieds) peu ramifiée. La carotte a des racines qui s'étendent verticalement (5 pieds) avec des branches courtes. Tandis que la tomate a des racines qui remplissent le sol autour de la plante, en s'étendant en profondeur vers de nouveaux territoires.

Pour définir le niveau de compétition pour l'eau entre arbres et cultures, nous avons classé les systèmes racinaires des cultures en fonction de la surface occupée en profondeur. Ce niveau, qui commence à deux pieds du sol, correspond au niveau où les racines des arbres fruitiers commencent à s'étendre en profondeur avec plus de branchement (Figure 7.8). De ce fait, l'ordre de classement est comme suit : (oignon = melon) $<$ salade $<$ carotte $<$ tomate.

Contrairement à cette compétition, associer des arbres et des cultures sur la même parcelle favorise une meilleure utilisation de l'eau grâce aux systèmes racinaires des arbres, qui transfèrent l'eau des couches profondes du sol vers les surfaces superficielles et récupèrent l'excès d'eau qui pourrait nuire aux cultures maraîchères (voir chapitre 6). Dans notre modèle, le surplus d'eau existe à toutes les saisons sauf l'été. A ces périodes, le partage de l'eau se produit automatiquement par la présence d'une culture auprès d'un arbre. Cependant, si la culture est plantée loin d'un arbre, elle pourrait être altérée par un excès d'eau.

Mixité des cultures maraîchères

Une fois que les cultures sont choisies, il faut déterminer combien produire de manière à diversifier la production à chaque saison. Pour répondre à cette question, nous avons ajouté des contraintes de mixité des cultures maraîchères dans le modèle, représentées par des fourchettes de production (Figure 7.10). Les valeurs minimales sont choisies de sorte qu'à chaque saison il y ait une mixité minimale de cultures dans la parcelle, en laissant suffisamment d'unités spatiales pour positionner des arbres. Tandis que les valeurs maximales correspondent au cas où toutes les unités spatiales sont attribuées à des cultures maraîchères sans arbres.

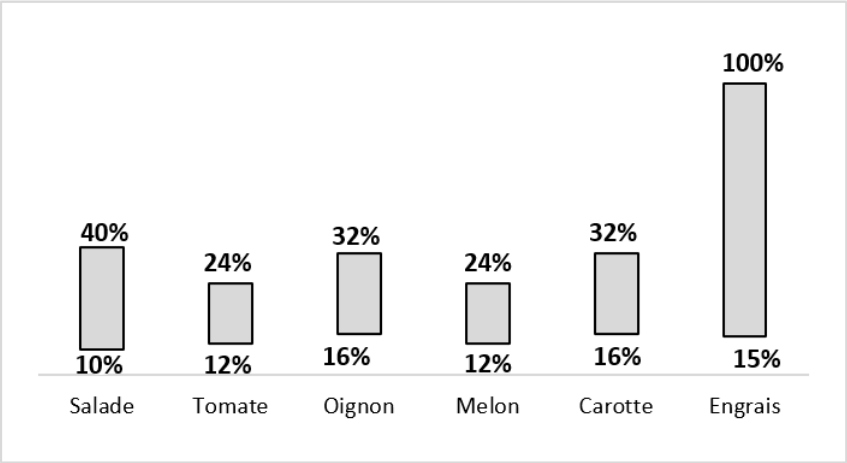


FIGURE 7.10 – Pourcentages min-max de la parcelle à attribuer à chaque culture sur une saison donnée

Règles d'organisation

Comme évoqué dans le chapitre 6, les délais de retour minimal des cultures sont de plusieurs années. Pour modéliser la rotation des cultures en respectant les délais de retour, il faut disposer de plusieurs cultures par saison. Or, nous n'avons sélectionné qu'un petit nombre d'espèces, six espèces sur toutes saisons confondues. Ce qui n'aide pas à modéliser correctement les délais de retour. En conséquence et pour prendre en considération, dans notre modèle, l'effet précédent des cultures, nous avons ajouté des contraintes qui interdissent de planter la même culture au même endroit deux années successives.

Un autre critère souhaité concerne le regroupement spatial des cultures maraîchères. Cela a un impact positif sur le temps de travail nécessaire pour cultiver ces cultures. Nous avons modélisé cette règle par une pénalisation des cultures dispersées sur la parcelle, évaluées en comparant la culture plantée sur une unité spatiale avec son adjacente horizontale ou verticale (Figure 7.11).

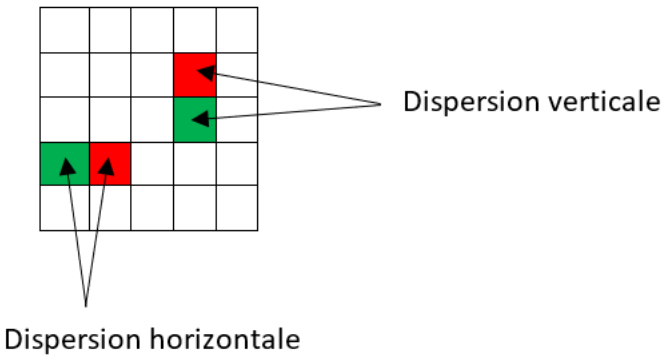


FIGURE 7.11 – Modélisation de la dispersion horizontale/verticale des cultures maraîchères

7.3 Formulation mathématique

Pour résoudre le problème de verger-maraîcher, nous avons reformulé le modèle conceptuel, décrit dans la section précédente, en un modèle mathématique. D'abord, nous avons élaboré un modèle basé sur la programmation quadratique en variables binaires sous contraintes linéaires [Maqrot et al., 2016, Maqrot et al., 2017a], que nous avons ensuite reformulé en un modèle de programmation linéaire mixte [Maqrot et al., 2017b]. Puis en un modèle un modèle de programmation linéaire en variables binaires [Maqrot et al., 2018b].

7.3.1 Modèle quadratique en variables binaires (BQP)

Comme définit dans le chapitre 1, un problème de programmation quadratique en variables binaires est un problème d'optimisation, qui consiste à trouver un vecteur x qui minimise une fonction objectif quadratique. La forme mathématique la plus générale sous contraintes linéaires est la suivante :

$$\min \quad \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} q_{ij} x_i x_j \tag{7.1}$$

$$s.c \qquad \sum_{i=1}^{n} a_{ki} x_i \leq b_k \qquad k = 1, ..., m \tag{7.2}$$

$$x_i, x_j \in \{0, 1\} \qquad i = 1, ..., n; j = 1, ..., n \tag{7.3}$$

où $Q^{n \times n}$ est une matrice de coefficients carrée symétrique ($q_{ij} = q_{ji}$), et c un vecteur de coefficients linéaire.

Variables

Pour reformuler le problème de verger-maraîcher sous cette forme quadratique, nous considérons quatre variables binaires :

- $crop^{t,c}_{x,y} = 1$ ssi l'unité spatiale de coordonnées x, y a une culture c au temps t
- $tree_{x,y} = 1$ ssi l'unité spatiale de coordonnées x, y a un arbre
- $shade_{x,y} = 1$ ssi l'unité spatiale de coordonnées x, y a de l'ombre d'un arbre
- $root^{p}_{x,y} = 1$ ssi l'unité spatiale de coordonnées x, y a des racines d'un arbre à la période p

Paramètres

Soit $L = \{1, ..., l\}$ l'ensemble des positions horizontales/verticales des unités spatiales d'une parcelle de forme carrée, représentée sous forme d'une grille de taille $l \times l$. La numérotation des positions commence en haut à gauche (i.e, nord ouest) avec les coordonnées ($x = 1, y = 1$). Soit ζ l'ensemble des cultures, T l'ensemble des pas de temps (chaque pas de temps représente une saison, commençant en automne), et P l'ensemble des périodes de croissances des arbres, avec T_p l'ensemble des pas de temps de chaque période $p \in P$. Dans nos expériences, nous faisons varier la taille de la parcelle $l \times l$ en utilisant un ensemble fixe de ζ, T, P et T_p :

- $\zeta = \{oignon, melon, salade, carotte, tomate, engrais, sol\ nu\}$
- $T = \{1, ..., 9\}$

- $P = \{1,2,3\}$
- $T_1 = \{1\} \sim \{automne\}, T_2 = \{2,3,4,5\}, T_3 = \{6,7,8,9\} \sim \{hiver, printemps, \text{été}, automne\}$.

Fonction objectif : optimisation des interactions arbres-cultures

Rappelons que l'objectif de notre problème est de concevoir un système de verger-maraîcher, qui permet d'optimiser les interactions aériennes et souterraines entre arbres et cultures. Basés sur les interactions décrites à la section précédente, nous distinguons trois situations possibles pour chaque unité spatiale :

- SR : où l'ombre et racines sont présentes,
- $\bar{S}R$: où les racines sont présentes sans ombre,
- $\bar{S}\bar{R}$: où l'ombre et racines sont absentes.

Chaque situation s est liée à une valeur v^s qui peut être négative ou positive selon le type des interactions (facilitation ou compétition). Cela se traduit par la fonction objectif suivante :

$$\min \sum_{\substack{p \in P, t \in T^p, \\ c \in \zeta, x,y \in L}} [v^{SR} \times shade_{x,y} \times root^p_{x,y} + v^{\bar{S}R} \times (1 - shade_{x,y}) \times root^p_{x,y} + v^{\bar{S}\bar{R}} \times (1 - shade_{x,y}) \times (1 - root^p_{x,y})] \times crop^{t,c}_{x,y} \tag{7.4}$$

Notons que, durant les périodes P2 et P3, la présence de l'ombre (Figure 7.6) à une unité spatiale implique la présence des racines (Figure 7.8) :

$$shade_{x,y} \Rightarrow roots^p_{x,y} \tag{7.5}$$

Avec cette implication (équation 7.5), la fonction objectif 7.4 devient :

$$\min \sum_{\substack{p \in P, t \in T^p, \\ c \in \zeta, x,y \in L}} [A^{t,c} \times shade_{x,y} + B^{t,c} \times root^p_{x,y} + C^{t,c}] \times crop^{t,c}_{x,y} \tag{7.6}$$

Avec :

- $A^{t,c} = v^{SR} - v^{\bar{S}R}$: valeurs liées au degré de sensibilité des cultures à l'ombre à une saison quelconque (pas d'ombre à la période 1, i.e., $A^{1,c} = 0$). Selon le tableau 7.2, nous affectons un coût de 10 à un négatif (-), un coût de 0 à un effet neutre (0) et un coût de -10 à un effet positif (+).

- $B^{t,c} = v^{\bar{S}R} - v^{\bar{S}\bar{R}}$: valeurs liées aux interactions générées par les systèmes racinaires des arbres fruitiers et des cultures maraîchères (compétition ou partage pour l'eau). Basés sur la figure 7.9, nous avons déjà défini le niveau de compétition pour l'eau en classant les cultures maraîchères dans un ordre croissant selon leurs systèmes racinaires : (oignon = melon) < salade < carotte < tomate. Ainsi, basés sur ce classement, nous attribuons respectivement (en été) les coûts suivants : 0, 10, 20 et 30. Ensuite, nous réduisons les coûts pour les cultures permettant de partager les ressources en eau avec des arbres en toutes saisons sauf en été (tableau 7.3) : -10 pour les cultures maraîchères et -20 pour l'engrais vert.

	Compétition	Partage
Salade printemps	0	-10
Salade été	10	0
Salade automne	0	-10
Tomate	30	-10
Melon	0	0
Carotte	20	-10
Oignon	0	-10
Engrais	0	-20
Sol nu	0	10

TABLE 7.3 – Valeurs liées aux interactions souterraines (compétition et partage pour l'eau).

- $C^{t,c} = v^{\bar{S}\bar{R}}$: valeurs liées à l'absence de l'ombre et des racines des arbres. Nous évaluons cette situation par un coût de 10 quand le partage n'est pas en place (valeurs opposées de la colonne partage du tableau 7.3). Plus un coût dû à l'absence de l'ombre en été : 20 pour la salade et 10 pour la tomate.

Le tableau 7.4 résume les valeurs $A^{t,c}$, $B^{t,c}$ et $C^{t,c}$ associées à chaque culture maraîchère dans une saison quelconque. Ces coûts sont déterminant pour la quantité à planter et le placement des cultures sur les unités spatiales. Pour réduire le nombre des unités spatiales affectées au sol nu, nous ajoutons respectivement aux coefficients associés ($A^{t,c}$, $B^{t,c}$ et $C^{t,c}$) des valeurs égales à $max(A^{t,c})+10$, $max(B^{t,c})+10$ et $max(C^{t,c})+10$.

	Hiver			Printemps			Été			Automne		
	A	B	C	A	B	C	A	B	C	A	B	C
Salade printemps				0	-10	10						
Salade été							-30	-10	20			
Salade automne											-10	10
Tomate				0	-10	10	-20	30	10			
Melon							10	0	0			
Carotte							0	20	0		-10	10
Oignon				10	-10	10	0	0	0			
Engrais vert 1											-10	10
Engrais vert 2		-10	10								-10	10
Sol nu		40	30	20	40	30	20	40	30		40	30

TABLE 7.4 – Coefficients de la fonction objectif (7.6), représentés par des couleurs selon les espèces et les durées de plantation. Les coefficients $A^{t,c}$ en hiver ($t \in \{2,6\}$) et en automne ($t \in \{1,5,9\}$) sont en gris pour indiquer l'absence de l'ombre à ces deux saisons.

Fonction objectif : optimisation de la dispersion des cultures

En plus de l'objectif principal qui consiste à optimiser les interactions arbres-cultures, nous modélisons le regroupement spatial des cultures maraîchères comme un objectif secondaire de notre problème. Selon Figure 7.11, l'équation (7.7) traduit la pénalisation de la dispersion spatiale des cultures, évaluée en comparant (deux à deux) les cultures présentes dans les unités spatiales adjacentes horizontales et verticales. Nous accordons à chaque dispersion un coefficient de pénalisation, $C_{disp} > 0$. Pour avoir la dispersion comme critère secondaire par rapport aux interactions, nous mettons $\frac{1}{2\times L^2+1}$, avec L la dimension de la

grille, pour que dans le pire cas la somme des dispersions soit inférieure à 1 , en supposant que les coefficients des interactions ($A^{t,c}$, $B^{t,c}$ et $C^{t,c}$) soient des entiers.

$$C_{disp} \times [\sum_{t,c,X-1,y} ((1-crop^{t,c}_{x,y}) \times crop^{t,c}_{x+1,y} + (1-crop^{t,c}_{x+1,y}) \times crop^{t,c}_{x,y}) \\ + \sum_{t,c,x,Y-1} ((1-crop^{t,c}_{x,y}) \times crop^{t,c}_{x,y+1} + (1-crop^{t,c}_{x,y+1}) \times crop^{t,c}_{x,y})] \tag{7.7}$$

Contraintes

Afin de compléter le modèle mathématique, nous avons défini les contraintes linéaires suivantes :

1. Une culture ou un arbre sur chaque unité spatiale

$$tree_{x,y} + \sum_{c \in C} crop^{t,c}_{x,y} = 1 \quad (\forall t \in T, \forall x,y \in L^2) \tag{7.8}$$

2. Espacement minimal entre arbres (Figure 7.2)

$$tree_{x,y} + tree_{x+1,y} \leq 1 \quad (\forall x \in L - \{l\}, \forall y \in L) \tag{7.9}$$
$$tree_{x,y} + tree_{x,y+1} \leq 1 \quad (\forall x \in L, \forall y \in L - \{l\}) \tag{7.10}$$

3. Aucun arbre sur les bordures Est et Ouest de la parcelle

$$tree_{x,l} = 0 \quad (\forall x \in L) \tag{7.11}$$
$$tree_{1,y} = 0 \quad (\forall y \in L) \tag{7.12}$$

4. Définition de l'ombre au Nord, Est et Ouest d'un arbre (Figure 7.6)

$$-6 \times shade_{x,y} + \sum_{\substack{i \in \{\max(1-x,-1),\ldots,\min(l-x,1)\} \\ j \in \{0,\ldots,\min(l-y,1)\}}} tree_{x+i,y+j} \leq 0 \quad (\forall x,y \in L^2) \tag{7.13}$$
$$shade_{x,y} - \sum_{\substack{i \in \{\max(1-x,-1),\ldots,\min(l-x,1)\} \\ j \in \{0,\ldots,\min(l-y,1)\}}} tree_{x+i,y+j} \leq 0 \quad (\forall x,y \in L^2) \tag{7.14}$$

5. Évolution des racines d'un arbre (Figure 7.8)

$$root^1_{x,y} - tree_{x,y} = 0 \quad (\forall x,y \in L^2) \tag{7.15}$$
$$-9 \times root^p_{x,y} + \sum_{i \in I, j \in J} root^{p-1}_{x+i,y+j} \leq 0 \quad (\forall x,y \in L^2, \forall p \in \{2,3\}) \tag{7.16}$$
$$root^p_{x,y} - \sum_{i \in I, j \in J} root^{p-1}_{x+i,y+j} \leq 0 \quad (\forall x,y \in L^2, \forall p \in \{2,3\}) \tag{7.17}$$

où $I = \{\max(1-x,-1),\ldots,\min(l-x,1)\}$ et $J = \{\max(1-y,-1),\ldots,\min(l-y,1)\}$

6. Contrainte de rupture de symétrie verticale des solutions

$$\sum_{x \in \{(l-\lfloor l/2 \rfloor+1),\ldots,l\}, y \in L} tree_{x,y} - \sum_{x \in \{1,\ldots,\lfloor l/2 \rfloor\}, y \in L} tree_{x,y} \leq 0 \tag{7.18}$$

7. Quantités minimales et maximales des cultures (Figure 7.10)

$$minBalance(t,c) \leq \sum_{x,y} crop_{x,y}^{t,c} \leq maxBalance(t,c) \qquad (\forall t \in T, \forall c \in \zeta) \quad (7.19)$$

8. Rotation de cultures

$$crop_{x,y}^{t_2,c} + crop_{x,y}^{t_3,c} \leq 1 \qquad \forall t_2 \in T^2, \forall t_3 \in T^3, \quad \forall c \in \{oignon, tomate, carotte, melon\} \quad (7.20)$$

9. Cultures sur deux saisons consécutives (Tableau 7.1)

$$crop_{x,y}^{t,c} - crop_{x,y}^{t+1,c} = 0 \quad (\forall p \in P, t = f_c(T^p), \forall x,y \in L^2, \forall c \in \{oignon, tomate, carotte, engrais\}) \quad (7.21)$$

Pour cette dernière contrainte (7.21), $f_c(T^p)$ représente le pas de temps qui correspond à une culture c plantée à la saison T^p de la période p et dure deux saisons. Pour illustrer, l'engrais vert est planté en automne, i.e., $f_{engrais}(T^1) = 1$, l'oignon et la tomate sont plantées au printemps, i.e., $f_{oignon \vee tomate}(T^2) = 3$, et carotte est plantée en été, $f_{carotte}(T^3) = 8$).

7.3.2 Modèle linéaire en variables mixtes (MIP)

Pour reformuler le problème de conception de verger-maraîcher sous forme d'un programme linéaire mixte, nous avons simplifié le modèle précédent, en supprimant les contraintes sur la rotation de cultures (7.20) et l'objectif secondaire sur leur regroupement spatial (7.7). Ensuite, nous avons appliqué l'approche de décomposition de Benders (voir chapitre 1) qui consiste à séparer les variables en deux ensembles (deux niveaux de décisions) : variables appartenant au problème maître et variables contrariantes appartenant au sous-problème primal. Généralement, le problème maître est plus facile à résoudre que le problème d'origine car il s'agit d'une relaxation i.e., il considère les variables contrariantes comme des valeurs constantes fournies par la résolution du sous-problème dual. En d'autres termes, le sous-problème dual alimente le problème maître restreint par des contraintes (coupes) qui renforcent le problème maître jusqu'à obtenir un problème équivalent au problème de départ.

En appliquant ce principe, nous séparons le problème principal de verger-maraîcher en deux sous-problèmes : (i) positionnement des arbres et (ii) optimisation des interactions arbres-cultures au fil du temps. Le problème (i) correspond au problème maître restreint (PMR) dans l'approche de Benders et le problème (ii) correspond au sous-problème (SP). Le problème maître garde les variables binaires $tree_{x,y}$, $shade_{x,y}$ et $root_{x,y}^p$, et les contraintes associées (7.9)-(7.18). Les contraintes restantes (7.19)-(7.21) appartiennent au sous-problème (Figure 7.12).

Le problème maître communique avec le sous-problème en introduisant de nouvelles variables entières qui correspondent au nombre d'arbres, i.e.,

$$trees = \sum_{x,y \in L^2} tree_{x,y},$$

nombre des unités spatiales à l'ombre, i.e.,

$$shades = \sum_{x,y \in L^2} shade_{x,y} - trees,$$

Problème maître (Positionnement des arbres)	**Sous-problème** (Optimisation des interactions ⇒ quantités de production)
Variables discrètes	Variables continues (solutions entières)
$tree_{x,y}$	$crop^{t,c}$
$shade_{x,y}$	$crop^{t,c}_{sr}$
$root^p_{x,y}$	$crop^{t,c}_{\bar{s}r}$

FIGURE 7.12 – Objectifs et variables du problème maître et du sous-problème.

et nombre des unités spatiales de présence de racines à chaque période p, i.e.,

$$roots_p = \sum_{x,y \in L^2} root^p_{x,y} - trees$$

Ces variables supplémentaires $trees, shades, roots_p$ sont considérées comme des constantes dans le sous-problème, i.e., des informations sur le positionnement des arbres nécessaires pour optimiser les interactions arbres-cultures.

Concernant les variables du sous-problème, Au lieu d'avoir $crop^{t,c}_{x,y}$ variables binaires qui définissent exactement l'emplacement (x,y) de toute culture c à toute saison t, nous les remplaçons par des quantités de production pour chaque type d'unités spatiales. Nous distinguons trois configurations possibles : unités spatiales ayant ombre et racines (sr), unités spatiales avec seulement des racines $(\bar{s}r)$, et unités spatiales sans ombre ni racines $(\bar{s}\bar{r})$. Rappelons que la présence d'ombre aux périodes P2 et P3 implique la présence des racines (7.5). Les variables continues correspondantes sont respectivement : $q_{crop^{t,c}_{sr}}$, $q_{crop^{t,c}_{\bar{s}r}}$, et $q_{crop^{t,c}_{\bar{s}\bar{r}}}$. Elles représentent des partitions de l'ensemble des unités spatiales de la parcelle (sauf celles attribuées aux arbres), leur somme est égale à $q_{crop^{t,c}}$ (7.23)-(7.24). Nous ajoutons les contraintes (7.25)-(7.26) pour assurer une allocation faisable sur des unités spatiales ayant ombre ou racines sans présence d'arbre. Avec cette reformulation, le SP a maintenant une fonction objectif linéaire :

$$\min \sum_{t \in T, c \in \zeta} (A^{t,c} + B^{t,c}) q_{crop^{t,c}_{sr}} + B^{t,c} q_{crop^{t,c}_{\bar{s}r}} + C^{t,c} q_{crop^{t,c}} \tag{7.22}$$

Sous contraintes,

$$q_{crop^{t,c}_{sr}} + q_{crop^{t,c}_{\bar{s}r}} + q_{crop^{t,c}_{\bar{s}\bar{r}}} - q_{crop^{t,c}} = 0 \quad (\forall t \in T, \forall c \in \zeta) \tag{7.23}$$

$$\sum_{c \in \zeta} q_{crop^{t,c}} - l^2 + trees = 0 \quad (\forall t \in T) \tag{7.24}$$

$$\sum_{c \in \zeta} q_{crop^{t,c}_{sr}} - shades = 0 \quad (\forall t \in T \setminus \{1\}) \tag{7.25}$$

$$\sum_{c \in \zeta} (q_{crop^{t,c}_{sr}} + q_{crop^{t,c}_{\bar{s}r}}) - roots_p = 0 \quad (\forall p \in P, \forall t \in T^p) \tag{7.26}$$

$$minBalance(t,c) \leq q_{crop^{t,c}} \leq \quad maxBalance(t,c) \quad (\forall t \in T, \forall c \in \zeta) \tag{7.27}$$

$$q_{crop^{t,c}} - q_{crop^{t+1,c}} = 0 \quad \begin{matrix}(\forall p \in P, t = f_c(T^p),\\ \forall c \in \{oignon, tomate, carotte, engrais\})\end{matrix} \tag{7.28}$$

$$q_{crop^{t,c}_{sr}} - q_{crop^{t+1,c}_{sr}} = 0 \quad \begin{matrix}(\forall p \in P \setminus \{1\}, t = f_c(T^p),\\ \forall c \in \{oignon, tomate, carotte, engrais\})\end{matrix} \tag{7.29}$$

$$q_{crop^{t,c}_{\bar{s}r}} - q_{crop^{t+1,c}_{\bar{s}r}} = 0 \quad \begin{matrix}(\forall p \in P \setminus \{1\}, t = f_c(T^p),\\ \forall c \in \{oignon, tomate, carotte\})\end{matrix} \tag{7.30}$$

$$q_{crop^{t,engrais}_{\bar{s}r}} - q_{crop^{t+1,engrais}_{\bar{s}r}} \leq 0 \quad (\forall p \in P, t = f_{engrais}(T^p)) \tag{7.31}$$

La variable $q_{crop_{\bar{sr}}^{t,c}}$ apparaît uniquement dans la contrainte (7.23) en tant que variable d'écart. Nous pouvons la supprimer en remplaçant cette (7.23) par la contrainte suivante :

$$q_{crop_{sr}^{t,c}} + q_{crop_{\bar{sr}}^{t,c}} - q_{crop^{t,c}} \leq \quad 0 \quad (\forall t \in T, \forall c \in \zeta) \tag{7.32}$$

Notons que les unités spatiales de présence d'ombre ou de racines ne changent pas durant les périodes de production P2 et P3. En conséquence, les cultures plantées sur deux saisons consécutives de la même période doivent garder les mêmes quantités de production (le même nombre des unités spatiales à l'ombre et aux racines) (7.29)-(7.30). En revanche, des cultures sur deux saisons, comme l'engrais, qui chevauchent deux périodes peuvent être affectées à plus d'unités spatiales de racines à la seconde période qu'à la première (7.31). Cela crée une configuration supplémentaire : des unités spatiales sans racines à la période P2 et avec racines à la période P3.

En conclusion, l'objectif du sous-problème est de définir les quantités de production dans le temps sans spécifier l'emplacement des cultures sur la parcelle. Ce qui rend la fonction objectif (initialement quadratique) linéaire. Il est facile de vérifier que l'équation (7.22) est équivalente à l'équation (7.6).

Sans les contraintes (7.28)-(7.31) traitant les cultures sur deux saisons, le sous-problème peut facilement être transformé en un ensemble de problèmes indépendants de flot maximal à coût minimal. Un problème pour chaque saison t, avec un flot maximal égal à à $l^2 - trees$ en respectant des informations sur les demandes ($minBalance$) et les capacités ($maxBalance$, $l^2 - trees$, $shades$, $roots_p$). La figure 7.13 présente un exemple de production pendant la saison d'été de la période P2 pour une parcelle de taille 10×10. Malheureusement, le problème est beaucoup plus difficile. Les cultures sur deux saisons ajoutent un ensemble de contraintes d'égalité/inégalité (7.28)-(7.31) pour préserver les mêmes quantités de production sur deux saisons de deux périodes différentes, tout en considérant l'évolution des racines des arbres (Figure 7.8).

Pour assurer la faisabilité du sous-problème, le positionnement des arbres (dans le problème maître) doit se faire de manière à laisser suffisamment d'unités spatiales pour l'allocation de cultures (dans le sous-problème) permettant de satisfaire les demandes de production (Figure 7.10). Nous ajoutons donc, au problème maître les contraintes suivantes :

$$trees \leq l^2 - \sum_{c \in \zeta} minBalance^{t,c} \qquad (\forall t \in T) \tag{7.33}$$

$$\sum_{c \in \zeta} minBalance^{t,c} \leq l^2 \leq \sum_{c \in \zeta} maxBalance^{t,c} \qquad (\forall t \in T) \tag{7.34}$$

Pour résumer, nous avons décomposé le problème initial en un problème maître restreint (PMR) et un sous-problème (SP) en ajoutant des variables supplémentaires ($trees, shades, roots_p$), entraînant une fonction objectif linéaire. Ces variables se présentent comme des termes linéaires dans le PMR et le SP (7.24)-(7.26). Nous pouvons donc reformuler le problème initial en combinant le PMR et le SP, i.e., (7.9)-(7.18) ;(7.24)-(7.33) avec la fonction objectif linéaire (7.22). Il s'agit donc d'un seul problème linéaire mixte en nombres entiers (Mixed Integer Programming (MIP)). Avec cette reformulation, nous pouvons résoudre le problème directement (séparation automatique du PMR et du SP) en utilisant l'approche de décomposition de Benders implémentée dans IBM ILOG CPLEX à partir de la version v12.7. Nous avons opté pour l'utilisation de CPLEX avec

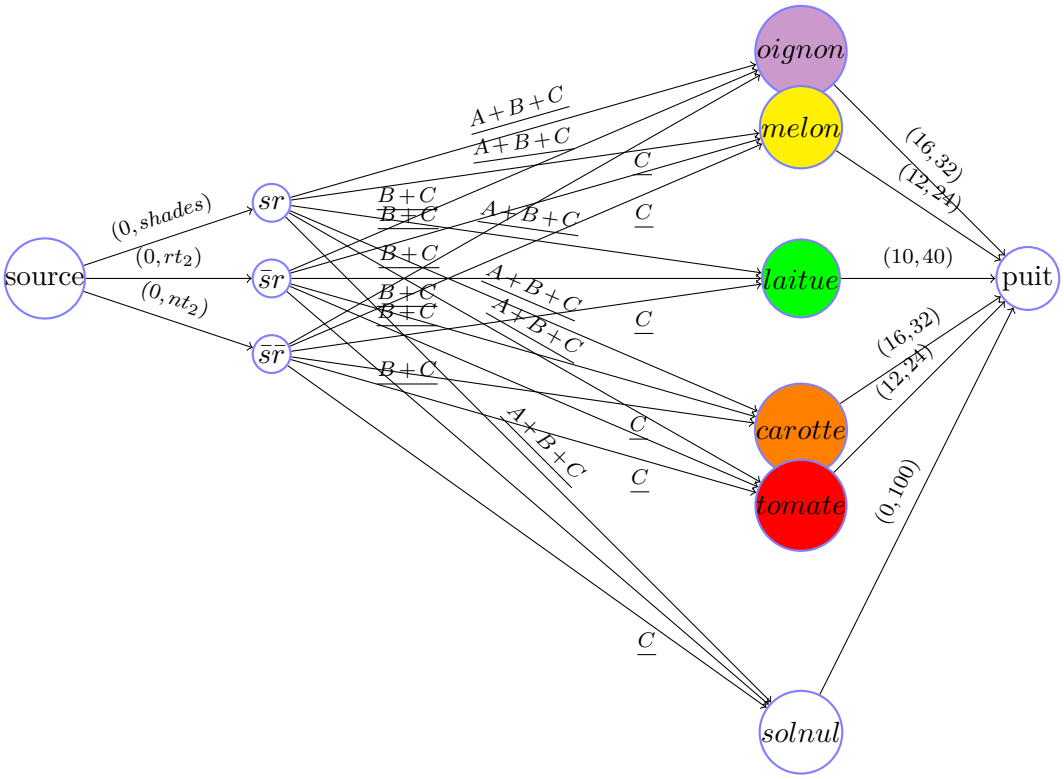


FIGURE 7.13 – Production en été à la période P2 pour une parcelle (grille) de taille 10x10, représentée comme un problème de flot maximal à coût-minimal. Les arcs au mileu représentent les coûts et celles entre parenthèses les demandes et capacités. Soit $rt_2 = roots_2 - shades$ and $nt_2 = 100 - trees - roots_2$.

l'approche de décomposition de Benders plutôt que d'utiliser notre version implémentée de la décomposition de Benders qui est moins performante que CPLEX.

Construction d'un plan d'allocation complet pour compléter le MIP (Algorithm 4)

Par rapport au modèle quadratique qui permet de définir les positions exactes des arbres $tree_{x,y}$ et des cultures $crop^{t,c}_{x,y}$, le modèle mixte permet de positionner uniquement les arbres $tree_{x,y}$ et par conséquent positionne l'ombre $shade_{x,y}$ et les racines $root_{x,y}$. Il permet également de définir les quantités des cultures selon les configurations possibles des unités spatiales ($q_{crop^{t,c}_{sr}}$, $q_{crop^{t,c}_{\bar{s}r}}$, $q_{crop^{t,c}_{\bar{s}\bar{r}}}$).

Pour construire un pan d'allocation complet, nous pouvons attribuer aléatoirement, à une culture c, $q_{crop^{t,c}_{sr}}$ (respectivement $q_{crop^{t,c}_{\bar{s}r}}$, $q_{crop^{t,c}_{\bar{s}\bar{r}}}$) unités spatiales parmi celles libres ayant ombre et racines (resp. racines seulement, ni ombre ni racines) à chaque pas de temps t. Par contre, attribuer une culture c qui dure deux saisons successives n'est pas si simple, car elle doit être attribuée à la même unité spatiale (x, y) au temps t et $t+1$, i.e., $crop^{t,c}_{x,y} = crop^{t+1,c}_{x,y}$.

Concernant l'engrais vert qui chevauche deux saisons t et $t+1$ sur deux périodes différentes, nous avons une procédure d'affectation spécifique. Dans le cas où nous avons $q_{crop^{t,engrais}_{\bar{s}r}} < q_{crop^{t+1,engrais}_{\bar{s}r}}$, nous devons alors trouver $q_{crop^{t+1,engrais}_{\bar{s}r}} - q_{crop^{t,engrais}_{\bar{s}r}}$ unités spatiales sans racines au temps t et avec racines au temps $t+1$ lors de l'allocation de

$q_{crop_{\bar{s}\bar{r}}}^{t,engrais}$. Cela a pour objectif de conserver une quantité totale constante de l'engrais, qui sera affectée aux mêmes unités spatiales au temps t et $t+1$. Les unités spatiales restantes $q_{crop_{\bar{s}\bar{r}}}^{t,engrais} - (q_{crop_{\bar{s}r}}^{t+1,engrais} - q_{crop_{\bar{s}r}}^{t,engrais})$ sont choisies arbitrairement parmi les unités spatiales sans racines.

Algorithm 4: Construction d'un plan d'allocation de cultures

Input: $tree_{x,y}, shade_{x,y}, root_{x,y}^{p}, q_{crop_{sr}}^{t,c}, q_{crop_{\bar{s}r}}^{t,c}, q_{crop_{\bar{s}\bar{r}}}^{t,c}$
Output: $crop_{x,y}^{t,c}$

1 **Procedure** `Allouer`$(q_{crop_{type}}^{t,c}, t, c, p, type, p', type')$
2 **if** $(t < \max(T) \wedge twoseason(c) \wedge f_c(T_p) = t)$ **then return** ;
4 **for** $y \leftarrow 1$ **to** l **do**
6   **for** $x \leftarrow 1$ **to** l **do**
7   **if** $(q_{crop_{type}}^{t,c} = 0)$ **then return** ;
8   **if** $(free_{x,y}^{t} \wedge \texttt{Type}(p,x,y) = type \wedge \texttt{Type}(p',x,y) = type')$ **then**
9     $free_{x,y}^{t} \leftarrow$ **false** ;
10     $q_{crop_{type}}^{t,c} \leftarrow q_{crop_{type}}^{t,c} - 1$;
11     $crop_{x,y}^{t,c} \leftarrow 1, \forall c' \in \zeta \setminus \{c\}, crop_{x,y}^{t,c'} \leftarrow 0$;
12   **if** $(twoseason(c) \wedge f_c(T_p) < t)$ **then**
13      $free_{x,y}^{t-1} \leftarrow$ **false** ;
14      $crop_{x,y}^{t-1,c} \leftarrow 1, \forall c' \in \zeta \setminus \{c\}, crop_{x,y}^{t-1,c'} \leftarrow 0$;
15   **end**
16   **end**
17   **end**
18 **end**
19 Tableau booléen $free_{x,y}^{t}$ pour identifier les unités spatiales disponibles pour les cultures;
20 $\forall x \in L, y \in L, t \in T$, **if** $(tree_{x,y} = 1)$ **then** $free_{x,y}^{t} \leftarrow$ **false else** $free_{x,y}^{t} \leftarrow$ **true**;
22 **for** $t \leftarrow \max(T)$ **to** 2 **do**
23 Ordre inverse des cultures entre les périodes P2 et P3;
25 **if** $(t \geq \min(T_3))$ **then** $\zeta \leftarrow \zeta 3$; $p \leftarrow 3$ **else** $\zeta \leftarrow \zeta 2$; $p \leftarrow 2$;
26 **foreach** $(c \in \zeta)$ **do**
27   `Allouer`$(q_{crop_{sr}}^{t,c}, t, c, p, sr, p, sr)$;
29   **if** $(c = engrais \wedge t < \max(T))$ **then**
30   `Allouer`$(q_{crop_{\bar{s}r}}^{t-1,c}, t, c, p, \bar{s}r, p-1, \bar{s}r)$;
31   `Allouer`$(q_{crop_{\bar{s}r}}^{t,c} - q_{crop_{\bar{s}r}}^{t-1,c}, t, c, p, \bar{s}r, p-1, \bar{s}\bar{r})$;
32   **end**
33   **else** `Allouer`$(q_{crop_{\bar{s}r}}^{t,c}, t, c, p, \bar{s}r, p, \bar{s}r)$;
34   `Allouer`$(q_{crop_{\bar{s}\bar{r}}}^{t,c}, t, c, p, \bar{s}\bar{r}, p, \bar{s}\bar{r})$;
35 **end**
36 **end**

Pour satisfaire ces contraintes complexes d'engrais entre deux périodes (7.31) puis les contraintes des cultures maraîchères sur deux saisons de la même période (7.28)-(7.30), nous avons choisi de construire un plan d'allocation avec un ordre de pas de temps décroissant, commençant en automne de la période P3 (voir Figure 7.14 et Algorithm 4, ligne 22). Concernant les dimensions spatiales, nous commençons l'attribution par le côté en haut à gauche puis nous cherchons les unités spatiales les plus proches (distance minimum par

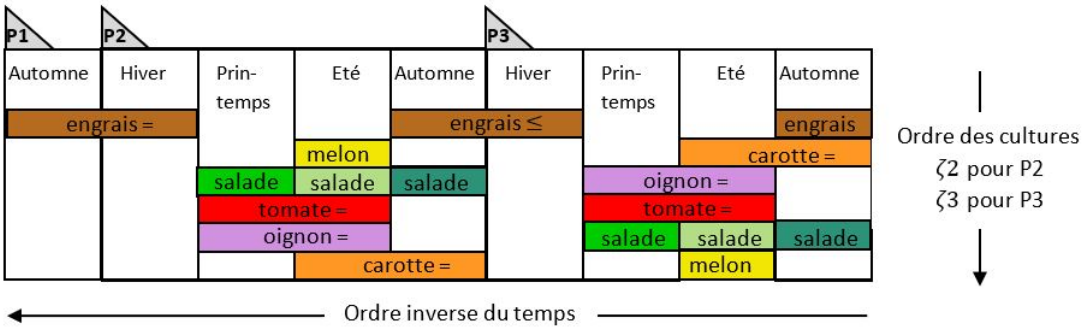


FIGURE 7.14 – Règles de construction d'un plan d'allocation complet. Ordre inverse des pas de temps à partir de la saison automne de la période P3 pour satisfaire les contraintes d'égalité / d'inégalité des cultures qui durent deux saisons. Ordre inverse des cultures entre les périodes P2 et P3 pour favoriser la rotation des cultures.

rapport à l'axe des y puis à l'axe des x. Ce qui permet de réduire la dispersion horizontale des cultures. De plus, nous attribuons les cultures selon un ordre inverse entre les périodes P2 et P3 afin de favoriser la rotation des cultures (voir Figure 7.14 et Algorithm 4, ligne 25).

7.3.3 Modèle linéaire en variables binaires

L'objectif de cette thèse est de concevoir des systèmes de verger-maraîcher de grande taille, une parcelle de surface de 1ha à 2ha que nous pouvons traduire par une grille de 100×100 unité spatiales. Comme évoqué dans le chapitre 1, les méthodes exactes deviennent inefficaces à mesure que la taille du problème devient importante. Une alternative est d'utiliser des méthodes approchées, permettant de trouver des solutions de bonne qualité en temps raisonnable. Pour cela, nous avons choisi l'heuristique de Wedelin (voir Partie II), une méthode conçue initialement pour résoudre des problèmes linéaires en variables 0-1 avec des contraintes d'égalité ayant des coefficients 0/1. Néanmoins, notre version implémentée de la méthode de Wedelin, solveur BARYONYX [Maqrot et al., 2018a], permet de résoudre des problèmes avec une structure plus générale : fonction objectif linéaire avec des variables binaires et des contraintes d'égalité/inégalité ayant des coefficients en $-1/0/1$ (voir chapitre 5).

Dans ce sens, nous proposons de reformuler le modèle quadratique du problème de verger-maraîcher en un modèle linéaire selon la structure imposée par notre version de l'algorithme de Wedelin. Pour cela, nous devons linéariser la fonction objectif quadratique (7.6) et reformuler les contraintes ayant des coefficients dans l'ensemble $\mathbb{N}$: les contraintes sur la définition de l'ombre d'un arbre (7.13)-(7.14) et les contraintes sur l'évolution de ses racines (7.16)-(7.17).

Linéarisation de la fonction objectif

Concernant la linéarisation de la fonction objectif, nous avons choisi la méthode classique [Fortet, 1959] (voir chapitre 1), qui consiste à remplacer chaque produit quadratique $(x_i \times x_j)$ par une variable additionnelle y_{ij} en introduisant de nouvelles contraintes (1.12)-(1.15). En appliquant cette méthode sur la fonction (7.6), nous remplaçons respectivement les produits $(shade_{x,y} \times crop^{t,c}_{x,y})$ et $(root^{p}_{x,y} \times crop^{t,c}_{x,y})$ par deux variables binaires supplé-

mentaires $S^{t,c}_{x,y}$ et $R^{t,c}_{x,y}$. Ce qui donne la fonction objectif linéaire suivante :

$$\min \sum_{\substack{p\in P, t\in T^p,\\ c\in\zeta, x,y\in L}} A^{t,c}\times S^{t,c}_{x,y} + B^{t,c}\times R^{t,c}_{x,y} + C^{t,c}\times crop^{t,c}_{x,y} \tag{7.35}$$

De plus, pour rester dans le cadre linéaire, nous introduisons les contraintes linéaires suivantes :

$(\forall t\in T, \forall c\in\zeta, x,y\in L^2, \forall p\in P)$

$$S^{t,c}_{x,y} - shade_{x,y} \leq 0 \tag{7.36}$$
$$S^{t,c}_{x,y} - crop^{t,c}_{x,y} \leq 0 \tag{7.37}$$
$$shade_{x,y} + crop^{t,c}_{x,y} - S^{t,c}_{x,y} \leq 1 \tag{7.38}$$

$$R^{t,c}_{x,y} - root^{p}_{x,y} \leq 0 \tag{7.39}$$
$$R^{t,c}_{x,y} - crop^{t,c}_{x,y} \leq 0 \tag{7.40}$$
$$root^{p}_{x,y} + crop^{t,c}_{x,y} - R^{t,c}_{x,y} \leq 1 \tag{7.41}$$

Reformulation des contraintes ayant des coefficients dans $\mathbb{N}$ en $\{-1,0,1\}$

Pour définir l'ombre au nord, est et ouest d'un arbre avec des contraintes ayant des coefficients en $-1/0/1$, nous proposons une reformulation explicite des contraintes (7.13)-(7.14) décrites dans le modèle quadratique.

$$tree_{x,y+1} - shade_{x,y} \leq 0 \quad (\forall x\in L, \forall y\in\{1,...,l-1\}) \tag{7.42}$$
$$tree_{x+1,y+1} - shade_{x,y} \leq 0 \quad (\forall x\in\{1,...,l-1\}, \forall y\in\{1,...,l-1\}) \tag{7.43}$$
$$tree_{x-1,y+1} - shade_{x,y} \leq 0 \quad (\forall x\in\{2,...,l\}, \forall y\in\{1,...,l-1\}) \tag{7.44}$$
$$tree_{x-1,y} - shade_{x,y} \leq 0 \quad (\forall x\in\{2,...,l\}, \forall y\in L) \tag{7.45}$$
$$tree_{x+1,y} - shade_{x,y} \leq 0 \quad (\forall x\in\{1,...,l-1\}, \forall y\in L) \tag{7.46}$$

$$shade_{x,y} - tree_{x+1,y} + tree_{x+1,y+1} + tree_{x-1,y} + tree_{x-1,y+1} + tree_{x,y+1} \leq 0 \quad (\forall x\in\{2...l-1\}, y\in\{1...l-1\}) \tag{7.47}$$

$$shade_{1,y} - tree_{2,y} + tree_{2,y+1} + tree_{1,y+1} \leq 0 \quad (\forall y\in\{1...l-1\}) \tag{7.48}$$
$$shade_{l,y} - tree_{l-1,y} + tree_{l_1,y+1} + tree_{l,y+1} \leq 0 \quad (\forall y\in\{1...l-1\}) \tag{7.49}$$
$$shade_{x,l} - tree_{x-1,l} + tree_{x+1,l} \leq 0 \quad (\forall x\in\{2...l-1\}) \tag{7.50}$$
$$shade_{1,l} - tree_{2,l} \leq 0 \tag{7.51}$$
$$shade_{l,l} - tree_{l-1,l} \leq 0 \tag{7.52}$$

Idem pour les contraintes (7.16)-(7.17) sur l'évolution des racines des arbres, nous présentons une reformulation explicite avec des coefficients en $-1/0/1$.

$(\forall p \in \{2,3\})$:

$$root^{p-1}_{x,y} - root^{p}_{x,y} \leq 0 \quad (\forall x,y \in L^2) \tag{7.53}$$
$$root^{p-1}_{x+1,y-1} - root^{p}_{x,y} \leq 0 \quad (\forall x \in \{1,...,l-1\}, \forall y \in \{2,...,l\}) \tag{7.54}$$
$$root^{p-1}_{x+1,y} - root^{p}_{x,y} \leq 0 \quad (\forall x \in \{1,...,l-1\}, \forall y \in L) \tag{7.55}$$
$$root^{p-1}_{x+1,y+1} - root^{p}_{x,y} \leq 0 \quad (\forall x,y \in \{1,...,l-1\}^2) \tag{7.56}$$
$$root^{p-1}_{x,y+1} - root^{p}_{x,y} \leq 0 \quad (\forall x \in L, \forall y \in \{1,...,l-1\}) \tag{7.57}$$
$$root^{p-1}_{x-1,y+1} - root^{p}_{x,y} \leq 0 \quad (\forall x \in \{2,...,l\}, \forall y \in \{1,...,l-1\} \tag{7.58}$$
$$root^{p-1}_{x-1,y} - root^{p}_{x,y} \leq 0 \quad (\forall x \in \{2,...,l\}, \forall y \in L) \tag{7.59}$$
$$root^{p-1}_{x-1,y-1} - root^{p}_{x,y} \leq 0 \quad (\forall x,y \in \{2,...,l\}) \tag{7.60}$$
$$root^{p-1}_{x,y-1} - root^{p}_{x,y} \leq 0 \quad (\forall x \in L, \forall y \in \{2,...,l\}) \tag{7.61}$$

$$root^{p}_{x,y} - \sum_{i,j \in \{-1,0,1\}} root^{p-1}_{x+i,y+j} \leq 0 \quad (\forall x,y \in \{2,...,l-1\}^2) \tag{7.62}$$
$$root^{p}_{x,y} - \sum_{i \in \{-1,0,1\}, j \in \{0,1\}} root^{p-1}_{x+i,1+j} \leq 0 \quad (\forall x \in \{2,...,l-1\}) \tag{7.63}$$
$$root^{p}_{x,l} - \sum_{i \in \{-1,0,1\}, j \in \{-1,0\}} root^{p-1}_{x+i,l+j} \leq 0 \quad (\forall x \in \{2,...,l-1\}) \tag{7.64}$$
$$root^{p}_{1,y} - \sum_{i \in \{0,1\}, j \in \{-1,0,1\}} root^{p-1}_{1+i,y+j} \leq 0 \quad (\forall y \in \{2,...,l-1\}) \tag{7.65}$$
$$root^{p}_{l,y} - \sum_{i \in \{-1,0\}, j \in \{-1,0,1\}} root^{p-1}_{l+i,y+j} \leq 0 \quad (\forall y \in \{2,...,l-1\}) \tag{7.66}$$

$$root^{p}_{1,1} - \sum_{i,j \in \{0,1\}} root^{p-1}_{i+1,j+1} \leq 0 \tag{7.67}$$
$$root^{p}_{l,1} - \sum_{i \in \{-1,0\}, j \in \{0,1\}} root^{p-1}_{l+i,1+j} \leq 0 \tag{7.68}$$
$$root^{p}_{l,l} - \sum_{i,j \in \{-1,0\}} root^{p-1}_{l+i,l+j} \leq 0 \tag{7.69}$$
$$root^{p}_{1,l} - \sum_{i \in \{0,1\}, j \in \{-1,0\}} root^{p-1}_{1+i,l+j} \leq 0 \tag{7.70}$$

En guise de comparaison, nous avons reformulé ce modèle linéaire en variables 0-1 pour se rapprocher au maximum du problème de Set Partitioning étendu (voir chapitre 2). Pour cela, nous avons reformulé les contraintes de manière à avoir que des coefficients en 0/1.

7.4 Résultats et discussions

7.4.1 Choix des instances du problème de conception de verger-maraîcher

Pour évaluer l'impact des interactions aériennes et souterraines, nous considérons trois scénarios :

- *Above* qui donne une grande importance aux interactions aériennes, à savoir le positionnement des cultures à l'ombre ou au soleil. Dans ce scénario, nous multiplions les coûts associés aux interactions aériennes (valeurs A, tableau 7.4) par 10.
- *Below* qui accorde une importance significative aux interactions souterraines, notamment la compétition ou le partage des ressources en eau. Dans ce scénario, nous multiplions les coûts associés aux interactions souterraines (valeurs B, tableau 7.4) par 10.
- *Equilibrate* est un intermédiaire entre les deux scénarios *Above* et *Below*, il correspond à un compromis entre les différentes interactions sans en privilégier aucun type (valeurs du tableau 7.4).

Chaque scénario est étudié dans quatre modèles mathématiques :

- Quadratique en variables binaires (BQP)
- Linéaire en variables mixtes (MIP)
- Linéaire en variables binaires (01LP)
- Linéaire en variables binaires sous forme du problème de Set Partitioning étendu (01LP-SPP)

Pour fixer le choix des objectifs et des contraintes à modéliser, nous avons considéré dans un premier temps trois modèles relâchés de la reformulation quadratique : 1) modèle *Basic* où nous optimisons l'objectif principal (7.6) sur les interactions sans les contraintes (7.20) sur la rotation des cultures, 2) modèle *Rotate* qui est équivalent au modèle *Basic* avec les contraintes sur la rotation des cultures, et 3) modèle *Dispersion* où nous optimisons l'objectif principal sur les interactions et l'objectif secondaire (7.7) sur la dispersion des cultures.

Nous avons présenté les résultats de ces modèles dans [Maqrot et al., 2016], que nous avons calculé par le solveur IBM ILOG CPLEX V12.6.32 avec les options par défaut. Le calcul a été effectué en utilisant 32 processeurs d'un CPU AMD OPTERON 6176 à 2,3 GHz, avec 378 Go de RAM, sous Linux 2.6.32. Mais malgré ce nombre important de processeurs, la résolution était lente. Le tableau 7.5 présente le temps (ecoulé) effectué par CPLEX pour trouver les optima des différents modèles : représentation optimisée d'un système de verger-maraîcher discrétisé en une grille de taille 10×10.

Equilibrate			*Above*			*Below*		
Basic	*Rotation*	*Dispersion*	*Basic*	*Rotation*	*Dispersion*	*Basic*	*Rotation*	*Dispersion*
31 :57	71 :40	238 :55*	00 :10	00 :23	46 :03	00 :32	04 :32	71 :40

TABLE 7.5 – Temps hh :mm (écoulé) de résolution effectué par IBM ILOG CPLEX V12.6.32 pour trouver les optima d'un problème de conception d'un verger-maraîcher de taille 10×10 (* : gap de 3%).

D'après le tableau 7.5, la résolution du scénario *Equilibrate* est beaucoup plus lente que celle des scanérios *Above* et *Below*, vu que le premier doit équilibrer l'optimisation des interactions aériennes et souterraines, contrairement aux deux derniers qui donnent de l'importance à seul type d'interactions. Concernant les modèles, nous observons une différence significative (plusieurs heures de calcul) entre le modèle *Dispersion* et les modèles *Basic* et *Rotate*. Par exemple, le modèle *Dispersion* du scénario *Equilibrate* ne parvient pas à trouver une solution optimale après 238 heures de calcul (%3 à l'optimum).

Pour cela, nous avons décidé de garder que le modèle *Basic* (Optimisation des interactions sans optimiser la dispersion et sans considérer les contraintes sur la rotation des cultures) pour les différentes formulations mathématiques. De plus, nous avons choisi des modèles avec des tailles différentes : des parcelles de tailles (10×10), (15×15), (20×20), (30×30) et (50×50). Au total nous avons 15 instances avec 4 formulations mathématiques différentes, comme montré dans la figure 7.15.

Problème de conception de verger-maraîcher

15 instances (3 scénarios $\times$ 5 tailles) avec 4 formulations mathématiques différentes :

- 3 scénarios : *Equilibate*, *Above* et *Below*.
- 5 tailles différentes de parcelles : (10×10), (15×15), (20×20), (30×30) et (50×50).
- 4 modèles matématiques : BQP, MIP, 01LP et 01LP-SPP.

FIGURE 7.15 – Instances étudiées pour la résolution du problème de conception de verger-maraîcher.

7.4.2 Protocole expérimental

Pour la résolution du problème de conception de verger-maraîcher, nous utilisons trois logiciels d'optimisation : un logiciel open source, BARYONYX V0.4 (voir chapitre 5) et deux logiciels commerciaux, CPLEX V12.8 et LOCALSOLVER V8.5 (voir chapitre 1). Le paramétrage utilisé est celui par défaut pour les deux derniers. Or, pour BARYONYX, nous appliquons le protocole de réglage automatique des paramètres décrit dans le chapitre 5. Ce dernier consiste à faire une analyse de sensibilité (méthode de Morris) pour classer les paramètres selon leur influence sur les résultats, fixer les paramètres avec effet négligeable, et régler les paramètres les plus importants en utilisant la méthode de GENOUD. Nous avons effectué l'ensemble des expérimentations (réglage des paramètres et résolution) sur un cluster de 32 processeurs Intel Xeon CPU E5-2683 v4 de 2,1 GHz et de 4 Go de RAM par processeur.

Réglage des paramètres de BARYONYX pour les formulations 01LP

Rappelons que le solveur BARYONYX ne permet de résoudre que des problèmes linéaires en variables 0-1. Il s'agit donc des deux formulations mathématiques (PL01) et (PL01-SPP). Pour résoudre les 15 instances (voir Figure 7.15) de chaque formulation, nous avons choisi de régler les paramètres de BARYONYX sur deux instances de taille 10×10, 01LP_Equilibrate_10 et 01LP-SPP_Equilibrate_10. Un réglage pour chaque instance.

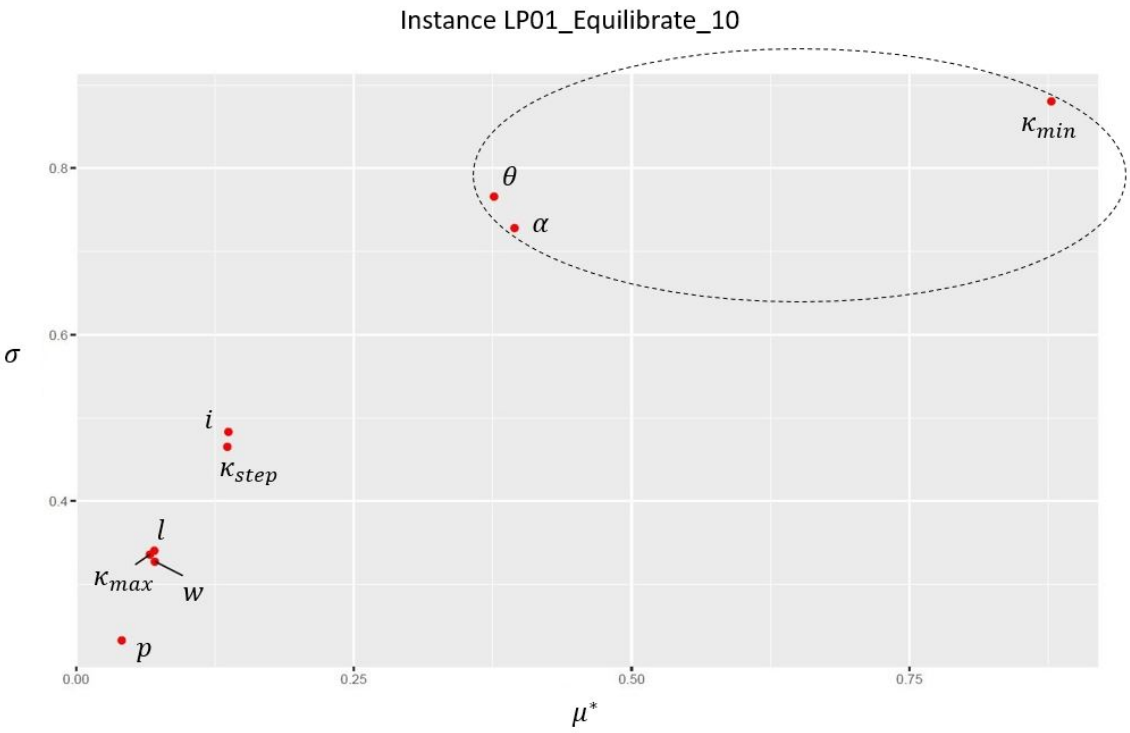


FIGURE 7.16 – Graphique des indices de Morris de l'instance 01LP_Equilibrate_10 .

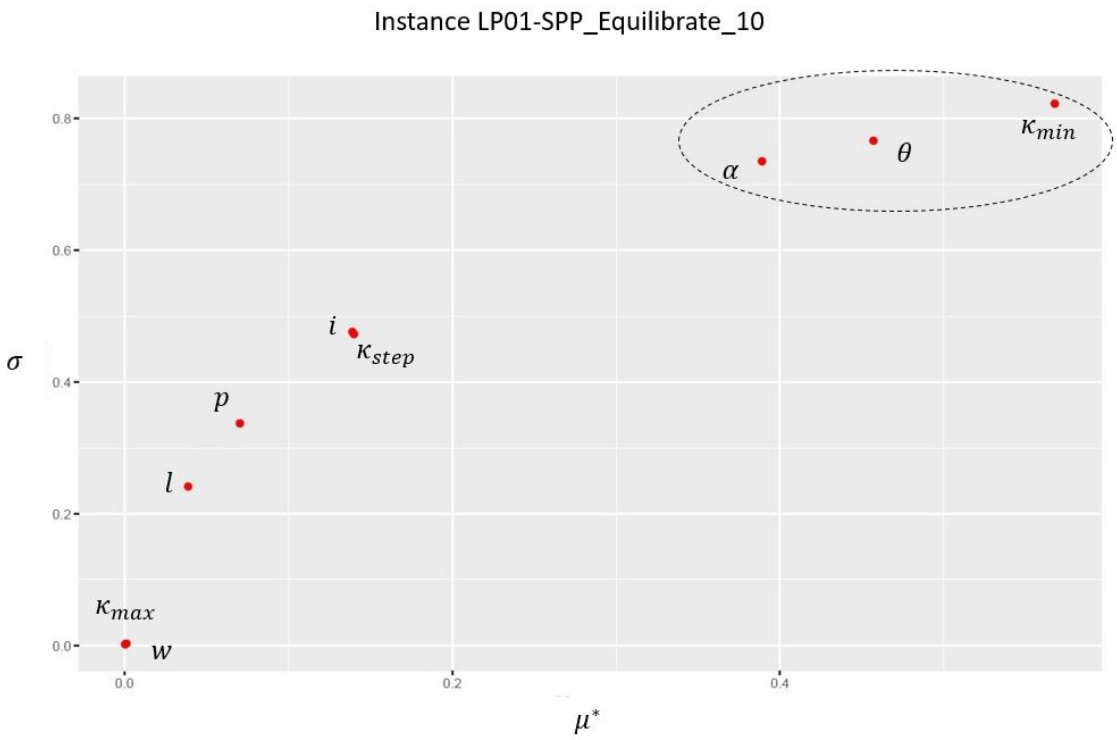


FIGURE 7.17 – Graphique des indices de Morris de l'instance 01LP-SPP_Equilibrate_10.

Instances	Paramètres		
	κ_{min}	θ	α
01LP_Equilibrate_10	1,92 $\times 10^{-1}$	9,34 $\times 10^{-1}$	1,37
01LP-SPP_Equilibrate_10	4,84 $\times 10^{-2}$	1,87 $\times 10^{-1}$	1,67

TABLE 7.6 – Meilleurs jeux de paramètres trouvés par GENOUD pour les deux instances 01LP_Equilibrate_10 et 01LP-SPP_Equilibrate_10.

Nous avons effectué une analyse de sensibilité en utilisant la méthode de Morris avec les mêmes paramètres décrits dans le chapitre 5. Une discrétisation de la gamme de variation de (N=9) paramètres de BARYONYX (voir Table 5.7, chapitre 5) en $Q = 10$ niveaux, avec une longueur de pas égale à $\delta = 5$. Un plan d'échantillonnage de $r = 50$ trajectoires aléatoires, passant chacun par $(N + 1 = 10)$ noeuds. L'analyse est donc effectuée avec $50 \times (9 + 1) = 500$ jeux de paramètres candidats pour explorer le comportement du solveur. Pour chaque jeu de paramètre, BARYONYX est exécuté pendant 60 secondes en mode optimisation en utilisant 30 processeurs. A la fin de l'analyse, Morris calcule les indices de sensibilité : la moyenne des effets élémentaires μ^* et leur variance σ sur les r trajectoires.

D'apès les figures 7.16 et 7.17 qui synthétisent respectivement les résultats de Morris pour les deux instances 01LP_Equilibrate_10 et 01LP-SPP_Equilibrate_10, nous remarquons que les trois paramètres κ_{min}, θ et α ont des valeurs très élevées de (μ^*, σ) en comparaison avec les autres paramètres. Donc, la performance de BARYONYX dans la résolution de ces instances est sensible à la variation de ces paramètres, changer donc leurs valeurs peut modifier complètement les résultats. Pour régler les paramètres avec effet négligeable, nous avons utilisé le réglage de paramètres de WEDELIN GOOD (voir Table 5.9, chapitre 5).

Nous avons effectué le réglage automatique des paramètres importants (κ_{min}, θ et α) en utilisant la méthode de GEOUD (voir chapitre 5) avec un nombre maximal de générations égal à 10 et des populations de 100 individus. Nous avons donc approximativement 1000 individus et par conséquent 1000 jeux de paramètres. GENOUD appelle BARYONYX pour chaque jeu de paramètre pour une résolution qui dure 60 secondes en utilisant 30 processeurs. Au bout de 10 générations, GENOUD affiche le jeu de paramètres optimisé. Le tableau 7.6 présente les meilleurs valeurs trouvées pour les deux instances 01LP_Equilibrate_10 et 01LP-SPP_Equilibrate_10. Le résultat de ce réglage automatique sera généralisé pour résoudre la totalité des instances (01LP) et (01LP-SPP).

Analyse des résultats

Le tableau 7.7 récapitule les résultats de résolution des 15 instances du problème de conception de verger-maraîcher. Chaque instance est reformulée sous forme de quatre modèles mathématiques (BQP, MIP, 01LP et 01LP-SPP), qui représentent le même objectif de minimisation avec un nombre de variables et de contraintes différent (voir Table 7.8).

Instances	LocalSolver V8.5	baryonyx V0.4		cplex V12.8			
	BQP	01LP	01LP-SPP	BQP	MIP	01LP	01LP-SPP
Equilibrate_10	∞	19.840	3.950	**3.540**	**3.540**	3.620	3.790
Equilibrate_15	∞	45.980	8.910	8.210	**7.740**	8.550	8.870
Equilibrate_20	∞	82.360	16.050	20.370	**13.470**	15.420	16.550
Equilibrate_30	∞	191.540	36.150	77.540	**30.600**	77.220	∞
Equilibrate_50	∞	539.080	∞	**207.280**	211.500	229.330	∞
Above_10	∞	52.260	4.460	**-2.670**	**-2.670**	-2.610	-2.310
Above_15	∞	124.490	11.150	-5.460	**-6.050**	-5.630	-2.920
Above_20	∞	227.810	21.190	1.440	**-10.660**	-8.910	-7.960
Above_30	∞	526.970	∞	175.130	**-23.160**	191.770	∞
Above_50	∞	1.475.380	∞	**443.170**	463.500	544.150	∞
Below_10	∞	160.430	38.550	**31.720**	**31.720**	**31.720**	33.430
Below_15	∞	372.560	87.700	70.410	**68.590**	70.870	84.830
Below_20	∞	682.520	156.330	161.240	**118.500**	134.700	152.620
Below_30	∞	1.583.880	356.300	674.870	**261.990**	672.230	∞
Below_50	∞	4.401.390	**1.097.620**	1.836.910	1.863.000	1.818.630	∞

Table 7.7 – Résultats de résolution parallèle (30 processeurs pendant 600 secondes) du problème de conception de verger-maraîcher en utilisant 3 logiciels d'optimisation : LocalSolver, baryonyx et cplex (gras : meilleure solution trouvée).

Instance	Modèles mathématiques			
Equilibrate_50	BQP	MIP	01LP	01LP-SPP
Nb. variables binaires	582.147	12.500	687.500	2.047.500
Nb. variables réelles	0	276	0	0
Nb. contraintes	244.396	22.826	976.125	1.976.073

TABLE 7.8 – Taille de l'instance Equilibrate_50 dans les différents modèles mathématiques.

Instance	LOCALSOLVER V7.0	CPLEX V12.7	
	BQP	BQP	MIP
Equilibrate_50	118.850	349.050	**112.490**

TABLE 7.9 – Résultats de résolution de l'instance Equilibrate_50 pendant 3600 secondes en utilisant 10 processeurs [Maqrot et al., 2018b] (gras : meilleure solution trouvée).

Rappelons que le modèle MIP permet de définir l'emplacement des arbres dans la parcelle et la quantité des cultures à planter dans les zones de présence ou d'absence de l'ombre et des racines des arbres, sans préciser l'emplacement de plantation des cultures maraîchères comme dans les modèles BQP, 01LP et 01LP-SPP. Ce qui explique la différence significative en nombre de variables et de contraintes entre MIP et les autres modèles mathématiques (voir Table 7.8). Cela joue un rôle important lors de la résolution. Avec le modèle MIP, CPLEX trouve (en utilisant la décomposition de Benders) les meilleures solutions puisqu'il s'agit d'un modèle simplifié.

Concernant le modèle quadratique BQP résolu par LOCALSOLVER et CPLEX, nous constatons que le premier ne trouve aucune solution pour la totalité des instances. Tandis que le second résout toutes les instances, avec de meilleures solutions pour les instances de grande taille comme l'instance Equilibrate_50 (solution de 207.280). Nous expliquons cela par le fait que LOCALSOLVER a besoin de plus de temps (plus que 600 secondes) pour résoudre ces instances. Dans [Maqrot et al., 2018b], nous avons présenté les résultats de résolution (pendant 3600 secondes en utilisant 10 processeurs) de l'instance Equilibrate_50 avec la version V7.0 de LOCALSOLVER et V12.7 de CPLEX. Les résultats présentés dans le tableau 7.9 montre que LocalSolver fournit une meilleure solution (118.850) que celle trouvée par CPLEX pour le modèle BQP (349.050) et proche de celle du modèle MIP(112.490).

Les modèles linéaires 01LP et 01LP-SPP sont les plus volumineux en termes du nombre de variables et de contraintes. Avec le modèle 01LP, BARYONYX et CPLEX résolvent toutes les instances avec une qualité moins bonne pour BARYONYX. Toutefois avec le modèle 01LP-SPP, les deux solveurs ne parviennent pas à résoudre toutes les instances. BARYONYX résout 12/15 instances contre 9/12 par CPLEX. De plus, BARYONYX trouve la meilleure solution pour l'instance Below_50.

Ces résultats montrent que BARYONYX est compétitif avec les autres solveurs (CPLEX et LOCALSOLVER) dans la résolution des problèmes de type Set Partitioning, notamment les problèmes avec de grande taille. Cependant, il est moins performant pour fournir des solutions de bonne qualité pour des problèmes avec des formes plus générales que le Set Partitioning.

Instance	BARYONYX V0.4	CPLEX V12.8	
	01LP-SPP	BQP	MIP
Equilibrate_50	104.890	207.280	**84.960**

TABLE 7.10 – Résultats de résolution de l'instance Equilibrate_50 pendant 3600 secondes en utilisant 30 processeurs (gras : meilleure solution trouvée).

7.4.3 Discussion et visualisation des solutions

Pour évaluer la capacité du modèle à prendre en compte les différentes interactions, nous avons considéré trois scénarios : *Above*, *Below* et *Equilibrate*. Pour chaque scénario testé, nous analysons les solutions obtenues en termes du 1) nombre d'arbres, 2) mixité des cultures maraîchères, et 3) leur positionnement par rapport aux zones d'interaction délimitées autour des arbres. Pour cette analyse, nous considérons les meilleures solutions trouvées pour une parcelle de taille 10×10. Il s'agit des solutions obtenues par CPLEX pour le modèle BQP (voir Table 7.7). La figure 7.18 représente, pour chaque scénario, la position des arbres et la répartitions des cultures aux périodes P2 et P3. Pour la période P1, nous considérons que les arbres n'ont pas d'influence hors de leurs propres unités spatiales.

En regardant la figure 7.18, nous remarquons que les trois représentations respectent les contraintes modélisées, comme l'espace minimal entre les arbres (selon Figure 7.2) et les dates et durées de plantation des cultures maraîchères (selon Table 7.1). La grande différence entre les trois scénarios consiste dans le nombre et la position des arbres (18 arbres pour *Above*, 21 pour *Below* et 24 pour *Equilibrate*). Dans le scénario *Above*, les interactions liées à l'ombre sont plus coûteuses que les autres. Pour avoir donc moins d'unités spatiales à l'ombre, la solution retenue comporte moins d'arbres (18 contre 24 pour *Equilibrate)*. Cela permet de réduire le coût lié aux interactions aériennes de coût négatif (valeurs A, Table 7.4). Nous remarquons la présence de quatre arbres espacées. Cela a pour objectif d'assurer un nombre suffisant d'unités spatiales à l'ombre pour les cultures qui en bénéficient comme la salade. De manière similaire, dans le scénario *Below*, les interactions liées à la présence de racines sont les plus coûteuses. Les arbres sont donc regroupés pour avoir moins d'unités spatiales liées aux interactions souterraines de coût négatif. Par exemple, les cultures qui sont en compétition pour l'eau avec les racines des arbres sont attribuées avec des quantités minimales (selon Figure 7.10). Par exemple, 12 unités pour la tomate (contre 24 dans le scénario *Above*). Concernant le scénario *Equilibrate* qui correspond à un compromis entre les interactions aériennes et souterraines, la topologie des unités spatiales est équilibrée entre à l'ombre, au soleil avec ou sans racines. Il s'agit d'un intermédiaire entre les solutions des deux autres scénarios, des arbres tassés regroupés et d'autres espacés.

Pour comparer la représentation de différentes solutions (approchées) d'une même instance de grande taille. Nous avons relancé la résolution de l'instance Equilibrate_50 (modèle 01LP-SPP) par BARYONYX V0.4 [1] et CPLEX V12.8 (modèles BQP et MIP) pendant 3600 secondes en utilisant 30 processeurs. Table 7.10, qui présente les résultats de résolution, montre que BARYONYX (qui n'avait pas de solution à 600 secondes d'exécution) trouve une solution meilleure (104.890) que celle obtenue par CPLEX pour le modèle BQP (207.280), mais de moins bonne qualité par rapport à celle du modèle MIP (84.960). Figure 7.19 présente une visualisation spatiale et temporelle de ces trois solutions.

D'après la figure 7.19, nous constatons une grande différence entre les trois représentations. BARYONYX trouve une solution avec 307 arbres qui occupent 307 unités spatiales sur

[1] avec le même jeu de paramètres utilisé avant (calcul des résultats du tableau 7.7)

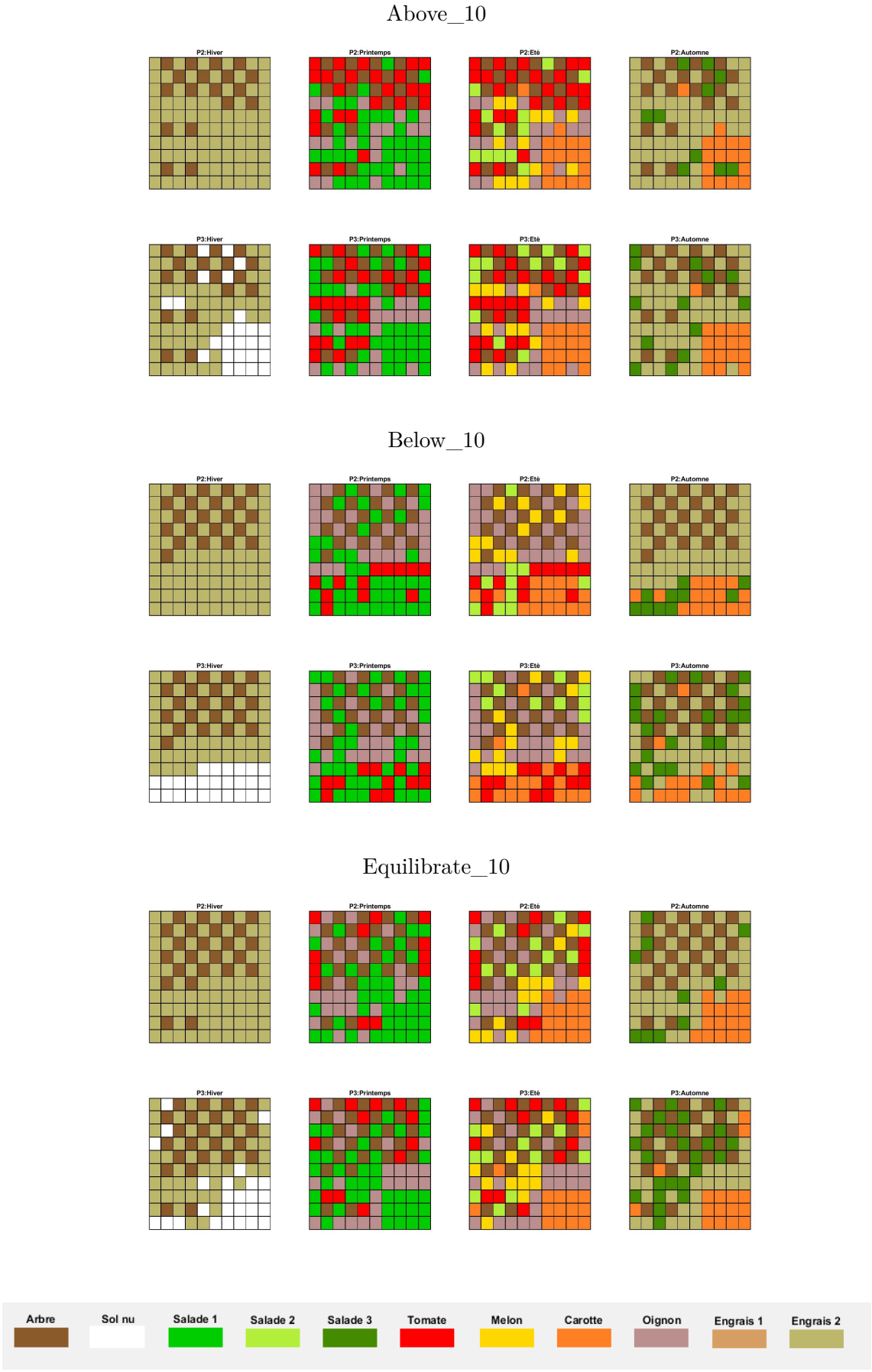
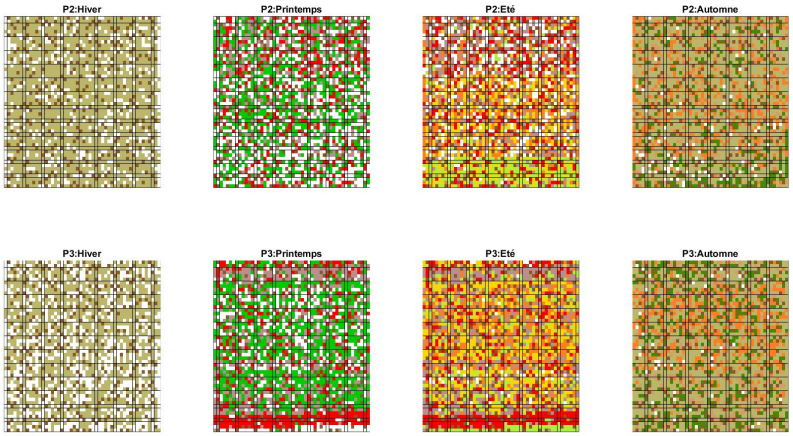
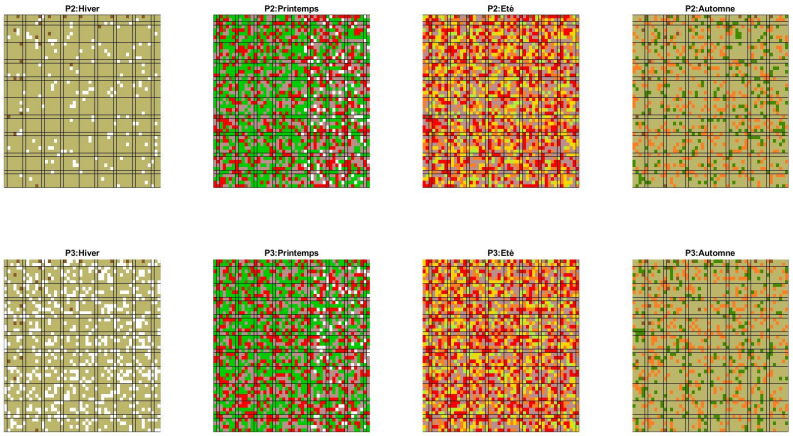


FIGURE 7.18 – Représentations d'une parcelle de taille 10×10 (solution de CPLEX, modèle BQP) aux périodes P2 et P3 pour les trois scénarios *Above*, *Below* et *Equilibrate*.

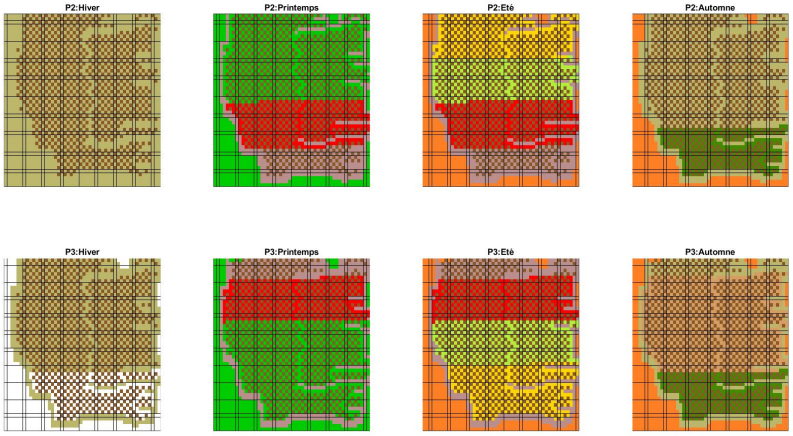
Résultat de BARYONYX, modèle 01LP-SPP (Solution : 104.890)



Résultat de CPLEX, modèle BQP (Solution : 207.280)



Résultat de CPLEX, modèle MIP (Solution : 84.960)



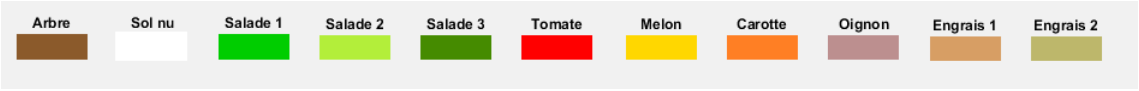


FIGURE 7.19 – Représentations des solutions trouvées par BARYONYX et CPLEX pour la conception d'une parcelle de verger-maraîcher de taille 50×50 pour le scénario *Equilibrate* aux périodes P2 et P3.

2500, i.e., 12% de la parcelle. Cela génère un nombre important d'interactions entre ces arbres et les cultures maraichères de proximité. La solution représentée respecte la totalité des contraintes, comme l'espace minimale entre les arbres, la mixité des cultures (quantité minimale de chaque espèce) et la date et durée de plantation notamment les cultures sur deux saisons qui doivent occuper les mêmes unités spatiales). Nous remarquons que le sol nu est beaucoup présent dans cette solution. Sa présence a un coût très pénalisant par rapport aux autres cultures. Cela justifie la valeur de cette solution approchée (coût élevé des interactions).

Concernant les résultats de CPLEX, la solution du modèle BQP est une solution approchée qui respecte la totalité des contraintes mais avec un coût des interactions très élevé. Dans cette solution, nous avons uniquement 23 arbres qui représente 0,9% de l'exploitation agricole. Cela correspond à une première solution où le solveur cherche à respecter les contraintes avant d'optimiser les interactions entre arbres et cultures. Pour le modèle MIP, la solution comprend 800 arbres qui représente 32% de la parcelle, i.e, 68% de la parcelle est attribué aux cultures maraîchères. Nous remarquons que l'exploitation agricole est couverte des cultures maraîchères sur toutes les saisons sauf en hiver de la période P2 où la présence du sol nu est obligatoire, car aucune culture n'est plantée en hiver après la salade d'automne et la carotte.

Contrairement aux deux premières représentations 01LP-SPP et BQP où les cultures sont dispersées sur la parcelle, la solution du modèle MIP regroupe les cultures par espèce, en essayant d'appliquer la rotation de cultures pour préserver les propriétés du sol, i.e., ne pas planter la même culture au même endroit pendant deux périodes successives. Cette différence vient du fait que les modèles 01LP et BQP ont pour objectif de définir les positions des arbres et des cultures (présentées dans la figure 7.19). Tandis que le modèle MIP permet de définir uniquement les positions des arbres et les quantités de cultures à planter dans chaque type d'unité spatiale. En utilisant ces informations, nous avons appliqué une méthode gloutonne (Algorithm 4) pour représenter l'allocation des cultures du modèle MIP (Figure 7.19).

7.5 Conclusion du chapitre

Contrairement aux approches existantes sur l'allocation de cultures annuelles dans les systèmes agroforestiers, nous avons proposé différents modèles de conception d'un système de verger-maraîcher, associant des cultures saisonnières et des cultures pérennes. Nous avons explicitement décrit les modèles conceptuels et mathématiques, en détaillant l'optimisation des effets positifs et négatifs des interactions potentielles générées par la combinaison des arbres et des cultures.

Malgré la taille importante (nombre de variables et de contraintes) du problème de conception de verger-maraîcher notamment pour des parcelle de grande taille (50×50), nous avons obtenu des solutions approchées de bonne qualité en temps raisonnable. Les solutions de CPLEX avec la méthode de Décomposition de Benders appliquée sur le modèle mixte MIP sont meilleures que celles trouvées pour le modèle quadratique BQP et celles obtenues par BARYONYX pour le modèle linéaire 01LP. Les résultats de ce dernier sont sensibles aux choix des valeurs de ses paramètres. Un réglage automatique semble intéressant à condition de bien choisir l'intervalle de variation des paramètres.

Conclusion Générale et perspectives

Cette thèse s'inscrit dans le domaine de la recherche opérationnelle avec une application en agronomie. L'objectif est de développer une méthode approchée générique capable de résoudre des problèmes de grandes tailles, en fournissant des solutions de bonne qualité en temps raisonnable. Des problèmes comme le problème de conception d'un système agroforestier associant des arbres fruitiers et des cultures maraîchères.

Bilan

Dans cette thèse, nous avons adopté une méthode approchée nommée algorithme de Wedelin ou In-the-middle. Cette méthode basée sur la relaxation Lagrangienne est conçue pour résoudre des problèmes linéaires en variables 0-1 de type problème Set Partitioning. Nous avons étudié une version généralisée pour résoudre des problèmes linéaires avec des formes plus générales, que nous avons implémenté en C++. Le solveur nommé BARYONYX a deux types de paramètres : catégoriques et numériques. Nous avons fixé les premiers suite à des expérimentations. Quant aux seconds, nous avons procédé à une analyse de sensibilité pour classer les paramètres selon leur influence sur la variabilité du résultat de résolution. Ceci a pour objectif de réduire le temps de réglage automatique en se focalisant uniquement sur les paramètres les plus importants. Ce réglage automatique a été effectué par familles de problèmes, optimisé sur un sous-ensemble d'apprentissage et généralisé sur la totalité des problèmes. Pour évaluer la performance de BARYONYX, nous l'avons comparé à d'autres logiciels d'optimisation, à savoir CPLEX, LOCALSOLVER et TOULBAR2. Les résultats montrent que BARYONYX atteint des performances meilleures que les autres solveurs pour la résolution du problème des N reines pondérées et le problème de Set Partitioning. Cependant, il trouve des difficultés pour fournir des solutions de bonne qualité pour d'autres types de problèmes comme le problème de Set Covering.

Comme application de la thèse, nous nous sommes intéressés à la modélisation et la résolution du problème de conception des systèmes de verger-maraîcher, que nous avons défini comme un problème d'allocation des arbres et des cultures dans les dimensions spatiales et temporelles. Ce système repose sur un ensemble complexe des interactions arbres-cultures modifiant l'utilisation des ressources en lumière, eau et nutriments. Concevoir donc un tel système nécessite d'optimiser l'utilisation de ces ressources en maximisant les interactions positives et en minimisant celles négatives. Nous avons proposé différentes formulations mathématiques de ce problème pour comparer les résultats de résolution de différentes méthodes : 1) un modèle quadratique en variables 0-1 résolu par CPLEX et LOCALSOLVER, 2) un modèle linéaire avec des variables mixtes résolu par Cplex (Méthode de Décomposition de Benders) et un modèle linéaire en variables 0-1 résolu par BARYONYX (Heuristique de Wedelin). Ces modèles représentent des problèmes de grande taille qui peuvent atteindre 2 millions de variables et 2 millions de contraintes pour la conception d'un verger-maraîcher

discrétisé en 50×50 unités spatiales. Les résultats des expérimentations ont montré que la méthode de décomposition de Benders appliquée via CPLEX est la meilleure pour résoudre le problème de verger-maraîcher. BARYONYX fournit des solutions approchées en temps raisonnable, dont la qualité dépend fortement du choix de ses paramètres.

Perspectives

Ce travail de thèse offre des perspectives variées :

- Le comportement de BARYONYX pourrait être davantage généralisé dans un plus grand nombre de situations en particulier avec l'extension à une fonction objectif quadratique, en s'inspirant des problèmes de somme de fonctions de coûts [Werner, 2005, Wedelin, 2013]. De plus, comme la méthode se base sur la relaxation Lagrangienne, les valeurs des multiplicateurs Lagrangiens peuvent être exploitées pour calculer la borne duale. Cela permet d'estimer la qualité des solutions obtenues (distance à l'optimum), une fonctionnalité disponible dans différents solveurs, comme CPLEX et LOCALSOLVER.

- Concernant le réglage des paramètres de BARYONYX, il serait intéressant d'auto-régler les valeurs de ses paramètres pendant la résolution et à les contrôler dynamiquement au fur et à mesure de la recherche de solution. Cela permet d'avoir un réglage adapté au problème à résoudre, permettant de fournir des solutions de bonne qualité en temps raisonnable.

- Il serait également intéressant de combiner plusieurs solveurs, notamment BARYONYX avec CPLEX et TOULBAR2, outil d'optimisation développé dans le laboratoire. Une piste de travail serait d'étendre les techniques de reformulation développées dans les réseaux de fonctions de coûts pour prendre en compte les contraintes linéaires à l'instar de [Wedelin, 2013]. Par ailleurs d'autres méta-heuristiques pour sortir des minima locaux, tel que l'algorithme Guided Local Search [Hutter et al., 2005], pourraient venir renforcer l'approche actuellement développée.

- Concernant le modèle de conception de verger-maraîcher, nous pouvons améliorer le côté écologique en optimisant les interactions entre les cultures et les insectes nuisibles [Batish et al., 2007]. Concernant le côté économique, nous pouvons intégrer dans le modèle la répartition du temps de travail (récolte, plantation, etc.) qui sera considérée lors de l'allocation des cultures. Nous pouvons également améliorer le critère d'optimisation pour prendre en considération le rendement de production (critère économique). L'idée est de faire le positionnement en fonction des rendements attendus des arbres et des cultures maraîchères, où les interactions arbres-cultures contribuant positivement ou négativement dans la production. Nous quantifions cette contribution en terme de pourcentage, par exemple +10% de rendement si la salade est positionnée à l'ombre pendant la saison d'été.

- Pour soutenir les agronomes dans leurs stratégies de conception, nous proposons de simplifier la procédure de modélisation et de résolution du problème de verger-maraîcher, en automatisant le déroulement des étapes de cette procédure. Une application avec une interface où l'agriculteur pourrait remplir les informations nécessaires (comme le choix des cultures et le rendement de production attendu) sans avoir besoin de comprendre le modèle mathématique ou les étapes de résolution semble intéressante.

Annexe

TABLE 11 – Résultats de résolution des instances SPP (en gras : ensemble d'apprentissage et meilleures solutions, "-" : aucune solution trouvée)

Instances	Columns	Rows	BARYONYX				cplex	LocalSolver	Beasley[Beasley, 1990b]	Umetani[Umetani, 2017]
			baryonyx	WEDELIN FAST	WEDELIN GOOD	baryonyx^{rgn}				
sppaa01	7564	614	56255	57439	57396	56178	**56137**	-	**56137**	**56137**
sppaa02	3875	362	**30494**	**30494**	**30494**	**30494**	**30494**	**30494**	**30494**	**30494**
sppaa03	6776	561	49695	49876	50213	**49649**	**49649**	-	**49649**	**49649**
sppaa04	6139	343	26518	26636	27007	26485	**26374**	44591	**26374**	**26374**
sppaa05	6269	536	54012	54368	54307	**53839**	**53839**	-	**53839**	**53839**
sppaa06	5963	510	27093	27069	27176	**27040**	**27040**	**27040**	**27040**	**27040**
sppnw01	50069	135	**114852**	**114852**	115536	**114852**	**114852**	**114852**	**114852**	N/A
sppnw02	85258	145	**105444**	105684	107934	**105444**	**105444**	202986	**105444**	N/A
sppnw03	38956	53	**24492**	26517	27363	**24492**	**24492**	36402	**24492**	N/A
sppnw04	46189	35	**16862**	19036	18004	**16862**	**16862**	27610	**16862**	N/A
sppnw05	202593	62	132950	133180	133376	132920	**132878**	142110	**132878**	N/A
sppnw06	5956	38	**7810**	8592	9392	**7810**	**7810**	**7810**	**7810**	N/A
sppnw07	3105	34	**5476**	6200	6200	**5476**	**5476**	**5476**	**5476**	N/A
sppnw08	352	21	**35894**	**35894**	**35894**	**35894**	**35894**	**35894**	**35894**	N/A
sppnw09	2301	38	**67760**	**67760**	68402	**67760**	**67760**	**67760**	**67760**	N/A
sppnw10	655	21	**68271**	68286	68286	**68271**	**68271**	**68271**	**68271**	N/A
sppnw11	6482	34	**116256**	116856	117651	116259	**116256**	**116256**	**116256**	N/A
sppnw12	451	25	**14118**	14124	14190	**14118**	**14118**	**14118**	**14118**	N/A
sppnw13	10903	50	**50146**	50158	51934	**50146**	**50146**	**50146**	**50146**	N/A
sppnw14	95172	70	61846	62046	62452	61850	**61844**	66158	**61844**	N/A
sppnw15	441	29	**67743**	67746	67746	**67743**	**67743**	**67743**	**67743**	N/A
sppnw16	138947	135	**1181590**	1181638	**1181590**	**1181590**	**1181590**	**1181590**	**1181590**	N/A
sppnw17	78173	54	**11115**	13698	13083	**11115**	**11115**	27546	**11115**	N/A
sppnw18	8438	108	340162	341518	343538	**340160**	**340160**	**340160**	**340160**	N/A
sppnw19	2134	32	**10898**	11888	11888	**10898**	**10898**	**10898**	**10898**	N/A
sppnw20	536	22	**16812**	17556	17556	**16812**	**16812**	**16812**	**16812**	N/A
sppnw21	426	25	**7408**	**7408**	7436	**7408**	**7408**	**7408**	**7408**	N/A
sppnw22	521	23	**6984**	7144	7244	**6984**	**6984**	**6984**	**6984**	N/A
sppnw23	473	18	**12534**	**12534**	**12534**	**12534**	**12534**	**12534**	**12534**	N/A
sppnw24	926	19	**6314**	6568	6568	**6314**	**6314**	**6314**	**6314**	N/A
sppnw25	844	20	**5960**	**5960**	6286	**5960**	**5960**	**5960**	**5960**	N/A
sppnw26	516	23	**6796**	**6796**	**6796**	**6796**	**6796**	**6796**	**6796**	N/A
sppnw27	817	22	**9933**	**9933**	11091	**9933**	**9933**	**9933**	**9933**	N/A
sppnw28	599	18	**8298**	**8298**	**8298**	**8298**	**8298**	**8298**	**8298**	N/A
sppnw29	2034	18	**4274**	4392	4666	**4274**	**4274**	**4274**	**4274**	N/A
sppnw30	1884	26	**3942**	4450	4294	**3942**	**3942**	**3942**	**3942**	N/A
sppnw31	1823	26	**8038**	8484	8106	**8038**	**8038**	**8038**	**8038**	N/A
sppnw32	227	15	**14877**	**14877**	15120	**14877**	**14877**	**14877**	**14877**	N/A
sppnw33	2415	23	**6678**	6724	6724	**6678**	**6678**	**6678**	**6678**	N/A
sppnw34	736	20	**10488**	**10488**	10701	**10488**	**10488**	**10488**	**10488**	N/A
sppnw35	1403	23	**7216**	7316	**7216**	**7216**	**7216**	**7216**	**7216**	N/A
sppnw36	1408	20	**7314**	7824	8272	**7314**	**7314**	**7314**	**7314**	N/A
sppnw37	639	19	**10068**	10233	**10068**	**10068**	**10068**	**10068**	**10068**	N/A
sppnw38	908	21	**5558**	5688	5592	**5558**	**5558**	**5558**	**5558**	N/A
sppnw39	565	25	**10080**	**10080**	10758	**10080**	**10080**	**10080**	**10080**	N/A
sppnw40	336	19	**10809**	10848	10896	**10809**	**10809**	**10809**	**10809**	N/A
sppnw41	177	17	**11307**	**11307**	11766	**11307**	**11307**	**11307**	**11307**	N/A
sppnw42	893	22	**7656**	7684	7684	**7656**	**7656**	**7656**	**7656**	N/A
sppnw43	982	17	**8904**	9012	9012	**8904**	**8904**	**8904**	**8904**	N/A
sppkl01	5957	47	1091	**1086**	**1086**	**1086**	**1086**	**1086**	**1086**	N/A
sppkl02	16542	69	221	**219**	**219**	**219**	**219**	**219**	**219**	N/A
sppus01	351018	86	10051	10176	10101	10038	**10036**	-	**10036**	**10036**
sppus02	8433	45	5977	6015	6316	**5965**	**5965**	**5965**	**5965**	**5965**
sppus03	23207	50	**5338**	5544	5544	**5338**	**5338**	**5338**	**5338**	**5338**
sppus04	4422	98	**17854**	**17854**	**17854**	17862	**17854**	**17854**	**17854**	**17854**

TABLE 12 – Résultats de résolution des instances Telebus (en gras : ensemble d'apprentissage et meilleures solutions, "-" : aucune solution trouvée)

Instances	Columns	Rows	BARYONYX				cplex	LocalSolver	Borndörfer[Borndörfer, 1998]	Umetani[Umetani, 2017]	Bastert [Bastert et al., 2010]	
			baryonyx	WedelinFast	WedelinGood	baryonyx^{rgn}					WedelinFast	WedelinGood
t0415	3172	870	5346798	**5339422**	5349625	5346798	**5339422**	-	5590096	5572626	-	5593065
t0416	3152	974	**6088264**	**6088264**	**6088264**	**6088264**	6093843	-	6130271	**6088264**	-	6095736
t0417	3623	897	**5952247**	5955570	**5952247**	**5952247**	5955570	-	6043157	6024760	-	6034848
t0418	3921	999	**6442906**	**6442906**	**6442906**	6447571	6442908	-	6550898	6446019	-	6470351
t0419	3168	904	5908538	5908538	5908538	5908538	**5907874**	-	5916956	5910913	5910913	5910913
t0420	1847	562	**4153696**	**4153696**	**4153696**	**4153696**	4276444	-	4276444	-	-	-
t0421	1656	557	**4290809**	**4290809**	**4290809**	**4290809**	4340929	-	4354411	**4290809**	-	-
t1716	11952	467	173161	220247	171481	**146871**	169429	330789	161636	165972	181488	306453
t1717	16428	551	192123	196599	178810	**165881**	190181	397845	184692	180757	199024	178948
t1718	16310	523	175772	206736	164318	**153166**	169806	-	162992	174338	197136	175517
t1719	15846	556	195103	223397	189027	**167523**	187717	474975	187677	184354	213194	200215
t1720	16195	538	183560	217548	176376	**154881**	175643	-	172752	181868	183673	188880
t1721	9043	357	135647	134842	128337	**117602**	123567	296830	127424	130047	154951	138819
t1722	6581	338	128574	136612	118822	**113213**	121025	-	122472	114508	132369	125586
v0415	4012	598	2431155	2432482	2432378	2436076	**2429415**	2429420	**2429415**	2429568	2434157	2450287
v0416	10723	812	2732616	2730395	2731091	2739534	**2725602**	2726930	**2725602**	2726156	2731356	2737662
v0417	55232	715	2619345	2613689	2615766	2624641	**2611518**	2617710	**2611518**	**2611518**	2614749	2623710
v0418	4411	742	2847522	2847305	2847075	2852267	**2845425**	2845790	**2845425**	**2845425**	2865326	2864652
v0419	7356	650	2595906	2596932	2591893	2598319	**2590326**	2590330	**2590326**	**2590326**	2597016	2602660
v0420	2350	417	1697940	1697803	1698449	1700336	**1696889**	1696890	**1696889**	**1696889**	1714155	1720504
v0421	823	286	1853951	1854929	1855285	1855170	**1853951**	1853950	**1853951**	**1853951**	1857033	-
v1616	52775	1230	1009279	1016815	1021077	1019797	**1006460**	1066280	**1006460**	1007402	1022969	1029986
v1617	85300	1409	1108113	1119114	1121974	1126315	**1102586**	1216370	**1102586**	1103651	1129989	-
v1618	90805	1396	1159231	1170565	1175678	1173355	**1153871**	1237340	1154458	1155986	1179617	-
v1619	85565	1424	1162669	1169928	1176576	1178951	**1156338**	1248790	**1156338**	1157537	1184010	1189709
v1620	89367	1365	1145978	1154107	1159002	1158319	**1140604**	1233050	**1140604**	1141976	1159246	-
v1621	16606	807	830329	829829	832269	830175	**825563**	838979	**825563**	825605	835860	844014
v1622	10990	736	794530	799151	801465	799438	**793445**	811134	**793445**	793708	801475	-

TABLE 13 – Résultats de résolution des instances CSPLib02 (en gras : ensemble d'apprentissage et meilleures solutions, "-" : aucune solution trouvée)

Instances	Columns	Rows	BARYONYX				cplex	LocalSolver	Curtis [Curtis et al., 2000]
			baryonyx	WEDELIN FAST	WEDELIN GOOD	baryonyx^{rgn}			
c1a	7543	186	29	**26**	**26**	**26**	**26**	30	34
c1	3829	186	29	**26**	**26**	**26**	**26**	**26**	33
c2	14771	205	33	**29**	**29**	**29**	**29**	**29**	N/A
r1	2503	53	12	**11**	**11**	**11**	**11**	**11**	15
r1a	4273	53	12	**11**	**11**	**11**	**11**	**11**	16
r2	3001	54	**14**	**14**	**14**	**14**	**14**	**14**	17
r3	19091	160	**16**	**16**	**16**	**16**	**16**	-	20
r4	2484	203	27	**25**	**25**	**25**	**25**	**25**	31
r5a	14764	242	31	**28**	**28**	**28**	**28**	**28**	N/A
r5	2202	242	30	**29**	**29**	**29**	**29**	-	N/A
t1	77	24	**7**	**7**	**7**	**7**	**7**	**7**	**7**
t2	3015	125	20	**19**	**19**	**19**	**19**	**19**	N/A

TABLE 14 – Résultats de résolution des instances VCS (en utilisant le réglage de paramètres de SPP pour baryonyx^{rgn}). "-" : aucune solution trouvée)

Instances	Columns	Rows	BARYONYX (1800s)				cplex (1800s)	cplex (36000s)
			baryonyx	WEDELIN FAST	WEDELIN GOOD	baryonyx^{rgn}		
vcs1200	127014	1180	849	-	-	819	844	**749**
vcs1600	537412	1576	1449	-	-	1451	-	**1378**

TABLE 15 – Résultats de résolution des instances nqueens (en gras : ensemble d'apprentissage et meilleures solutions, "-" : aucune solution trouvée)

Instances	Columns	Rows	BARYONYX				cplex	LocalSolver
			baryonyx	WEDELIN FAST	WEDELIN GOOD	baryonyx^{rgn}		
8queens	64	42	**18**	**18**	**18**	**18**	**18**	**18**
10queens	100	54	**25**	**25**	**25**	**25**	**25**	**25**
20queens	400	114	**63**	**63**	**63**	**63**	**63**	**63**
30queens	900	174	108	103	103	103	**101**	144
40queens	1600	234	165	161	164	155	**154**	284
50queens	2500	294	197	190	194	180	**176**	340
500queens	250000	2994	-	8343	6856	**5864**	96859	24509
1000queens	1000000	5994	-	-	20013	**17667**	384140	191532

Bibliographie

[Aarts et al., 2003] Aarts, E., Aarts, E. H., and Lenstra, J. K. (2003). Local search in combinatorial optimization. Princeton University Press. *Cité page 42*

[Adenso-Diaz and Laguna, 2006] Adenso-Diaz, B. and Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. Operations research, 54(1) :99–114. *Cité page 68*

[Agarwal et al., 1989] Agarwal, Y., Mathur, K., and Salkin, H. M. (1989). A set-partitioning-based exact algorithm for the vehicle routing problem. Networks, 19(7) :731–749. *Cité page 52*

[Akplogan, 2013] Akplogan, M. (2013). Approche modulaire pour la planification continue : application à la conduite des systèmes de culture. PhD Thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier. *2 citations pages 14 and 139*

[Akplogan et al., 2013] Akplogan, M., De Givry, S., Metivier, J.-P., Quesnel, G., Joannon, A., and Garcia, F. (2013). Solving the crop allocation problem using hard and soft constraints. RAIRO-Operations Research, 47(2) :151–172. *Cité page 141*

[Alefragis et al., 2000] Alefragis, P., Sanders, P., Takkula, T., and Wedelin, D. (2000). Parallel integer optimization for crew scheduling. Annals of Operations Research, 99(1-4) :141–166. *Cité page 76*

[Alfandari et al., 2011] Alfandari, L., Lemalade, J.-L., Nagih, A., and Plateau, G. (2011). A MIP flow model for crop-rotation planning in a context of forest sustainable development. Annals of operations research, 190(1) :149–164. *Cité page 140*

[Alfandari et al., 2015] Alfandari, L., Plateau, A., and Schepler, X. (2015). A branch-and-price-and-cut approach for sustainable crop rotation planning. European Journal of Operational Research, 241(3) :872–879. *Cité page 140*

[Argouarc'H, 2005] Argouarc'H, J. (2005). Fiches technico-économiques des principaux légumes - Culture de plein champ et sous abri. Technical report, CFPPA Rennes - Le Rheu (35). *Cité page 137*

[Atkinson, 1983] Atkinson, D. (1983). The growth, activity and distribution of the fruit tree root system. Plant and Soil, 71 :23–35. *2 citations pages 134 and 148*

[Avella et al., 2004] Avella, P., Boccia, M., and Sforza, A. (2004). Solving a fuel delivery problem by heuristic and exact approaches. European Journal of Operational Research, 152(1) :170–179. *Cité page 52*

[Avella et al., 2006] Avella, P., D'Auria, B., and Salerno, S. (2006). A LP-based heuristic for a time-constrained routing problem. European Journal of Operational Research, 173(1) :120–124. *2 citations pages 48 and 52*

[Balaprakash et al., 2007] Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the F-Race algorithm : Sampling design and iterative refinement. In International workshop on hybrid metaheuristics, pages 108–122. Springer. *2 citations pages 67 and 70*

[Balas and Padberg, 1976] Balas, E. and Padberg, M. W. (1976). Set partitioning : A survey. SIAM review, 18(4) :710–760. *Cité page 50*

[Baldacci et al., 2008] Baldacci, R., Christofides, N., and Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. Mathematical Programming, 115(2) :351–385. *Cité page 52*

[Barbulescu et al., 2000] Barbulescu, L., Watson, J.-P., and Whitley, L. D. (2000). Dynamic representations and escaping local optima : Improving genetic algorithms and local search. AAAI/IAAI, 2000 :879–884. *Cité page 56*

[Barnhart et al., 1998] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price : Column generation for solving huge integer programs. Operations research, 46(3) :316–329. *Cité page 40*

[Bartz-Beielstein, 2010] Bartz-Beielstein, T. (2010). SPOT : an R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. Technical report, Package for R, a free software environment for statistical computing and graphics. *Cité page 67*

[Bartz-Beielstein et al., 2005] Bartz-Beielstein, T., Lasarczyk, C. W., and Preuß, M. (2005). Sequential parameter optimization. In 2005 IEEE congress on evolutionary computation, volume 1, pages 773–780. IEEE. *Cité page 67*

[baryonyx library, 2017] baryonyx library (2017). n-queens. 2017 problèmes des 10-reines pondérées. https://github.com/quesnel/baryonyx/tree/master/lib/test/n-queens. Accessed : 2019-02-12. *2 citations pages 17 and 94*

[Bastert et al., 2010] Bastert, O., Hummel, B., and de Vries, S. (2010). A generalized Wedelin heuristic for integer programming. INFORMS Journal on Computing, 22(1) :93–107. *16 citations pages 18, 30, 53, 76, 84, 85, 86, 87, 88, 89, 92, 95, 102, 116, 117, and 179*

[Batish et al., 2007] Batish, D., Kohli, R., Jose, S., and Singh, H. (2007). Ecological Basis of Agroforestry. CRC Press, crc press/taylor&francis edition. *3 citations pages 130, 131, and 176*

[Beasley, 1990a] Beasley, J. E. (1990a). OR-Library 1990 a collection of test data sets for a variety of operations research (or) problems. http://people.brunel.ac.uk/~mastjjb/jeb/info.html. Accessed : 2019-03-04. *6 citations pages 17, 18, 97, 101, 104, and 111*

[Beasley, 1990b] Beasley, J. E. (1990b). Or-library : distributing test problems by electronic mail. The Journal of the Operational Research Society, 41 :1069–1072. *7 citations pages 18, 53, 111, 114, 123, 124, and 178*

[Beasley and Chu, 1996] Beasley, J. E. and Chu, P. C. (1996). A genetic algorithm for the set covering problem. European journal of operational research, 94(2) :392–404. *Cité page 53*

[Benders, 1962] Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. Numerische mathematik, 4(1) :238–252. *Cité page 40*

[Benoist et al., 2011] Benoist, T., Estellon, B., Gardi, F., Megel, R., and Nouioua, K. (2011). Localsolver 1. x : a black-box local-search solver for 0-1 programming. 4OR : A Quarterly Journal of Operations Research, 9(3) :299–316. *Cité page 44*

[Berzsenyi et al., 2000] Berzsenyi, Z., Győrffy, B., and Lap, D. (2000). Effect of crop rotation and fertilisation on maize and wheat yields and yield stability in a long-term experiment. European Journal of Agronomy, 13 :225–244. *Cité page 138*

[Birattari et al., 2002] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, pages 11–18. Morgan Kaufmann Publishers Inc. *Cité page 67*

[Birattari et al., 2010] Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-Race and iterated F-Race : An overview. Springer. *Cité page 70*

[Blot and Hoos, 2016] Blot, A. and Hoos, H. (2016). MOParamILS : une plateforme multi-objectif pour la configuration automatique d'algorithmes. In ROADEF2016 : 17ème Conférence ROADEF de la société Française de Recherche Opérationnelle et d'Aide à la Décision. *Cité page 69*

[Boller et al., 1997] Boller, E., Malavolta, C., and Jörg, E. (1997). Guidelines for Integrated Production of Arable Crops in Europe : IOBC Technical Guideline III, volume 20. International Organization For Biological and Integrated Control. *Cité page 138*

[Borndörfer, 1998] Borndörfer, R. (1998). Aspects of set packing, partitioning, and covering. *11 citations pages 17, 18, 52, 53, 97, 101, 104, 111, 122, 123, and 179*

[Bozinovski and Bozinovski, 2004] Bozinovski, A. and Bozinovski, S. (2004). n-queens pattern generation : An insight into space complexity of a backtracking algorithm. In ACM International Conference Proceeding Series ; Proceedings of the 2004 International Symposium on Information and Communication Technologies, pages 281–286. *Cité page 51*

[Bramel and Simchi-Levi, 1997] Bramel, J. and Simchi-Levi, D. (1997). On the effectiveness of set covering formulations for the vehicle routing problem with time windows. Operations Research, 45(2) :295–301. *Cité page 52*

[Brulard, 2018] Brulard, N. (2018). Outils d'aide à la conception de systèmes de production maraîchers urbains optimisés pour la vente en circuits courts et de proximité. PhD thesis, Grenoble Alpes. *Cité page 141*

[Brulard et al., 2016] Brulard, N., Catusse, N., and Cung, V.-D. (2016). Intégration de l'ordonnancement de tâches avec setups séquence-dépendants dans le dimensionnement stratégique d'une exploitation agricole. *Cité page 140*

[Bruno et al., 2001] Bruno, J. F., Stachowicz, J. J., and Mark Bertness, D. (2001). Allelopathic effects of some volatile substances from the tomato plant. Journal of Crop Protection, 4 :313–321. *Cité page 131*

[Bruno et al., 2003] Bruno, J. F., Stachowicz, J. J., and Mark Bertness, D. (2003). Inclusion of facilitation into ecological theory. TRENDS in Ecology and Evolution, 18 :15–29. *Cité page 130*

[Brusco et al., 1999] Brusco, M. J., Jacobs, L. W., and Thompson, G. M. (1999). A morphing procedure to supplement a simulated annealing heuristic for cost-andcoverage-correlated set-covering problems. Annals of Operations Research, 86 :611–627. *Cité page 53*

[Burgess et al., 1998] Burgess, S. S., Adams, M. A., Turner, N. C., and Ong, C. K. (1998). The redistribution of soil water by tree root systems. Oecologia, 115 :306–311. *Cité page 134*

[Campolongo et al., 2007] Campolongo, F., Cariboni, J., and Saltelli, A. (2007). An effective screening design for sensitivity analysis of large models. Environmental modelling & software, 22(10) :1509–1518. *2 citations pages 13 and 61*

[Cannell et al., 1996] Cannell, M., Van Noordwijk, M., and Ong, C. (1996). The central agroforestry hypothesis : the trees must acquire resources that the crop would not otherwise acquire. Agroforestry systems, 34(1) :27–31. *Cité page 134*

[Castellazzi et al., 2008] Castellazzi, M. S., Wood, G. A., Burgess, P. J., Morris, J., Conrad, K. F., and Perry, J. N. (2008). A systematic representation of crop rotations. Agricultural Systems, 97(1) :26–33. *3 citations pages 14, 138, and 139*

[CFLib, 2018] CFLib (2018). nqueens. 2019 générateur de problèmes des n-reines pondérées. https://forgemia.inra.fr/thomas.schiex/cost-function-library/tree/master/random/wqueens. Accessed : 2019-02-06. *6 citations pages 17, 18, 97, 101, 104, and 110*

[Chambre d'agriculture, 2016] Chambre d'agriculture, T.-e.-G. (2016). Chiffres repères pour différentes activités agricoles développées sur le Tarn et Garonne. Fiches technico-économiques en Arboriculture (année 2014-2015). Accessed : 2019-07-08. *Cité page 137*

[Chetty and Adewumi, 2014] Chetty, S. and Adewumi, A. O. (2014). Comparison Study of Swarm Intelligence Techniques for the Annual Crop Planning Problem. IEEE Transactions on Evolutionary Computation, 18(2) :258–268. *Cité page 140*

[Chu and Beasley, 1998] Chu, P. C. and Beasley, J. E. (1998). Constraint handling in genetic algorithms : the set partitioning problem. Journal of Heuristics, 4(4) :323–357. *Cité page 53*

[Coello, 2005] Coello, C. A. C. (2005). An introduction to evolutionary algorithms and their applications. In International Symposium and School on Advancex Distributed Systems, pages 425–442. Springer. *2 citations pages 42 and 43*

[Coulon et al., 2000] Coulon, F., Dupraz, C., Liagre, F., Pointereau, P., and des Espaces Naturels, S.-D. (2000). Etude des pratiques agroforestières associant des arbres fruitiers de haute tige à des cultures et pâtures. Rapport au ministère de l'environnement, Solagro. *Cité page 128*

[Coy et al., 2001] Coy, S. P., Golden, B. L., Runger, G. C., and Wasil, E. A. (2001). Using experimental design to find effective parameter settings for heuristics. Journal of Heuristics, 7(1) :77–97. *Cité page 68*

[CPLEX, 2015] CPLEX, I. I. (2015). User's Manual for CPLEX, 2016. Version 12 Release. CPLEX division, page 564. *Cité page 41*

[Crescenzi and Kann, 1997] Crescenzi, P. and Kann, V. (1997). Approximation on the web : A compendium of NP optimization problems. In International Workshop on Randomization and Approximation Techniques in Computer Science, pages 111–118. Springer. *2 citations pages 48 and 49*

[Cukier et al., 1975] Cukier, R. I., Schaibly, J. H., and Shuler, K. E. (1975). Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. III. Analysis of the approximations. The Journal of Chemical Physics, 63(3) :1140–1149. *Cité page 62*

[Curtis et al., 2000] Curtis, S. D., Smith, B. M., and Wren, A. (2000). Constructing driver schedules using iterative repai. Research report series. *3 citations pages 122, 123, and 180*

[Cygan, 2013] Cygan, M. (2013). Improved approximation for 3-dimensional matching via bounded pathwidth local search. In Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on, pages 509–518. IEEE. *Cité page 48*

[Dantzig and Wolfe, 1960] Dantzig, G. B. and Wolfe, P. (1960). Decomposition principle for linear programs. Operations research, 8(1) :101–111. *Cité page 40*

[Darby-Dowman and Mitra, 1983] Darby-Dowman, K. and Mitra, G. (1983). An extension of set partitioning with application to scheduling problems. Brunel University Mathematics Technical Papers collection ;. *2 citations pages 50 and 51*

[Davis and Norman, 1988] Davis, J. and Norman, J. (1988). Effects of shelter on plant water use. Agriculture, Ecosystems and Environment, 393 :393–402. *Cité page 133*

[Delorme et al., 2004] Delorme, X., Gandibleux, X., and Rodriguez, J. (2004). GRASP for set packing problems. European Journal of Operational Research, 153(3) :564–580. *Cité page 53*

[Desaulniers et al., 1997] Desaulniers, G., Desrosiers, J., Dumas, Y., Marc, S., Rioux, B., Solomon, M. M., and Soumis, F. (1997). Crew pairing at air france. European journal of operational research, 97(2) :245–259. *2 citations pages 50 and 52*

[Desrosiers et al., 1988] Desrosiers, J., Sauvé, M., and Soumis, F. (1988). Lagrangian relaxation methods for solving the minimum fleet size multiple traveling salesman problem with time windows. Management Science, 34(8) :1005–1022. *Cité page 38*

[Detlefsen and Jensen, 2007] Detlefsen, N. K. and Jensen, A. L. (2007). Modelling optimal crop sequences using network flows. Agricultural Systems, 94(2) :566–572. *Cité page 140*

[Diaz-Perez, 2013] Diaz-Perez, J. C. (2013). Bell pepper (capsicum annum l.) crop as affected by shade level : Microenvironment, plant growth, leaf gas exchange, and leaf mineral nutrient concentration. HORTSCIENCE, 48 :175–182. *Cité page 132*

[Dogliotti et al., 2005] Dogliotti, S., van Ittersum, M. K., and Rossing, W. A. H. (2005). A method for exploring sustainable development options at farm scale : a case study for vegetable farms in South Uruguay. Agricultural Systems, 86(1) :29–51. *2 citations pages 139 and 140*

[Doucet et al., 1999] Doucet, C., Weaver, S. E., Hamill, A. S., and Zhang, J. (1999). Separating the effects of crop rotation from weed management on weed density and diversity. Weed science, 47 :729–735. *Cité page 137*

[Dupraz and Liagre, 2006] Dupraz, C. and Liagre, F. (2006). L'agroforesterie : une voie de diversification écologique de l'agriculture européenne ? Cahier d'étude DEMETER – Économie et Stratégies agricoles. *Cité page 128*

[Dupraz and Liagre, 2011] Dupraz, C. and Liagre, F. (2011). Agroforesterie : Des arbres et des cultures. Editions france agricole, 2ème éd. edition. *7 citations pages 14, 132, 133, 134, 135, 136, and 137*

[Dury et al., 2012] Dury, J., Schaller, N., Garcia, F., Reynaud, A., and E., B. J. (2012). Models to support cropping plan and crop rotation decisions. A review. Agron. Sustain. Dev, 32((2)) :567–580. *Cité page 139*

[Edwards et al., 1998] Edwards, L., Richter, G., Bernsdorf, B., Schmidt, R.-G., and Burney, J. (1998). Measurement of rill erosion by snowmelt on potato fields under rotation in Prince Edward Island (Canada). Canadian journal of soil science, 78(3) :449–458. *Cité page 137*

[Eiben et al., 1999] Eiben, Á. E., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. IEEE Transactions on evolutionary computation, 3(2) :124–141. *Cité page 56*

[Eiben et al., 2004] Eiben, A. E., Marchiori, E., and Valko, V. A. (2004). Evolutionary algorithms with on-the-fly population size adjustment. In International Conference on Parallel Problem Solving from Nature, pages 41–50. Springer. *Cité page 56*

[Eichhorn et al., 2006a] Eichhorn, M. P., Paris, P., Herzog, F., Incoll, D., Liagre, F., Mantzanas, K., Mayus, M., Moreno, G., Papanastasis, V., Pilbeam, D., Pisanelli, A., and Dupraz, C. (2006a). Silvoarable systems in europe – past, present and future prospects. Agroforestry Systems, 67 :29–50. *Cité page 128*

[Eichhorn et al., 2006b] Eichhorn, M. P., Paris, P., Herzog, F., Incoll, L. D., Liagre, F., Mantzanas, K., Mayus, M., Moreno, G., Papanastasis, V. P., Pilbeam, D. J., Pisanelli, A., and Dupraz, C. (2006b). Silvoarable Systems in Europe – Past, Present and Future Prospects. Agroforestry Systems, 67(1) :29–50. *Cité page 134*

[Elhallaoui et al., 2005] Elhallaoui, I., Villeneuve, D., Soumis, F., and Desaulniers, G. (2005). Dynamic Aggregation of Set-Partitioning Constraints in Column Generation. Operations Research. *Cité page 52*

[Ernst et al., 2006] Ernst, A., Jiang, H., and Krishnamoorthy, M. (2006). A new Lagrangian heuristic for the task allocation problem. In Industrial Mathematics, pages 137–158. Oxford : Alpha Science International Ltd. *Cité page 76*

[Faivre et al., 2013] Faivre, R., Iooss, B., Mahévas, S., Makowski, D., and Monod, H. (2013). Analyse de sensibilité et exploration de modèles : application aux sciences de la nature et de l'environnement. Editions Quae. *6 citations pages 13, 59, 61, 62, 63, and 64*

[Filippi et al., 2017] Filippi, C., Mansini, R., and Stevanato, E. (2017). Mixed integer linear programming models for optimal crop selection. Computers & Operations Research, 81 :26–39. *Cité page 140*

[Fisher, 1985] Fisher, M. L. (1985). An applications oriented guide to Lagrangian relaxation. Interfaces, 15(2) :10–21. *4 citations pages 9, 27, 35, and 38*

[Fisher, 2004] Fisher, M. L. (2004). The Lagrangian relaxation method for solving integer programming problems. Management science, 50(12_supplement) :1861–1871. *3 citations pages 13, 34, and 38*

[Fortet, 1959] Fortet, R. (1959). Applications de l'algèbre de Boole en recherché opérationnelle. Revue française d'automatique d'informatique et de recherche opérationnelle, 4 :5–36. *2 citations pages 30 and 161*

[Fortet, 1960] Fortet, R. (1960). L'algebre de boole et ses applications en recherche opérationnelle. Trabajos de Estadistica y de Investigación Operativa, 11(2) :111–118. *Cité page 30*

[Fortier, 2016] Fortier, J.-M. (2016). Le jardinier-maraîcher - Manuel d'agriculture biologique sur petite surface. Ecosociété, édition revue et augmentée edition. *Cité page 138*

[Friedman, 1937] Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the american statistical association, 32(200) :675–701. *Cité page 67*

[Froger et al., 2015] Froger, A., Guyon, O., and Pinson, E. (2015). A set packing approach for scheduling passenger train drivers : the French experience. In RailTokyo2015. *Cité page 52*

[Galán-Martín et al., 2015] Galán-Martín, n., Pozo, C., Guillén-Gosálbez, G., Antón Vallejo, A., and Jiménez Esteller, L. (2015). Multi-stage linear programming model for optimizing cropping plan decisions under the new Common Agricultural Policy. Land Use Policy, 48 :515–524. *Cité page 139*

[Gandibleux et al., 2004] Gandibleux, X., Delorme, X., and T'Kindt, V. (2004). An ant colony optimisation algorithm for the set packing problem. In International Workshop on Ant Colony Optimization and Swarm Intelligence, pages 49–60. Springer. *Cité page 53*

[Gent et al., 2017] Gent, I. P., Jefferson, C., and Nightingale, P. (2017). Complexity of n-Queens Completion. Journal of Artificial Intelligence Research, 59 :815–848. *Cité page 51*

[Glen, 1987] Glen, J. J. (1987). Mathematical models in farm planning : a survey. Operations Research, 35(5) :641–666. *Cité page 139*

[Glover, 1975] Glover, F. (1975). Improved linear integer programming formulations of nonlinear integer problems. Management Science, 22(4) :455–460. *Cité page 30*

[Glover, 1990] Glover, F. (1990). Tabu search : A tutorial. Interfaces, 20(4) :74–94. *Cité page 43*

[Griffiths et al., 1998] Griffiths, J., Phillips, D. S., Compton, S. G., Wright, C., and IncollL, L. D. (1998). Responses of slug numbers and slug damage to crops in a silvoarable agroforestry landscape. Journal of applied ecology, 35 :252–260. *Cité page 133*

[Grigoryan, 2018] Grigoryan, E. (2018). Investigation of the regularities in the formation of solutions n-queens problem. Modeling of Artificial Intelligence, 5 :3–21. *Cité page 51*

[Grohe and Wedelin, 2008] Grohe, B. and Wedelin, D. (2008). Cost Propagation–Numerical Propagation for Optimization Problems. In International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, pages 97–111. Springer. *Cité page 76*

[Groiez, 2013] Groiez, M. (2013). Étude et séparation des inégalités valides pour des problèmes de partitionnement et de couverture. PhD Thesis, École Polytechnique de Montréal. *Cité page 49*

[Gupta et al., 2000] Gupta, A. P., Harboe, R., and Tabucanon, M. T. (2000). Fuzzy multiple-criteria decision making for crop area planning in Narmada river basin. Agricultural Systems, 63(1) :1–18. *2 citations pages 139 and 140*

[Hamadi, 2013] Hamadi, Y. (2013). Autonomous search. In Combinatorial Search : From Algorithms to Systems, pages 99–122. Springer. *Cité page 56*

[Handler and Zang, 1980] Handler, G. Y. and Zang, I. (1980). A dual algorithm for the constrained shortest path problem. Networks, 10(4) :293–309. *Cité page 38*

[Haneveld and Stegeman, 2005] Haneveld, W. K. and Stegeman, A. W. (2005). Crop succession requirements in agricultural production planning. European Journal of Operational Research, 166(2) :406–429. *Cité page 140*

[Heady, 1948] Heady, E. O. (1948). The Economics of Rotations with Farm and Production Policy Applications. Journal of Farm Economics, 30(4) :645–664. *Cité page 139*

[Heady, 1954] Heady, E. O. (1954). Simplified Presentation and Logical Aspects of Linear Programming Technique. Journal of Farm Economics, 36(5) :1035–1048. *Cité page 139*

[Held et al., 1974] Held, M., Wolfe, P., and Crowder, H. P. (1974). Validation of subgradient optimization. Mathematical programming, 6(1) :62–88. *Cité page 34*

[HERZOG, 1998] HERZOG, F. (1998). Streuobst : a traditional agroforestry system as a model for agroforestry development in temperate europe. Agroforestry Systems, 42 :61–80. *Cité page 128*

[Hinterding et al., 1997] Hinterding, R., Michalewicz, Z., and Eiben, A. E. (1997). Adaptation in evolutionary computation : A survey. In Evolutionary Computation, 1997., IEEE International Conference on, pages 65–69. IEEE. *Cité page 56*

[Hoffman and Padberg, 1993] Hoffman, K. L. and Padberg, M. (1993). Solving airline crew scheduling problems by branch-and-cut. Management science, 39(6) :657–682. *3 citations pages 50, 52, and 111*

[Holmberg and Yuan, 2000] Holmberg, K. and Yuan, D. (2000). A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. Operations Research, 48(3) :461–481. *Cité page 38*

[Huisman, 2007] Huisman, D. (2007). A column generation approach for the rail crew re-scheduling problem. European Journal of Operational Research, 180(1) :163–173. *2 citations pages 49 and 52*

[Hunter and Aarssen, 1988] Hunter, A. F. and Aarssen, L. W. (1988). Plants helping plants. BioScience JSTOR, 38 :34–40. *4 citations pages 14, 130, 131, and 132*

[Hurley et al., 2016] Hurley, B., O'Sullivan, B., Allouche, D., Katsirelos, G., Schiex, T., Zytnicki, M., and de Givry, S. (2016). Multi-language evaluation of exact solvers in graphical model discrete optimization. Constraints, An International Journal, 21 :413–434. *2 citations pages 112 and 122*

[Hutter, 2009] Hutter, F. (2009). Automated configuration of algorithms for solving hard computational problems. PhD Thesis, University of British Columbia. *4 citations pages 66, 67, 68, and 70*

[Hutter et al., 2010] Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. In International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, pages 186–202. Springer. *Cité page 70*

[Hutter et al., 2005] Hutter, F., Hoos, H. H., and Stützle, T. (2005). Efficient stochastic local search for MPE solving. In Proc. of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), pages 169–174. *Cité page 176*

[Hutter et al., 2007] Hutter, F., Hoos, H. H., and Stützle, T. (2007). Automatic algorithm configuration based on local search. In Aaai, volume 7, pages 1152–1157. *Cité page 68*

[Iooss, 2011] Iooss, B. (2011). Revue sur l'analyse de sensibilité globale de modèles numériques. Journal de la Société Française de Statistique, 152(1) :1–23. *2 citations pages 13 and 65*

[Iooss and Lemaître, 2015] Iooss, B. and Lemaître, P. (2015). A review on global sensitivity analysis methods. In Uncertainty Management in Simulation-Optimization of Complex Systems, pages 101–122. Springer. *Cité page 63*

[Itoh et al., 2003] Itoh, T., Ishii, H., and Nanseki, T. (2003). A model of crop planning under uncertainty in agricultural management. International Journal of Production Economics, 81-82 :555–558. *2 citations pages 139 and 140*

[J.E.Beasley, 1987] J.E.Beasley (1987). An algorithm for set covering problems. European Journal of Operational Research, 31 :85–93. *Cité page 111*

[Jha et al., 2018] Jha, R., Das, D., Dash, A., Jayaraman, S., Behera, B., and Panigrahi, P. (2018). A novel quantum n-queens solver algorithm and its simulation and application to satellite communication using IBM quantum experience. arXiv, arXiv :1806.10221. *Cité page 51*

[Jones, 2001] Jones, D. R. (2001). A taxonomy of global optimization methods based on response surfaces. Journal of global optimization, 21(4) :345–383. *2 citations pages 62 and 67*

[Jones and Perry, 1978] Jones, F. G. W. and Perry, J. N. (1978). Modelling Populations of Cyst-Nematodes (Nematoda : Heteroderidae). Journal of Applied Ecology, 15(2) :349–371. *Cité page 138*

[Jose et al., 2004] Jose, S., Gillespie, A. R., and Pallardy, S. G. (2004). Interspecific interactions in temperate agroforestry. In New Vistas in Agroforestry, pages 237–255. Springer. *3 citations pages 18, 131, and 133*

[Jose et al., 2000a] Jose, S., Gillespie, A. R., Seifert, J., and Biehle., D. J. (2000a). Defining competition vectors in a temperate alley cropping system in the midwestern usa 2. competition for water. Agroforestry Systems, 48 :41–59. *Cité page 133*

[Jose et al., 2000b] Jose, S., Gillespie, A. R., Seifert, J., Mengel, D., and Pope, P. (2000b). Defining competition vectors in a temperate alley cropping system in the mid-western usa. 3. competition for nitrogen and litter decomposition dynamics. Agroforestry Systems, 48 :61–77. *Cité page 134*

[Kaeser et al., 2010] Kaeser, A., Sereke, F., Dux, D., Herzog, F., and Reckenholz-Tänikon, A. R. T. (2010). Agroforesterie moderne en Suisse. Vergers novateurs : productivité et rentabilité. *3 citations pages 14, 133, and 136*

[Kargupta, 1995] Kargupta, H. (1995). SEARCH, polynomial complexity, and the fast messy genetic algorithm. Urbana, 51 :61801. *Cité page 63*

[Kelly and Xu, 1999] Kelly, J. P. and Xu, J. (1999). A set-partitioning-based heuristic for the vehicle routing problem. INFORMS Journal on Computing, 11(2) :161–172. *2 citations pages 52 and 53*

[Kim et al., 2006] Kim, J. G., Hunke, E. C., and Lipscomb, W. H. (2006). Sensitivity analysis and parameter tuning scheme for global sea-ice modeling. Ocean Modelling, 14(1) :61–80. *Cité page 56*

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. science, 220(4598) :671–680. *Cité page 42*

[Kleijnen, 2005] Kleijnen, J. P. (2005). An overview of the design and analysis of simulation experiments for sensitivity analysis. European Journal of Operational Research, 164(2) :287–300. *Cité page 63*

[Klincewicz and Luss, 1986] Klincewicz, J. G. and Luss, H. (1986). A Lagrangian relaxation heuristic for capacitated facility location with single-source constraints. Journal of the Operational Research Society, 37(5) :495–500. *Cité page 39*

[Kosters, 2013] Kosters, W. (2013). n-Queens bibliography 336 references. http://liacs.leidenuniv.nl/~kosterswa/nqueens/. Accessed : 2010-09-30. *Cité page 51*

[Kuhnt and Rudak, 2011] Kuhnt, S. and Rudak, N. (2011). Jop : Joint optimization plot. R package version, 2(1). *Cité page 71*

[Land and Doig, 1960] Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. Econometrica : Journal of the Econometric Society, pages 497–520. *Cité page 39*

[Lehmann et al., 1998] Lehmann, J., Peter, I., Steglich, C., Gebauer, G., Huwe, B., and Zech, W. (1998). Below-ground interactions in dryland agroforestry. Forest Ecology and Management, 111 :157–169. *Cité page 133*

[Leteinturier et al., 2006] Leteinturier, B., Herman, J. L., Longueville, F. d., Quintin, L., and Oger, R. (2006). Adaptation of a crop sequence indicator based on a land parcel management system. Agriculture, Ecosystems & Environment, 112(4) :324–334. *Cité page 138*

[Malézieux, 2012] Malézieux, E. (2012). Designing cropping systems from nature. Agronomy for sustainable development, 32(1) :15–29. *Cité page 128*

[Maqrot et al., 2016] Maqrot, S., De Givry, S., Quesnel, G., and Tchamitchian, M. (2016). Designing mixed fruit-vegetable cropping systems by integer quadratic programming. In 8th International Congress on Environmental Modelling and Software (iEMSs), volume 2, page 8 p., toulouse, France. *3 citations pages 22, 152, and 164*

[Maqrot et al., 2017a] Maqrot, S., de Givry, S., Quesnel, G., and Tchamitchian, M. (2017a). Designing mixed fruit-vegetable cropping systems by integer quadratic programming. In Acta Horticulturae, pages 265–274. International Society for Horticultural Science (ISHS), Leuven, Belgium. *2 citations pages 22 and 152*

[Maqrot et al., 2017b] Maqrot, S., de Givry, S., Quesnel, G., and Tchamitchian, M. (2017b). A Mixed Integer Programming Reformulation of the Mixed Fruit-Vegetable Crop Allocation Problem. In Advances in Artificial Intelligence : From

Theory to Practice, Lecture Notes in Computer Science, pages 237–250. Springer. *2 citations pages 22 and 152*

[Maqrot et al., 2018a] Maqrot, S., De Givry, S., Quesnel, G., and Tchamitchian, M. (2018a). Improving wedelin's heuristic with sensitivity analysis for set partitioning problem. In 23rd International Symposium on Mathematical Programming (ISMP-18) Bordeaux, France. *3 citations pages 22, 92, and 161*

[Maqrot et al., 2018b] Maqrot, S., De Givry, S., Tchamitchian, M., and Quesnel, G. (2018b). Improving wedelin's heuristic with sensitivity analysis. In In Proc. (ROADEF-18) Lorient, France. *5 citations pages 19, 22, 92, 152, and 169*

[Maron and Moore, 1994] Maron, O. and Moore, A. W. (1994). Hoeffding races : Accelerating model selection search for classification and function approximation. In Advances in neural information processing systems, pages 59–66. *Cité page 67*

[Mason, 2001] Mason, A. J. (2001). Elastic constraint branching, the Wedelin/Carmen Lagrangian heuristic and integer programming for personnel scheduling. Annals of Operations Research, 108(1-4) :239–276. *6 citations pages 76, 84, 85, 86, 87, and 102*

[McCarl et al., 1977] McCarl, B. A., Candler, W. V., Doster, D. H., and Robbins, P. R. (1977). Experiences with farmer oriented linear programming for crop planning. Canadian Journal of Agricultural Economics/Revue canadienne d'agroeconomie, 25(1) :17–30. *Cité page 140*

[Mebane and Sekhon, 2012] Mebane, W. R. and Sekhon, J. S. (2012). R version of genetic optimization using derivatives, version 5.8-2.0. Technical report. *Cité page 117*

[Mebane Jr et al., 2011] Mebane Jr, W. R., Sekhon, J. S., et al. (2011). Genetic optimization using derivatives : the rgenoud package for r. Journal of Statistical Software, 42(11) :1–26. *2 citations pages 69 and 71*

[Medard and Sawhney, 2007] Medard, C. P. and Sawhney, N. (2007). Airline crew scheduling from planning to operations. European Journal of Operational Research, 183(3) :1013–1027. *Cité page 52*

[Mingozzi et al., 1999] Mingozzi, A., Boschetti, M. A., Ricciardelli, S., and Bianco, L. (1999). A set partitioning approach to the crew scheduling problem. Operations Research, 47(6) :873–888. *Cité page 52*

[MIPLIB, 2010] MIPLIB (2010). MIPLIB. 2010 mixed integer problem library. http://miplib2010.zib.de/miplib2010-BP.phpm. Accessed : 2019-01-17. *6 citations pages 17, 18, 97, 101, 104, and 111*

[Mitchell, 2011] Mitchell, J. E. (2011). Branch and cut. Wiley Encyclopedia of Operations Research and Management Science. New York : Wiley. *Cité page 40*

[Morgan, 2014] Morgan, J. W. (2014). Boolean Binary Response Models. Technical report, Package for R, a free software environment for statistical computing and graphics. *Cité page 71*

[Morris, 1991] Morris, M. D. (1991). Factorial sampling plans for preliminary computational experiments. Technometrics, 33(2) :161–174. *Cité page 59*

[Nerlich et al., 2013] Nerlich, K., Graeff-Ho¨nninger, S., and Claupein, W. (2013). Agroforestry in europe : a review of the disappearance of traditional systems and development of modern agroforestry practices, with emphasis on experiences in germany. Agroforestry Systems, 87 :475–492. *2 citations pages 14 and 136*

[Neveu et al., 2004] Neveu, B., Trombettoni, G., and Glover, F. (2004). ID Walk : A Candidate List Strategy with a Simple Diversification Device. In Wallace, M., editor, Principles and Practice of Constraint Programming – CP 2004, Lecture Notes in Computer Science, pages 423–437. Springer Berlin Heidelberg. *Cité page 44*

[Nevo et al., 1994] Nevo, A., Oad, R., and Podmore, T. H. (1994). An integrated expert system for optimal crop planning. Agricultural Systems, 45(1) :73–92. *Cité page 140*

[Newman, 1986] Newman, S. (1986). A pear and vegetable interculture system : Land equivalent ratio, light use efficiency and productivity. Experimental Agriculture, 22 :383–392. *Cité page 134*

[Nissen et al., 1999] Nissen, T., Midmore, D., and Cabrera, M. (1999). Aboveground and belowground competition between intercropped cabbage and young eucalyptus torelliana. Agroforestry Systems, 46 :83–93. *Cité page 134*

[Niu et al., 2016] Niu, G., Li, Y. P., Huang, G. H., Liu, J., and Fan, Y. R. (2016). Crop planning and water resource allocation for sustainable development of an irrigation region in China under multiple uncertainties. Agricultural Water Management, 166 :53–69. *Cité page 139*

[Ong et al., 1991] Ong, C., Corlett, J., Singh, R., and Black, C. (1991). Above and below ground interactions in agroforestry systemss. Agroforestry Systems, 45 :45–57. *Cité page 132*

[Ouali et al., 2017] Ouali, A., Allouche, D., De Givry, S., Loudni, S., Lebbah, Y., Eckhardt, F., and Loukil, L. (2017). Iterative Decomposition Guided Variable Neighborhood Search for Graphical Model Energy Minimization. In Conference on Uncertainty in Artificial Intelligence, UAI 2017. *2 citations pages 44 and 112*

[Palmer, 1997] Palmer, J. W. (1997). Diurnal light interception and a computer model of light interception by hedgerow apple orchards. Journal of Applied Ecology, 14 :601–614. *Cité page 132*

[Papadimitriou and Steiglitz, 1998] Papadimitriou, C. H. and Steiglitz, K. (1998). Combinatorial optimization : algorithms and complexity. Courier Corporation. *Cité page 28*

[Paschos, 2013] Paschos, V. T. (2013). Applications of combinatorial optimization, Volume 3. John Wiley & Sons. *Cité page 28*

[Plateau, 2006] Plateau, M.-C. (2006). Reformulations quadratiques convexes pour la programmation quadratique en variables 0-1. These de doctorat en informatique, Conservatoire National des Arts et Métiers, Paris. *Cité page 30*

[Pujol et al., 2012] Pujol, G., Iooss, B., and Janon, A. (2012). The R package "sensitivity", version 1.6-1. *Cité page 113*

[Rao et al., 1993] Rao, M. R., Muraya, P., and Huxley, P. A. (1993). Observations of some tree root systems in agroforestry intercrop situations, and their graphical representation. Experimental Agriculture, 29 :183–194. *Cité page 134*

[Rasmussen, 2011] Rasmussen, M. (2011). Optimisation-Based Solution Methods for Set Partitioning Models. PhD thesis, Technical University of Denmark (DTU). *Cité page 49*

[ROSAT, 2016] ROSAT, S. (2016). Méthodes pour favoriser l'intégralité de l'amélioration dans le simplexe en nombres entiers - Application aux rotations d'équipages aériens. PhD Thesis, École polytechnique de Montréal. Département de mathématiques et de génie industriel. Accessed : 2019-03-06. *Cité page 52*

[Rubin, 1973] Rubin, J. (1973). A technique for the solution of massive set covering problems, with application to airline crew scheduling. Transportation Science, 7(1) :34–48. *Cité page 52*

[Saltelli et al., 1999] Saltelli, A., Tarantola, S., and Chan, K.-S. (1999). A quantitative model-independent method for global sensitivity analysis of model output. Technometrics, 41(1) :39–56. *Cité page 62*

[Santos et al., 2015] Santos, L. M. R., Munari, P., Costa, A. M., and Santos, R. H. S. (2015). A branch-price-and-cut method for the vegetable crop rotation scheduling problem with minimal plot sizes. European Journal of Operational Research, 245 :581–590. *Cité page 141*

[Sarker and Ray, 2009] Sarker, R. and Ray, T. (2009). An improved evolutionary algorithm for solving multi-objective crop planning models. Computers and Electronics in Agriculture, 68(2) :191–199. *Cité page 140*

[Sarker and Quaddus, 2002] Sarker, R. A. and Quaddus, M. A. (2002). Modelling a nationwide crop planning problem using a multiple criteria decision making tool. Computers & Industrial Engineering, 42(2) :541–553. *Cité page 140*

[Sekhon and Mebane, 1998] Sekhon, J. S. and Mebane, W. R. (1998). Genetic optimization using derivatives. Political Analysis, 7 :187–210. *Cité page 69*

[Seki et al., 2010] Seki, T., Yamashita, N., and Kawamoto, K. (2010). New local search methods for improving the Lagrangian-relaxation-based unit commitment solution. IEEE Transactions on Power Systems, 25(1) :272–283. *Cité page 38*

[Sethi et al., 2006] Sethi, L. N., Panda, S. N., and Nayak, M. K. (2006). Optimal crop planning and water resources allocation in a coastal groundwater basin, Orissa, India. Agricultural Water Management, 83(3) :209–220. *Cité page 140*

[Sherali and Adams, 2013] Sherali, H. D. and Adams, W. P. (2013). A reformulation-linearization technique for solving discrete and continuous nonconvex problems, volume 31. Springer Science & Business Media. *Cité page 30*

[Simon et al., 2010] Simon, S., Bouvier, J., Debras, J., and Sauphanor, B. (2010). Biodiversity and pest management in orchard systems. a review. Agronomy for Sustainable Development, 30 :139–152. *Cité page 133*

[Simpson et al., 2001] Simpson, T. W., Poplinski, J. D., Koch, P. N., and Allen, J. K. (2001). Metamodels for computer-based engineering design : survey and recommendations. Engineering with computers, 17(2) :129–150. *Cité page 62*

[Smith and Wren, 1988] Smith, B. M. and Wren, A. (1988). A bus crew scheduling system using a set covering formulation. Transportation Research Part A : General, 22(2) :97–108. *Cité page 52*

[Smith and Fogarty, 1996] Smith, J. and Fogarty, T. C. (1996). Self adaptation of mutation rates in a steady state genetic algorithm. In Evolutionary Computation, 1996., Proceedings of IEEE International Conference on, pages 318–323. IEEE. *Cité page 56*

[Sobol, 1993] Sobol, I. M. (1993). Sensitivity estimates for nonlinear mathematical models. Mathematical modelling and computational experiments, 1(4) :407–414. *2 citations pages 60 and 62*

[Sobol, 2001] Sobol, I. M. (2001). Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. Mathematics and computers in simulation, 55(1-3) :271–280. *Cité page 62*

[Sosic and Jun Gu, 1994] Sosic, R. and Jun Gu (1994). Efficient local search with conflict minimization : a case study of the n-queens problem. IEEE Transactions on Knowledge and Data Engineering, 6(5) :661–668. *Cité page 51*

[Storlie B. et al., 2009] Storlie B., C., Swiler, L. P., Helton, J. C., and Sallaberry, C. J. (2009). Implementation and evaluation of nonparametric regression procedures for sensitivity analysis of computationally demanding models. Reliability Engineering & System Safety, 94(11) :1735–1763. *Cité page 62*

[Suniel, 1999] Suniel, C. (1999). CSPLib problem 022 : Bus driver scheduling. http://www.csplib.org/Problems/prob022. Accessed : 2019-03-04. *Cité page 112*

[Taillandier et al., 2012] Taillandier, P., Therond, O., and Gaudou, B. (2012). A new BDI agent architecture based on the belief theory. Application to the modelling of cropping plan decision-making. In International environmental modelling and software society (iEMSs). *Cité page 141*

[Tarantola et al., 2006] Tarantola, S., Gatelli, D., and Mara, T. A. (2006). Random balance designs for the estimation of first order global sensitivity indices. Reliability Engineering & System Safety, 91(6) :717–727. *Cité page 62*

[Tchamitchian and Godin, 2014] Tchamitchian, M. and Godin, E. (2014). Designing mixed horticultural systems. Building Organic Bridges, 1 :179–182. *2 citations pages 21 and 144*

[Teodoro et al., 2016] Teodoro, G., Kurç, T. M., Taveira, L. F., Melo, A. C., Gao, Y., Kong, J., and Saltz, J. H. (2016). Algorithm sensitivity analysis and parameter tuning for tissue image segmentation pipelines. Bioinformatics, 33(7) :1064–1072. *Cité page 57*

[Thie and Keough, 2011] Thie, P. R. and Keough, G. E. (2011). An introduction to linear programming and game theory. John Wiley & Sons. *Cité page 31*

[Tissot and Prieur, 2012] Tissot, J.-Y. and Prieur, C. (2012). Bias correction for the estimation of sensitivity indices based on random balance designs. Reliability Engineering and System Safety, 107 :205–213. *Cité page 62*

[Tjomson, 1978] Tjomson, J. D. (1978). Effects of stand composi- tion on insect visitation in two species mix- tures of hieracium. The American Midland Naturalist, 100 :431–440. *Cité page 131*

[Tsakiris and Spiliotis, 2006] Tsakiris, G. and Spiliotis, M. (2006). Cropping pattern planning under water supply from multiple sources. Irrigation and Drainage Systems, 20(1) :57–68. *2 citations pages 139 and 140*

[Umetani, 2017] Umetani, S. (2017). Exploiting variable associations to configure efficient local search algorithms in large-scale binary integer programs. European Journal of Operational Research. *6 citations pages 18, 53, 122, 123, 178, and 179*

[Umetani and Yagiura, 2007] Umetani, S. and Yagiura, M. (2007). Relaxation heuristics for the Set Covering Problem (< Special Issue> the 50th Anniversary of the Operations Research Society of Japan). Journal of the Operations Research Society of Japan, 50(4) :350–375. *4 citations pages 18, 53, 123, and 124*

[Valli et al., 1965] Valli, V., Bryan, H., and Young, H.W. et Davis, D. (1965). The effect of shade on the bio-climate and production of vegetable crops. Florida state horticultural society, 78 :95-101. *Cité page 132*

[van Noordwijk et al., 1996] van Noordwijk, M., Lawson, G., K.Hairiah, and Wilso, J. (1996). Root distribution of trees and crops : competition and/or complementarity. Tree–crop interactions : Agroforestry in a changing climate. cabi, wallingford, uk edition. *2 citations pages 133 and 134*

[Vemuganti, 1998] Vemuganti, R. R. (1998). Applications of set covering, set packing and set partitioning models : A survey. In Handbook of combinatorial optimization, pages 573–746. Springer. *Cité page 50*

[Vercambre et al., 2003] Vercambre, G., Pagès, L., Doussan, C., and Habib, R. (2003). Architectural analysis and synthesis of the plum tree root system in an orchard using a quantitative modelling approach. Plant and Soil, 251(1) :1–11. *Cité page 148*

[Villa-Vialaneix et al., 2012] Villa-Vialaneix, N., Follador, M., Ratto, M., and Leip, A. (2012). A comparison of eight metamodeling techniques for the simulation of N2o fluxes and N leaching from corn crops. Environmental Modelling & Software, 34 :51–66. *Cité page 62*

[Weaver and Bruner, 1927] Weaver, J. E. and Bruner, W. E. (1927). Root Development Of Vegetable Crops Ist Edn. Mcgraw-Iill Book Co, ; London. *Cité page 149*

[Wedelin, 1995a] Wedelin, D. (1995a). An algorithm for large scale 0/1 integer programming with application to airline crew scheduling. Annals of operations research, 57(1) :283–301. *8 citations pages 52, 53, 76, 78, 83, 85, 87, and 88*

[Wedelin, 1995b] Wedelin, D. (1995b). The design of a 0/1 integer optimizer and its application in the Carmen system. European Journal of Operational Research, 87(3) :722–730. *4 citations pages 21, 22, 39, and 76*

[Wedelin, 2013] Wedelin, D. (2013). Revisiting the in-the-middle algorithm and heuristic for integer programming and the max-sum problem. preprint. https://pdfs.semanticscholar.org/a98c/0447e55f4feee688f1f6e4eb0e77f17a418f.pdf (Accessed : 2019-10-19). *3 citations pages 76, 88, and 176*

[Werner, 2005] Werner, T. (2005). A linear programming approach to max-sum problem : A review. Research Reports of CMP. 2005, No. 25. *Cité page 176*

[Xie and Turnquist, 2011] Xie, C. and Turnquist, M. A. (2011). Lane-based evacuation network optimization : An integrated Lagrangian relaxation and tabu search approach. Transportation Research Part C : Emerging Technologies, 19(1) :40–63. *Cité page 39*

[Yaghini et al., 2015] Yaghini, M., Karimi, M., and Rahbar, M. (2015). A set covering approach for multi-depot train driver scheduling. Journal of Combinatorial Optimization, 29(3) :636–654. *Cité page 53*

[Yang and Abbass, 2003] Yang, S. and Abbass, B. (2003). Adaptive crossover in genetic algorithms using statistics mechanism. Artificial Life, 8 :182–185. *Cité page 56*

[Zaghrouti et al., 2014] Zaghrouti, A., Soumis, F., and Hallaoui, . I. E. (2014). Integral simplex using decomposition for the set partitioning problem. Operations Research, 62 :435–449. *3 citations pages 52, 112, and 122*

[Zhu et al., 1997] Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). Algorithm 778 : L-BFGS-B : Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on Mathematical Software (TOMS), 23(4) :550–560. *2 citations pages 57 and 69*

Titre Méthodes exactes et approchées d'optimisation combinatoire en programmation mathématique. Application à la conception des systèmes de verger-maraîcher.

Résumé Dans le cadre du développement durable et des innovations dans les systèmes agroalimentaires, les systèmes mixtes horticoles (vergers et maraîchage) visent à répondre aux enjeux actuels auxquels l'agriculture est confrontée, à savoir une diminution de la pollution des sols, une meilleure gestion des ressources (eau, énergies) et un enrichissement de la biodiversité, tout en continuant d'assurer des fonctions alimentaires. Ils combinent des productions à la fois diversifiées et relativement intensifiées, leur permettant de s'insérer en périphérie urbaine. Ces systèmes agroforestiers reposent sur un ensemble complexe d'interactions modifiant l'utilisation de la lumière, de l'eau et des nutriments. La conception d'un tel système doit donc optimiser l'utilisation de ces ressources en maximisant les interactions positives (facilitations) et en minimisant celles négatives (compétitions). Nous définissons le problème de verger-maraîcher comme un problème d'allocation des arbres et des cultures dans les dimensions spatio-temporelles. Nous proposons trois formulations mathématiques : modèle quadratique en variables binaires (BQP), modèle linéaire en variables mixtes (MILP) et modèle linéaire en variables binaires (01LP). Les limites des méthodes exactes pour résoudre ce problème sont présentées, montrant la nécessité d'appliquer des méthodes approchées, capables de résoudre des systèmes à grande échelle avec des solutions de bonne qualité en temps raisonnable. Pour cela, nous avons développé un solveur open source, BARYONYX, qui est une version parallèle de l'heuristique de Wedelin (généralisée). Nous avons utilisé l'analyse de sensibilité pour identifier les paramètres les plus influents. Une fois trouvés, nous avons fixé les autres et utilisé un algorithme génétique pour régler les plus importants sur un ensemble d'instances d'entraînement. Le jeu de paramètres optimisé peut alors être utilisé pour résoudre d'autres instances de plus grande taille du même type de problème. BARYONYX avec son réglage automatique obtient des résultats améliorant l'état-de-l'art sur des problèmes de partitionnement. Les résultats sont plus mitigés sur le problème de verger-maraîchage, bien que capable de passer à l'échelle.

Mots-clés optimisation combinatoire, heuristiques, analyse de sensibilité, réglage de paramètres, problème d'allocation spatio-temporelle des cultures, agroécologie.

Title Exact and Approximate methods for combinatorial optimization in mathematical programming. Application to agroforestry system design.

Abstract Mixed fruit-vegetable cropping systems (MFVCS) are a promising way of ensuring environmentally sustainable agricultural production systems in response to the challenge of being able to fulfill local market requirements. They combine productions and make a better use of biodiversity. These agroforestry systems are based on a complex set of interactions modifying the utilization of light, water and nutrients. Thus, designing such systems requires to optimize the use of these resources : by maximizing positive interactions (facilitations) and minimizing negative ones (competitions). To reach these objectives, the system's design has to include the spatial and temporal dimensions, taking into account the evolution of above- and belowground interactions over a time horizon. For that, we define the MFVCAP using a discrete representation of the land and the interactions between vegetable crops and fruit trees. We formulate the problem as three models : binary quadratic program (BQP), mixed integer linear programming (MILP) and binary linear programming (01LP). We explore large models using exact solvers. The limits of exact methods in solving the MFVCS problem show the need for approximate methods, able to solve a large-scale system with solutions of good quality in reasonable time, which could be used in interactive design with farmers and advisers. We have implemented a C++ open-source solver, called BARYONYX, which is a parallel version of a (generalized) Wedelin heuristic. We used a sensitivity analysis method to find useful continuous parameters. Once found, we fixed other parameters and let a genetic optimization algorithm using derivatives adjust the useful ones in order to get the best solutions for a given time limit. The optimized configuration could be used to solve larger instances of the same problem type. BARYONYX got competitive results compared to state-of-the-art exact and approximate solvers on crew and bus driver scheduling problems expressed as set partitioning problems. The results are less convincing on MFVCS but still able to produce valid solutions on large instances.

Keywords discrete optimization, heuristics, sensitivity analysis, parameter tuning, spatial and temporal crop allocation problem, agroecology.