REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE.
UNIVERSITE M'HAMED BOUGARA, BOUMERDES
FACULTE DES SCIENCES

Thèse de Doctorat

Présentée par :
ALOUANE Basma

Filière : Informatique
Option : Informatique

Gestion de contraintes dans les approches évolutionnaires

Devant le jury :

Dr. MAOUCHE Amin Riad	Professeur UMBB	Président de jury
Dr. GUERROUMI Mohamed	Professeur USTHB	Examinateur
Dr. BOUGHACI Dalila	Professeur USTHB	Examinateur
Dr. ATIF Karim	MCA USTHB	Examinateur
Dr. BOULIF Menouar	Professeur UMBB	Directeur de thèse

Année Universitaire 2022/2023

Dédicace

Je dédié ce manuscrit

A mes chères parents pour leur amour inconditionnel
A mon mari mon meilleur ami
A mes chères enfants ma raison de vivre
A mes frères et sœurs mes piliers
A mes beaux parents

Remerciements

Je tiens à exprimer ma profonde gratitude à mon directeur de thèse, le professeur BOULIF Menouar, pour son aide, sa compétence et sa disponibilité tout au long de mes recherches et de la rédaction de ma thèse. Ses suggestions et ses réponses à mes questions ont considérablement amélioré la qualité de mon travail. Sa bienveillance, que je considère comme celle d'un frère, m'a motivée à continuer et à aller de l'avant. Je suis très reconnaissante de l'avoir comme directeur de thèse.

Je tiens à exprimer mes sincères remerciements à Pr. MAOUCHE Amin Riad, professeur à l'UMBB, pour avoir accepté de présider le jury.

Je suis reconnaissante à Dr. ATIF Karim, MCA à l'USTHB, Pr. GUERROUMI Mohamed, professeur à l'USTHB, et Pr. BOUGHACI Dalila, professeur à l'USTHB, qui ont généreusement donné de leur temps et de leur expertise pour évaluer mon travail.

Je remercie également le professeur Mohamed MEZGHICHE, Directeur de laboratoire LIMOSE, et le chef de département informatique, professeur Ali BERICHI, ainsi que tous les collègues enseignants du département informatique.

Je remercie mes chères amies, Mme GUNADIZ Safia, Mme OTHMANIN Wahiba et Mlle ROSTANE Kanza, pour leurs soutiens et encouragements. J'apprécie également leur patience face à mes sautes d'humeur.

Je remercie mon frère, Dr. ALOUANE Mohamed Amine, docteur en Robotique Automatique, pour son aide précieuse et pour ses suggestions tout au long de ce travail.

Je remercie mes parents, mes frères et sœurs pour leur confiance et leur soutien.

Je remercie mon mari pour son soutien, ses encouragements, pour sa confiance et pour sa patience. Je le remercie d'avoir toujours été là pour moi dans les moments difficiles. Je remercie également mes enfants à qui je dois mes excuses pour tous les moments de leur vie où je n'étais pas présente.

Résumé

Les problèmes d'optimisation sont souvent difficiles à résoudre efficacement par des méthodes exactes, notamment lorsque ces problèmes sont combinatoires, complexes ou NP-hard, en raison de la complexité exponentielle de celles-ci. Face à ces limitations, des méthodes approximatives telles que les approches évolutionnaires ont été proposées. En général, les problèmes d'optimisation sont soumis à des contraintes. Dans les approches évolutionnaires, les opérateurs d'exploitation et d'exploration (sélection, mutation et croisement) ne prennent pas en compte les contraintes, ce qui conduit à la division de l'espace de recherche en deux sous-ensembles disjoints : l'espace réalisable (contenant uniquement les solutions satisfaisant toutes les contraintes) et l'espace irréalisable (contenant les solutions violant au moins une contrainte). Ceci a incité les chercheurs à proposer d'équiper ces approches de méthodes de gestion des contraintes qui les gèrent et explorent l'information cachée dans les solutions irréalisables afin d'aider l'approche évolutionnaire à converger vers l'optimum.

Dans cette thèse, nous proposons une technique de gestion des contraintes basée sur des fonctions de transformation. Nous étudions l'impact de la prise en compte de différents ordres de priorité des contraintes et proposons un système d'inférence floue pour gérer l'ordre des contraintes et aider l'algorithme évolutionnaire à converger vers l'espace réalisable.

Mots-clés : Gestion de contraintes - Algorithme génétique - Problème de partitionnement de graphe semi supervisé - Problème fortement contraint - Fonction de transformation - Raisonnement flou.

Abstract

Optimization problems are often challenging to solve efficiently using exact methods, especially when they are combinatorial, complex, or NP-hard, given the exponential complexity of such approaches. As a result, approximate methods like evolutionary algorithms have been proposed. Typically, optimization problems come with constraints. However, in evolutionary algorithms, the exploitation and exploration operators (selection, mutation, and crossover) do not consider constraints, leading to the partitioning of the search space into two distinct subsets : the feasible space (containing only solutions satisfying all constraints) and the infeasible space (containing solutions violating at least one constraint). This limitation has prompted researchers to propose enhancing these methods with constraint handling techniques to deal with constraints effectively and explore the information embedded in infeasible solutions, thereby aiding the evolutionary process in converging towards the optimum.

In this thesis, we propose a constraint handling technique based on transformation functions. We investigate the impact of different orders of constraint priorities and introduce a fuzzy inference system to manage the order of constraints, assisting the evolutionary algorithm in navigating towards the feasible space.

Key words : Constraint handling - Genetic algorithm - Semi-supervised graph partitioning problem - Heavily constrained problems - Transformation function - Fuzzy logic.

اَلْملخص

غالبًا ما تكون مشكلات التحسين صعبة الحل بكفاءة باستخدام الطرق الدقيقة، خاصة عندما تكون اندماجية أو معقدة ونتيجة لذلك، تم اقتراح طرق تقريبية مثل الخوارزميات التطورية. عادة، تأتي مشاكل التحسين مع القيود. ومع ذلك، في الخوارزميات التطورية، لا تأخذ عوامل الاستغلال والاستكشاف (الاختيار، والطفرة، والتقاطع) في الاعتبار القيود، مما يؤدي إلى تقسيم مساحة البحث إلى مجموعتين فرعيتين متميزتين : المساحة الممكنة (التي تحتوي فقط على الحلول التي تلبي جميع القيود) مساحة غير ممكنة (تحتوي على حلول تنتهك قيدًا واحدًا على الأقل). وقد دفع هذا القيد الباحثين إلى اقتراح تعزيز هذه الأساليب باستخدام تقنيات التعامل مع القيود للتعامل مع القيود بشكل فعال واستكشاف المعلومات المضمنة في الحلول غير الممكنة، وبالتالي مساعدة العملية التطورية في التقارب نحو المستوى الأمثل.

في هذه الأطروحة، نقترح تقنية معالجة القيد على أساس وظائف التحويل. نحن ندرس تأثير الترتيبات المختلفة لأولويات القيود ونقدم نظام استدلال غامض لإدارة ترتيب القيود، مما يساعد الخوارزمية التطورية في التنقل نحو المساحة الممكنة.

الكلمات الرئيسية : إدارة الشروط الخوارزمية الجينية ـ مشكلة تقسيم الرسم البياني شبه الخاضعة للإشراف ـ مشكلة مقيدة بشدة ـ دوال التحويل ـ التفكير الضبابي.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Trouver une solution optimale qui réalise le(s) objectif(s) et respecte les contraintes liées aux problèmes que l'on rencontre dans notre vie quotidienne, tels que les problèmes d'ordonnancement et le problème de l'emploi du temps, est généralement une tâche très difficile. Cela conduit la communauté scientifique à essayer de trouver des méthodes efficaces pour résoudre de tels problèmes (Bertsekas, 1997) (Golberg, 1989).

La plupart des méthodes sont proposées pour traiter les problèmes d'optimisation sans contrainte et sont classées en deux catégories (Bertsekas, 1997) (Golberg, 1989) : les méthodes exactes et les méthodes approximatives. Les méthodes exactes sont des méthodes qui contiennent des algorithmes garantissant l'exploration des solutions optimales, mais ces algorithmes entraînent des défis supplémentaires, tels que le coût en termes de temps d'exécution et d'espace mémoire utilisé, ainsi que le défi de l'optimum local. Les méthodes approchées sont considérées comme une alternative aux méthodes exactes. Les algorithmes de cette méthode permettent de trouver une ou plusieurs bonnes solutions, non nécessairement optimales, en un temps raisonnable parmi un ensemble de solutions finies.

On distingue deux types de méthodes approchées : les heuristiques et les méta heuristiques. Les heuristiques sont dédiées à des problèmes spécifiques, dont le principe consiste à partir d'une solution approximative, du moins potentiellement bonne, d'essayer de l'améliorer de manière itérative. Les méta-heuristiques s'inspirent des systèmes naturels et peuvent être appliquées à de nombreux problèmes. Les algorithmes méta heuristiques sont classés en deux catégories : les méta heuristiques basées sur une solution unique, qui traitent une solution approximative, même si elle est moins performante, et tentent de l'améliorer de manière itérative au cours de la procédure de recherche, et les méta-heuristiques basées sur une population, qui traitent un ensemble de solutions en parallèle, telles que les Approches Evolutionnaires (AEs).

Certains problèmes d'optimisation, en plus de leur fonction objectif, sont soumis à des contraintes qui doivent être respectées par les solutions, rendant ainsi le problème d'optimisation de plus en plus difficile. La présence de ces contraintes induit la division de l'espace de recherche en deux sous-espaces disjoints, mettant en évidence deux types de solutions : les solutions réalisables (satisfaisant les contraintes) se trouvent dans l'espace réalisable, et les solutions irréalisables (violant au moins une contrainte) sont incluses dans l'espace irréalisable.

L'AE(s) consiste à générer un ensemble d'individus appelé population et, à partir de cette population et sur la base des mesures de qualité qui leur ont été attribuées par la fonction objectif, nous appliquons un certain nombre d'opérateurs (sélection, mutation et croisement). L'AE(s) essaie de trouver une solution réalisable de bonne qualité. Cependant, les opérateurs de mutation et de croisement sont aveugles aux contraintes (ils ne

traitent pas les contraintes), ce qui signifie que les AE(s) excluent les solutions considérées comme irréalisables.

Néanmoins, ces individus irréalisables peuvent aider le processus de recherche à produire des individus réalisables de bonne qualité. Afin de remédier à ce problème, la littérature a proposé d'équiper l'AE(s) de techniques capables de traiter les contraintes et de réévaluer les solutions irréalisables, connues sous le nom de techniques de gestion des contraintes.

Les techniques de gestion de contraintes sont classées en cinq classes (Coello, 2002) qui sont :

1. Fonction de pénalité : Elles sont largement utilisées en mathématiques. Elles ont été proposées par R. Courant dans les années 1940, puis étendues par Carroll et al. et Fiacco et al(Fiacco and McCormick, 1966). L'idée d'une fonction de pénalité est de transformer le problème d'optimisation sous contraintes en un problème sans contraintes en ajoutant une valeur dégradante à la fonction objectif de chaque solution irréalisable. Dans cette catégorie, nous trouvons la pénalité de mort, la pénalité statique, la pénalité dynamique et la pénalité adaptative.
2. Approches basées sur la préservation de la réalisabilité des solutions réalisables : ces approches maintiennent la réalisabilité des solutions en utilisant des représentations d'encodage et des opérateurs génétiques spécifiques.
3. Algorithmes de réparation : ces méthodes réparent chaque solution irréalisable pour ne laisser que les solutions réalisables dans la population.
4. Technique de séparation de la fonction objectif et des contraintes : ces techniques ne combinent pas le degré de violation des contraintes avec la valeur de la fonction objectif.
5. Méthodes hybrides : ces approches couplent l'une des approches précédentes avec une technique d'optimisation numérique pour traiter les contraintes.

Dans (Boulif and Atif, 2006), les auteurs proposent une méthode de gestion de contraintes qui peut être assimilée comme faisant partie d'un sixième groupe que nous appelons une approche par des fonctions de transformation. La méthode proposée réalise une transformation de fonction d'adaptation linéaire que l'AG puisse distinguer entre les solutions.

Cependant, la méthode est présentée de façon concise et se base uniquement sur une transformation linéaire. En outre, en adoptant une telle forme de fonction de transformation, il s'avère qu'au terme du processus de recherche, les solutions réalisables deviennent très proches, au point que l'algorithme évolutionnaire n'arrive pas à distinguer entre les solutions réalisables, ce qui influe sur la qualité de la recherche en phase d'intensification.

Dans cette thèse,

1. Nous étudions l'impact de l'utilisation de différentes formes de fonctions de transformation telles que la fonction exponentielle et la fonction hybride (linéaire, exponentielle). De plus, nous analysons l'impact de la valeur de la barrière entre la région réalisable et la région irréalisable. Les résultats de cette étude ont été rapportés dans l'article (Alouane and Boulif, 2021).
2. Nous avons également constaté que les contraintes n'ont pas toutes le même degré d'importance, ce qui nous a incités à à examiner leur interaction mutuelle. Pour mener notre étude, nous utilisons l'algorithme génétique pour gérer les contraintes en nous appuyant sur la gestion de contraintes basée sur les fonctions de transformation proposées dans (Alouane and Boulif, 2021). De plus, nous proposons un

système d'inférence flou basé sur le modèle de Mamdani (Mamdani and Assilian, 1975), qui nous permettra de déterminer le bon ordre de contraintes favorisant la convergence de l'AG vers la région réalisable.

Les conclusions de l'étude sont présentées dans l'article (Alouane and Boulif, 2023).

Notre thèse est organisée comme suit :

Le chapitre 1 présente les approches évolutionnaires, plus précisément les algorithmes génétiques.

Le chapitre 2 intitulé 'Gestion de contraintes', aborde la notion de gestion de contraintes. Nous présentons une classification des techniques de gestion de contraintes.

Le chapitre 3 présente le problème fortement contraint, et nous détaillons le Problème de Partitionnement de Graphe (PPG) comme exemple de ce problème, qui constitue notre domaine d'application.

Le chapitre 4 présente la gestion de contraintes basée sur les fonctions de transformation pour résoudre un problème fortement contraint. Nous présentons également les différents résultats obtenus.

Le chapitre 5 aborde la priorisation floue des contraintes pour résoudre un problème fortement contraint. Nous présentons également l'analyse et la discussion des résultats obtenus.

Enfin, nous concluons notre thèse par une conclusion générale et des perspectives

Chapitre 1

Problème d'optimisation et approches évolutionnaires

1.1 Introduction

L'optimisation, une discipline des mathématiques dont le but est de trouver une solution à un problème donné.

Cependant, ce n'est pas toujours évident d'obtenir une solution exacte en raison de la complexité du problème d'optimisation (qui peut être simple ou difficile). Cela a mené les mathématiciens à proposer des stratégies pour trouver des solutions proches de l'optimalité, telles que les approches évolutionnaires.

Dans ce chapitre, nous allons explorer les différents types de problèmes d'optimisation et leur classification, nous intéressons aux problèmes d'optimisation combinatoire. Nous allons également présenter les approches évolutionnaires utilisées pour les résoudre. Une attention particulière est accordée aux algorithmes génétiques.

1.2 Définition de problème d'optimisation

L'optimisation consiste à trouver la meilleure solution (optimale) parmi un ensemble de solutions. Il s'agit d'un domaine dérivé des mathématiques, largement utilisé en informatique. Un problème d'optimisation cherche à minimiser (ou maximiser) une fonction objectif (Winston, 2004). Mathématiquement, un problème d'optimisation est défini par la fonction 1.1 :

$$\min_{x \in R^n} f(x) \texttt{sous la contrainte } x \in F \tag{1.1}$$

où $x = (x_1, x_2, ..., x_n)$ est le vecteur de solution, f est la fonction objectif, F est la région réalisable qui contient les solutions réalisables.

Maximiser une fonction f est équivalent à minimiser la fonction $-f$.

$$\max_{x \in F} f(x) = -\min_{x \in F} -f(x) \tag{1.2}$$

1.2.1 Classification des problèmes d'optimisation

Eiben et Smith (Eiben and Smith, 2015) propose la classification suivante basée sur le type de la fonction objectif, le nombre de fonctions objectif, et le type des variables considérées dans l'espace de recherche.

1. Le type des solutions dans l'espace de recherche S dépend de la nature de S. Si S est de type continu (par exemple, des variables réelles), alors il s'agit d'un **problème d'optimisation numérique**. En revanche, si S est de type discret (par exemple, booléen ou entier), alors c'est un **problème d'optimisation combinatoire**,
2. Le nombre de contraintes détermine la nature du problème d'optimisation. Si le problème d'optimisation est soumis à des contraintes alors il s'agit d'un **un problème d'optimisation sous contraintes**, sinon c'est un **problème d'optimisation sans contraintes**,
3. Le nombre de fonction objectif : si le problème d'optimisation optimise une seule fonction objectif donc il s'agit **d'un problème d'optimisation mono objectif**, s'il doit optimiser plusieurs fonctions objectifs donc **c'est un problème d'optimisation multi-objectifs**.

En plus des critères considérés dans (Eiben and Smith, 2015), Yang, dans (Yang, 2010), prend également en considération l'allure de la fonction objectif et le type des contraintes.

1. Si la fonction objectif admet un seul optimum qui est à la fois local et global, alors il s'agit d'un **problème d'optimisation unimodal (convexe)**. En revanche, si la fonction objectif admet plusieurs optima alors c'est un **problème d'optimisation multimodal**,
2. Si les contraintes et la (les) fonction(s) objectif(s) sont linaires donc on parle d'un **problème d'optimisation linaire** sinon c'est un **problème d'optimisation non linaire**.

1.2.2 La théorie de complexité

La théorie de complexité (Hartmanis, 1982) classe les problèmes d'optimisations combinatoire selon leur complexité en terme de temps de calcul et d'espace mémoire en deux classes : **les problèmes P (Polynomial time)** et **les problèmes NP (Non-deterministic Polynomial time).**

Généralement la complexité d'un problème dépend de la taille du problème et du temps d'exécution nécessaire pour le résoudre par un algorithme. La taille du problème représente le nombre de variables dédiées à ce problème ainsi que le nombre de valeurs nécessaires pour chaque variable.

Le temps d'exécution est exprimé par le temps que l'algorithme utilise pour trouver la solution à un problème, autrement dit, il représente le nombre d'instructions parcourues par l'algorithme pour aboutir à la solution.

Classe P

Les problèmes de cette classe sont résolus par un algorithme en un temps polynomial par rapport à la taille du problème. Par exemple si le problème est de taille n donc le

temps d'exécution doit être $\leq O(n)$. Dans cette classe, on trouve les problèmes considérés comme étant faciles.

Classe NP

La classe NP comprend les problèmes qui peuvent être résolus en temps polynomial par une machine de Turing non déterministe. Autrement dit, elle englobe les problèmes pour lesquels la validité peut être vérifiée en temps polynomial, ce qui inclut tous les problèmes de la classe P (Cook, 2021).

1.3 Optimisation combinatoire

Definition 1 *Un problème d'optimisation combinatoire consiste à chercher le minimum $\tilde{x}$ d'une fonction f, le plus souvent à valeurs entières ou réelles, sur un ensemble fini S, tel que $f(\tilde{x}) = f(\tilde{x})$*

Où f la fonction objectif.

Une caractéristique des problèmes d'optimisations combinatoires (Papadimitriou and Steiglitz, 1998) (optimisation discrète) est qu'ils nécessitent l'exploration d'un grand ensemble de solutions, parcourant ainsi tout l'espace de recherche afin d'en extraire une solution optimale parmi un ensemble fini de solutions.

Un exemple d'un problème d'optimisation combinatoire est le voyageur de commerce, où la solution est codée par des entiers naturels qui indiquent le numéro de la ville à visiter.

Bien que les problèmes d'optimisation combinatoire sont souvent faciles à définir, ils sont généralement difficiles à résoudre. La résolution de problème d'optimisation combinatoire manipulent des algorithmes (Baghel et al., 2012) permettant la maximisation ou la minimisation d'un ou de plusieurs fonction objectifs sous ou sans contraintes.

1.3.1 Optimisation combinatoire sans contraintes

Pour les problèmes d'optimisations sans contraintes, la recherche de la solution optimale dépend de la (les) mesure(s) de la (les) fonction(s) objectif(s). Il peut s'agir soit d'un problème mono-objectif soit d'un problème multi-objectif.

1.3.2 Optimisation combinatoire sous contraintes

Un problème d'optimisation combinatoire sous contraintes en plus de sa (ses) fonction (s) objectif (s) à optimiser, est soumis à un certain nombre de contraintes qui doivent être satisfaites. Les contraintes se présentent sous deux catégories : les contraintes d'égalité et les contraintes d'inégalité. Selon la présence ou l'existence de l'un des types de contraintes on distingue deux types de problèmes :

1. Un problème d'optimisation sous forme canonique est caractérisé par sa fonction objectif et ses contrainte d'inégalité.

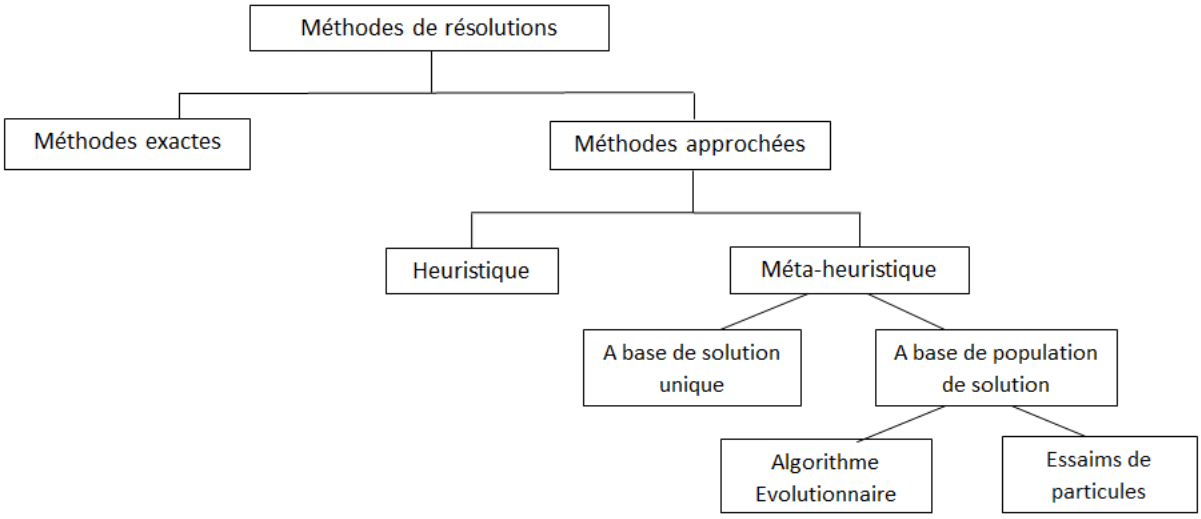


FIGURE 1.1 – Classification de méthodes de résolutions.

2. Un problème sous forme standard est caractérisé par sa fonction objectif et ses contraintes d'égalité.

Le passage de la forme canonique à la forme standard se fait par la transformation des contraintes d'inégalités ou contraintes d'égalités par l'ajout des variable d'écart (Bazaraa et al., 2011).

Un problème d'optimisation combinatoire sous contraintes est défini mathématiquement par la fonction 1.3 :

$$\begin{aligned} &\min f(x) \\ g_i(x) <= 0 \qquad &i = 1, ..., p \\ h_j(x) = 0 \qquad &j = 1, ..., m \end{aligned} \tag{1.3}$$

Où $g(x) <= 0$ est une contrainte d'inégalité, p est le nombre de contraintes d'inégalité et $h(x)$ une contrainte d'égalité et m le nombre de contraintes d'égalité.

Pour le cas de notre étude, nous nous concentrerons sur les problèmes d'optimisation combinatoire sous contraintes.

1.4 Méthodes de résolutions des problèmes d'optimisation

Afin de trouver une solution optimale au problème d'optimisation combinatoire, la litterature (Bertsekas, 1997) (Golberg, 1989) (Pochet and Wolsey, 2006) (Aarts et al., 1987)(Clerc, 2010) propose beaucoup de méthodes de résolution qu'elle classe généralement en deux catégories principales : **les méthodes exactes** et **les méthodes approchées**. Certaines propositions ont combiné les deux méthodes de résolution pour obtenir de méthodes hybrides.

1.4.1 Les méthodes exactes

Les méthodes exactes (Pochet and Wolsey, 2006) parcourent tout l'espace de recherche jusqu'à ce qu'elles trouvent une meilleure solution, Elles sont efficaces pour résoudre des problèmes appartenant à la **classe P**. Cependant, les algorithmes de ce type sont gourmands en temps et en mémoire pour les problèmes de taille importante.

Parmi ces méthodes, citons les plus connues :

1. **La méthode Branch-and-Bound** : Elle fonctionne en divisant récursivement l'espace de recherche en sous-problèmes plus petits en utilisant des contraintes supplémentaires. A chaque étape, elle évalue la valeur de la solution courante et la compare à celle de la meilleure solution connue. Si la valeur de la solution courante est supérieure à celle de la meilleure solution connue, le sous-problème est abandonné. sinon, le sous-problème est résolu récursivement jusqu'à ce qu'une solution optimale soit trouvée.
2. **la programmation dynamique** : Est une méthode de résolution de problèmes consistant à décomposer un problème complexe en sous-problèmes plus simples, puis à utiliser les solutions de ces sous-problèmes pour résoudre le problème global.
3. **L'algorithme de retour arrière** : Est une méthode pour résoudre des problèmes en explorant toutes les possibilités d'une solution. Il consiste à générer tous les sous-ensembles possibles d'une solution, en testant chaque option à chaque étape, et en revenant en arrière si l'option choisie ne conduit pas à une solution valide.

1.4.2 Les méthodes approchées

Sont plus pratiques dans le cas où l'on cherche une solution de bonne qualité en peu de temps, mais elles ne garantissent pas l'optimalité de la solution. Le point positif majeur des méthodes approchées est qu'elles sont dotées de mécanismes de diversification qui leurs permettent d'éviter de tomber dans des optima locaux. Les algorithmes de cette classe (Vazirani, 2001) sont divisés en deux catégories : les méthodes heuristiques dédiées à un problème donné et les méta-heuristiques basées sur une solution unique ou sur une population de solutions.

Heuristiques

Les principaux avantages des algorithmes heuristiques sont que ces algorithmes sont (souvent) conceptuellement plus simples et (presque toujours) beaucoup moins coûteux en termes de calcul que les algorithmes exacts. Parmi les méthodes heuristiques, on distingue les heuristiques constructives et les méthodes de recherche locale.

Méta-heuristique

(Talbi, 2009) décrit la méta-heuristique comme une stratégie de recherche itérative qui guide le processus sur l'ensemble de l'espace de recherche dans l'espoir de trouver la solution optimale. Cette classe d'algorithme comprend **les méta-heuristiques à solution unique** et **les méta-heuristiques à population**.

a- **Méta-heuristique à solution unique** : Les approches à solution unique se concentrent sur la modification et l'amélioration d'une seule solution candidate, les méta-heuristiques à solution unique comprennent le recuit simulé, la recherche locale itérée, la recherche par voisinage variable et la recherche locale guidée.

1. **Recuit simulé** (Simulated annealing) est un algorithme d'optimisation stochastique inspiré du processus de recristallisation de la matière. Le fonctionnement de base de l'algorithme consiste à générer une solution aléatoire et à l'améliorer en utilisant une probabilité de transition dépendant de la différence d'énergie entre la solution actuelle et la solution proposée. Il utilise une température initiale élevée qui est progressivement abaissée pour diriger la recherche vers les solutions de meilleure qualité (Kirkpatrick et al., 1983) .
2. **la recherche par voisinage** (Variable Neighborhood Search (VNS)) est une métaheuristique qui explore diverses structures de voisinage en utilisant des opérateurs de perturbation et d'optimisation. Elle alterne entre les voisinages de manière cyclique, en appliquant des opérateurs de perturbation suivis d'une recherche locale. La VNS est dynamique, passant de voisinages plus locaux à des voisinages plus perturbateurs. Elle combine intensification (exploration de voisinages locaux) et diversification (exploration de voisinages éloignés) pour trouver des solutions de haute qualité (Hansen and Mladenovic, 2003).

b- **Méta-heuristique à Population** : Les approches basées sur la population maintiennent et améliorent plusieurs solutions candidates, en utilisant souvent les caractéristiques de la population pour guider la recherche.

Dans la catégorie des méta-heuristiques à population, on trouve les algorithmes évolutionnaires (AEs).

1.5 Les algorithmes évolutionnaires

Les algorithmes évolutionnaires (AEs) sont conçus pour résoudre des problèmes d'optimisation combinatoire, ils s'inspirent de l'évolution naturelle. L'idée d'appliquer le principe de Darwin aux problèmes d'optimisation remonte à 1940. Il existe quatre variantes principales d'AEs (Eiben and Smith, 2015) : la **programmation évolutionnaire**, introduite par Fogel, la **stratégie évolutionnaire**, les **algorithmes génétiques**, introduits par Holland (Holland, 1975) et vulgarisés par Goldberg (Golberg, 1989), et la **programmation génétique**.

Les points fondamentaux communs qui déterminent leurs caractéristiques sont décrits dans (Calégari et al., 1999) et (Eiben and Smith, 2015), qui sont les suivantes :

`Codage de la solution` : Il existe de nombreuse façon de coder les individus (Chaouche and Boulif, 2019). Le procédé de codage est principalement déterminé par les informations qui doivent être échangées entre les individus, et ces informations dépendent du problème considéré.

`La population` : En AEs, on appelle population l'ensemble des solutions (candidats) générées de manière aléatoire ou par l'intermédiaire d'un algorithme. La taille de la population peut être soit rester constante, soit évoluer au cours de son évolution. Les AEs visent à améliorer les individus en échangeant des informations via des opérateurs spécifiques, adaptés à chaque AE, pour trouver le meilleur individu de la population, évalué par

une fonction objectif. Le changement d'information permet de générer d'autres individus appelés progéniture (fils) à partir des individus parents ou de modifier ceux qui existent déjà (les parents).

`La sélection` : Les AEs cherchent, au cours du processus d'évolution, à améliorer la population grâce aux échanges d'informations qui s'effectuent entre les individus. Mais la question la plus importante qui se pose est selon quel critère s'effectue le choix (la sélection) des individus.

`Évolution de la population` : L'évolution de la population s'effectue à chaque itération du processus d'évolution, ce qu'on appelle une *génération*.

`Évaluation de la fonction objectif` : Les AEs dépendent de la mesure de la fonction objectif assignée aux individus, ce qui permet d'évaluer les individus. Elle est appelée fonction *d'adaptation* ou fonction de *fitness*.

`Perturbation` L'un des principaux problèmes rencontrés dans l'optimisation combinatoire est la convergence prématurée de la solution vers un optimum local. Afin d'éviter ce problème, les AEs introduisent un peu de perturbation dans la population. Par exemple, pour les algorithmes génétiques, l'opérateur de mutation permet de générer une perturbation aléatoire de certains individus.

1.5.1 Fonctionnement des AEs

Le principe de fonctionnement des AEs est le même (Eiben and Smith, 2015). Étant donné une population d'individus créée au hasard, chaque individu (candidat) est caractérisé par une liste d'informations (pour un problème d'optimisation combinatoire, exprimée par des entiers). Étant donné une fonction objectif, qui attribue à chaque individu une mesure de qualité, les meilleurs candidats sont choisis pour ensemencer la prochaine génération en appliquant un croisement et/ou une mutation. Le croisement est un opérateur appliqué à deux ou plusieurs candidats sélectionnés (les parents), produisant un ou plusieurs nouveaux candidats (les enfants). La mutation est appliquée à un candidat, produisant ainsi un nouveau candidat. L'exécution des opérations de croisement et de mutation sur les parents conduit à la création d'un ensemble de nouveaux candidats (la progéniture). Ce processus peut être répété jusqu'à ce qu'un candidat avec une qualité suffisante (une solution) soit trouvé ou qu'une limite de calcul précédemment définie soit atteinte.

Le fonctionnement des AEs est présenté par l'algorithme 1 :

```
[1] Initialisation de la population de manière aléatoire;
[2] Évaluer chaque individu;
    repeat
[3] |  Sélectionner les parents;
[4] |  Croiser les parents deux à deux;
[5] |  Muter les progénitures;
[6] |  Évaluer les nouveaux candidats;
[7] |  Sélectionner les nouveaux candidats pour la nouvelle génération;
    until nombre de générations;
```

Algorithme 1 : Fonctionnement des AEs

Dans ce travail, nous nous sommes intéressés aux algorithmes génétiques.

1.6 Algorithmes génétiques

L'algorithme génétique (AG), proposé en 1975 par J. Holland (Holland, 1975) et vulgarisé ensuite par Goldberg (Golberg, 1989), est une technique d'optimisation méta-heuristique inspirée du principe de sélection naturelle de Darwin, basée sur la "survie des plus forts". Les individus les plus forts ont plus de chances de contribuer à la production de nouveaux individus (la progéniture) que les plus faibles, qui peuvent même ne pas contribuer du tout. Les AG ne garantissent pas de trouver la solution optimale du problème, cependant, les solutions trouvées se situent généralement à des niveaux acceptables.

Les AG fonctionnent avec une population d'individus où chaque individu est représenté par un ensemble de valeurs appelés gènes. L'ensemble de tous les gènes d'un individu est appelé chromosome, qui est fondamentalement une solution candidate. Les individus sont évalués à l'aide d'une fonction d'adaptation, et ceux qui ont une bonne valeur de fonction objectif sont sélectionnés.

Les descendants sont produits par recombinaison, ce qui leur permet d'hériter des caractéristiques de chacun des parents, et par mutation, qui peut également leur conférer des caractéristiques tout à fait nouvelles.

1.7 Fonctionnent des Algorithme Génétiques

Pour implémenter un algorithme génétique, les aspects suivants doivent être définis (Boulif and Atif, 2006) :

1. Un codage des individus pour représenter les solutions du problème,
2. Une méthode qui génère la population initiale,
3. Une fonction d'adaptation qui associe une valeur de fitness à chaque individu de la population,
4. Les opérateurs génétiques tels que la sélection, le croisement et la mutation,
5. Les paramètres de contrôle tels que la taille de la population, le taux de croisement, le taux de mutation et le nombre de générations pour le critère d'arrêt.

1.7.1 Codage

Afin de résoudre un problème il faut d'abord coder convenablement les solutions de l'algorithme génétique. En général, il existe plusieurs méthodes de codage des solutions (Chaouche and Boulif, 2019). Les plus fréquemment utilisées sont :

— **Codage binaire :** Le codage binaire était le premier à être utilisé dans le domaine des AGs. Les solutions sont représentées sous forme de séquences binaires, où chaque bit correspond à un attribut de la solution. Il présente plusieurs avantages : Un ensemble de symboles minimum de 0 et 1, permettant de coder facilement divers types d'objets.

— **Codage entiers :** Les solutions sont représentées par des nombres entiers. Chaque nombre entier peut correspondre à une caractéristique ou à une décision dans le problème.
— **Codage réel :** Les solutions sont représentées par des nombres réels.

1.7.2 Génération de la population initiale

Cette phase initiale affecte largement la diversité et la qualité des solutions explorées par l'AG au cours de son processus d'optimisation. Une pratique courante pour créer cette population consiste à générer des individus de manière aléatoire, ce qui permet une large exploration initiale de l'espace des solutions. Il est également possible d'adopter un algorithme déterministe pour générer une population initiale sur la base de critères spécifiques, tels que la connaissance préalable du problème ou une heuristique spécialisée. Cette approche peut parfois conduire à une convergence plus rapide en orientant dès le départ la recherche vers des régions de l'espace de solution susceptibles de contenir des solutions optimales.

1.7.3 Fonction d'évaluation

La fonction d'évaluation ou d'adaptation détermine la capacité d'un individu à participer à la prochaine génération en évaluant son aptitude à résoudre le problème donné. Elle permet de déterminer la qualité de chaque solution potentielle. En effet, l'algorithme génétique vise principalement à maximiser la fonction d'évaluation, cherchant ainsi à améliorer continuellement les solutions au fil des générations.

1.7.4 Les opérateurs génétiques

Pour réaliser le processus de reproduction d'un AG, on utilise différents opérateurs génétiques. Dans les paragraphes suivants, nous présentons les opérateurs les plus utilisés : les opérateurs de sélection, de croisement et de mutation.

Sélection

Pendant chaque génération, une partie de la population existante est sélectionnée pour engendrer une nouvelle génération. Les individus sont choisis en fonction de leur valeur de fitness, les solutions les plus adaptées ont généralement plus de chances d'être sélectionnées.

Certaines méthodes de sélection évaluent la valeur de chaque solution et choisissent les meilleures. D'autres méthodes évaluent uniquement un échantillon aléatoire de la population, ce qui rend le processus moins fastidieux. La plupart des fonctions de sélection sont conçues de manière à sélectionner une petite proportion des solutions les moins adaptées. Cela permet de maintenir une grande diversité dans la population et d'éviter une convergence prématurée vers des solutions médiocres. Parmi les méthodes les plus connues, citons :

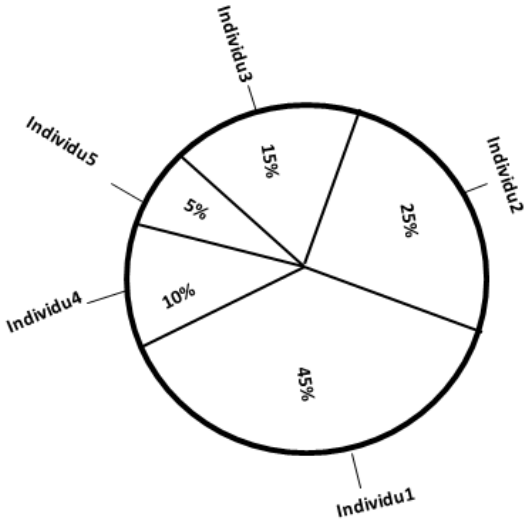


FIGURE 1.2 – Sélection par roue de loterie.

1. **La sélection par roue de loterie** : les chromosomes ont une probabilité p d'être choisis en fonction de leur valeur de fitness relative. Pour un chromosome i dans la population, la probabilité de sélection pi est calculée par l'équation 1.4 :

$$p_i = \frac{fitness_i}{\sum_{j=1}^{n} fitness_j} \tag{1.4}$$

   Où $fitness_i$ est la valeur de fitness du chromosome i, et n est le nombre de chromosomes dans la population.

   Les solutions candidates ayant une meilleure fitness auront moins de chances d'être éliminées. Néanmoins, avec la sélection proportionnelle à la fitness, les solutions les plus faibles ont une chance de survivre au processus de sélection, est un avantage car, bien qu'une solution puisse être faible, elle peut inclure certains composants qui pourraient s'avérer utiles après le processus de recombinaison. L'analogie avec une roulette peut être envisagée en imaginant une roue de roulette dans laquelle chaque solution candidate occupe un secteur de la roue. La taille des secteurs est proportionnelle à la probabilité de sélection de la solution. Sélectionner n chromosomes dans la population équivaut à jouer n parties sur la roulette, car chaque candidat est tiré au sort de manière indépendante, (voir la figure 1.2, illustrant le processus de sélection par roulette)
2. **Sélection par tournois** : La sélection par tournoi consiste à prendre k chromosomes aléatoires dans la population. et à choisir le plus apte. La pression de sélection peut être facilement ajustée en modifiant la taille du tournoi. Si la taille du tournoi est plus grande, les individus faibles ont moins de chances d'être sélectionnés (voir figure 1.3).
3. **Elitisme** : Elle garantit que le ou les meilleurs individus de la population survivent de génération en génération. La stratégie élitiste la plus élémentaire copie le meilleur élément de la population actuelle vers la population suivante.
4. **Sélection par rang** : Un rang numérique est attribué à chaque individu de la population en fonction de sa fitness, et la sélection est basée sur ce rang plutôt que sur des différences absolues de fitness. L'avantage de cette méthode est qu'elle

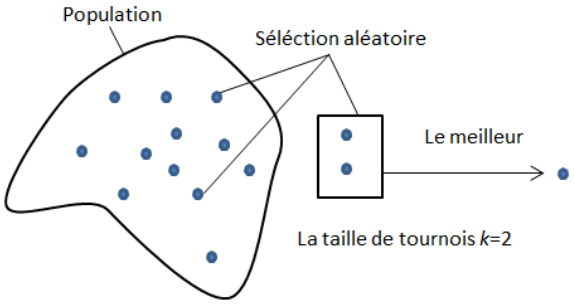


FIGURE 1.3 – Sélection par tournoi.

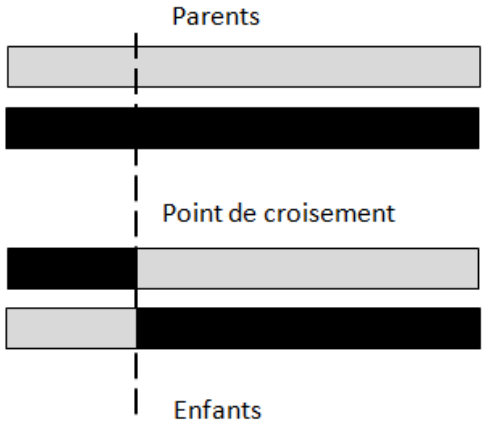


FIGURE 1.4 – Exemple de croisement un point.

permet d'éviter que des individus très aptes ne deviennent rapidement dominants au détriment d'individus moins aptes, ce qui réduirait la diversité génétique de la population.

Croisement

Le croisement est un opérateur génétique qui combine (accouple) deux chromosomes (parents) pour produire de nouveaux chromosomes (progéniture). L'idée derrière le croisement est que le nouveau chromosome peut être meilleur que les deux parents s'il prend les meilleures caractéristiques de chacun des parents. Le croisement se produit pendant l'évolution selon une probabilité de croisement définissable par l'utilisateur. Il existe de nombreuses techniques de croisement, parmi lesquels :

1. **Croisement simple avec un point :** Le croisement simple avec un point consiste à choisir aléatoirement un seul point sur les chromosomes des deux parents et à échanger les segments de chaque parent à partir de ce point pour générer deux descendants (voir figure : 1.4).
2. **Le croisement à deux points** : Le croisement à deux points sélectionne deux points de séparation aléatoires à partir des deux parents, ce qui génère trois segments dans chaque parent, comme illustré dans la figure "1.5.
3. **Croisement uniforme** : Le croisement uniforme peut être considéré comme un croisement multi-points, où le nombre de coupures est déterminé aléatoirement lors de l'opération. La figure 1.6 illustre cette technique.

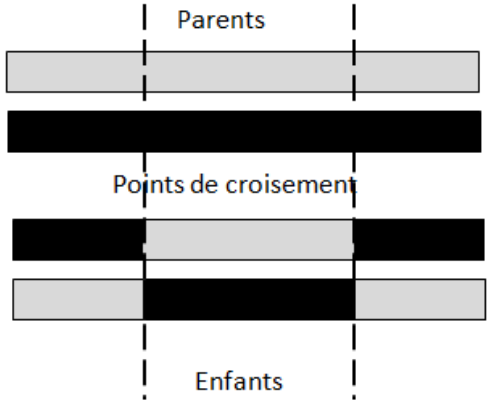


FIGURE 1.5 – Exemple de croisement deux points.

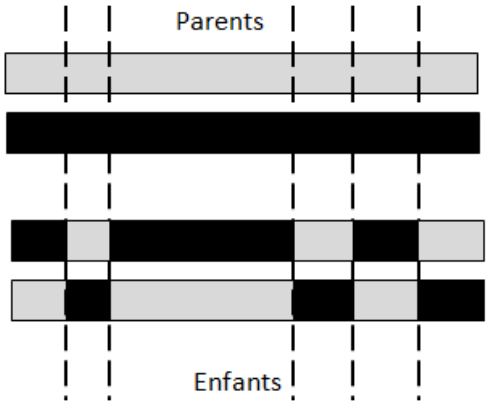


FIGURE 1.6 – Exemple de croisement uniforme.

Mutation

La mutation est un opérateur utilisé pour explorer un espace de solutions plus large, afin d'éviter de converger vers un minimum local non adéquat. Elle sélectionne aléatoirement des individus de la population avec une probabilité suffisamment faible p_m, puis choisit aléatoirement un gène du chromosome et le modifie.

1.7.5 Conditions d'arrêt

Divers critères sont proposés, y compris :
— L'algorithme peut être arrêté après un nombre prédéfini d'itérations.
— L'exécution de l'algorithme peut être interrompue dès lors que la disparité entre deux chromosomes est en dessous d'un seuil préétabli.
— L'algorithme peut être arrêté lorsque la durée prédéfinie est atteinte.

1.8 Conclusion

Dans ce chapitre, nous avons présenté les problèmes d'optimisation et leurs diverses classifications, ainsi que les méthodes de résolution disponibles, telles que les méthodes exactes et les méthodes approchées.

Un intérêt particulier a été accordé aux algorithmes génétiques, qui font partie des Algorithmes Évolutionnaires et sont largement utilisés pour résoudre les problèmes d'optimisation combinatoire.

Résoudre un problème d'optimisation combinatoire consiste à choisir la meilleure solution parmi une ou plusieurs solutions candidates, en prenant en compte les contraintes liées à ce problème. La sélection de cette meilleure solution entraîne parfois le rejet de certains individus de la population (ceux qui violent les contraintes). Cependant, il s'avère que certains éléments rejetés pendant le processus de sélection peuvent être intéressants, car ils aident les AG à converger plus rapidement vers la solution optimale. Afin de faire face à cette situation, la gestion des contraintes a été proposée.

Dans le prochain chapitre, nous nous concentrerons sur les méthodes de gestion des contraintes

Chapitre 2

Gestion de contraintes

2.1 Introduction

Ce chapitre aborde une notion très importante pour résoudre les problèmes d'optimisation combinatoire sous contraintes, qui est la notion de gestion de contraintes. Nous présentons les classifications proposées dans la littérature et les différentes techniques de gestion de contraintes existantes.

2.2 Gestion des contraintes

Les algorithmes génétiques (AGs), fondamentalement proposés pour des problèmes d'optimisation sans contraintes, sont confrontés à des situations où les problèmes d'optimisation comportent des contraintes qu'il est nécessaire de respecter pendant le processus de recherche de la meilleure solution.

La présence de contraintes divise l'espace de recherche S en deux sous ensemble disjoints : l'ensemble réalisable F qui contient les individus réalisables (ceux qui respectent toutes les contraintes), et l'ensemble irréalisable U, qui contient les individus irréalisables (ceux qui ne satisfont ou moins une contraintes) (voir figure 2.1). Le problème qui se pose pendant le processus de recherche de la meilleure solution est de savoir comment gérer les individus qui se trouvent dans l'ensemble irréalisable. La solution la plus simple consiste à rejeter complètement ces individus, car un individu de la population qui viole une contrainte se verra attribuer une mauvaise mesure de qualité et aura une forte probabilité d'être éliminé par le processus de sélection, sans même considérer à quel point cet élément rejeté est proche de l'optimum (voir figure 2.1). Une autre solution simple est de conserver ces solutions irréalisables. Dans ce cas, l'algorithme de recherche peut converger vers une solution irréalisable, ce qui n'est pas souhaitable. Par conséquent, la meilleure solution consiste à essayer de trouver un compromis entre le rejet et la rétention de l'individu.

Il peut être intéressant de conserver, tout en les pénalisant, les individus irréalisables car ils peuvent permettre de générer des individus réalisables de bonne qualité. Étant donné que les opérateurs associés aux AGs sont aveugles aux contraintes (Craenen et al., 2001), la littérature a proposé d'équiper les AGs de techniques de gestion de contraintes (Michalewicz, 1995) (Coello, 2002)(Takahama and Sakai, 2006) (Rahimi et al., 2023).

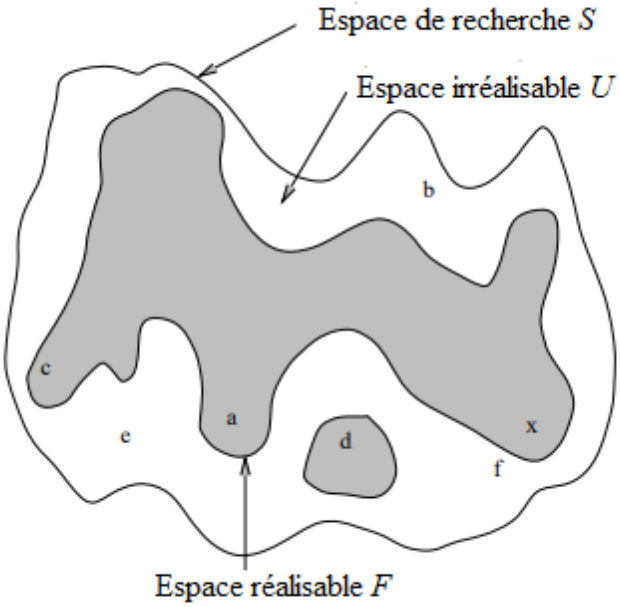


FIGURE 2.1 – L'espace de recherche (Michalewicz, 1995).

2.3 Classification des méthodes de gestion de contraintes

Les techniques de la gestion de contraintes ont été regroupées en deux approches (Coello, 2002; Barbosa and Lemonge, 2008) :

— Approche indirecte : regroupe les approches basées sur la fonction de pénalité, qui visent à transformer le problème sous contraintes en un problème sans contraintes en utilisant une fonction de pénalité $p(x)$. Pour évaluer les solutions irréalisables, une modification de la fonction objectif par l'application de la fonction de pénalité est utilisée dans le but de réduire l'adaptation de ces solutions irréalisables (dans le cas d'un problème de minimisation). La fonction de pénalité est définie en fonction du nombre de contraintes violées ou de la distance entre la solution irréalisable et l'espace réalisable F. Dans cette catégorie, on trouve pénalité de mort, pénalité statique, pénalité dynamique et pénalité adaptative (Barbosa and Lemonge, 2008; Coello, 2002).

— Approche directe : regroupe les approches suivantes :

1. Méthodes basées sur la préservation de la réalisabilité des solutions : développées par Michalewicz (Michalewicz and Nazhiyath, 1995), ces méthodes visent à générer uniquement des solutions réalisables.
2. Méthodes basées sur les algorithmes de réparation,
3. Méthodes basées sur la séparation de la fonction objectif et des contraintes,
4. Méthodes hybrides : Ces approches combinent l'une des approches précédentes avec une technique d'optimisation numérique pour traiter les contraintes.

Parmi toutes ces méthodes de gestion de contraintes existantes, la méthode de pénalité est la plus utilisée.

2.3.1 Méthodes de pénalités

La méthode de pénalité est la méthode la plus utilisée. Les fonctions de pénalité ont été proposées par Richard Courant dans les années 1940 (Courant, 1943) et plus tard ont été élargis par Carroll et Fiacco et McCormick (Carroll, 1961)

L'idée de la fonction de pénalité est de transformer un problème d'optimisation sous contrainte à un problème d'optimisation sans contrainte en ajoutant (ou soustrayant en cas de maximisation) une fonction de pénalité $p(x)$. Cette fonction de pénalité sera basée sur la quantité de violation de contrainte (Coello, 2002) ou bien par la distance entre la solution irréalisable et la région réalisable (Barbosa and Lemonge, 2008).

La méthode de pénalité transforme le problème d'optimisation sous contraintes à un problème d'optimisation sans contraintes en deux manières.

1. La forme additive consiste à ajouter une fonction de pénalité à la fonction objectif, de telle sorte que si la solution est réalisable, sa valeur de pénalité sera égale à zéro.

$$F(x) = \begin{cases} f(x) & \text{si x est réalisable} \\ f(x) + p(x) & \text{sinon} \end{cases} \tag{2.1}$$

2. La forme multiplicative, consiste à amplifier la fonction objectif par un facteur de pénalité positif, qui est égale à 1 en cas d'une solution réalisable est $p(x) > 0$ en cas d'une solution irréalisable.

$$F(x) = \begin{cases} f(x) & \text{si x est réalisable} \\ f(x) \times p(x) & \text{sinon} \end{cases} \tag{2.2}$$

Dans les deux cas additive et multiplicative, $F(x)$ est la fonction de fitness.

La forme additive est la plus utilisée (Eiben and Smith, 2015; Helio J.C. Barbosa and Bernardino, 2015). Pour cette forme, deux cas sont à distinguer : la technique **intérieure** et la technique **extérieure**.

Dans la technique **intérieure**, elle consiste à commencer à partir d'une population où tous ses individus sont réalisables, tandis que dans la technique **extérieure** (la plus utilisée), il n'est pas nécessaire de commencer avec une population d'individus réalisables. On "commence par une solution où ses individus ne sont pas réalisables puis se diriger vers la région réalisable", car dans de nombreuses applications pour lesquelles les EAs sont destinés, le problème de la recherche d'une solution réalisable est lui-même NP-hard (Smith et al., 1997), donc il est déjà difficile de trouver une solution réalisable.

La formule générale d'une fonction de pénalité extérieure est la suivante (Coello, 2002) (équation 2.3) :

$$F(x) = f(x) + [\sum_{i=1}^{n} r_i \times G_i + \sum_{j=1}^{p} c_j \times L_j]. \tag{2.3}$$

Où $F(x)$ est la fonction adaptative, G_i et L_j sont les fonctions de contraintes, r_i et c_j sont des constantes positives appelées les facteurs de pénalité. Les fonctions G_i et L_j sont définies par les équations 2.4 et 2.5 respectivement.

$$G_i = \max[0, g_i(x)^{\beta}] \tag{2.4}$$

$$L_j = |h_j(x)|^\gamma \tag{2.5}$$

β et γ sont égaux à 1 ou 2.

Type de fonctions de pénalités

1. **pénalité de mort** : c'est la technique la plus simple, elle consiste à rejeter les solutions irréalisable de la population, car si une solution viole une contrainte la valeur 0 sera assignée à sa valeur de fitness.
2. **pénalité statique** : D'une manière générale, la pénalité statique correspond à des approches dans lesquelles les facteurs de pénalité sont des valeurs constantes tout au long du processus de recherche de solution. Les valeurs des facteurs de pénalité ne dépendent pas du nombre de générations actuel.
   — `La méthode de Homaifar et al` (Homaifar et al., 1994) : Cette méthode propose de créer l niveaux de violation pour chaque m contraintes, et pour chaque contrainte, un coefficient de pénalité $R_{i,j}, i = 1...l, j = 1...m$ sont associées à chaque niveau, de sorte que le niveau de violation le plus élevé aura le plus grand coefficient. La nouvelle fonction d'adaptation est donnée par l'équation 2.6.

$$F(x) = f(x) + \sum_{i=1,j=1}^{l,m} R_{i,j} \max[0, g_i(x)]^2 \tag{2.6}$$

   La pénalité statique est simple, mais il n'est pas judicieux de garder les mêmes facteurs de pénalité tout au long du processus d'évolution, car ces paramètres dépendent du problème. Par exemple, l'approche proposée par Homaifar et al dans (Homaifar et al., 1994) nécessite un grand nombre de paramètres ($m(2l+1)$ paramètres) et Michalewicz dans (Michalewicz, 1995) indique que la qualité de la solution dépend des valeurs de paramètres.
   — **Kuri Morales** proposé dans (Morales and Quezada, 1998) une pénalité statique décrite par Coello dans (Coello, 2002) comme une approche intéressante. La fonction d'adaptation $F(x)$ est définie par l'équation 2.7.

$$fitness(x) = \begin{cases} f(x) & \text{x est réalisable} \\ K - \sum_{i=1}^{s} K/m & \text{sinon} \end{cases} \tag{2.7}$$

   Où s est le nombre de contraintes satisfaites, m est le nombre total de contraintes, et K est une constante qui prend une grande valeur s'il s'agit d'un problème à minimiser et une petite valeur s'il s'agit d'un problème à maximiser. Si un individu n'est pas réalisable, sa fitness F est égale à la valeur de sa fonction objectif. Tous les individus qui violent le même nombre de contraintes reçoivent la même pénalité, quelle que soit la distance qui les sépare de l'espace réalisable.
3. **pénalité dynamique** : Dans cette catégorie, les paramètres de la pénalité dynamique sont généralement déterminés en fonction du numéro de génération courante.

   Parmi les méthodes de pénalité dynamique, on peut citer :
   — **La méthode Joines and Houck**, Cette méthode (Joines and Houck, 1994) accroît la pénalité à mesure que l'on progresse dans les générations, les individus sont évalués à la génération t par l'éqaution 2.8.

$$F(x) = f(x) + (C + t)^\alpha \times \mathrm{SVC}(\beta, x) \tag{2.8}$$

C, α et β sont des constantes définies par l'utilisateur, SVC est défini par l'équation 2.9.

$$\mathrm{SVC}(\beta, x) = \sum_{i=1}^{n} D_i^{\beta}(x) + \sum_{j=1}^{p} D_j(x) \tag{2.9}$$

$$D_i(x) = \begin{cases} 0, & g_i(x) \leq 0 \\ |g_i(x)| & sinon \end{cases}$$

où $1 \leq i \leq n$.

$$D_j(x) = \begin{cases} 0, & -\epsilon \leq h_j(x) \leq \epsilon \\ |h_j(x)| & sinon \end{cases}$$

où $1 \leq j \leq p$.

4. **pénalité adaptative** : La pénalité adaptative (Helio J.C. Barbosa and Bernardino, 2015) ajuste automatiquement les valeurs de tous les paramètres impliqués en utilisant les retours du processus évolutif, sans intervention de l'utilisateur. On peut citer l'exemple de :

   — **Hadj-Alouane** (Bean and ben Hadj-Alouane, 1993) : Les individus sont évalués à l'aide de la fonction présentée par l'équation :

   $$F(x) = f(x) + \lambda(t)[\sum_{i=1}^{n} g_i^2(x) + \sum_{j=1}^{p} |h_j(x)|] \tag{2.10}$$

   $\lambda(t)$ est mis a jour à chaque génération t par l'équation 2.11 :

   $$\lambda(t+1) = \begin{cases} (1/\beta_1) \times \lambda(t), & cas1 \\ \beta_2 \times \lambda(t), & cas2 \\ \lambda(t) & sinon, \end{cases} \tag{2.11}$$

   En d'autres termes, la composante de pénalité $\lambda(t+1)$ pour la génération $t+1$ est diminuée si tous les meilleurs individus des k dernières générations étaient réalisables (cas 1) ou est augmentée s'ils étaient tous irréalisables (cas 2). Sinon s'il y a des individus réalisables et irréalisables à égalité comme meilleurs dans la population, alors la pénalité ne change pas.

   — **Pénalité adaptative basée sur la logique floue** proposée par Saha et al (Saha et al., 2014), la méthode basée sur les règles d'inférences floues de type *if .... then....* de type mamdani (Mamdani and Assilian, 1975). Saha et al. proposent de prendre en compte non seulement la fonction objectif et la violation des contraintes, mais aussi le pourcentage de solution réalisable rf défini par l'équation 2.14). De plus, ils proposent de normaliser les valeurs de la violation des contraintes $V(x)$ par l'équation 2.16 et les valeurs de la fonction objectif $f(x)$ par l'équation 2.15. Les valeurs antécédentes de (FIS) sont rf, v_{norm} et f_{norm} (équation 2.15) et la valeur conséquente est $p(x)$ la valeur de la pénalité. La fonction d'adaptation est définit par équation 2.12 :

   $$F(x) = p(x) + d(x). \tag{2.12}$$

   $d(x)$ est définit par l'équation 2.13 :

$$d(x) = rf \times f_{norm}(x) + (1 - rf) \times v_{norm}(x). \tag{2.13}$$

$$rf = \frac{\text{nombre total des solutions réalisables}}{\text{la taille de la population}}. \tag{2.14}$$

$$f_{norm}(x) = \frac{f(x) - f_{min,k}}{f_{max,k} - f_{min,k}} \tag{2.15}$$

$f(x)$ est la fonction objectif, $f_{max,k}$ et $f_{min,k}$ sont réspectivement le maximum et le minimum de la valeur de la fonction objectif à la génération k.

$$v_{norm}(x) = \frac{v(x) - v_{min,k}}{v_{max,k} - v_{min,k}} \tag{2.16}$$

$v(x)$, le nombre de contraintes violées, $v_{max,k}$ et $v_{min,k}$ sont respectivement le maximum et le minimum de contraintes violées à la génération k.

2.3.2 Méthodes basées sur la préservation de la réalisabilité des solutions

Les méthodes basées sur la préservation de la réalisabilité des solutions tentent de rendre un individu irréalisable réalisable en appliquant des opérateurs génétiques spéciaux. Elles sont utilisées dans les problèmes d'optimisation combinatoire dans lesquels il est extrêmement difficile de trouver des solutions réalisables.

— **GENOCOP** : L'algorithme génétique pour l'optimisation numérique des problèmes sous contraintes, développé par Zbigniew Michalewicz (Zbigniew, 1996). GENOCOP, élimine les contraintes d'égalité ainsi qu'un nombre égal de variables du problème. Cela réduit une partie de l'espace de recherche et simplifie le problème pour l'AG. Les contraintes restantes sont des inégalités linéaires, formant un ensemble convexe recherché par l'AG. GENOCOP tente de trouver une solution initiale réalisable en échantillonnant la région réalisable. Si cela n'est pas possible après plusieurs tentatives, l'utilisateur est invité à fournir un tel point de départ. La population initiale sera alors composée de copies identiques de ce point de départ. Les opérateurs génétiques adoptés effectuent des combinaisons linéaires d'individus pour garantir que leur progéniture soit également réalisable.

2.3.3 Méthodes basées sur les algorithmes de réparation

Les Méthodes basées sur les algorithmes de réparation consistent à réparer un individu irréalisable pour le rendre réalisable. Les algorithmes de réparation peuvent être utilisés soit pour l'évaluation seulement, soit pour remplacer (avec une certaine probabilité) l'individu original dans la population.

— **GENOCOP III** (Michalewicz and Nazhiyath, 1995) : Utilise également des algorithmes de réparation. L'idée est d'incorporer le système GENOCOP original (Zbigniew, 1996) (qui ne traite que les contraintes linéaires) et de l'étendre en maintenant deux populations distinctes, où les résultats d'une population influencent

les évaluations des individus dans l'autre population. La première population est constituée des points de recherche qui satisfont les contraintes linéaires du problème ; la réalisabilité (au sens des contraintes linéaires) de ces points est maintenue par des opérateurs spécialisés. La deuxième population est constituée de points de référence réalisables. Comme ces points de référence sont déjà réalisables, ils sont évalués directement par la fonction objectif, tandis que les points de recherche sont "réparés" pour être évalués.

2.3.4 Méthodes basées sur la séparation de la fonction objectif et des contraintes

La violation des contraintes et la fonction objectif sont optimisées séparément. Ces méthodes adoptent un ordre lexicographique, dans lequel la violation de la contrainte précède généralement la fonction objectif.

— **Méthode de Runarsson et Yao** (Runarsson and Yao, 2000), Runarsson et Yao ont introduit une méthode de classement stochastique pour obtenir un équilibre entre la fonction objectif et la pénalité de manière stochastique. Un facteur de probabilité Pf est utilisé pour déterminer si la valeur de la fonction objectif ou la valeur de la violation de la contrainte détermine le rang de chaque individu. Bien que la méthode ait produit de très bons résultats pour $Pf = 0.45$, elle n'a fourni aucune preuve que $Pf = 0.45$ est un choix optimal.

2.4 Conclusion

Dans ce chapitre, les techniques de gestion des contraintes a été abordé. Ces techniques sont très importantes pour les AEs lorsqu'ils sont confrontés à des problèmes d'optimisation sous contraintes. Nous avons également présenté les méthodes existantes.

On peut en déduire que l'efficacité des AEs dépend de la façon dont les contraintes sont traitées et leur succès repose sur le choix de la technique appropriée.

Dans le chapitre suivant, un exemple d'un problème d'optimisation combinatoire sera exposé.

Chapitre 3

Problèmes fortement contraints et le partitionnement de graphe

3.1 Introduction

Dans ce chapitre, nous allons explorer la notion des problèmes fortement contraints. Nous passerons en revue les théories existantes à leur sujet et fournirons les différentes applications possibles. Nous examinerons également le partitionnement des graphes de manière plus approfondie, en abordant les concepts de base de la théorie des graphes et en détaillant les domaines d'application du Problème de Partitionnement de Graphe (PPG).

3.2 Problèmes fortement contraints

Les problèmes fortement contraints sont des problèmes d'optimisation NP-complet qui se retrouvent dans de nombreuses applications pratiques, telles que la planification de la production, la conception de réseaux, la gestion des ressources, etc. Ils nécessitent des heuristiques pour les résoudre. Les problèmes fortement contraints sont largement utilisés dans la pratique mais sont moins abordés dans la littérature. Peu de recherche académique a décrit ce type de problème.

— D'aprés Colorni et al. (1990) les problèmes fortement contraints sont caractérisés comme étant ceux pour lesquels une modification minimale d'une solution réalisable peut très souvent conduire à une solution irréalisable,
— D'aprés (Kim and Myung, 1997) ces problèmes sont décrits comme ayant de nombreuses contraintes non linéaires, dont la majorité nécessitent une heuristique lors de l'application des opérateurs évolutionnaires et ne garantit pas la convergence vers l'optimum,
— D'aprés (Myung and Kim, 1996) ce problème est abordé du point de vue de l'efficacité des calculs et de la précision des solutions. Il n'est pas évident de converger vers l'optimum, malgré le temps et les efforts considérables consacrés à l'atteindre.

3.3 Exemples de problèmes fortement contraints

Les problèmes fortement contraints sont des problèmes d'optimisation NP-complet et qui sont très liée à leurs contraintes. Résoudre un problème de ce type nécessite des heuristique ou méta-heuristiques.

Il existe de nombreux exemples de problème fortement contraints dans divers domaines tels que :

- **Problème de planification :** Planification des emplois du temps des employés où des étudiants, en tenant en compte des contraintes de temps, de disponibilité de ressource, etc...
- **Problème d'ordonnancement :** Par exemple dans une usine, l'ordonnancement de la production consiste à planifier les opérations de fabrication en tenant compte des contraintes de délais, des contraintes de disponibilité des ressources (machines, main-d'œuvre) et des contraintes de séquence de production.
- **Problème de transport :** Par exemple planification des itinéraires de livraison en tenant compte des contraintes de distance, de temps et de ressources.

Beaucoup de ces problèmes peuvent être modélisés par des graphes. Dans ce qui suit, nous présentons le problème de partitionnement de graphes.

3.4 Notion de graphes

On appelle graphe G le couple (V, E), où V est un ensemble fini non vide de sommets et E un ensemble d'arêtes tel que $E \subseteq \{(u, v)|u, v \in V\}$. Si $e = (u, v)$ est une arête de E, chacun des sommets u et v est appelé un sommet terminal de e.

On dit d'une arête $(u, v) \in E$ qu'elle est incidente à chacun des sommets terminaux $u, v \in V$, et u et v sont dits adjacents ou voisins par rapport à E.

Un graphe est dit simple s'il ne contient pas d'arêtes multiples ni de boucles. C'est-à-dire que chaque arête $e \in E$ avec des sommets d'extrémité u et v est unique $(u, v) = e = (v, u)$ et les sommets d'extrémité sont distincts $(u \neq v)$.

Un graphe complet est un graphe simple où tous les deux sommets distincts sont adjacents. Le graphe complet sur n sommets est noté $K_n = (V_n, E_n)$, c'est-à-dire $E_n = \{(u, v)|u, v \in V_n\}$ et $|V_n| = n$.

3.5 Problème de partitionnement de graphe

Le problème de partitionnement de graphe (PPG) est un modèle mathématique qui se pose dans plusieurs domaines d'application tels que la conception de circuits intégrés, la visualisation de réseaux (Kim et al., 2011; Kim and Kim, 2018). Il est considéré comme un problème combinatoire NP-complet (Kim and Kim, 2018; Boulif and Atif, 2006). Le PPG revient à décomposer un graphe $G = (V, E)$ où E un ensemble d'arêtes pondérées, en plusieurs sous-ensembles disjoints de sommets appelés clusters, de sorte que la somme des poids des arêtes dont les extrémités se trouvent dans différents clusters soit réduite au minimum.

PPG consiste à diviser le graphe $G = (V, E)$ en k sous-ensembles disjoints $P = p_1, p_2, ..., p_k$. On dit que P est une partition du graphe G si les partitions p_i satisfont les conditions suivantes 3.1 :

$$\begin{aligned} \forall i \in \{1, 2, \cdots, k\} : p_i \neq \phi \\ \bigcup_{i=1}^{k} p_i = V \\ \forall i, j \in \{1, 2, ..., k\}, i \neq j : p_i \bigcap p_j = \phi \end{aligned} \tag{3.1}$$

on définit le poids d'une partition P par l'équation 3.2

$$W(P) = \sum_{e \in E'} w(e) \tag{3.2}$$

Où $E' \subset E$ est le sous-ensemble des arêtes inter-cluster de P. Il s'agit des arêtes $(v_i, v_j) \in E$ telles que v_i et v_j appartiennent à deux clusters différents.

3.6 Fonction objectif

La fonction objectif à minimiser, définie par l'équation 3.3, est la somme des poids des arêtes inter-cluster.

$$W(p^*) = \min W(P) \tag{3.3}$$

La fonction objectif est transformée en une fonction de maximisation par l'équation 3.4.

$$W'(P) = B - W(P) \tag{3.4}$$

où P est une solution candidate (partition) et $B = \sum_{e \in E} W(e)$ est une borne supérieure de $W(P)$.

3.7 Contraintes

Le PPG consiste à diviser le graphe en partitions en minimisant la fonction objectif cité dans l'équation 3.3. Pour qu'une partition soit acceptable (réalisable), elle doit aussi respecter les contraintes suivantes :

1. Contraintes sur le nombre de clusters (Alouane and Boulif, 2023) : Soit k_{max}, k_{min} le nombre maximal et minimal de clusters, respectivement. Par conséquent, le nombre de clusters k doit satisfaire ce qui suit :

$$k_{min} \leqslant k \leqslant k_{max}$$

2. Contraintes sur la taille des clusters (Alouane and Boulif, 2023) : La taille des clusters ne doit pas dépasser une valeur donnée N. C'est-à-dire :

$$\forall p_i \in P (i = 1, 2, \cdots, k) |p_i| \leq N$$

3. Contraintes de cohabitation (Alouane and Boulif, 2023) : Soit CC l'ensemble des paires de sommets pour lesquelles nous savons qu'ils appartiennent au même cluster. Chaque partition réalisable doit satisfaire :

$$\forall (v_i, v_j) \in CC, \exists p \in P : v_i, v_j \in p$$

4. Contraintes de non-cohabitation (Alouane and Boulif, 2023) : Soit CNC l'ensemble des paires de sommets dont nous savons qu'ils sont dans des clusters différents. Ceci peut être matérialisé par :

$$\forall (v_i, v_j) \in CNC, \exists p \in P : v_i \in p \text{ and } v_j \notin p$$

3.8 Problème de partitionnement semi supervisé de graphes

Le PPG a plusieurs variantes, le problème de partitionnement semi-supervisé du graphe (PPSSG) est l'une des variantes les plus importantes de PPG (Chaouche and Boulif, 2019; Song et al., 2021), dans laquelle des informations partielles sur le contenu des clusters sont disponibles. Ces informations partielles peuvent être complétée par la présence des contraintes de cohabitation et de non-cohabitation.

3.9 Applications du partitionnement de graphe

Il existe de nombreux problèmes du monde réel qui peuvent être modélisés par des graphes et considérés comme des variantes de PPGs. Dans ce qui suit, nous présentons quelques domaines d'application.

Analyse des réseaux sociaux : Le principal objectif dans l'étude des réseaux sociaux est l'identification de la structure des communautés. Ce problème est résolu comme un PPG, consistant à diviser les nœuds des réseaux en clusters ou en communautés afin d'optimiser la similarité au sein des partitions et de maximiser l'hétérogénéité entre les clusters. Le nombre de clusters n'est pas connu à priori (Newman, 2013).

Circuits intégrés VLSI : Le VLSI (Very Large Scale Integration) est un circuit intégré à très grande échelle, composé de millions de portes logiques. Une porte est un composant de base d'un circuit intégré. Le partitionnement est utilisé pour faire face à la complexité croissante de la conception de VLSI (Kahng et al., 2011), en divisant le système en composants plus petits et plus faciles à gérer. Les portes logiques sont considérées comme des sommets et les interconnexions entre les portes logiques sont les arêtes (voir Figure 3.1).

La segmentation d'image : La segmentation d'image est un problème fondamental dans toutes les applications de traitement d'image. Le partitionnement de graphe est l'un des outils les plus attrayants pour diviser une image en plusieurs composantes (Chang et al., 2017) (Tolliver and Miller, 2006) . Les pixels sont désignés comme des sommets et s'il existe des similitudes entre les pixels, ils sont représentés comme des arêtes (voir Figure 3.2).

Le chemin le plus court : Les réseaux de transport et la recherche du chemin le plus court à partir d'une grande carte sont un sujet d'actualité de l'informatique. L'algorithme du plus court chemin calcule le plus court chemin entre deux destinations lorsqu'il

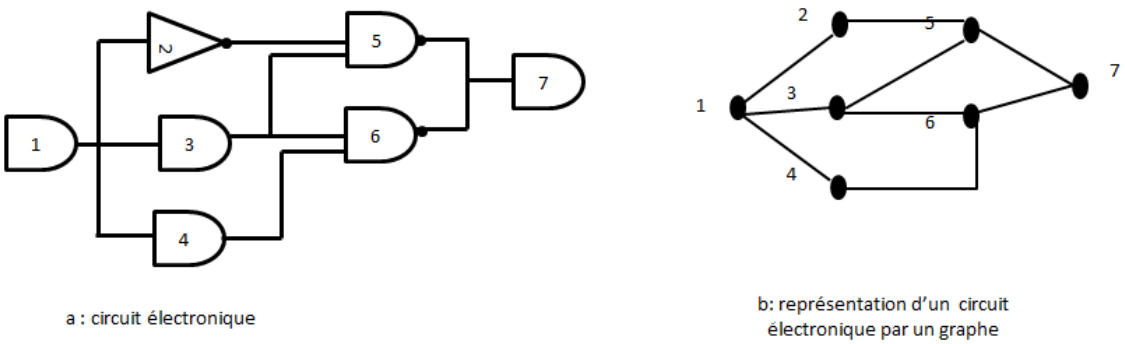


FIGURE 3.1 – PPG pour la conception d'un VLSI

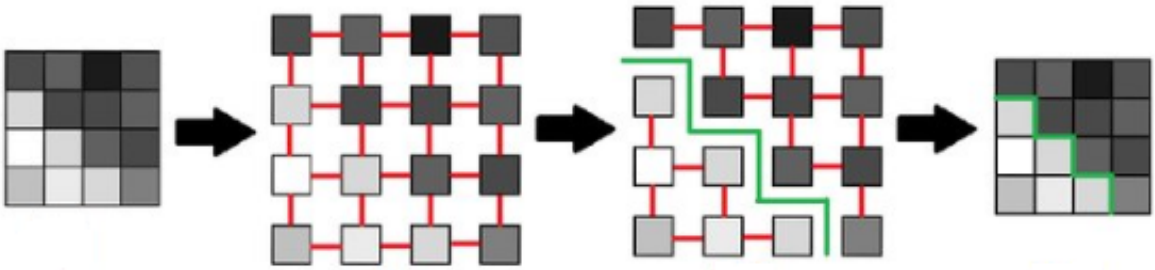


FIGURE 3.2 – PPG pour la segmentation d'image (Eppel, 2016)

existe deux ou plusieurs façons d'atteindre une destination de u à v, par exemple, le système GPS que les gens utilisent pour trouver le chemin le plus court pour conduire d'une destination à une autre. Les transporteurs aériens utilisent un itinéraire cartographique qui peut naturellement être formé comme un graphe ; les sommets sont les aéroports et il y a un chemin de u à v, s'il y a un vol de l'emplacement u à l'emplacement v. Ce chemin peut être dirigé (u, v) ou non dirigé $(u, v)(v, u)$. En observant de tels réseaux, on remarque qu'il y a un petit nombre de sommets (lieu), avec un nombre énorme d'arêtes incidentes (connexion entre les lieux). De manière similaire, un autre réseau de transport peut également être formé. Par exemple, les réseaux ferroviaires, dont chaque terminal est un sommet et dont une arête est une route d'un terminal à un autre. En employant le partitionnement de graphe, il devient possible d'accélérer la recherche du plus court chemin en simplifiant la structure du graphe, et ainsi réduire sa complexité (Schulz et al., 2002).
Calculs parallèles : Une autre utilisation importante du partitionnement de graphes est le calcul parallèle. Le partitionnement permet de diviser la charge de calcul de manière égale entre les machines parallèles, afin d'obtenir des calculs plus rapides et de meilleures performances (Hendrickson and Leland, 1995).

3.10 Conclusion

Dans ce chapitre, nous avons abordé le problème fortement contraints et quelques exemples de ce type.

Nous avons examiné le problème d'optimisation combinatoire NP-complet, le partitionnement de graphe (PPG), qui est souvent utilisé pour traiter des problèmes tels que les VLSI, segmentation d'images, etc....

Nous avons présenté la version du PPG avec contraintes, le problème de partitionnement semi-supervisé de graphe (PPSSG), qui le rend plus difficile à résoudre à cause des contraintes ajoutées.

Comme le PPSSG est un problème d'optimisation fortement contraints on a besoin d'équiper la méthode de résolution d'une méthode de gestion de contraintes. Dans le chapitre suivant, nous introduisons notre proposition pour résoudre de tels problèmes.

Chapitre 4

Gestion de contraintes basée sur les fonctions de transformation pour résoudre un problème fortement contraint

4.1 Introduction

De nombreux problèmes réels sont fortement contraints, ce qui nécessite l'utilisation de méta-heuristiques pour les résoudre, comme les algorithmes génétiques. En effet, dans de nombreux cas, une solution doit respecter strictement ces contraintes pour être considérée comme valide. Les algorithmes génétiques ne sont pas initialement conçus pour traiter des problèmes contraints, car leurs opérateurs de base - le croisement et la mutation - ne prennent pas en compte ces contraintes. Ainsi, pour trouver une solution satisfaisante, il est indispensable de gérer efficacement les contraintes.

Dans ce chapitre, nous présentons notre contribution (Alouane and Boulif, 2021) qui consiste à proposer une approche de gestion des contraintes pour les algorithmes génétiques (AGs) lorsqu'ils traitent des problèmes fortement contraints. Nous avons choisi le problème de partitionnement de graphe (PPG), présenté au chapitre 3, comme exemple de ce type de problème. Pour évaluer notre approche, nous utilisons diverses formes de fonctions de transformation et les appliquons au problème de partitionnement de graphes semi-supervisé. Enfin, nous réalisons une étude comparative avec une méthode de gestion de contraintes statique et présentons les résultats obtenus.

4.2 Fonction d'adaptation de PPG

La définition de la fonction d'adaptation est un élément crucial dans la conception d'un algorithme génétique (AG). Cette fonction mesure la qualité des individus, ce qui influe directement sur le processus de sélection.

Dans le cas des problèmes d'optimisation sans contrainte, cette évaluation se base uniquement sur la fonction objectif. Cependant, dans la pratique, la plupart des problèmes d'optimisation sont soumis à des contraintes.

La méthode la plus utilisée propose d'ajouter une pénalité $penalty(x)$ à la fonction objectif en fonction du nombre de contraintes violées, fonction 4.1.

$$F(x) = f(x) + penalty(x) \tag{4.1}$$

Dans le cas de notre étude, la fonction d'adaptation est définie de la manière suivante : d'abord, la fonction objectif est transformée en une fonction de maximisation comme illustré dans l'equation 4.2.

$$W'(p) = B - W(p) \tag{4.2}$$

tel que $B = \sum_{e \in E} W(e)$ est une borne supérieure de $W(p)$.

Ensuite, la fonction est adaptée à l'aide de l'équation 4.3.

$$Z(p) = W'(p) + (u - V(p)) * B \tag{4.3}$$

Où u est le nombre de contraintes, p est le candidat (partition) et $V(p)$ le nombre de contraintes violées.

Dans (Boulif and Atif, 2006), les auteurs constatent que si l'AG distingue les solutions irréalisables des solutions réalisables, il n'est pas capable de discerner les bonnes et mauvaises solutions réalisables, car leurs valeurs d'adaptation sont très proches.

4.3 Gestion de contraintes basée sur les fonctions de transformation

La première application de (FT) a été proposée par Boulif et al. (Boulif and Atif, 2006), où les auteurs proposent d'équiper l'AG par une fonction de transformation linéaire permettant d'étendre la région réalisable, mais de manière statique, c'est-à-dire que r (r barrière qui sépare la région réalisable de la région irréalisable) reste constant pendant le processus d'optimisation.

Ici, on propose d'analyser l'impact de l'utilisation d'autres fonctions telles que l'exponentielle et l'hybride, et on propose de donner différente valeur pour le coefficient r ($r = 20\%$, $r = 50\%$ et $r = 80\%$) qui divise l'espace de recherche en deux régions délimitées par la région irréalisable $[0, r]$ et la région réalisable $[r, 1]$ (Alouane and Boulif, 2021).

4.3.1 Transformation linéaire

Une fonction linéaire possède la forme suivante : $f(x) = a(x) + b$ où a et b sont des constantes. Pour la région irréalisable, l'intervalle $[0, r]$ de l'axe des ordonnées est subdivisé en u segments de même taille. Ceci définit sur cet axe les ordonnées $0, r/u, 2 * r/u$ et r. Ces valeurs combinées avec les abscisses $0, 1, 2, ...(u-1)$ et u donnent les coordonnées des points à considérer pour la construction des fonctions linéaire. Ainsi, la fonction linéaire utilisée pour transformer la valeur de la fonction objectif d'une solution qui viole toutes les contraintes est définie par les points $(0, 0)$ and $(1, r/u)$. Celle qui suit est associée aux solutions qui violent toutes les contraintes sauf une. Elle est définie par les points $(1, r/u)$ and $(2, 2 * r/u)$, jusqu'à définition de la fonction de transformation associée aux solutions

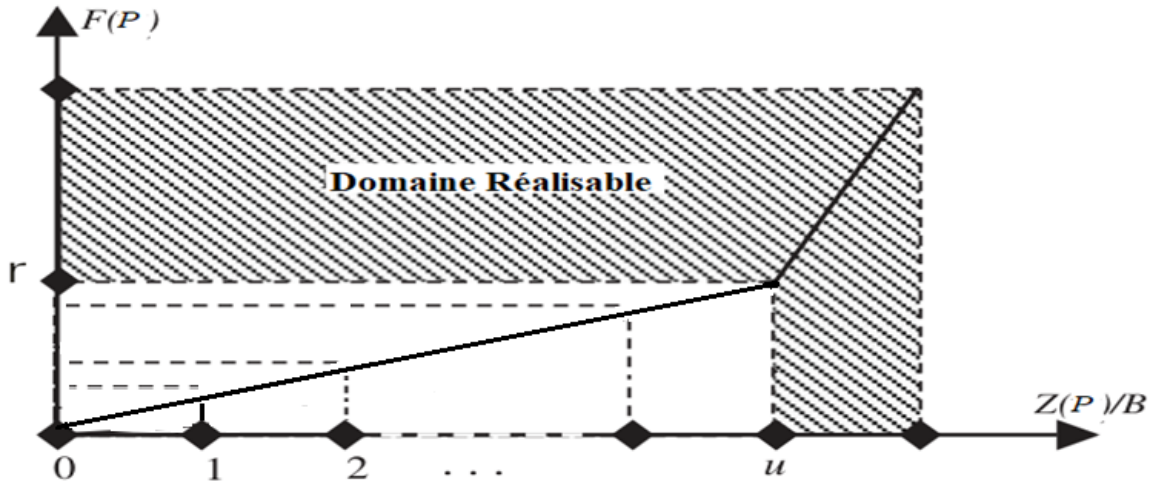


FIGURE 4.1 – Fonction de transformation principale, (Boulif and Atif, 2006)

qui violent une seule contrainte et qui est définie par les points $((u-1),(u-1)*r/u$ et (u,r) (figure 4.1)

Les paramètres a et b de la fonction linéaire qui représente la région irréalisable sont donnés par les équations 4.4 et 4.5 :

$$b = 0 \tag{4.4}$$

$$a = r/u \tag{4.5}$$

Pour la région de l'espace réalisable, elle est définie par les deux points $(u,r).(u+1,1)$. Donc la fonction qui représente cette région est définie par l'équation 4.6 :

$$f(x) = (1-r)x + ((u+1)r - u) \tag{4.6}$$

On destingue deux fonctions d'adaptation : pour l'espace irréalisable, la fonction presentée par l'équation 4.7, et pour l'espace de recherche réalisable, la fonction présentée par l'équation 4.8.

$$F(p) = (r/u) \times Z(p) \tag{4.7}$$

$$F(p) = (1-r) \times Z(p) + ((u+1)r - u) \tag{4.8}$$

4.3.2 Fonction exponentielle

Pour des raisons de simplification, la fonction exponentielle que nous utilisons est de la forme : $f(x) = k \times \exp(\alpha x)$. Les mêmes points définis lors de la détermination des fonctions de transformation linéaire sont utilisés pour déterminer les paramètres k et α. pour déterminer ces paramètres, on propose de considérer deux points qui sont $(0,0)$ et (u,r) pour l'espace irréalisable, et (u,r) et $(u+1,1)$ pour l'espace réalisable On aura deux équations pour déterminer les valeurs de k et α, les équations 4.9 et 4.10.

$$k = \frac{r}{\exp(\alpha * u)} \tag{4.9}$$

$$\alpha = \ln(\frac{1}{r}) \tag{4.10}$$

La fonction d'adaptation équipée d'une transformation exponentielle est décrite dans l'équation 4.11 :

$$F(p) = k \times \exp(\alpha \times Z(p)) \tag{4.11}$$

4.3.3 Fonction hybride

Nous appliquons une fonction linéaire pour l'espace irréalisable, et une fonction exponentielle pour l'espace réalisable.

4.3.4 Algorithme proposé pour la gestion de contraintes basé sur les fonctions de transformation

La gestion de contraintes basée sur les fonctions de transformation est proposée pour l'AG. Pour sa mise en œuvre, nous définissons les aspects suivants :

Codage

Les individus sont codés en utilisant une représentation entière shiftée à base de sommets (SVTC) (Boulif and Atif, 2006; Chaouche and Boulif, 2019). SVTC est une amélioration de la représentation d'encodage la plus naturelle et la plus utilisée dans la littérature. C'est-à-dire l'encodage qui associe à chaque nœud du graphe un nombre compris entre 1 et $|V|$, pour déterminer à quel cluster le sommet appartient. Par exemple, la partition de la figure 4.2 est encodée avec SVTC par l'unique individu de la figure 4.3 : le graphe a trois clusters, le premier contient les sommets 1 et 2, le second contient les sommets 3, 4 et 5, et le dernier ne contient que le sommet 6.

Génération de la population initial

La population initiale est générée aléatoirement, garantissant ainsi une diversité dès le départ du processus évolutif.

Fonction d'adaptation

La fonction d'adaptation est celle définie à la section 4.2

Opérateurs génétiques

Pour l'opérateur de sélection, nous avons utilisé la sélection par la roue de lotterie.

Pour l'opérateur de croisement, nous avons opté pour la méthode du croisement à un point. Le point de croisement est sélectionné de manière aléatoire, puis les parents sont remplacés par leur progéniture.

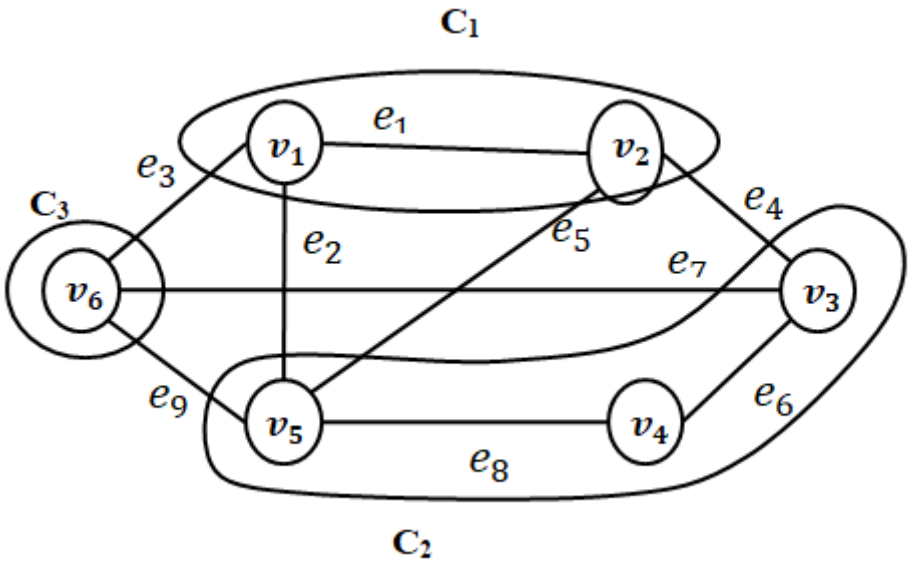


FIGURE 4.2 – Un graphe a trois clusters

V1	V2	V3	V4	V5	V6
1	1	2	2	2	3

FIGURE 4.3 – Exemple de codage SVTC

Quant à l'opérateur de mutation, nous avons employé la méthode du point de mutation unique. Encore une fois, le point de mutation est choisi aléatoirement.

L'algorithme génétique (AG) est présenté dans l'algorithme 2. Le même AG est appliqué pour toutes les fonctions de transformation proposées, avec différentes valeurs de r. Tout d'abord, la population initiale est générée aléatoirement. Ensuite, la valeur du paramètre de transformation est mise à jour pour évaluer chaque solution de la population. Ensuite, nous sauvegardons les $t1\%$ meilleures solutions pour les réinsérer dans la génération suivante. Ensuite, $t2\%$ des individus sont sélectionnés à l'aide de la roue de lotterie, puis $t3\%$ de la population subit l'opérateur de croisement. Pour la génération suivante, nous ne conservons que la descendance, puis $t4\%$ de celle-ci subit l'opérateur de mutation. La nouvelle génération se compose des $t1\%$ meilleures solutions sauvegardées précédemment, de l'ensemble de la descendance, et de $100-(t1+t2)\%$ d'individus générés aléatoirement. Ce processus est répété jusqu'à ce qu'un nombre donné de générations soit atteint. Les valeurs des paramètres $t1$, $t2$, $t3$ et $t4$ sont présentées dans le tableau 4.1.

[1] **Input :** La matrice de graphe
[2] **Output :** La solution trouvée
[3] Génération aléatoire de la population initiale;
[4] Mise à jour des paramètres de la fonction de transformation (voir 4.3);
repeat
[5] | Évaluer la population en utilisant la fonction d'adaptation (voir section 4.3);
[6] | Sauvegarder les $t1$ meilleurs individus;
[7] | Sélectionner une proportion $t2$ d'individus;
[8] | Appliquer le croisement;
[9] | Choisir aléatoirement des individus pour la mutation;
until *nombre de générations*;

Algorithme 2 : L'algorithme génétique implémenté

Paramètre	valeur
elitisme $t1$	10%
sélection $t2$	50%
croisement $t3$	30%
mutation $t4$	1%

TABLE 4.1 – Valeur de paramètre

4.4 Discussion de résultats

4.4.1 L'influence de la valeur r

Dans les figures 4.4 et 4.5, nous présentons respectivement les transformations linéaire (4.3.1) et exponentielle (4.3.2) avec un nombre de contraintes égal à 3 et différentes valeurs de r. Nous pouvons observer l'impact de r sur la valeur de la fonction d'adaptation $F(p)$. En effet, lorsque l'espace irréalisable est très petit, l'algorithme génétique (AG) éprouve des difficultés à distinguer les solutions irréalisables (voir Tableau 4.2). Cependant, dans l'intervalle de l'espace réalisable $[0.2, 1]$, les solutions réalisables sont plus nettement différenciées (voir Tableau 4.3). De plus, la fonction de transformation normalise la valeur de la fonction d'adaptation.

valeurs irréalisables	exponentielle			linéaire		
	r=20	r=50	r=80	r=20	r=50	r=80
1.2	0.015	0.033	0.036	0.08	0.2	0.32
1.5	0.021	0.049	0.054	0.1	0.25	0.4

TABLE 4.2 – L'effet de r pour les valeurs irréalisables

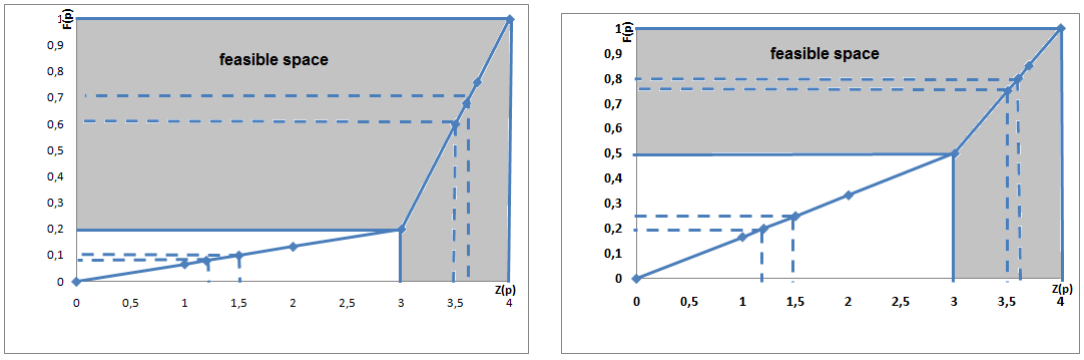
4.4.2 Etude expérimentale

L'algorithme utilisé (algorithme 2) sera appliqué à 30 instances de graphes avec trois valeurs différentes pour barrière réalisable/irréalisable $r = 20\%$, $r = 50\%$ et $r = 80\%$.

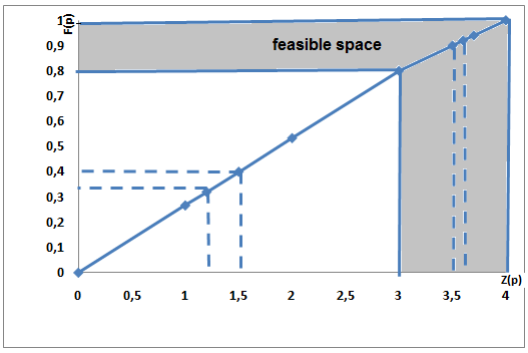
Dans certaines instances de graphes, nous avons modifié les poids des aretes. Les lignes

valeurs réalisables	exponentielle			linéaire		
	r=20	r=50	r=80	r=20	r=50	r=80
3.5	0.44	0.71	0.89	0.6	0.75	0.9
3.6	0.52	0.75	0.91	0.68	0.8	0.92

TABLE 4.3 – L'effet de r pour les valeurs réalisables



(a) $r = 0.2$ (b) $r = 0.5$



(c) $r = 0.8$

FIGURE 4.4 – Transformation linéaire avec différente valeur de r

de ces graphes sont mises en évidence par une étoile * dans le tableau 4.4. Nous avons désigné la somme des poids des arêtes par (**EWS**), nombre de cluster par (**CL**), nombre de contraintes de non cohabitation (**CNC**) et nombre de contraintes de cohabitation (**CC**)

Nous avons divisé les graphes en trois classes : petit (du graphe 1 au graphe 12), moyen (du graphe 13 au graphe 23), élevé (du graphe 24 au graphe 30).

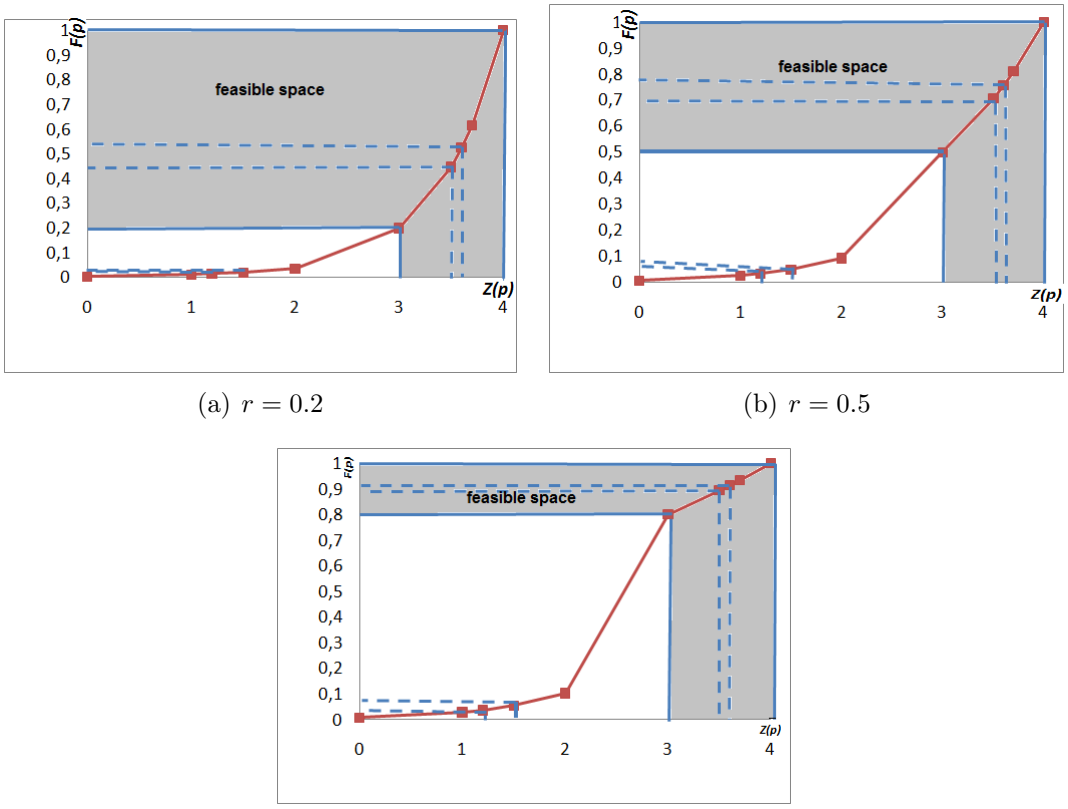
Dans chaque classe, nous avons choisi un graphe pour montrer la performance de chaque méthode, les graphes choisis sont 9, 18, et 24 ayant 74, 196 et 300 sommets respectivement.

Pour étudier la performance des méthodes de gestion de contraintes proposées, nous avons utilisé trois métriques :

1. La moyenne de la meilleure valeur de fitness (MBF), qui mesure la moyenne de la meilleure fitness sur 30 exécutions. Si une exécution ne donne aucune solution réalisable, nous utilisons la somme de poids global du graphe au lieu des valeurs trompeuses de la fonction objectif.
2. L'ecart-type de la meilleure fitness sur 30 exécutions, que nous avons notée SDBF.
3. Le temps d'exécution moyen, que nous avons noté vitesse (s).

TABLE 4.4 – Instance de graphes

G	Ref.	$\|V\|$	$\|E\|$	densité	EWS	CL	CC #	CNC #
1	(Bader et al., 2011)	8	10	0.36	10	4	1	1
2	(Merchichi and Boulif, 2015)	10	15	0.33	15	5	2	1
3	(Merchichi and Boulif, 2015)	12	22	0.33	102	4	2	1
4⋆	(Bader et al., 2011)	14	21	0.23	229	5	2	1
5⋆	(Merchichi and Boulif, 2015)	15	55	0.52	184	5	2	1
6⋆	(Bader et al., 2011)	24	36	0.13	383	8	2	2
7⋆	(Walshaw, 2016)	30	45	0.10	486	6	2	2
8⋆	(Walshaw, 2016)	36	290	0.46	2973	12	2	2
9⋆	(Walshaw, 2016)	49	476	0.40	5020	16	3	3
10⋆	(Walshaw, 2016)	74	301	0.11	3095	12	4	5
11⋆	(Walshaw, 2016)	96	1368	0.30	14548	16	4	8
12⋆	(Walshaw, 2016)	100	1470	0.30	15415	20	4	8
13⋆	(Walshaw, 2016)	121	1980	0.27	20667	16	5	8
14⋆	(Bader et al., 2011)	125	3891	0.50	13664	25	5	10
15⋆	(Bader et al., 2011)	128	190	0.02	1043	25	6	8
16⋆	(Walshaw, 2016)	144	2596	0.25	7887	30	6	9
17⋆	(Bader et al., 2011)	153	1135	0.10	3433	50	6	9
18⋆	(Walshaw, 2016)	169	3328	0.23	9971	30	8	7
19⋆	(Walshaw, 2016)	196	4147	0.22	14567	35	9	15
20⋆	(Walshaw, 2016)	225	5180	0.21	15492	40	9	19
21	(Bader et al., 2011)	250	613	0.02	613	30	9	22
22	(Bader et al., 2011)	250	3218	0.10	3218	50	9	22
23	(Bader et al., 2011)	250	15668	0.50	15668	50	4	26
24	(Bader et al., 2011)	300	21633	0.48	21633	50	4	33
25	(Bader et al., 2011)	420	3720	0.04	3720	60	8	44
26	(Bader et al., 2011)	450	9757	0.10	9757	50	9	49
27⋆	(Bader et al., 2011)	500	2355	0.02	8290	50	9	53
28	(Bader et al., 2011)	500	121275	0.97	121275	50	12	50
29	(Bader et al., 2011)	900	307350	0.76	307350	100	15	94
30	(Bader et al., 2011)	1000	10107	0.02	35526	100	15	110



(a) $r = 0.2$ (b) $r = 0.5$

(c) $r = 0.8$

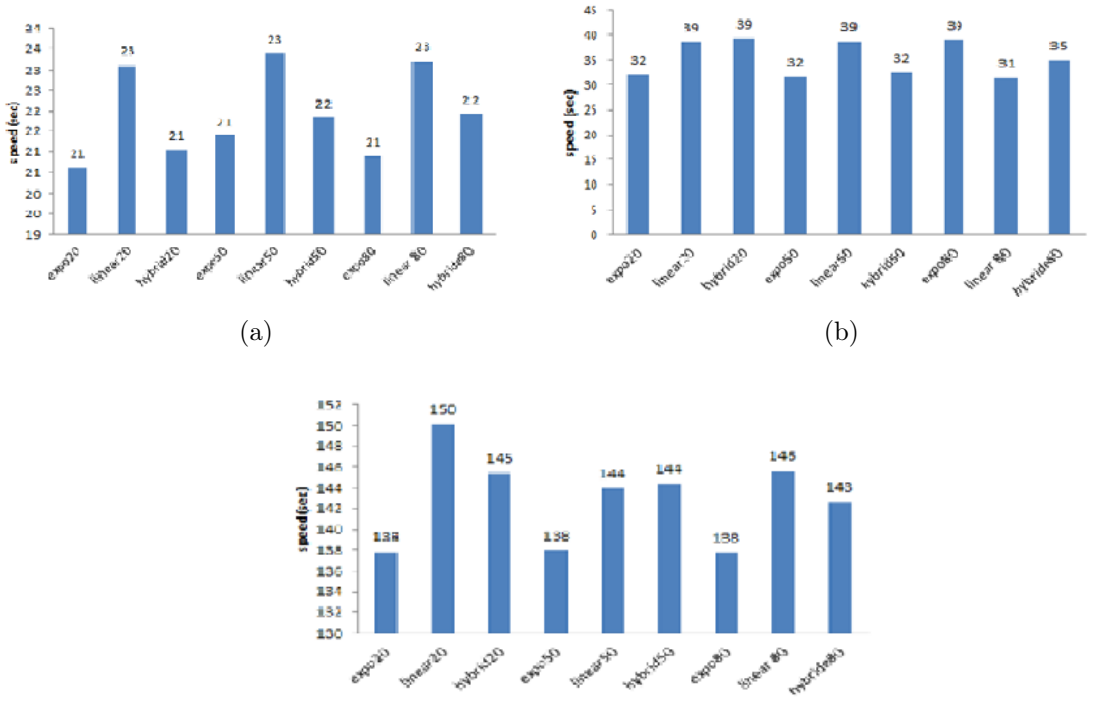
FIGURE 4.5 – Transformation exponentielle avec différentes valeur de r

Selon le temps d'exécution moyen (vitesse), comme le suggère la figure 4.6, le schéma de gestion de contraintes équipé de la fonction exponentielle avec $r = 20\%$ donne la meilleure performance sur toutes les instances. Le schéma de gestion de contraintes équipé de la fonction linéaire avec $r = 80\%$ donne la pire performance. Nous notons également que le schéma de gestion de contraintes avec la fonction exponentielle et hybride, avec chaque variante de r, donne une meilleure performance en moyenne du temps d'exécution que la gestion de contraintes avec la fonction linéaire.

Nous pouvons clairement remarquer dans la figure 4.7 que pour chaque fonction de transformation et pour les petits graphes (voir 4.7(a)) ou les graphes moyens (voir 4.7(b)), la valeur de la barrière réalisable/irréalisable $r = 80\%$ donne le pire MBF. Sauf pour le grand graphe (voir 4.7(c)), la fonction linéaire avec $r = 80\%$ donne la meilleure solution par rapport à la fonction linéaire avec $r = 20\%$ ou $r = 50\%$.

Nous pouvons expliquer cela par le fait que les grands graphes nécessitent plus d'exploration dans l'espace irréalisable, ce qui améliore l'exploitation dans l'espace réalisable même avec un intervalle d'espace réalisable étroit. Nous remarquons que la fonction exponentielle avec $r = 20\%$ donne la meilleure valeur de MBF, cela peut être dû au bon équilibre entre l'exploration et l'exploitation tout au long du processus de recherche.

La figure 4.8 montre l'écart-type de la meilleure valeur de fitness (SDBF) sur 30 exécutions. Pour les petits graphes avec $r = 20\%$, la méthode de la fonction hybride montre, sur 30 exécutions, des valeurs proches de la meilleure solution. En revanche, les solutions obtenues avec la fonction de transformation linéaire et des valeurs de $r = 50\%$ et $r = 80\%$ semblent plus dispersées.



(a) (b) (c)

FIGURE 4.6 – Temps d'exécution moyen (secondes) pour les petits (a), moyens (b) et (c) grands graphes

4.4.3 Etude comparative

Pour tester davantage les performances de notre approche, nous comparons notre gestion de contraintes basée sur les FT avec la gestion des contraintes statiques proposée par Kuri Morales, Dans cette dernière, la fitness d'un individu est calculée avec la formule suivante l'équation 4.12 :

$$fitness(x) = \begin{cases} f(x) & \text{x est réalisable} \\ K - \sum_{i=1}^{s} K/m & \text{x est irréalisable} \end{cases} \tag{4.12}$$

Où s est le nombre de contraintes satisfaites, m est le nombre total de contraintes et K est une constante (qui prend une grande valeur s'il s'agit d'un problème à minimiser et une petite valeur s'il s'agit d'un problème à maximiser). Si un individu n'est pas réalisable, sa fitness est égale à la valeur de sa fonction objectif. Tous les individus qui violent le même nombre de contraintes reçoivent la même pénalité, quelle que soit la distance qui les sépare de l'espace réalisable. Dans ce qui suit, nous appelons cette méthode **Kuri static**

Comme nous pouvons le voir dans le tableau 4.5, toutes les méthodes donnent des solutions réalisables, mais la méthode *Kuri static* donne les mauvaises valeurs de fitness dans toutes les instances.

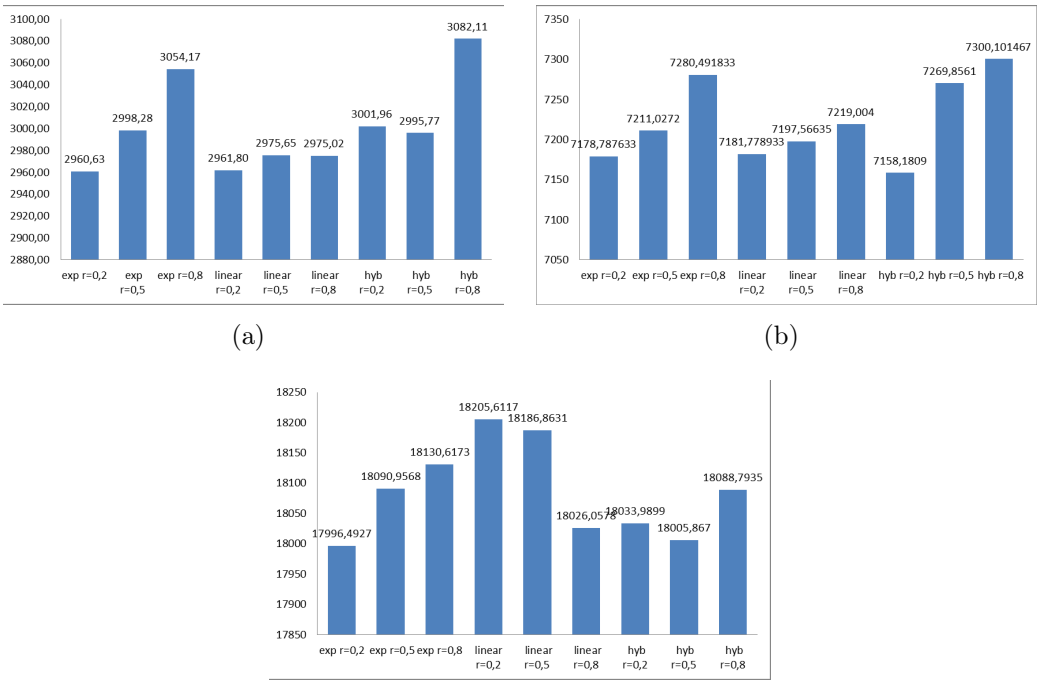
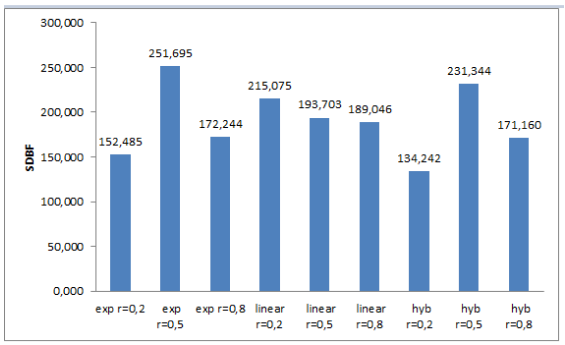


FIGURE 4.7 – MBF pour les petits (a), moyens (b) et (c) grands graphes

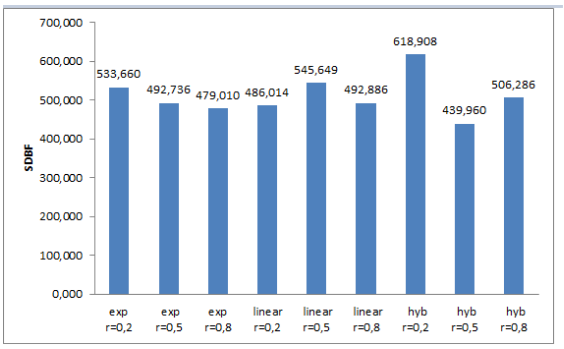
4.5 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle approche de gestion des contraintes pour les algorithmes génétiques (AG) traitant des problèmes fortement contraints. Cette méthode utilise différents types de fonctions de transformation.

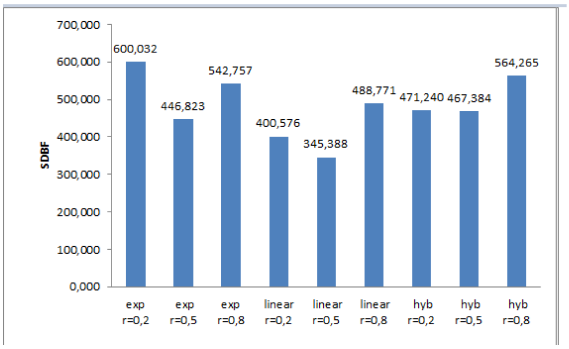
L'approche présentée consiste à équiper l'AG d'une fonction de transformation qui remplit deux rôles essentiels : en phase d'exploitation, elle permet à l'AG de sortir de la zone des solutions irréalisables, tandis qu'en phase d'intensification, elle améliore la capacité de l'AG à différencier les bonnes solutions réalisables des moins bonnes. Nous avons constaté que la valeur séparant la région réalisable de la région irréalisable a un impact significatif sur performance de l'AG. De plus, l'utilisation d'une fonction linéaire pour la région irréalisable et d'une fonction exponentielle pour la région réalisable permet à l'AG d'explorer efficacement l'espace de recherche en prenant en compte les bons gènes cachés parmi les solutions irréalisables, puis d'accélérer la phase d'intensification.



(a)



(b)



(c)

FIGURE 4.8 – SDBF pour les petits (a), moyens (b) et (c) grands graphes

TABLE 4.5 – Meilleure valeur d'adaptation

G	exponentielle			linéaire			hybride			Static
	r=20	r=50	r=80	r=20	r=50	r=80	r=20	r=50	r=80	
1	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
2	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
3	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
4	49	57	**48**	**48**	**48**	**48**	**48**	50	50	50
5	**64**	67	69	69	72	74	**64**	67	67	72
6	166	160	**150**	159	192	179	155	171	177	192
7	258	278	271	**251**	277	256	255	258	267	281
8	2185	2123	2197	2173	2203	**2111**	2182	2191	2141	2209
9	2711	**2314**	2580	2526	2645	2610	2676	2329	2741	2590
10	**1470**	1492	1532	1547	1510	1631	1582	1513	1504	1566
11	8641	8743	8452	8118	8641	8525	**7187**	8540	7609	8758
12	8971	9203	9573	**8894**	9249	9403	9018	9249	9295	9418
13	11945	12545	11139	**10705**	13392	12937	12772	12132	12132	12793
14	8636	**7802**	8048	8144	8403	8321	8185	8280	8198	8677
15	617	622	647	**599**	641	621	627	602	657	635
16	5174	5213	5269	**5048**	5237	5190	5190	5513	5379	5521
17	2249	**1696**	2111	1981	2125	1991	1933	2063	2159	2403
18	6262	6381	6381	6302	**5963**	6491	6102	6481	6182	7289
19	9993	10343	10124	9862	9993	**9469**	10488	9847	9745	10197
20	10922	11387	11170	11402	11511	11216	11015	11356	11309	**10736**
21	456	451	**441**	457	460	457	451	452	442	463
22	**2375**	2466	2481	2487	2436	2481	2386	2478	2410	2481
23	10576	10921	10764	10889	10654	10654	10654	10686	**10497**	10654
24	16225	16874	16441	16874	17090	16657	16874	16657	**15792**	17306
25	**2634**	**2634**	**2634**	**2634**	**2634**	**2634**	**2634**	**2634**	**2634**	2853
26	7757	7747	7757	7757	**7659**	7757	7747	7708	7757	7913
27	6964	6964	6964	6872	6964	6947	6872	6872	**6765**	6947
28	**94716**	**94716**	**94716**	**94716**	**94716**	94837	**94716**	**94716**	**94716**	101628
29	**282147**	**282147**	**282147**	283991	283684	283684	283684	283684	284299	287987
30	33359	33394	33465	33359	33501	33430	33501	33501	33501	33749

Chapitre 5

Priorisation floue

5.1 Introduction

Dans le monde réel, les problèmes sont souvent soumis à diverses contraintes, qui peuvent avoir des priorités différentes. Dans ce chapitre, nous présentons notre contribution, qui consiste à étudier l'impact de la prise en compte des priorités des contraintes sur le fonctionnement des algorithmes génétiques (AG) lorsqu'ils traitent des problèmes fortement contraints. Nous utilisons des méthodes de gestion de contraintes basées sur les fonctions de transformation présentées dans le chapitre précédent, ainsi qu'une étude comparative basée sur des tests statiques.

Nous décrivons également le système d'inférence floue (SIF), qui permettra de déterminer le meilleur ordre de contraintes. Enfin, nous examinons une étude expérimentale visant à évaluer les performances de l'approche proposée sur le problème de partitionnement semi-supervisé des graphes. Nous comparons l'approche floue à d'autres méthodes bio-inspirées.

5.2 La priorisation floue des contraintes

Dans la réalité, les contraintes auxquelles sont soumis les problèmes d'optimisation ne sont pas toutes de la même importance. Plutôt que d'attribuer la même priorité à tous les types de contraintes, nous proposons d'assigner à chaque contrainte un ordre de priorité différent. Pour ce faire, l'axe des abscisses de la fonction de transformation (FT) (présentée dans le chapitre précédent) est d'abord divisé en fonction du nombre de types de contraintes (noté c).

Pour le PPG semi-supervisé, il existe quatre types de contraintes. Par conséquent, l'axe des abscisses x est d'abord divisé en quatre sous-intervalles, puis chaque intervalle est à son tour divisé en fonction du nombre de contraintes associées. L'ordre dans lequel les types de contraintes sont disposés sur l'axe des x détermine les niveaux de priorité. En effet, si une solution viole plusieurs types de contraintes, elle sera placée dans l'intervalle le plus à gauche sur l'axe des x (voir Figure 5.1). Par conséquent, la fonction objectif définie par l'équation 4.3 sera redéfinie selon l'équation 5.1.

$$Z(P) = W'(P) + (i-1) \times B + (u_i - V_i(P)) \times B/u_i \tag{5.1}$$

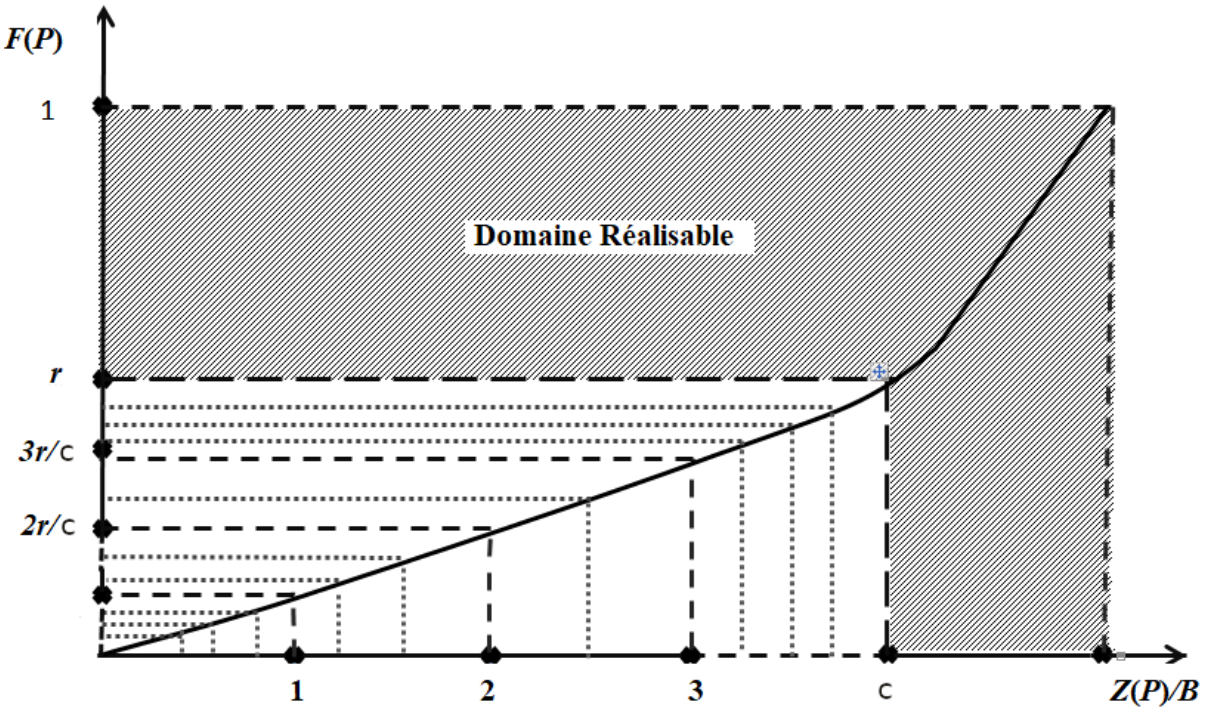


FIGURE 5.1 – Fonction de transformation avec priorisation

où :

— i est le rang de la contrainte de priorité la plus élevée violée par la solution P. Si P est réalisable, alors i est égal à $c+1$ (c'est-à-dire le nombre de types de contraintes plus un).
— u_i est le nombre de contraintes de type i.
— $V_i(P)$ est le nombre de contraintes de type i violées par P.

Après avoir effectué cette procédure de placement, nous appliquons une fonction de transformation (linéaire, exponentielle ou hybride) qui associe chaque valeur obtenue à partir de l'équation 5.1 à l'intervalle réel [0,1]. L'image de l'axe des abscisses $c \cdot B$ est la barrière irréalisable/réalisable r. Pour la région irréalisable, l'intervalle $[0, r]$ de l'axe des ordonnées est subdivisé en c intervalles, où c est le nombre de types de contraintes. Chaque intervalle i est divisé en u_i sous-intervalles, où u_i est le nombre de contraintes de type i. Si les sous-intervalles de type de contrainte sont tous de même taille, cette subdivision définit les ordonnées $0, r/c, 2{*}r/c, \cdots, r, 1$. Ensuite, en combinant ces valeurs respectivement avec les abscisses $0, B, 2 \times B, \cdots, c \times B, (c+1) \times B$, on obtient les coordonnées de ce que nous appelons les points de construction à considérer dans la définition des FT (voir figure 5.1).

La fonction de transformation linéaire, la fonction de transformation exponentielle et la fonction de transformation hybride seront réadaptées de la manière suivante :

1. **Fonction de transformation linéaire** :
   Pour la région réalisable, elle est définie par l'équation : 5.2.

$$F(P) = \frac{(1-r)}{B} \times Z(P) + ((c+1)r - c) \tag{5.2}$$

Pour la région irréalisable, elle est définie par l'équation 5.3

$$F(P) = (r/(c \cdot B)) \times Z(P) \tag{5.3}$$

2. **Fonction de transformation exponentielle** : La région réalisable est définie par l'équation 5.4 et la région irréalisable est définie par 5.5

$$F(P) = r \cdot e^{\frac{-\ln(r)}{B} \cdot Z(P) + c \cdot \ln(r)} \tag{5.4}$$

$$F(P) = e^{\frac{\ln(r+1)}{c \cdot B} \cdot Z(P)} - 1 \tag{5.5}$$

3. **Fonction hybride** La fonction hybride : consiste à appliquer une fonction linéaire pour l'espace réalisable et une fonction exponentielle pour l'espace irréalisable.

5.3 Approche floue

Afin de trouver le meilleur ordre de contraintes qui permettra à l'AG de converger vers l'espace réalisable, nous utilisons la logique floue.

5.3.1 Raisonnement flou

La logique floue, proposée par Zadeh (Zadeh, 1996) dans les années soixante, introduit la notion d'ensembles flous pour représenter des données imprécises ou imparfaitement décrites. En 1975, Mamdani et Assilian (Mamdani and Assilian, 1975) ont publié un ouvrage qui a concrétisé la théorie de Zadeh en proposant un système d'inférence floue pour contrôler un moteur à vapeur.

Les trois étapes principales pour concevoir un système d'inférence floue sont :

1. Fuzzification : transforme les données d'entrée numériques en valeurs linguistiques en utilisant des fonctions d'appartenance,
2. Évaluation des règles : calculer l'information linguistique de sortie en utilisant des règles floues,
3. Défuzzification : La défuzzification est la dernière étape du processus d'inférence floue, où l'ensemble flou combiné est transformé en une seule valeur scalaire. Contrairement à la fuzzification, qui convertit les valeurs nettes des variables d'entrée en degrés d'appartenance à des ensembles flous, La défuzzification trouve une valeur numérique à utiliser comme résultat pour la variable de sortie à partir de l'ensemble flou. Les méthodes les plus couramment utilisées sont :
   — **Méthode de centre de gravité :** Elle est donnée par l'expression algébrique :

$$Z^* = \frac{\int \mu_A(z) \cdot z \, dz}{\int \mu_A(z) \, dz}$$

   Où Z_* représente la valeur de la variable de sortie. $\mu_A(z)$ représente le degré d'appartenance à l'ensemble flou pour la valeur z.
   — **Méthode du maximum :** Cette approche est valide uniquement lorsque la fonction d'appartenance associée à l'ensemble de sortie présente un seul maximum. Dans ce cas, la sortie choisie est l'abscisse correspondant à ce maximum z_*.

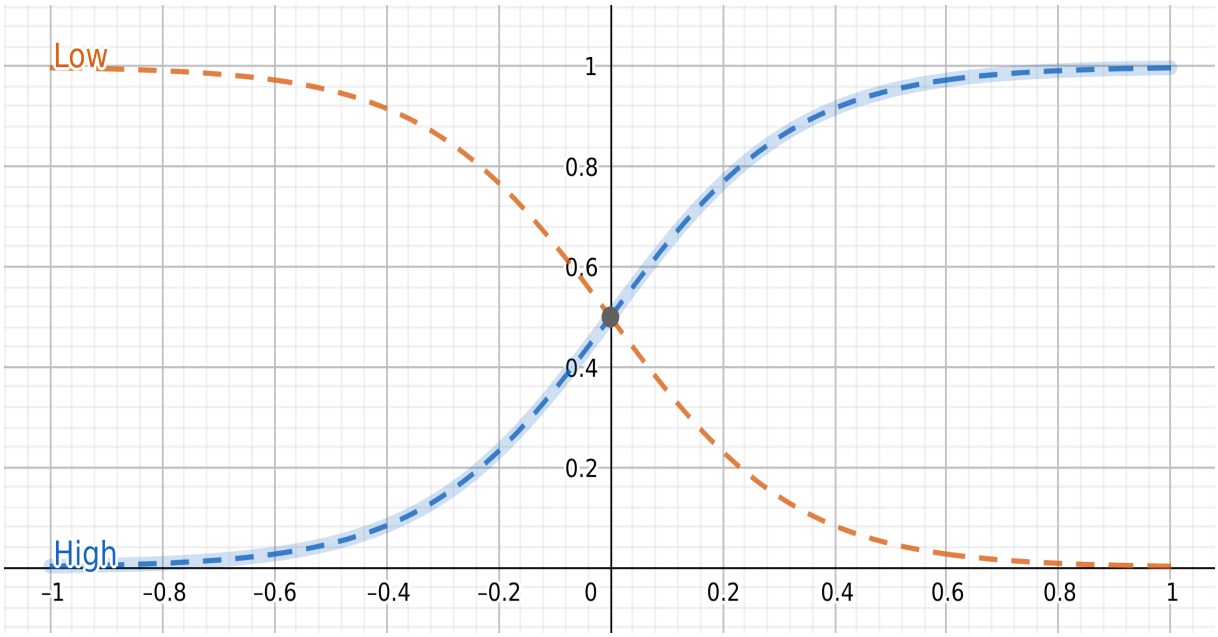


FIGURE 5.2 – La fonction d'appartenance de Convergence

— **Méthode de la moyenne des maxima :** Dans cette méthode, la valeur de sortie est estimée en prenant l'abscisse du point situé au centre de l'intervalle M caractérisé par la fonction d'appartenance maximale. Cette estimation est obtenue à partir de l'expression suivante :

$$Z^* = \frac{\inf(M) + \sup(M)}{2}$$

Où $\inf(M)$, $\sup(M)$ sont respectivement les bornes inférieure et supérieure de l'intervalle M.

5.3.2 Système d'inférence flou

Pour notre SIF proposée, nous utilisons une variable linguistique d'entrée appelée **convergence**, qui mesure la vitesse de convergence pendant le processus d'optimisation. La variable linguistique **convergence** possède deux valeurs : "**high**" (indique une bonne amélioration de la valeur de la fonction d'adaptation de la meilleure solution de la population) et "**low**" (indique une mauvaise amélioration de la valeur de la fonction d'adaptation). La figure 5.2 présente la fonction d'appartenance de la variable **convergence**.

Comme variable de sortie, nous proposons de considérer la mesure **perturbation**, qui nous donne l'information sur le nombre de modifications à appliquer sur l'ordre des contraintes. Pour **perturbation**, on associe également les termes **high**, qui indique une importante modification, et **low**, qui indique une légère modification. Sa fonction d'appartenance est présentée par la figure 5.3

Les équations associées aux deux fonctions d'appartenance (convergence et perturbation) sont respectivement les suivantes :

— Equations 5.6 et 5.7 pour la valeur *high* et *low* de la variable *convergence*, respec-

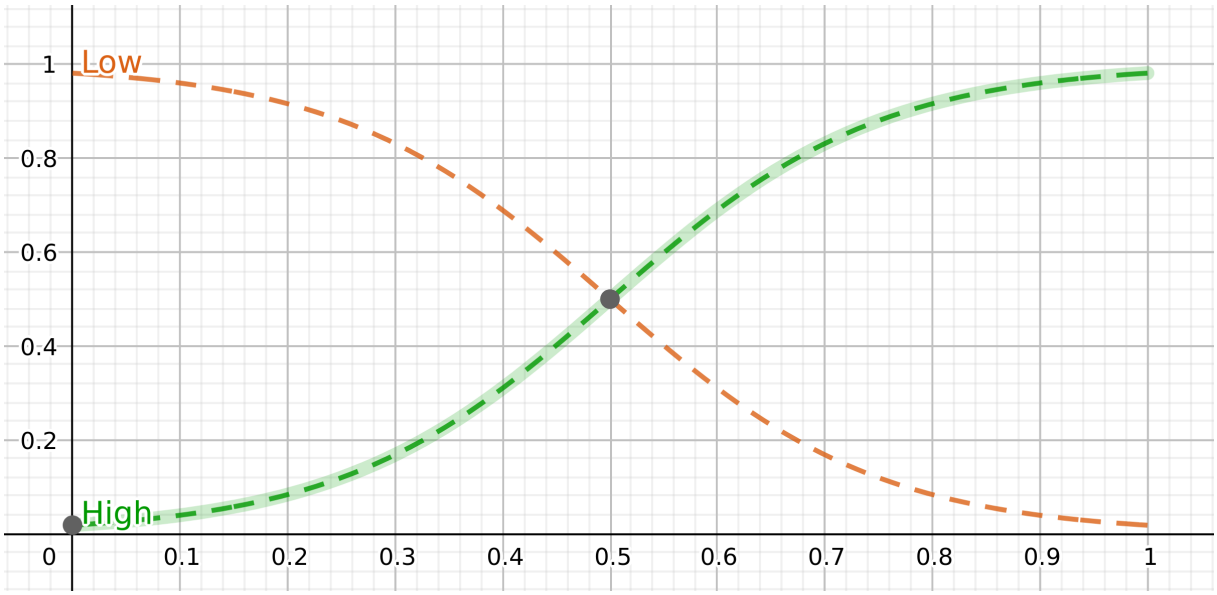


FIGURE 5.3 – La fonction d'appartenance de Perturbation

tivement.

$$f1(x) = 0.5 + 0.5 \cdot \tanh(3x) \tag{5.6}$$

$$f2(x) = 0.5 + 0.5 \cdot \tanh(-3x) \tag{5.7}$$

— Equations 5.8 et 5.9 pour les valeurs *high* et *low* de la variable *perturbation*, respectivement.

$$f3(x) = 0.5 + 0.5 * \tanh(4x - 2) \tag{5.8}$$

$$f4(x) = 0.5 + 0.5 * \tanh(-4x + 2) \tag{5.9}$$

5.3.3 Evaluation de la convergence

Pour évaluer la convergence, nous calculons la variation de la meilleure fitness tout au long des $t5$ dernières générations en utilisant l'équation 5.10.

$$conv = F^{\star}(t) - F^{\star}(t - t5). \tag{5.10}$$

Lorsque la convergence est "**high**", ce qui signifie que la qualité de la population s'est clairement améliorée au cours des $t5$ dernières générations, il n'est pas utile de changer l'ordre des contraintes. Mais s'il y a peu ou pas d'amélioration, le SIF calcule une valeur utilisée pour définir le montant de changements à appliquer sur l'ordre des types de contraintes. Afin de concrétiser ce comportement, la sortie du SIF est utilisée d'une manière qui exerce une influence sur les deux paramètres :

— Le nombre de types de contraintes à déplacer.
— La probabilité qu'un type de contrainte soit sélectionné.

1. **Calcul du nombre de types de contraintes à déplacer** : le nombre de mouvements nm est calculé selon l'équation 5.11, ce qui peut entraîner un maximum de $c - 1$ mouvements, suffisants pour générer n'importe quelle permutation possible.

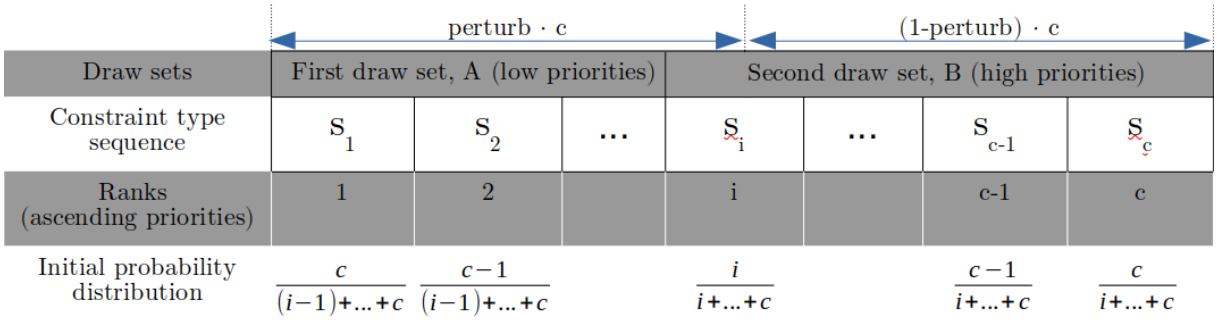


FIGURE 5.4 – Procédure de sélection par contrainte

$$nm = \lceil perturb \cdot (c-1) \rceil_2 \tag{5.11}$$

Dans cette équation, *perturb* est la valeur de sortie du SIF, c est le nombre de types de contraintes et $\lceil x \rceil_2$ désigne le premier nombre entier plus grand que x.

2. **Calcul de la probabilité qu'un type de contraintes soit déplacé** : Afin de réaliser les déplacements définis par l'équation 5.11, nous effectuons une série de permutations. Chaque permutation implique deux de ces déplacements et favorise l'échange à effectuer entre les types de contraintes de *"high"* et *"low"* priorité.

   Pour ce faire, étant donné la valeur de sortie du SIF *perturb* et la séquence des types de contraintes $(s_1, s_2, \cdots, s_c)$ ordonnés par priorités croissantes, nous identifions s_i qui satisfait l'équation 5.12.

$$i = \lceil perturb \cdot c \rceil . \tag{5.12}$$

   i, divise la séquence en deux, définissant les sous-ensembles de types de contraintes. $A = \{s_1, s_2, \cdots, s_{i-1}\}$ et $B = \{s_i, s_{i+1}, \cdots, s_c\}$ (voir figure. 5.4).

   Pour effectuer un échange, le premier type de contrainte est choisi dans A et le second dans B en utilisant un tirage sans remplacement. Si l'un des deux sous-ensembles devient vide et qu'il reste encore des mouvements, le tirage au sort est effectué sur le sous-ensemble non vide restant.

   La distribution de probabilité associée à la sélection d'un type de contrainte s_k parmi A est définie par l'équation 5.13

$$P(''get\ s_k \text{ from } A'') = \frac{c - rank(s_k) + 1}{\sum_{s_j \in A} c - rank(s_j) + 1} \tag{5.13}$$

   Alors que, à partir de B par l'équation 5.14

$$P(''get\ s_k \text{ from } B'') = \frac{rank(s_k)}{\sum_{s_j \in B} rank(s_j)} \tag{5.14}$$

Le pseudo-code de l'AG flou intégrant le SIF que nous venons de décrire est représenté dans l'algorithme 3.

Input : Le graphe et les contraintes
Output : La meilleure partition trouvée
[1] Générer une population initiale aléatoire;
[2] Générer un ordre de contraintes aléatoire;
[3] Calculer la valeur d'adaptation de chaque individu en utilisant FT (voir section 5.2);
repeat
[4] Enregistrer les $t1$ meilleurs individus;
[5] Sélectionner $t2$ individus par la roulette de loterie;
[6] Choisir $t3$ individus aléatoirement et appliquer le croisement;
[7] Choisir $t4$ individus aléatoirement et appliquer la mutation;
[8] Calculer la valeur d'adaptation de chaque individu;
[9] Utiliser un tournoi pour récupérer les $t1$ individus enregistrés à la place d'individus choisis au hasard s'ils sont plus adaptés;
[10] À chaque $t5$ itération, lancer la procédure de ré-ordonnement flou et réévaluer la valeur d'adaptation de chaque individu;
until *nombre de générations*;

Algorithme 3 : AG flou

Exemple de fonctionnement de SIF

Afin d'expliquer correctementde le fonctionnement de l'approche proposée, nous l'appliquons à un exemple de PPG de huit sommets où la somme des poids de ses arrêtes est égale à dix, avec sa matrice d'adjacence présentée dans le tableau 5.1. Le PPG est soumis aux contraintes suivantes :

1. **Contrainte de cohabitation(CC)** : Les sommets $v4$ et $v5$ doivent se trouver dans le même cluster,
2. **Contraintes de non cohabitation (CNC)** : Les sommets $v2$ et $v7$ doivent être placés dans des clusters différents.
3. **Contrainte de nombre de clusters (CL)** : Au moins 2 et au plus 7 clusters doivent être formés,
4. **Contraintes sur la taille de cluster (CS)** : La taille maximale des clusters est fixée à 4 sommets.

TABLE 5.1 – Matrice d'adjacence de l'instance GPP

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	0	1	0	0	0	0	0	0
v_2	1	0	1	0	1	0	0	0
v_3	0	1	0	1	1	0	0	0
v_4	0	0	1	0	0	1	1	0
v_5	0	1	1	0	0	1	0	0
v_6	0	0	0	1	1	0	1	0
v_7	0	0	0	1	0	1	0	1
v_8	0	0	0	0	0	0	1	0

Prenons deux partitions comme exemple : $P1 = \{\{v_1, v_2\}, \{v_4, v_5, v_8\}, \{v_3, v_6, v_7\}\}$ et $P2 = \{\{v_1, v_3, v_5, v_8\}, \{v_2, v_4, v_6, v_7\}\}$. La partition $P1$ est réalisable, car elle satisfait

toutes les contraintes définies précédemment, tandis que la partition $P2$ viole à la fois la contrainte de cohabitation et celle de non cohabitation.

Il est à noter que la fonction objectif de $P2$ est inférieure à celle de $P1$ ($6 \leq 8$), bien que la première soit irréalisable. La procédure de gestion des contraintes de l'AG proposée corrige ce problème pour s'assurer que les solutions réalisables ont une meilleure adaptation.

Dans l'approche de la fonction de transformation présentée dans la section 4.3, pour obtenir l'adaptation de chaque solution, la fonction objectif est d'abord transformée en une fonction de maximisation par l'équation 4.2, et nous obtenons ainsi l'équation 5.15.

$$W'(P) = 10 - W(P) \tag{5.15}$$

La fonction objectif définie par l'équation 4.3 nous donne une mesure de l'adaptation de chaque solution dans l'espace de recherche définie par l'équation 5.16 :

$$Z(P) = 10(i+1) - W(P) - 10 \cdot V_i(P) \tag{5.16}$$

l'équation 5.16 donne : $Z(P1) = 42$.

Pour évaluer chaque solution irréalisable, comme $P2$, nous devons disposer des priorités des contraintes. Nous supposons initialement que l'ordre décroissant de priorité des contraintes est $CC - CL - CNC - CS$ (ou 1324, en utilisant les indices d'énumération des contraintes présentés dans l'exemple). Cela signifie que la contrainte de cohabitation a une priorité plus élevée par rapport aux autres. Ainsi, $P2$ sera évalué selon le 1^{er} sous-intervalle au lieu du 3^{eme} sous-intervalle (parce que $P2$ viole à la fois CC et CNC, et l'évaluer selon le sous-intervalle de CC est plus pénalisant). En procédant ainsi, on obtient $Z(P2) = 4$.

Pendant la comparaison des solutions réalisables pour les problèmes fortement contraints, les valeurs de $Z()$ seront très proches les unes des autres. C'est là que les fonctions de transformation interviennent pour fournir à l'AG des valeurs de fitness sur lesquelles il peut se baser. Avec $c = 4$, la TF est construite sur cinq sous-intervalles : quatre pour les solutions irréalisables, et le dernier pour les solutions réalisable. Par la suite, en fixant $r = 20\%$, on obtient six points de construction dont les coordonnées sont : $(0, 0)$, $(10, 0.05)$, $(20, 0.10)$, $(30, 0.15)$, $(40, 0.20)$ En utilisant une transformation linéaire, par exemple, la TF à appliquer pour la région irréalisable est définie par l'équation 5.17 et pour la région réalisable par l'équation 5.18.

$$F(P) = 0.005 \times Z(P) \tag{5.17}$$

$$F(P) = 0.08 \cdot Z(P) - 3 \tag{5.18}$$

L'application de la TF sur $P1$ et $P2$ donne les valeurs $F(P1) = 0.36$ et $F(P2) = 0.12$.

A chaque $t5$ itérations, le SIF est lancé. Si à la génération t la meilleure valeur de fitness est $F^{\star}(t) = 0.33$, étant donné qu'avant $t5$ générations, elle était $F^{\star}(t - t5) = 0.12$, l'équation 5.10 donne $conv = 0.21$. Par conséquent, nous utilisons cette valeur comme une valeur d'entrée pour le SIF. Les fonctions d'appartenance de la variable linguistique *convergence* donnent les valeurs floues $high = 0.78$ et $low = 0.22$.

En appliquant la procédure du barycentre (Mamdani and Assilian, 1975) (voir figure 5.5), on obtient le résultat de sortie $perturb \approx 0.32$.

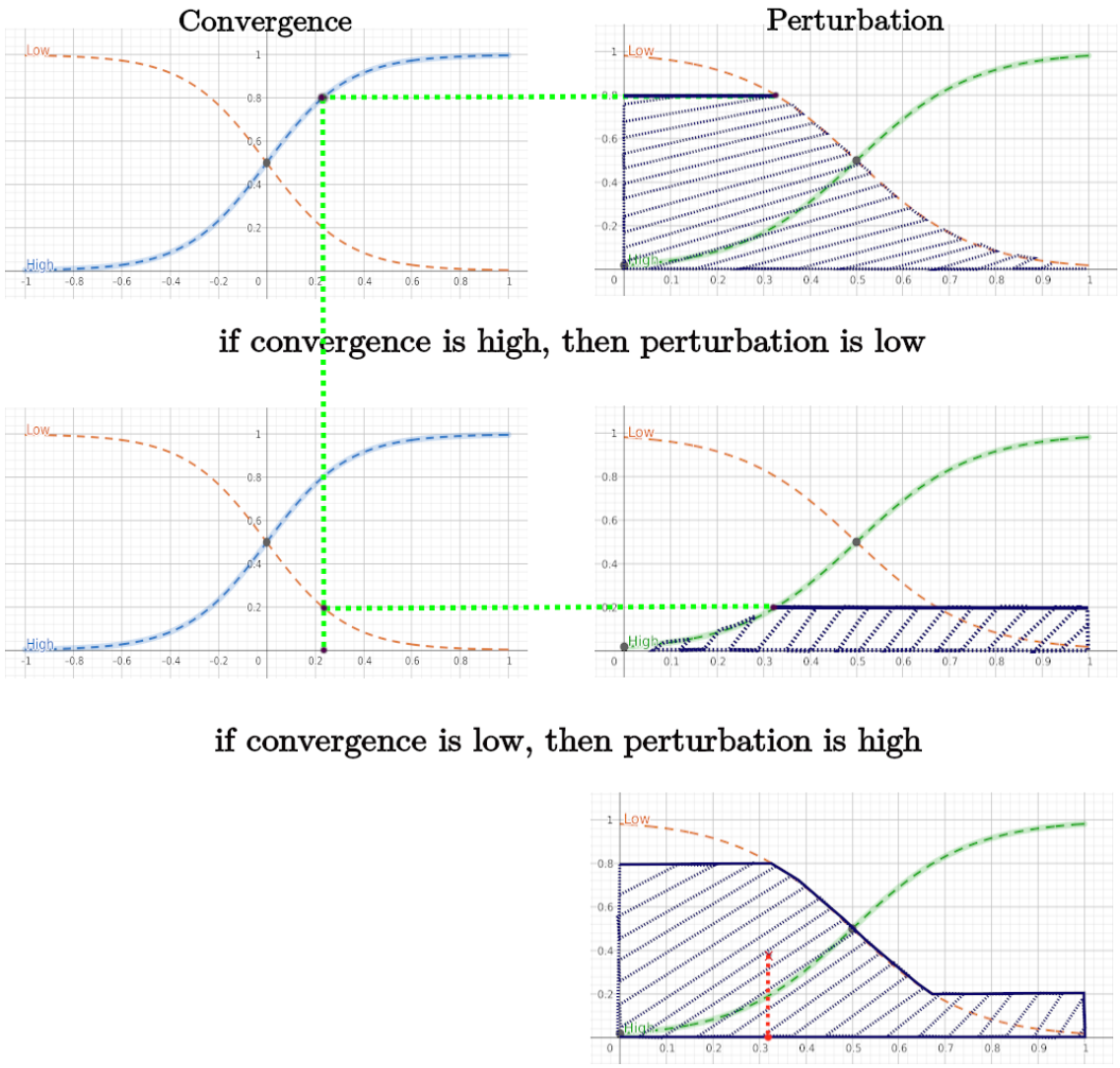


FIGURE 5.5 – Exemple de fonctionnement de SIF

Par conséquent, la séquence de contraintes 1324 subit $\lceil 0.32 \cdot 3 \rceil_2 = 2$ déplacements aléatoires qui seront effectués en réalisant une permutation.

La permutation est réalisée selon les distributions de probabilité représentées dans la table 5.2. Par exemple, si les deux types de contraintes choisies aléatoirement sont 2 de A, et 1 de B, la nouvelle séquence sera 2314.

5.4 Résultats et analyse

L'algorithme proposé (algorithme 3) est appliqué à seize instances de graphes tirées de la littérature, comme présenté dans le tableau 5.3.

Dans certaines de ces instances, nous avons modifié le poids des arêtes. Ces graphes sont mises en évidence par une étoile $\star$. Les détails des contraintes de cohabitation et de non-cohabitation sont disponibles dans (Alouane, 2022). Les instances ont été divisées en trois classes en fonction de leur taille :

TABLE 5.2 – Exemple de sélection de type de contrainte

tirages	1er ensemble, A(priorité faible)		2eme ensemble, B(priorité élevée)	
Type contrainte	4	2	3	1
Rang	1	2	3	4
probabilité	4/7	3/7	3/7	4/7

TABLE 5.3 – Les instances de Graphes

G	Ref.	$\|V\|$	$\|E\|$	Densité	EWS	CL	CS	CC #	NCC #
1	(Bader et al., 2011)	8	10	0.36	10	7	8	1	1
2	(Merchichi and Boulif, 2015)	10	15	0.33	15	5	5	2	1
3	(Merchichi and Boulif, 2015)	12	22	0.33	102	4	5	2	1
4 ⋆	(Bader et al., 2011)	24	36	0.13	383	8	10	2	2
5 ⋆	(Walshaw, 2016)	49	467	0.40	5020	16	25	3	3
6 ⋆	**(Walshaw, 2016)**	**74**	**301**	**0.11**	**3095**	**12**	**40**	**4**	**5**
7 ⋆	(Walshaw, 2016)	96	1368	0.30	14548	16	50	4	8
8 ⋆	(Bader et al., 2011)	125	3891	0.50	13664	25	60	5	10
9 ⋆	**(Bader et al., 2011)**	**128**	**190**	**0.02**	**1043**	**25**	**60**	**6**	**10**
10	(Bader et al., 2011)	250	613	0.02	613	30	80	9	22
11	(Bader et al., 2011)	250	3218	0.10	3218	50	80	4	27
12	(Bader et al., 2011)	300	21633	0.48	21633	50	100	4	33
13	(Bader et al., 2011)	420	3720	0.04	3720	60	150	8	44
14	(Bader et al., 2011)	450	9757	0.10	9757	50	150	9	47
15 ⋆	**(Bader et al., 2011)**	**500**	**2355**	**0.02**	**8290**	**50**	**100**	**9**	**53**
16	(Bader et al., 2011)	900	307350	0.76	307350	100	90	15	97

— petite : du premier graphe au 6ème,
— moyenne : du 7ème au 11ème graphe, et
— élevée : du graphe 12 au dernier.

Dans chaque classe, nous avons choisi un représentant pour montrer la performance détaillée de chaque méthode. Les graphes représentatifs choisis sont 6, 9 et 15 ayant respectivement 74, 128 et 500 sommets. Les lignes de ces graphes sont en caractères gras dans le tableau 5.3.

Pour les fonctions de transformation, nous utilisons trois valeurs différentes pour la barrière réalisable/irréalisable r, 20%, 50% et 80%. Pour chaque instance, nous avons réalisé 30 exécutions de l'AG flou avec les paramètres indiqués dans les tableaux 5.4 et 5.5.

Les méthodes ont été codées en C, et ont été exécutées sur un PC Windows avec un CPU intel Core(TM) $i5-5200U@22.20GHz$ et $4.00Go$ de RAM. Rappelons que nous considérons quatre types de contraintes :

(1) Les contraintes de cohabitation (CC),
(2) Contraintes de non cohabitation (CNC).
(3) Contrainte sur le nombre de clusters (CL),
(4) Contrainte sur la taille des clusters (CS).

Dans ce qui suit, nous utiliserons ces numéros d'énumération pour spécifier l'ordre des contraintes. C'est-à-dire qu'un ordre *CC-CNC-CL-CS*, par exemple, sera désigné par

1234. Initialement, l'ordre des contraintes est généré de manière aléatoire. Ensuite, à chaque itération t5, nous lançons le moteur d'inférence floue pour décider si nous devons modifier l'ordre des contraintes ou non.

TABLE 5.4 – Valeurs des paramètres de l'AG

G	taille population	# d'itérations
1	50	20
2	100	30
3	100	50
4	100	50
5	100	50
6	100	60
7	100	70
8	150	100
9	150	100
10	300	150
11	350	200
12	350	200
13	500	200
14	500	300
15	600	300
16	650	300

TABLE 5.5 – Paramètres des opérateurs de l'AG

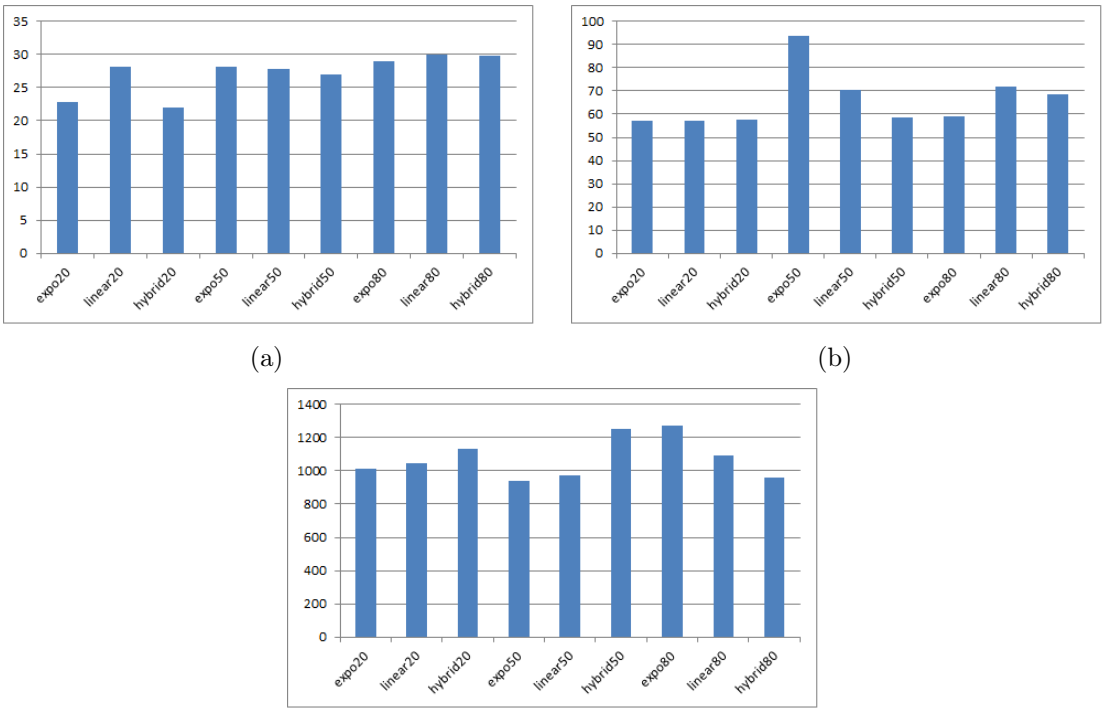
paramètre	valeur
elitism, $t1$	20%
sélection, $t2$	50%
croisement, $t3$	60%
mutation, $t4$	1%
pas SIF, $t5$	10%

5.4.1 Impact de priorisation floue des contraintes

Pour étudier l'impact de l'ordre des contraintes sur la performance de l'AG nous avons utilisé deux mesures : la moyenne de la meilleure valeur de fitness (MBF) et le temps d'exécution moyen.

La figure 5.6 montre le temps d'exécution moyen en secondes lorsque nous appliquons les trois variantes de TF (linéaire, exponentielle et hybride) avec différentes valeurs de r sur les trois instances représentatives. Nous observons que la valeur de la barrière réalisable/irréalisable r a un impact significatif sur les performances de l'AG. Globalement, les valeurs les plus performantes pour r sont de 20% pour la petite classe, 80% pour la moyenne et 50% pour la grande.

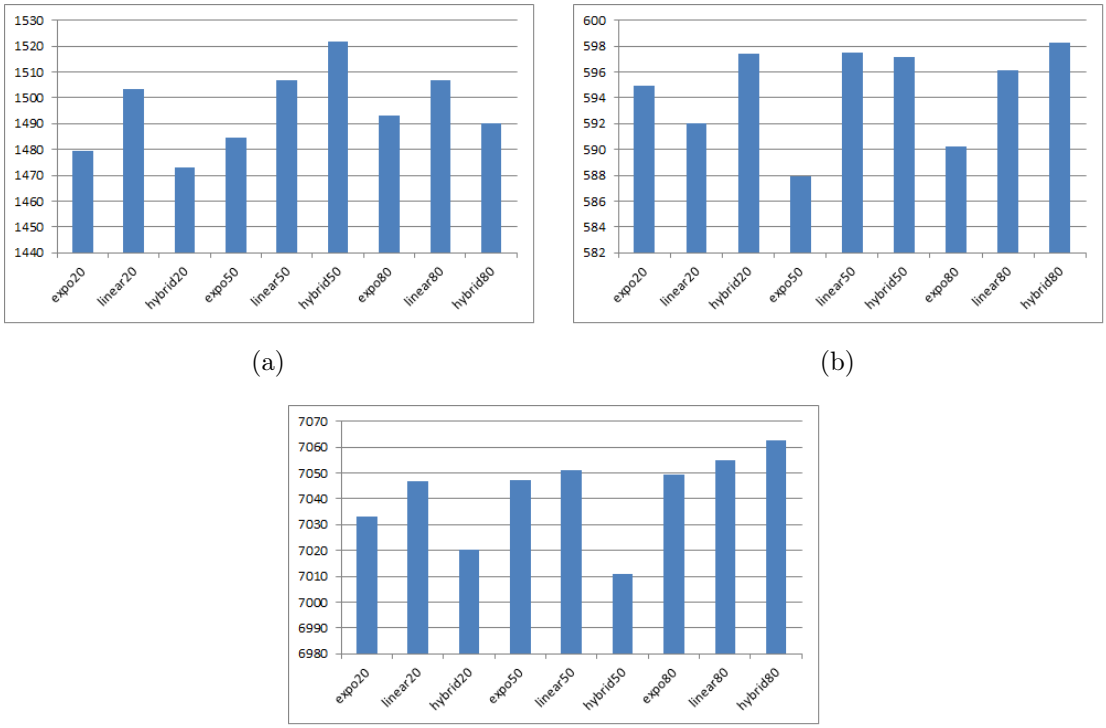
Selon la deuxième métrique (figure 5.7), nous remarquons que pour les petits et grands graphes, la FT hybride est la plus performante, alors que pour les graphes moyens, la FT exponentielle donne les meilleures performances. Par conséquent, pour les petites et



(a) (b)

(c)

FIGURE 5.6 – Temps d'exécution moyen (secondes) pour les petits (a), moyens (b) et (c) grands graphes



(a) (b)

(c)

FIGURE 5.7 – MBF pour les petits (a), moyens (b) et (c) grands graphes

grandes instances, les résultats les meilleurs sont obtenus avec la même FT mais avec différentes valeurs de r, $r = 20\%$ et $r = 50\%$, respectivement.

En plus des résultats précédents, le tableau 5.6 donne la meilleure valeur de la fonction objectif pendant 30 exécutions. Nous remarquons que la FT hybride est vainqueur dans la plupart des instances de graphe, quelle que soit la valeur de r. En fait, l'utilisation d'une FT linéaire pour la région irréalisable et d'une FT exponentielle pour le domaine réalisable semble permettre à l'AG de se diriger facilement vers l'exploitation de l'espace de recherche en tenant compte des bons gènes que les solutions irréalisables peuvent cacher, et ensuite, d'accélérer la phase d'intensification

TABLE 5.6 – Meilleure fitness BF

G	exp20	linear20	hybrid20	exp50	linear50	hybrid50	exp80	linear80	hybrid80
1	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
2	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
3	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
4	147	132	140	161	145	**124**	158	**124**	151
5	2473	2412	2439	2268	2318	**2258**	2390	2343	2347
6	1321	1260	1232	1246	1286	**1226**	1344	1289	1349
7	7133	7339	6939	7287	7115	**6679**	7051	7232	7598
8	**7408**	7727	7820	7762	7591	7591	7762	7814	7706
9	557	**508**	545	545	532	515	531	541	522
10	426	430	425	432	439	**424**	426	434	429
11	2226	2231	2224	2213	2205	**2139**	2205	2205	2244
12	15826	16072	15826	15836	16248	**15813**	16285	16293	15823
13	2635	2631	2854	2918	2912	**2425**	2610	2635	2596
14	7794	7752	7737	7850	7753	**7338**	7724	7753	7743
15	6945	6967	6765	6993	6993	**6873**	6945	6945	**6873**
16	**283689**	284282	284041	284002	284041	283698	284595	284084	**283689**

5.4.2 Application du test de Friedman

Pour vérifier que les performances obtenues sont statistiquement différentes, et que les meilleurs résultats obtenus par la TF hybride ne proviennet pas du hasard, nous avons utilisé la procédure du test de Friedman (Friedman, 1937). Les valeurs de classement présentées dans le tableau 5.7 montrent qu'il existe une nette différence entre les variantes de gestion des contraintes avec un $pvalue \approx 0.0005$ ce qui est bien inférieur au seuil de signification de 0.05.

La moyenne de rang relativement faible de la TF hybride avec $r = 50\%$ nous informe qu'il fait vraissemblablement partie des variantes qui ont provoqué le rejet de l'hypothèse nulle.

En représentant graphiquement les valeurs du tableau 5.7 (voir figure 5.8), nous remarquons une ligne sombre associée à la variante hybride $r = 50\%$. De plus, nous remarquons un segment assez sombre dans le coin supérieur droit, ce qui révèle que la variante hybride $r = 80\%$ qui a donné des performances relativement bonnes pour les grandes instances est également prometteuse.

TABLE 5.7 – Rangs du test de Friedman pour les variantes de gestion des contraintes basée sur FT

G	exp20	linear20	hybrid20	exp50	linear50	hybrid50	exp80	linear80	hybrid80
1	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
2	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
3	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
4	6.0	3.0	4.0	9.0	5.0	1.5	8.0	1.5	7.0
5	9.0	7.0	8.0	2.0	3.0	1.0	6.0	4.0	5.0
6	7.0	4.0	2.0	3.0	5.0	1.0	8.0	6.0	9.0
7	5.0	8.0	2.0	7.0	4.0	1.0	3.0	6.0	9.0
8	1.0	5.0	9.0	6.5	2.5	2.5	6.5	8.0	4.0
9	9.0	1.0	7.5	7.5	5.0	2.0	4.0	6.0	3.0
10	3.5	6.0	2.0	7.0	9.0	1.0	3.5	8.0	5.0
11	7.0	8.0	6.0	5.0	3.0	1.0	3.0	3.0	9.0
12	3.5	6.0	3.5	5.0	7.0	1.0	8.0	9.0	2.0
13	5.5	4.0	7.0	9.0	8.0	1.0	3.0	5.5	2.0
14	8.0	5.0	3.0	9.0	6.5	1.0	2.0	6.5	4.0
15	5.0	7.0	1.0	8.5	8.5	2.5	5.0	5.0	2.5
16	1.5	8.0	5.5	4.0	5.5	3.0	9.0	7.0	1.5
ave.	5.37500	5.43750	4.71875	6.09375	5.43750	2.15625	5.25000	5.65625	4.87500
pvalue	0.0005463								

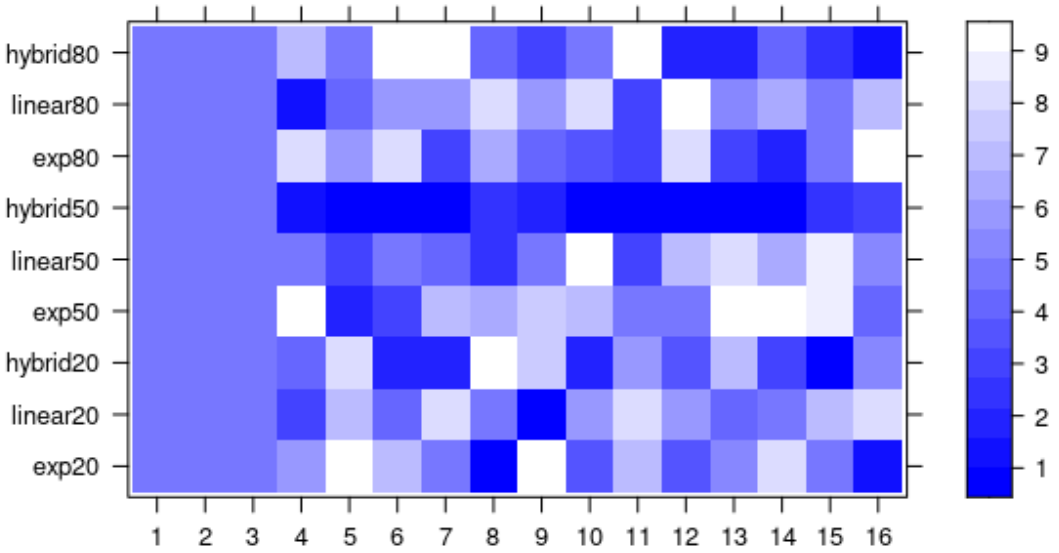


FIGURE 5.8 – Representation des rangs du test de Friedman.

5.4.3 Application du test de post-hoc de Nemenyi

Le rejet de l'hypothèse nulle par le test de Friedman signifie que les différences observées entre les moyennes des échantillons sont significatives et que les moyennes ne sont pas similaires. Il est donc nécessaire de continuer avec un test post-hoc effectuer des comparaisons par paires.

Le tableau 5.8 présente les résultats associés à l'application du test de Nemenyi (Nemenyi, 1963). Les valeurs de ce tableau confirment avec une assez forte évidence le bon

comportement de l'hybride avec $r = 50\%$ par rapport aux autres variantes.

TABLE 5.8 – Comparaisons posthoc

	exp20	linear20	hybrid20	exp50	linear50	hybrid50	exp80	linear80
linear20	1.0000	-	-	-	-	-	-	-
hybrid20	0.9991	0.9982	-	-	-	-	-	-
exp50	0.9982	0.9991	0.8905	-	-	-	-	-
linear50	1.0000	1.0000	0.9982	0.9991	-	-	-	-
hybrid50	0.0249	0.0201	0.1675	0.0016	0.0201	-	-	-
exp80	1.0000	1.0000	0.9998	0.9944	1.0000	0.0376	-	-
linear80	1.0000	1.0000	0.9887	1.0000	1.0000	0.0092	1.0000	-
hybrid80	0.9999	0.9997	1.0000	0.9430	0.9997	0.1129	1.0000	0.9967

5.4.4 Meilleur ordre de contraintes

Afin d'identifier l'ordre qui est à l'origine des performances de l'AG, le tableau 5.9 présente l'ordre des contraintes associé à l'itération dans laquelle la meilleure amélioration de la fitness a eu lieu. On peut constater que pour la même valeur de la barrière r réalisable/irréalisable, il existe différents ordres de contraintes.

De plus, on remarque que certaines séquences de contraintes sont plus répétées. En effet, en considérant les performances globales, les séquences 4123, 4312 et 2134 sont les plus fréquentes avec respectivement 13, 12 et 9 répétitions (dans le tableau 5.9, ces ordres sont en caractères gras).

TABLE 5.9 – Ordre des contraintes ayant donné la meilleure fitness

G	exp20	linear20	hybrid20	exp50	linear50	hybrid50	exp80	linear80	hybrid80
1	4132	**4312**	**4123**	4321	3124	1342	**4123**	1234	2314
2	2314	2341	3421	1432	**2134**	1234	1423	4321	2431
3	1243	2431	1342	2143	**2134**	3421	3124	3142	1234
4	**4123**	2314	**4312**	**2134**	**4213**	1423	**4123**	3142	**4123**
5	**4312**	**4123**	3412	**2134**	1234	3142	**2134**	**4123**	1234
6	4321	2143	1432	2143	4132	3412	1423	2413	**4123**
7	4132	2413	2413	1243	2431	2413	4231	1423	1432
8	**4123**	1243	**4312**	2314	3421	3142	1432	1423	1423
9	2314	2413	**4312**	3421	3241	2314	2143	4321	1243
10	3142	**2134**	3142	**4123**	**2134**	**4312**	1342	2314	1432
11	**4123**	**4312**	3241	4231	3214	2314	2134	**2134**	1324
12	1342	**4123**	4213	4213	3124	1243	3124	3421	4231
13	4213	1243	**4312**	3421	4321	**4123**	2143	1423	1234
14	3214	1243	**4312**	**4312**	**4312**	1324	1324	4321	1243
15	3124	3124	4321	3421	1243	2143	3241	3412	**4312**
16	1342	**4312**	1423	4231	3421	3241	**4123**	1432	2314
Occurrences fréquentes	4123	4312	4312	3421	2134	3142	1423	4321	1234

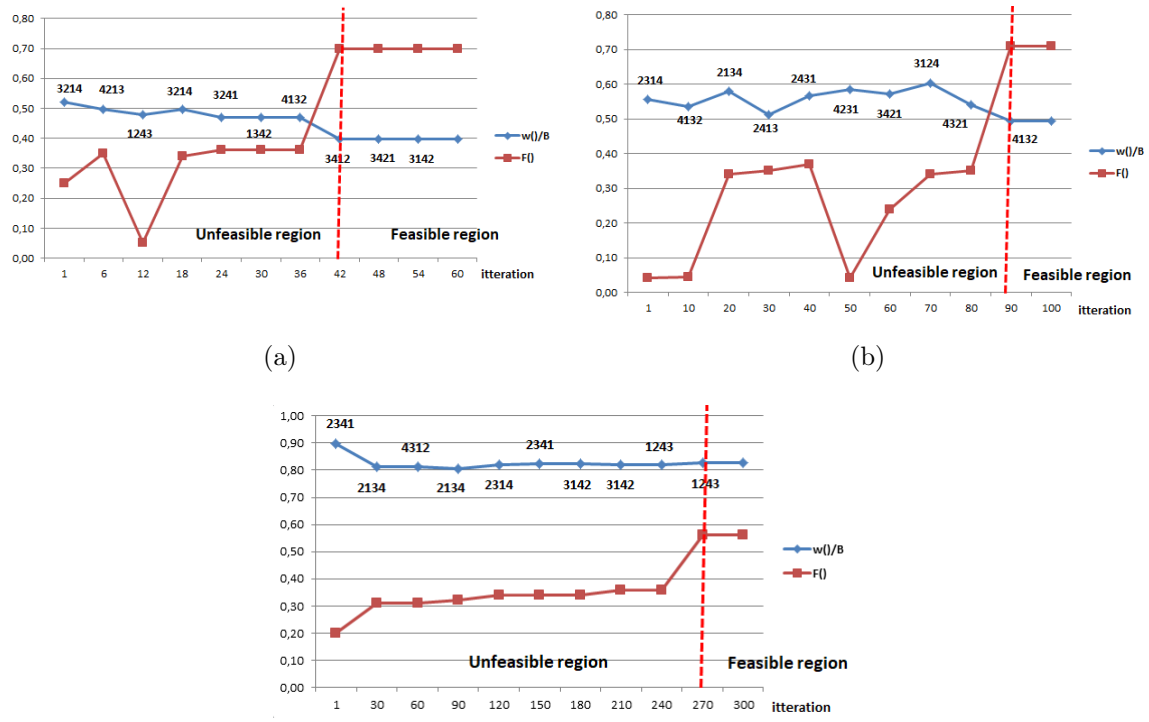
5.4.5 Impact de l'ordre de contraintes sur la performance de l'AG

Afin d'avoir une vue plus précise de l'impact de l'ordre des contraintes sur le GA lorsqu'il se dirige vers la meilleure solution trouvée, la figure 5.9 représente à la fois la valeur de la meilleure fitness et la fonction objectif associée pendant l'exécution de la variante FT *hybride*50. A chaque lancement du système d'inférence floue, nous mentionnons le nouvel ordre des contraintes.

Pour la classe de petits graphes (Figure 5.9(a)), l'ordre initial 3214 a permis une légère amélioration. Par la suite, la séquence 4213 a généré un mauvais effet sur la fitness. En effet, après l'application d'un nouvel ordre, toutes les solutions sont réévaluées, et donc, les meilleurs individus qui étaient l'élite de la population peuvent ne plus posséder le même statut. Ensuite, les séquences 1243 et 3214 ont donné deux améliorations consécutives. Après deux tours stables, l'ordre 4132 a permis une bonne amélioration qui a aidé l'AG à atteindre le domaine réalisable. La solution obtenue était la meilleure performance de l'AG dans ce cycle.

Pour la classe de moyens graphes (Figure 5.9(b)), les meilleurs sauts de fitness provenaient de l'ordre 4132 qui a donné la première amélioration, puis 4231, 3421 et 4321, ce qui révèle l'influence des contraintes CL et CS.

Pour les grandes instances (Figure 5.9(c)), l'ordre initial 2341 a permis à l'AG de donner une amélioration significative. Cependant, l'impact de l'ordre 1243 sur le fitness révèle que CNC a eu une plus grande influence dans l'amélioration du résultat de l'AG pour cette instance.



(a) (b)

(c)

FIGURE 5.9 – Évolution de l'ordre des contraintes pour les graphes représentatifs de petite (a), moyenne (b) et (c) grande taille pour *hybrid*50

Enfin, dans les trois instances, nous remarquons qu'après avoir atteint le domaine réalisable, l'AG flou a du mal à améliorer ses performances. Cela suggère que l'AG pourrait avoir besoin d'une aide supplémentaire dans sa phase d'intensification, par exemple en utilisant la recherche locale.

5.4.6 Etude comparative

Afin d'examiner la performance de l'AG flou proposé (algorithme 3) par rapport à d'autre approches de la littérature, nous avons choisi de le comparer aux méthodes suivantes :

— L'approche de gestion des contraintes statiques (Morales and Quezada, 1998) qui est, selon Coello (Coello, 2002), " une approche intéressante de la pénalité statique ".
— La FT hybride statique présentée dans le chapitre précédent.

Pour ces deux méthodes, que nous désignerons respectivement par $MoralesCH$ et $mixedTF$, nous utiliserons notre implémentation tout en respectant la description donnée par les auteurs de ces approches.

Le tableau 5.10 montre les résultats expérimentaux obtenus par la méthode hybride flou avec $r = 50\%$, $mixedTF$ et $MoralesCH$lors de 30 exécutions indépendantes. Comme on peut le constater, les meilleurs résultats sont donnés par la méthode méthode hybride flou proposée, dans la plupart des graphes. (Lorsqu'une méthode donne le meilleur fitness, médiane ou moyenne, la valeur correspondante est mise en gras). En outre, étant donné les valeurs relativement faibles de l'écart type σ, nous pouvons conclure que la méthode floue est plus stable, ses performances étant plus regroupées autour de la moyenne.

TABLE 5.10 – Résultats expérimentaux utilisant la TF hybride floue avec $r = 50\%$, $MoralesCH$ et $mixedTF$ pour 30 exécutions

	Fuzzy hybrid TF($r = 50\%$)					MoralesCH					mixedTF				
Graph	best	worst	median	stdiviation	mean	best	worst	median	stdiviation	mean	best	worst	median	stdiviation	mean
1	**2**	**2**	**2**	0	**2**	**2**	**2**	**2**	0	**2**	**2**	3	**2**	0.3	2.1
2	**5**	7	**5**	0.93	**5.6**	**5**	7	**5**	0.99	5.8	**5**	8	7	**0.71**	7.1
3	**2**	**2**	**2**	0	**2**	6	28	17	5.93	16.08	6	38	17	8.60	16.84
4	**125**	191	**168.5**	**15.21**	**167.43**	192	231	219	45.94	215	171	254	224	19.74	223.41
5	**2258**	2738	**2545.5**	**116.92**	**2540.6**	2590	3278	3062	198.76	2989.5	2329	3333	3035	231.34	2995.76
6	**1226**	1623	**1499**	**64.32**	**1483.57**	1566	2104	1857	159.62	1832.006	1513	2188	1823	151.51	1825.63
7	**6679**	8958	**8348.5**	**563.025**	**8175.26**	8758	11289	9718.06	847.02	10008.05	8540	12249	9776	754.26	9831.05
8	**7591**	8477	**8197**	**228.51**	**8161.83**	8677	11546	9578	819.098	9638	8280	10493	9271	531.50	9348.9
9	**515**	639	**603**	**25.87**	**597.13**	635	767	705	38.46	706.94	602	823	713	47.62	719.49
10	**424**	459	**448.5**	**9.16**	**445.56**	463	544	492	24, 79	498.74	472	535	508.48	14.25	506.68
11	**2139**	2420	**2352**	**63.85**	**2331.76**	2481	2854	2719	128.25	2715.34	2477	2813	2602	96.19	2609.26
12	**15813**	17133	**16565**	**349.98**	**16546.9**	17306	19340	18723	490.58	18636.81	16766	18691	18128	467.38	18005.86
13	**2425**	3187	**2998**	145.68	**2966.27**	2853	3720	3017	266.30	3078.32	2634	3188	3017	127.74	2988
14	**7338**	8686	**7967**	280.00	**8028.033**	7913	9757	8557	721.81	8796.81	7708	9757	8113	653.96	8401.75
15	6873	7133	**7008.5**	73.46	7010.86	6974	7287	7150	100.75	7140	**6872**	7096	7013	**63.45**	**7008.36**
16	283689	285646	**284301**	506.44	**284449.5**	284595	285646	284595	336.31	284858.96	**283685**	287065	285682	1032.01	285517.9

La figure 5.10 présente une vue graphique où les meilleures valeurs de la fonction objectif pendant les 30 exécutions sont compilées en utilisant des boxplots. Nous remarquons que pour presque tous les graphs, en plus d'avoir la médiane la plus basse et les moustaches les plus profondes, l'AG flou a les plus petits boxplots en comparaison avec ceux de $MoralesCH$ et $mixedTF$. Cela confirme que l'approche par ordre de contraintes floues est dans l'ensemble la meilleure méthode, et est plus stable que $MoralesCH$ et $mixedTF$.

5.4.7 Etude comparative avec DFA et SGA

Nous avons également comparé notre approche, l'AG flou, avec l'AG simple (SGA) et un algorithme inspiré de la nature, à savoir l'algorithme discret de la luciole (Discrete Firefly Algorithm (DFA)) (Yang, 2009; Osaba et al., 2016). Cette comparaison nous a permis d'obtenir une autre perspective dans l'analyse de notre approche proposée.

Nous avons mis en œuvre ces deux méta-heuristiques à base de population en utilisant une approche naïve pour la gestion des contraintes. C'est-à-dire que si une solution irréalisable est rencontrée, elle sera sévèrement pénalisée en lui attribuant la pire valeur de fitness possible (pour PPG, nous utilisons la somme de tous les poids des arêtes du graphe). Ainsi, pour toutes les solutions irréalisables, la valeur de fitness sera la même, quel que soit le nombre et le type de contraintes violées.

Les valeurs des paramètres que nous avons utilisés pour SGA sont similaires à celles de l'AG flou (voir tableaux 5.4 et 5.5). Pour DFA, les paramètres sont indiqués dans le tableau 5.11. Pour le paramètre γ, nous avons utilisé la valeur 0.95, comme indiqué dans (Osaba et al., 2016), et pour calculer la distance entre les lucioles, nous avons utilisé la métrique de Hamming.

Table 5.11 – valeur de paramètre DFA

G	# de firflies	# d'itérations
1	50	20
2	100	30
3	100	50
4	100	50
5	100	50
6	100	60
7	100	70
8	150	100
9	150	10
10	150	10
11	150	10
12	150	10
13	150	10
14	150	10
15	150	10
16	150	10

Le tableau 5.12 montre les résultats obtenus (meilleure fitness (BF) et meilleur temps d'exécution (BRT) en secondes) après avoir effectué trente (30) exécutions pour chaque

TABLE 5.12 – Résultats expérimentaux utilisant la TF hybride floue avec $r = 50\%$, SGA et DFA pour 30 exécutions indépendants

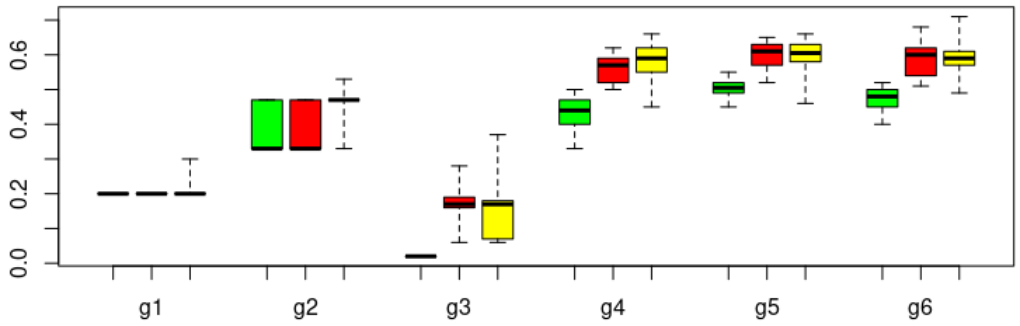
	Fuzzy hybrid TF		SGA		DFA	
Graph	BF	BRT	BF	BRT	BF	BRT
1	**2**	5.863	–	0.016	–	15.937
2	**5**	17.165	**5**	1.471	12	81.322
3	**2**	20.651	6	2.449	77	138.313
4	**125**	24.699	198	2.759	356	242.797
5	**2258**	20.31	2348	2.7	4069	148.803
6	**1226**	28.084	1586	3.916	2430	7.18
7	**6679**	30.896	8858	5.516	11567	17.417
8	**7591**	92.848	8300	10.263	11583	226.785
9	**515**	56.56	620	14.709	895	236.709
10	**424**	200.043	441	69.299	551	1560.06
11	**2139**	317.704	2294	100.69	–	–
12	**15813**	354.047	16963	154.908	–	–
13	**2425**	685.529	2854	459.261	–	–
14	**7338**	1078.71	8007	150.439	–	–
15	**6873**	955.608	7295	204.476	–	–
16	**283698**	2098.85	285646	1374.91	291469	60244.9

méthode. L'AG flou avec une FT hybride et $r = 50\%$ a donné les meilleures performances en termes de meilleure fitness pour tous les graphes. Nous avons également remarqué que DFA avait des difficultés à atteindre le domaine de solution réalisable (voir les performances du DFA du 11^{eme} au 15^{eme} graphe), ce qui est confirmé par la position des boxplots du DFA et l'absence de moustaches comme le montre la figure 5.11. En effet, à partir de la 10^{eme} instance, la quasi-totalité des résultats du DFA pour les trente exécutions sont condensés à la valeur de fitness normalisée 1.0, ce qui correspond à la valeur de fitness la plus élevée.

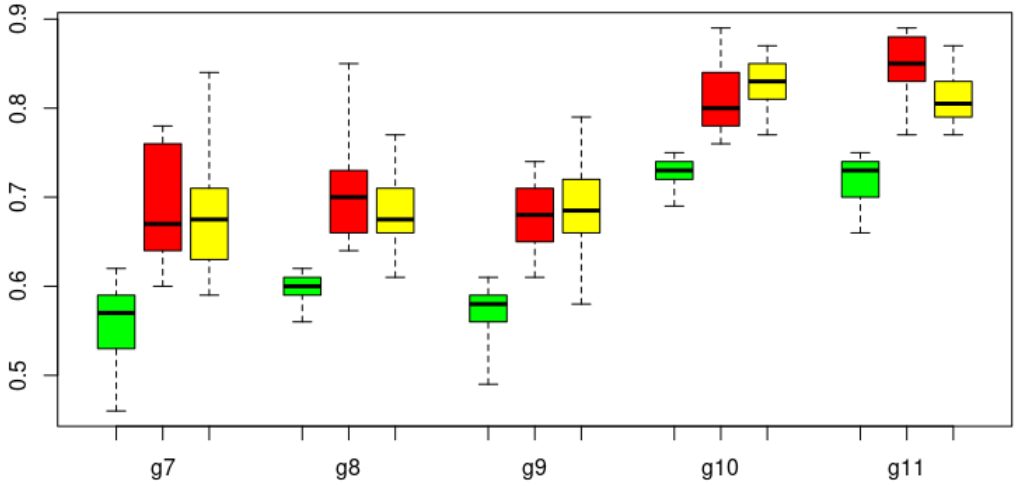
5.5 Conclusion

Dans ce chapitre, nous avons étudié les conséquences de la priorisation dynamique des contraintes sur l'algorithme génétique résolvant une version fortement contrainte du problème de partitionnement des graphes. Nous avons proposé d'équiper l'AG d'une technique de gestion des contraintes floue basée sur l'approche de fonction de transformation. Cette méthode proposée permet à l'AG de modifier les priorités des contraintes en fonction de ses performances actuelles afin de résoudre le problème. L'étude expérimentale présentée permet de conclure que la façon dont les contraintes sont ordonnées influe sur les performances de l'AG.

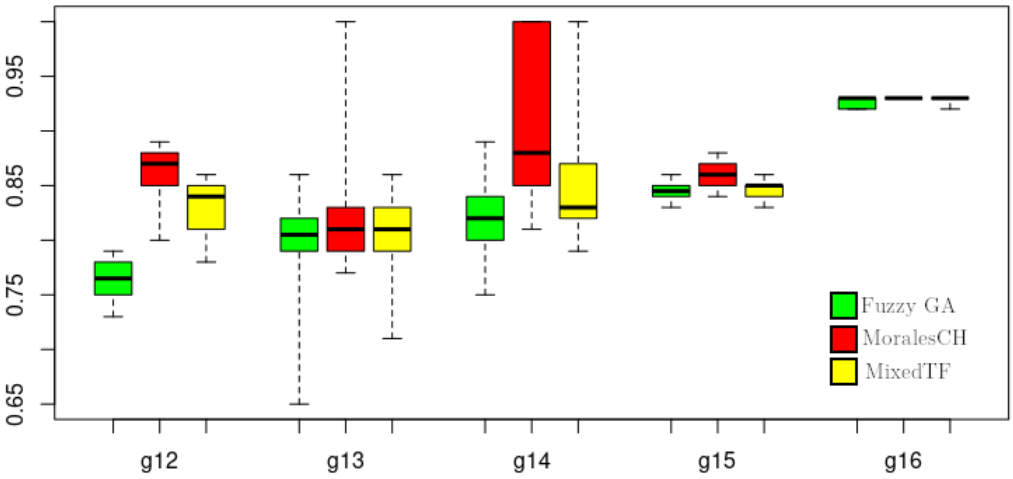
En outre, la différence dans l'ordre de contrainte optimal entre différentes instances GPP, et tout au long du processus d'optimisation dans une même instance, suggère qu'il n'y a pas qu'un seul ordre qui aide l'AG à se rapprocher à l'espace réalisable. Il est donc très important d'ajuster l'ordre au cours du processus de recherche.



(a) Small instances

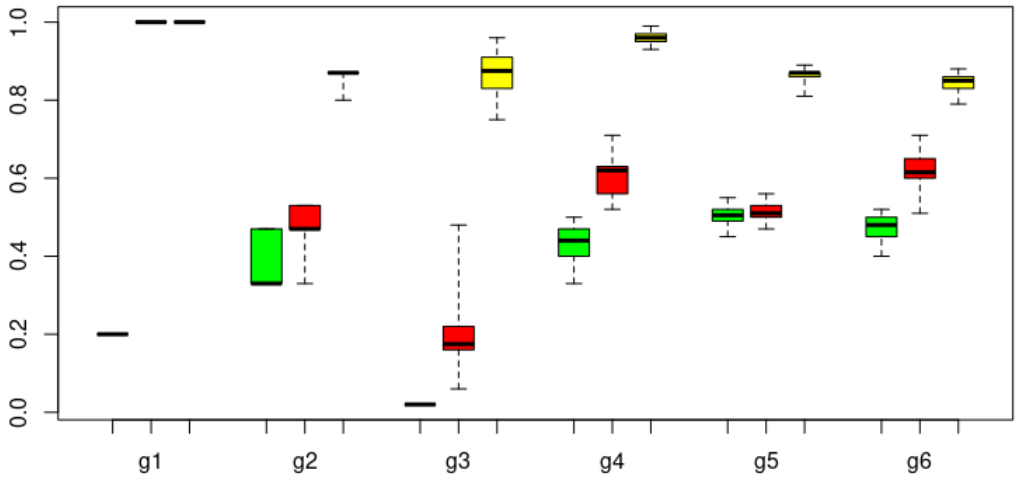


(b) Medium instances

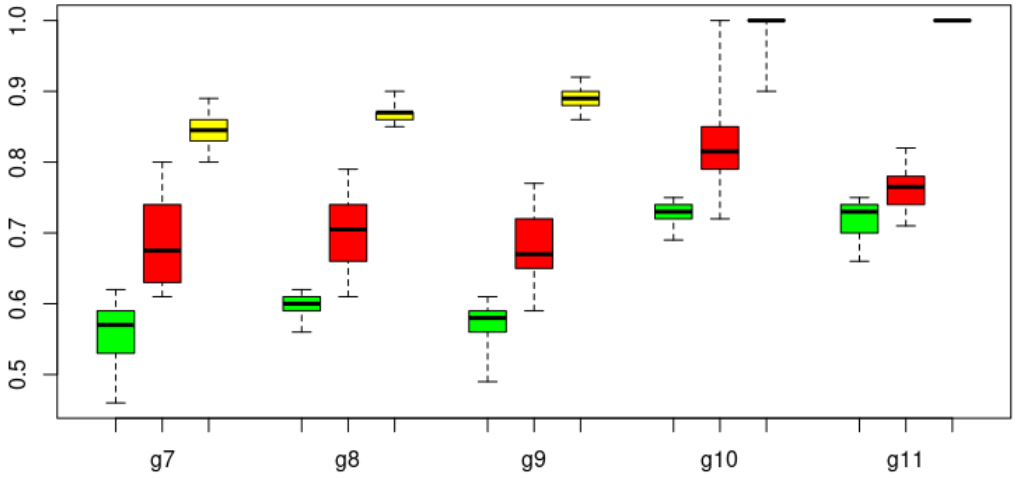


(c) Large instances

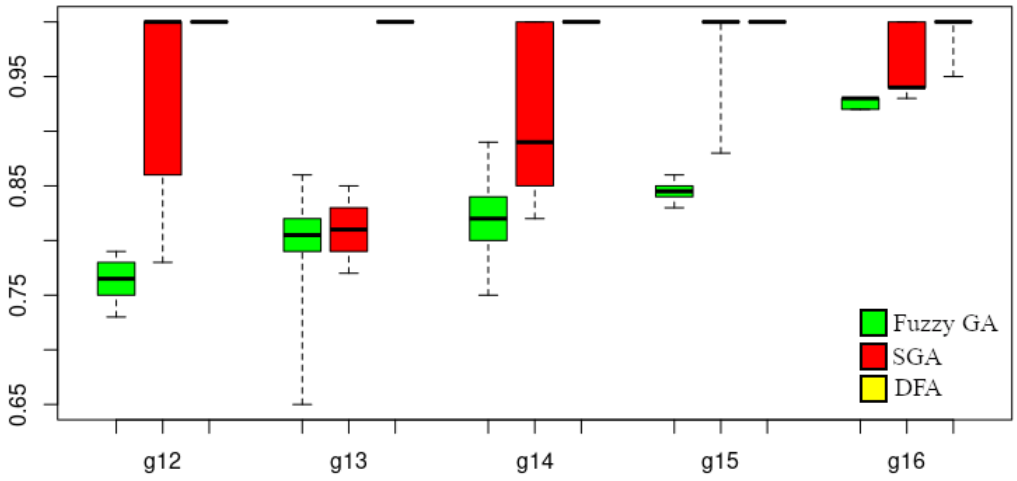
FIGURE 5.10 – Boxplots associés aux 30 exécutions des trois méthodes.



(a) Small instances



(b) Medium instances



(c) Large instances

FIGURE 5.11 – Boxplots associés aux 30 exécutions des trois méthodes bio-inspirées.

Conclusion générale

L'algorithme génétique (AG) est une méthode d'optimisation stochastique basée sur les principes de la théorie de l'évolution. L'AG imite les mécanismes de la sélection naturelle pour essayer de trouver de bonnes solutions à des problèmes difficiles à résoudre de manière optimale.

L'algorithme génétique entame son processus de recherche de bonnes solutions par la génération d'une population de solutions potentielles, appelées individus, qui sont ensuite évaluées en fonction de la valeur de la fonction objectif du problème. Cette évaluation procure une mesure de l'adaptation des individus qui est un facteur déterminant dans le mécanisme de « survie du meilleur » chez les espèces naturelles. Les individus sont ensuite sélectionnés selon leur degré d'adaptation pour subir des opérateurs génétiques tels que le croisement et la mutation, pour former une nouvelle génération. Ce processus est répété jusqu'à ce qu'une condition d'arrêt soit atteinte, par exemple, en fixant un nombre maximal d'itérations. Les meilleurs individus de la dernière génération sont les solutions trouvées par l'AG.

L'algorithme génétique, dans sa forme originelle, n'est pas conçu pour résoudre des problèmes avec contraintes. Les solutions qui ne respectent pas ces contraintes sont dites irréalisables ou inadmissibles. Il s'avère que ces solutions, bien qu'elles soient irréalisables, sont susceptibles de cacher des informations (ou code génétique) qui peuvent aider l'algorithme génétique à converger vers l'optimum, ou à défaut, vers de bonnes solution. Il convient donc de ne pas éliminer complètement ces solutions de la population, mais de les exploiter afin de booster le processus de recherche de bonnes solutions par l'AG. Des techniques de gestion de contraintes existent et offrent à l'AG la possibilité de gérer les solutions irréalisables. Parmi celles-ci, l'utilisation de fonctions de transformation (FT) est une voie d'investigation prometteuse.

Dans cette thèse, nous avons proposé deux contributions qui consistent à doter l'AG d'un mécanisme de gestion de contraintes basé sur les FT, afin de résoudre des problèmes difficiles faiblement ou fortement contraints. Pour mettre en œuvre notre approche, nous avons opté pour le problème de partitionnement de graphes semi-supervisé (Chaouche and Boulif, 2019). Ces contributions sont décrites dans ce qui suit :

Dans la première (Alouane and Boulif, 2021), nous avons étudié l'impact du changement de la valeur r (valeur qui, dans la définition d'une FT, sépare la région réalisable de la région irréalisable) sur les résultats de l'AG. Après une étude comparative, nous avons constaté que r affecte de manière significative les performances de l'AG. De plus, nous avons constaté que l'utilisation d'une FT linéaire pour la région irréalisable, et d'une FT exponentielle pour la région réalisable permet à l'AG de bien gérer sa phase d'exploitation en prenant en compte les bons gènes que cachent les solutions irréalisables, puis d'accélérer la phase d'intensification. Dans une autre étude, nous avons comparé notre approche

à la méthode proposée dans (Morales and Quezada, 1998). Cette étude nous a confirmé le bon comportement de notre approche.

Dans la deuxième contribution (Alouane and Boulif, 2023), nous avons étudié l'impact de l'ordre de priorité des contraintes sur la performance de l'AG utilisant les FT. Nous avons proposé d'équiper l'AG d'une procédure de gestion floue des contraintes. Dans cette approche, nous permettons à l'AG d'ajuster les priorités des contraintes afin de trouver l'ordre qui lui permet de mieux converger vers la région réalisable, puis vers de bonnes solutions. L'étude expérimentale avec différentes méthodes basées sur les FT nous a permis de constater la performance de notre approche floue. Afin de valider les résultats trouvés, nous avons réalisé des tests statistiques (tests de Friedman et Nemenyi). Les résultats des tests ont été en faveur de notre approche. Cependant, des insuffisances ont été constaté quant au comportement de l'AG flou une fois la partie réalisable atteinte. Ceci suggère que notre approche pourrait être améliorée dans la phase d'intensification.

Comme travail futur, il peut être très intéressant de voir comment l'AG flou se comporterait face à d'autres problèmes difficiles fortement contraints, tels que les problèmes d'ordonnancement et d'emploi du temps. En outre, nous sommes aussi intéressés par l'étude de l'inclusion de notre approche par FT, vu sa généricité, dans d'autres métaheuristiques bio-inspirées ou non, afin d'analyser son effet sur leur comportement.

Bibliographie

Aarts, E. H. et al. (1987). Simulated annealing : Theory and applications.

Alouane, B. (2022). figshare. dataset. volume 1. figshare.

Alouane, B. and Boulif, M. (2021). An approach for embedding constraint handling in the genetic algorithm fitness to solve heavily constrained problems. In *Recent Advances in Communication Technology, Computing and Engineering*, pages 242–259. Moulay Ismail University, Meknes, Morocco.

Alouane, B. and Boulif, M. (2023). Fuzzy constraint prioritization to solve heavily constrained problems with the genetic algorithm. *Engineering Applications of Artificial Intelligence*, 119 :105768.

Bader, D. A., Meyerhenke, H., Sanders, P., and Wagner, D. (2011). 10th dimacs implementation challenge-graph partitioning and graph clustering.

Baghel, M., Agrawal, S., and Silakari, S. (2012). Article : Survey of metaheuristic algorithms for combinatorial optimization. *International Journal of Computer Applications*, 58(19) :21–31.

Barbosa, H. J. and Lemonge, A. C. (2008). *An adaptive penalty method for genetic algorithms in constrained optimization problems*. Citeseer.

Bazaraa, M. S., Jarvis, J. J., and Sherali, H. D. (2011). *Linear programming and network flows*. John Wiley & Sons.

Bean, J. C. and ben Hadj-Alouane, A. (1993). *A dual genetic algorithm for bounded integer programs*.

Bertsekas, D. P. (1997). Nonlinear programming. *Journal of the Operational Research Society*, 48(3) :334–334.

Boulif, M. and Atif, K. (2006). A new branch-&-bound-enhanced genetic algorithm for the manufacturing cell formation problem. *Computers & operations research*, 33(8) :2219–2245.

Calégari, P., Coray, G., Hertz, A., Kobler, D., and Kuonen, P. (1999). A taxonomy of evolutionary algorithms in combinatorial optimization. *J. Heuristics*, 5(2) :145–158.

Carroll, C. W. (1961). The created response surface technique for optimizing nonlinear, restrained systems. *Operations research*, 9(2) :169–184.

Chang, J., Wang, L., Meng, G., Xiang, S., and Pan, C. (2017). Deep adaptive image clustering. In *Proceedings of the IEEE international conference on computer vision*, pages 5879–5887.

Chaouche, A. and Boulif, M. (2019). Solving the unsupervised graph partitioning problem with genetic algorithms : Classical and new encoding representations. *Computers & Industrial Engineering*, 137 :106025.

Clerc, M. (2010). *Particle swarm optimization*, volume 93. John Wiley & Sons.

Coello, C. A. C. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms : a survey of the state of the art. *Computer methods in applied mechanics and engineering*, 191(11-12) :1245–1287.

Colorni, A., Dorigo, M., and Maniezzo, V. (1990). Genetic algorithms and highly constrained problems : The time-table case. In *International Conference on Parallel Problem Solving from Nature*, pages 55–59. Springer.

Cook, S. A. (2021). The complexity of theorem-proving procedures (1971).

Courant, R. (1943). Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American mathematical Society*, 49(1) :1–23.

Craenen, B., Eiben, A., and Marchiori, E. (2001). How to handle constraints with evolutionary algorithms. *Practical Handbook Of Genetic Algorithms : Applications*, pages 341–361.

Eiben, A. and Smith, J. (2015). *Introduction to Evolutionary Computation*. Springer, Berlin, Heidelberg, 2 edition.

Eppel, S. (2016). Tracing liquid level and material boundaries in transparent vessels using the graph cut computer vision approach. *arXiv preprint arXiv :1602.00177*.

Fiacco, A. V. and McCormick, G. P. (1966). Extensions of sumt for nonlinear programming : equality constraints and extrapolation. *Management science*, 12(11) :816–828.

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.

Golberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989(102) :36.

Hansen, P. and Mladenovic, N. (2003). A tutorial on variable neighborhood search. *Les Cahiers du GERAD ISSN*, 711 :2440.

Hartmanis, J. (1982). Computers and intractability : a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1) :90.

Helio J.C. Barbosa, A. C. L. and Bernardino, H. S. (2015). *A Critical Review of Adaptive Penalty Techniques in Evolutionary Computation*, chapter 1. Infosys Science Foundation Series. R. Datta and K. Deb.

Hendrickson, B. and Leland, R. (1995). An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2) :452–469.

Holland, J. (1975). Adaptation in natural and artificial systems, univ. of mich. press. *Ann Arbor*.

Homaifar, A., Qi, C. X., and Lai, S. H. (1994). Constrained optimization via genetic algorithms. *Simulation*, 62(4) :242–253.

Joines, J. A. and Houck, C. R. (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's. In *Proceedings of the first IEEE conference on evolutionary computation. IEEE world congress on computational intelligence*, pages 579–584. IEEE.

Kahng, A. B., Lienig, J., Markov, I. L., and Hu, J. (2011). *VLSI physical design : from graph partitioning to timing closure*, volume 312. Springer.

Kim, H.-J. and Kim, Y.-H. (2018). Recent progress on graph partitioning problems using evolutionary computation. *arXiv preprint arXiv :1805.01623*.

Kim, J., Hwang, I., Kim, Y.-H., and Moon, B.-R. (2011). Genetic approaches for graph partitioning : a survey. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 473–480.

Kim, J.-H. and Myung, H. (1997). Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 1(2) :129–140.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598) :671–680.

Mamdani, E. H. and Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*, 7(1) :1–13.

Merchichi, S. and Boulif, M. (2015). Constraint-driven exact algorithm for the manufacturing cell formation problem. *European Journal of Industrial Engineering*, 9(6) :717–743.

Michalewicz, Z. (1995). A survey of constraint handling techniques in evolutionary computation methods. *Evolutionary programming*, 4 :135–155.

Michalewicz, Z. and Nazhiyath, G. (1995). Genocop iii : A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, volume 2, pages 647–651. IEEE.

Morales, A. K. and Quezada, C. V. (1998). A universal eclectic genetic algorithm for constrained optimization. In *Proceedings of the 6th European congress on intelligent techniques and soft computing*, volume 1, pages 518–522.

Myung, H. and Kim, J.-H. (1996). Hybrid evolutionary programming for heavily constrained problems. *BioSystems*, 38(1) :29–43.

Nemenyi, P. (1963). *Distribution-free Multiple Comparisons*. Princeton University.

Newman, M. E. (2013). Community detection and graph partitioning. *Europhysics Letters*, 103(2) :28003.

Osaba, E., Carballedo, R., Yang, X.-S., and Diaz, F. (2016). An evolutionary discrete firefly algorithm with novel operators for solving the vehicle routing problem with time windows. In *Nature-inspired computation in engineering*, pages 21–41. Springer.

Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial optimization : algorithms and complexity*. Courier Corporation.

Pochet, Y. and Wolsey, L. A. (2006). *Production planning by mixed integer programming*, volume 149. Springer.

Rahimi, I., Gandomi, A. H., Chen, F., and Mezura-Montes, E. (2023). A review on constraint handling techniques for population-based algorithms : from single-objective to multi-objective optimization. *Archives of Computational Methods in Engineering*, 30(3) :2181–2209.

Runarsson, T. P. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on evolutionary computation*, 4(3) :284–294.

Saha, C., Das, S., Pal, K., and Mukherjee, S. (2014). A fuzzy rule-based penalty function approach for constrained evolutionary optimization. *IEEE transactions on cybernetics*, 46(12) :2953–2965.

Schulz, F., Wagner, D., and Zaroliagis, C. (2002). Using multi-level graphs for timetable information in railway systems. In *Workshop on Algorithm Engineering and Experimentation*, pages 43–59. Springer.

Smith, A. E., Coit, D. W., Baeck, T., Fogel, D., and Michalewicz, Z. (1997). Penalty functions. *Handbook of evolutionary computation*, 97(1) :C5.

Song, Z., Yang, X., Xu, Z., and King, I. (2021). Graph-based semi-supervised learning : A comprehensive review. *arXiv preprint arXiv :2102.13303*.

Takahama, T. and Sakai, S. (2006). Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites. In *2006 IEEE international conference on evolutionary computation*, pages 1–8. IEEE.

Talbi, E.-G. (2009). *Metaheuristics : from design to implementation*. John Wiley & Sons.

Tolliver, D. A. and Miller, G. L. (2006). Graph partitioning by spectral rounding : Applications in image segmentation and clustering. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 1053–1060. IEEE.

Vazirani, V. V. (2001). *Approximation algorithms*, volume 1. Springer.

Walshaw, C. (2016). The graph partitioning archive, 2014. *URL http ://staffweb. cms. gre. ac. uk/wc06/partition. Zugegriffen am*, 30.

Winston, W. L. (2004). *Operations research : applications and algorithm*. Thomson Learning, Inc.

Yang, X.-S. (2009). Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms*, pages 169–178. Springer.

Yang, X.-S. (2010). *Engineering Optimization : An Introduction with Metaheuristic Applications*. Wiley Publishing, 1 edition.

Zadeh, L. A. (1996). Fuzzy sets. In *Fuzzy sets, fuzzy logic, and fuzzy systems : selected papers by Lotfi A Zadeh*, pages 394–432. World Scientific.

Zbigniew, M. (1996). Genetic algorithms+ data structures= evolution programs. In *Computational Statistics*, pages 372–373. Springer-Verlag.