PolyPublie
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Title:	Comparaison empirique de deux stratégies de recherche locale
Auteur: Author:	Robin Moine
Date:	2023
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Moine, R. (2023). Comparaison empirique de deux stratégies de recherche locale [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/54179/

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/54179/
Directeurs de recherche: Advisors:	Daniel Aloise
Programme: Program:	Génie informatique

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Comparaison empirique de deux stratégies de recherche locale

ROBIN MOINE
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Juillet 2023

© Robin Moine, 2023.

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Comparaison empirique de deux stratégies de recherche locale

présenté par **Robin MOINE**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Quentin CAPPART, président
Daniel ALOISE, membre et directeur de recherche
Alysson MACHADO COSTA, membre

DÉDICACE

À ma famille qui m'a soutenu
toutes ces années

REMERCIEMENTS

Tout d'abord je tiens à remercier sincèrement Daniel Aloise mon directeur de recherche pour m'avoir guidé dans ce projet et avoir été à mon écoute. J'ai particulièrement apprécié la richesse de nos échanges, ses retours constructifs tant durant la période d'expérimentations de ma maîtrise que lors de la rédaction de celle-ci. Il m'a fait bénéficier de son expertise scientifique pour me guider dans le choix d'expériences pertinentes. J'ai appris où concentrer mes efforts et savoir quand abandonner des idées. Il a toujours porté un regard critique et compréhensif sur mon travail et a pris en compte mes suggestions. Il m'a accordé du temps et de la compréhension quand j'éprouvais des difficultés. Dans la rédaction de mon travail, il a été d'une grande aide pour m'apprendre à rédiger de façon concise mes résultats. Finalement, je le remercie pour son attention : Il m'a laissé le temps de suivre mes cours pour ensuite me concentrer sur ma recherche. Il a été à l'écoute pour faciliter mon travail en alternant travail distanciel et présentiel.

Ensuite, je souhaite remercier Jonathan Jalbert pour son aide et son regard critique sur le pipeline statistique conçu.

Je souhaite également remercier mes amis du laboratoire : Clara, Andressa, Farin, Paul, Thiago, Quentin, Arnaud. Merci pour vos retours sur mon travail lors de ma présentation et pour le partage de vos expériences. Thiago merci pour tes conseils sur les tests, cela m'a permis d'améliorer la qualité de mon rendu. Merci Arnaud pour nos échanges et ton aide pour l'accès au serveur de calcul : tu m'as évité de longues nuits de travails et de calcul. Tu as toujours été disponible pour installer des packages et résoudre des problèmes d'accès au serveur.

Enfin, je souhaite remercier ma famille qui a toujours été là pour me soutenir et m'encourager. Merci pour votre écoute attentive et votre persévérance pour tenter de comprendre mon projet malgré nos domaines professionnels très différents. Avoir un soutient moral solide malgré la distance m'a permis d'avancer sur mon projet avec un esprit plus serein.

RÉSUMÉ

Le *Problème du Voyageur de Commerce* (TSP) est un problème d'optimisation combinatoire NP-difficile. Une étude comparative des stratégies de recherche locale "First Improvement" ou *Premier améliorant* (*PA*) et "Best Improvement" ou *Meilleur améliorant* (*MA*) a été réalisée pour le TSP dans l'article intitulé "First vs Best Improvement : an empirical study" par Hansen et Mladenovic, 2006 [1]. Cette étude a comparé empiriquement ces deux stratégies et a recommandé une règle pour choisir la meilleure option en fonction de la qualité de la solution initiale.

La règle proposée par Hansen et Mladenovic, 2006 [1] est la suivante : si la solution initiale est choisie de manière aléatoire, la stratégie de recherche locale *PA* donne en moyenne une solution finale de meilleure qualité. En revanche, si la solution initiale est obtenue à l'aide d'une heuristique gloutonne, la stratégie *MA* donne en moyenne de meilleurs résultats. Cette étude est devenue une référence dans la littérature de recherche, avec de nombreuses citations, et elle est utilisée comme base pour d'autres problèmes NP-difficiles.

Ce mémoire vise à vérifier la validité de la règle proposée par Hansen et Mladenovic, 2006 [1] pour le *Problème du Voyageur de Commerce* (TSP) et à déterminer si cette règle peut également s'appliquer à d'autres problèmes NP-difficiles. L'accent sera mis sur le TSP pour vérifier les résultats de Hansen et Mladenovic, 2006 [1]. Cependant, afin d'évaluer l'applicabilité de cette règle à d'autres problèmes NP-difficiles, nous examinerons également le *Problème de regroupements avec Minimisation de la Somme des Carrés* (MSSC) et le *MAXSAT-pondéré*. Ces problèmes serviront d'exemples pour montrer que les conclusions de [1] ne sont pas nécessairement applicables à d'autres problèmes NP-difficiles, en fonction des ensembles de données, des initialisations utilisées et des voisinages utilisés. Ainsi, l'objectif est de mener une étude approfondie pour comprendre dans quelle mesure la règle proposée par Hansen et Mladenovic, 2006 [1] peut être généralisée ou non à différents problèmes NP-difficiles.

ABSTRACT

The Traveling Salesman Problem (TSP) is a computationally challenging combinatorial optimization problem. A comparative study of the local search strategies "First Improvement" and "Best Improvement" was conducted for the TSP in the article titled "First vs Best Improvement: an empirical study" by Hansen et Mladenovic, 2006 [1]. This study empirically compared these two strategies and recommended a rule for choosing the best option based on the quality of the initial solution.

The rule proposed by Hansen et Mladenovic, 2006 [1] is as follows: if the initial solution is chosen randomly, the First Improvement strategy (PA) on average yields a better final solution. However, if the initial solution is obtained using a greedy heuristic, the Best Improvement strategy (MA) on average produces better results. This study has become a reference in the research literature, with numerous citations, and is used as a basis for other NP-hard problems.

This thesis aims to verify the validity of the rule proposed by Hansen et Mladenovic, 2006 [1] for the TSP and determine if this rule can also be applied to other NP-hard problems. The focus will be on the TSP to verify Hansen et Mladenovic, 2006 [1]'s results. However, in order to evaluate the applicability of this rule to other NP-hard problems, we will also examine the Minimum Set Cover (MSSC) and the Weighted Maximum Satisfiability (MAXSAT-pondéré) problems. These problems will serve as examples to demonstrate that the conclusions of Hansen et Mladenovic, 2006 [1] may not necessarily be applicable to other NP-hard problems, depending on the datasets, initializations used, and neighborhoods considered. Thus, the objective is to conduct a comprehensive study to understand to what extent the rule proposed by Hansen et al. [1] can be generalized or not to different NP-hard problems.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

PA	Premier améliorant
MA	Meilleur améliorant
TSP	Problème du Voyageur de Commerce
MSSC	Problème de regroupements avec Minimisation de la Somme des Carrés

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

Les méthodes de descente locale jouent un rôle central en optimisation mathématique. Elles sont souvent utilisées pour trouver un minimum local (ou maximum) d'une fonction différentiable, bien qu'elles puissent également obtenir des optima globaux garantis pour certains problèmes d'optimisation (par exemple, les problèmes convexes). Pour la classe des problèmes d'optimisation combinatoire, les méthodes de descente locale sont communément appelées *recherches locales*, en raison de leur similitude avec les méthodes de recherche classiques.

Sans perte de généralité, définissons un problème d'optimisation combinatoire comme un problème de minimisation de la manière suivante. Soit $Z = 1, \ldots, z$ un ensemble fini et soit $c = (c_1, \ldots, c_z)$ un vecteur réel de taille z. Pour $F \subseteq Z$, nous définissons $c(F) = \sum_{j \in F} c_j$. Le problème d'optimisation combinatoire s'exprime alors comme $\min c(F) : F \in \mathcal{F}$, où $\mathcal{F}$ est une collection de sous-ensembles de Z.

En définissant chaque solution réalisable $F \in \mathcal{F}$ par son vecteur caractéristique $\chi^F \in 0, 1^Z$ tel que $\chi^F_j = 1$ si $j \in F$, et $\chi^F_j = 0$ sinon, le problème d'optimisation combinatoire peut être vu comme une minimisation sur un polytope, c'est-à-dire $\min\{c^T x | x \in conv\{\chi^F | F \in \mathcal{F}\}\}$. Ainsi, un minimum local x^* est tel que

$$c^T x^* \leq c^T x, \forall x \in \mathcal{N}(x^*), \tag{1.1}$$

où $\mathcal{N}(x)$ désigne le *voisinage* de x. Dans les problèmes d'optimisation combinatoire, les voisins d'une solution peuvent être obtenus de plusieurs manières, par exemple en complétant un ou deux des composants de la solution ou en échangeant leurs valeurs.

Dans ce cadre, une recherche locale peut être vue comme une recherche de parcours dans un graphe $G = (N, E)$, où chaque nœud de N correspond à une solution du problème, et deux nœuds sont connectés dans G s'ils sont voisins selon le voisinage $\mathcal{N}$ considéré. À partir d'un nœud initial $n \in N$, la recherche locale procède en examinant parmi les nœuds adjacents à n s'il existe un nœud de solution améliorante n'. Si tel

est le cas, n' remplace n en tant que nouveau nœud en cours dans la recherche (c'est-à-dire $n \leftarrow n'$). Cela est répété jusqu'à ce qu'aucun nœud améliorant ne soit trouvé parmi les voisins du nœud courant n.

La manière dont les nœuds sont examinés dans G à partir du nœud de recherche en cours n donne lieu aux deux principaux types de méthodes de recherche locale. Dans la recherche locale du *Meilleur améliorant* (*MA*), tous les voisins de n sont évalués et le meilleur est conservé pour la comparaison. Si ce nœud est meilleur, la recherche locale se poursuit à partir de celui-ci. Dans la recherche locale du *Premier améliorant* (*PA*), la recherche change de nœud en cours dès qu'un nœud meilleur est trouvé parmi les voisins de n, puis elle recommence la recherche à partir de ce nœud. La Figure 1.1 illustre les résultats possibles des recherches locales de première et de meilleure amélioration pour un problème d'optimisation combinatoire pour lequel une solution est représentée par 4 bits.

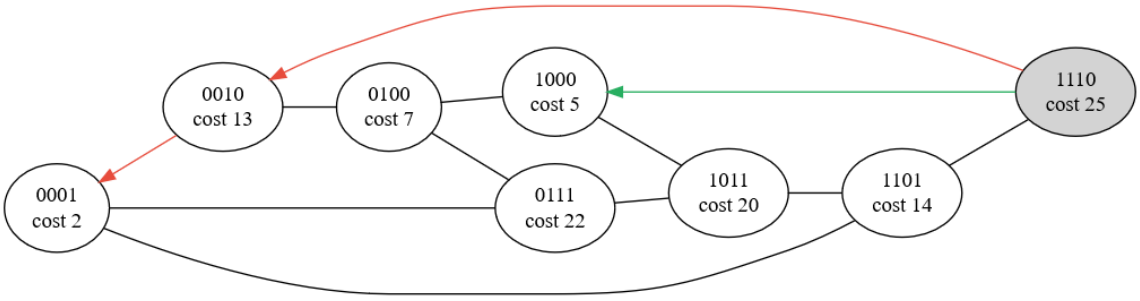


FIGURE 1.1 Exemple de recherche de parcours de graphe effectuée par la méthode de *Premier améliorant* (en vert) et *Meilleur améliorant* (en rouge)

Le graphe de voisinage est défini sur un voisinage de basculement de blocs : c'est-à-dire que, étant donné une solution $x = (b_1 b_2 b_3 b_4)$, ses voisins sont obtenus en inversant les valeurs de deux de ses bits consécutifs b_i, b_{i+1}. Les coûts de chaque solution sont également représentés dans la figure. Ils sont calculés de telle sorte que $b_1 = 1$ ait un coût de 5, $b_2 = 1$ ait un coût de 7, b_3 ait un coût de 13 et b_4 ait un coût de 2. Nous observons que, à partir de la solution initiale grise 1101, deux minima locaux différents peuvent être atteints. Le chemin vert correspond au chemin de solution

visité lors de la recherche locale de meilleure amélioration qui aboutit à la solution 1000 d'un coût égal à 5. Inversement, le chemin rouge représente un chemin possible suivi par une recherche locale de première amélioration qui atteint la solution locale minimale 0001 dont le coût est égal à 2.

1.1 Choisir une stratégie de recherche locale

Cependant, il n'existe pas de consensus dans la littérature qui définit la meilleure version de recherche locale. Les auteurs comparent souvent la version *Meilleur améliorant* et la version *Premier améliorant* d'une recherche locale à l'aide de mesures statistiques agrégées, telles que la moyenne et les écarts types des optima locaux. Ces mesures sont calculées sur plusieurs initialisations et instances de données d'un problème d'optimisation combinatoire particulier.

Cependant, l'étude empirique de Hansen et Mladenović publiée dans [1] compare les performances des recherches locales de première et de meilleure amélioration sur le voisinage 2-opt du *Problème du Voyageur de Commerce* (TSP). Des tests ont été réalisés sur des instances du TSP générées aléatoirement, à la fois euclidiennes et non-euclidiennes, ainsi que sur un sous-ensemble d'instances de TSPLIB. La principale conclusion de l'article est indiquée dans son résumé et citée ci-dessous :

> *Lorsqu'on applique l'heuristique 2-opt au problème du voyageur de commerce, choisir la meilleure amélioration à chaque itération donne en moyenne de moins bons résultats que choisir la première amélioration, si la solution initiale est choisie au hasard. Cependant, en commençant par des heuristiques constructives "gloutonnes" ou "plus proches voisins", la meilleure amélioration est meilleure et plus rapide en moyenne.*

L'article propose également des conclusions sur la comparaison des versions de recherche locale sur d'autres mesures que le coût de la solution (par exemple, le temps CPU, le nombre d'itérations de recherche locale, etc.). Dans notre revue de la littérature présentée dans la section suivante, on observe que cette affirmation a été

utilisée dans plusieurs autres articles, dont beaucoup traitent d'autres problèmes d'optimisation combinatoire NP-difficiles.

1.2 Objectifs de recherche

Dans cette étude, l'objectif est de revisiter l'étude de [1] afin d'examiner en détail les résultats qui ont soutenu cette affirmation. On émet également l'hypothèse que cette règle n'est pas applicable telle quelle pour tous problèmes NP-difficiles. Pour ces raisons on formule les deux objectifs de recherche suivant :

OR1 Vérifier pour les jeux de données étudiés par [1] si il y a une différence significative entre la stratégie du *Premier améliorant* et du *Meilleur améliorant* en appliquant une étude statistique approfondie

OR2 Vérifier si la règle de [1] est valide pour d'autres problèmes d'optimisation combinatoire en reproduisant l'étude statistique pour deux cas d'étude (*Problème de regroupements avec Minimisation de la Somme des Carrés* et *MAXSAT-pondéré*)

1.3 Plan du mémoire

Le présent mémoire est organisé de la façon suivante : le chapitre 2 de revue de littérature mesure l'impact et l'utilisation de la règle de [1] dans la littérature. Le chapitre 3 présente le protocole expérimental, qui comprend l'explication des initialisations utilisées, la description des jeux de données choisis, la justification de l'utilisation de la métrique de performance de différence en amélioration, ainsi que la présentation du protocole d'analyse statistique. Dans le chapitre 4, les résultats sont analysés pour identifier les problèmes, les jeux de données et les initialisations pour lesquels la règle de [1] est vérifiée. Finalement le chapitre 5 revient sur l'étude accomplie, ses perspectives, limitations et les futures travaux qui pourraient être réalisés.

CHAPITRE 2 REVUE DE LITTÉRATURE

Dans cette section, on analyse les articles qui citent Hansen et Mladenovic, 2006 [1] depuis sa publication, dans le but de comprendre son influence sur la littérature en optimisation discrète et combinatoire. Ensuite, on fait un retour sur les 2 stratégie de recherche locale, leur fonctionnement et on montre qu'il existe d'autres stratégies dans la littérature.

2.1 Impact de Hansen et Mladenovic, 2006 [1]

Parmi les 132 citations selon Google Scholar en mars 2023, 36 mentionnent directement la principale règle de [1]. Cependant, son utilisation est variée. On classifie ci-dessous ces citations en trois catégories :

Catégorie A : Manuscrits qui utilisent la règle de [1] pour choisir une version de recherche locale (premier ou meilleur amélioration).

Catégorie B : Manuscrits qui mentionnent la règle de [1], mais réalisent leurs propres expériences pour démontrer quelle stratégie de recherche locale est la plus efficace.

Catégorie C : Manuscrits qui mentionnent la règle de [1], mais ne l'utilisent pas comme contribution méthodologique.

On a pu trouver 13 manuscrits dans la catégorie A, 13 manuscrits dans la catégorie B et 10 manuscrits dans la catégorie C. Les 96 autres articles sont exclus de cette étude car ils citent [1] sans mentionner sa découverte principale. La séparation des catégories est résumée sur la figure 2.1. Ces citations sont généralement utilisées pour mentionner l'existence des deux types de recherche locale. Cependant, il est important de noter qu'ils renvoient également le lecteur à cet article, et par conséquent, à l'affirmation examinée de plus près ici.

Le tableau 2.1 détaille les domaines d'application influencés par le travail de [1]. Ces

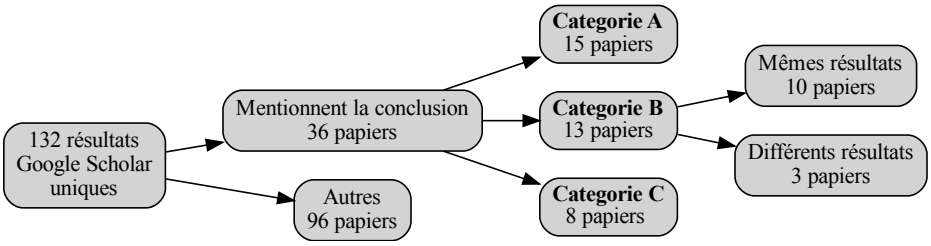


FIGURE 2.1 Nombre de références de [1]

articles traitent de problèmes d'optimisation combinatoire liés au TSP, mais aussi de divers problèmes NP-difficiles. Il est intéressant de noter qu'une grande partie des travaux du tableau 2.1 sont liés à des problèmes de routage similaires au TSP. En fait, comme le montre le tableau 2.2, un grand nombre d'articles des catégories A et B exploitent également l'espace de recherche locale en utilisant le voisinage 2-opt.

TABLEAU 2.1 Problèmes d'optimisation abordés pour chaque catégorie.

Catégories	Domaine d'application
Catégorie A	TSP et routage [2], [3] ; Localisation et distribution [4], [5], [6] ; Ordonnancement [7] ; Exploration de données [8], [9], [10] ; Problèmes de bin packing [11], [12] ; Optimisation de portefeuille [13] ; Systèmes de file d'attente [14] ;
Catégorie B	TSP et routage [15], [16], [17], [18], [19], [20] ; Localisation [21] ; Exploration de données [22], [23] ; Problèmes de programmation quadratique binaire [24] ; Théorie des graphes [25] ; Gestion de projets [26] ; Ordonnancement [27].
Catégorie C	TSP et routage [28], [29], [30] ; Localisation [31] ; Emballage [32] ; Apprentissage par renforcement [33] ; Ordonnancement [34], [35] ;

Alors que les articles de la catégorie A sont effectivement directement influencés par la découverte de [1], les articles de la catégorie B réalisent plutôt leurs propres expériences pour vérifier sa validité. Comme le souligne [15], il n'est pas certain que

Tableau 2.2 Articles qui mentionnent l'utilisation du voisinage 2-opt dans les recherches locales.

Catégorie	Articles qui utilisent le voisinage 2-opt
Catégorie A	[5], [6], [4], [2], [3]
Catégorie B	[18], [17], [16], [15], [19], [20]

les résultats de [1] soient toujours valables pour d'autres problèmes d'optimisation ou algorithmes.

Cela dit, 13 articles de la catégorie B réalisent des expériences informatiques pour évaluer quelle version de recherche locale (*Premier améliorant* ou *Meilleur améliorant*) est meilleure pour leurs problèmes d'optimisation cibles. Parmi eux, 9 articles confirment la découverte de [1], tandis que 4 autres la réfutent. La proportion d'articles confirmant [1] est presque la même si l'on ne prend en compte que les articles de la catégorie B qui explorent le voisinage 2-opt. 4 sur 6 articles confirment la découverte de [1] tandis que 2 articles ne le font pas. Il est important de souligner que la proportion d'œuvres réfutant le résultat de [1] sans le citer est probablement plus élevée. D'une part, les auteurs pourraient être tentés de ne pas citer directement [1] si leurs expériences ne sont pas en accord avec lui. D'autre part, cette même découverte pourrait également biaiser les travaux expérimentaux en faveur de sa confirmation afin de légitimer les résultats obtenus.

Les mesures moyennes sont souvent utilisées par les articles de la catégorie B comme la principale mesure statistique pour comparer les recherches locales [17], [15], [27], [19], [16], [23], [24], [20]. Cependant cette mesure est influencée par des valeurs extrêmes et ne donne pas une idée de la variation de la distribution de la métrique. C'est pourquoi d'autres travaux utilisent également des écarts-types [15,20,23,24,27]. Les meilleures et pires valeurs objectives sont également utilisées par [23], [24]. De plus, la distance par rapport aux solutions optimales peut également être prise en compte lorsque cela est disponible [26], [22], [25], [21], [20]. Cependant, dans le cas de l'optimisation multi-objectifs, d'autres mesures peuvent être utilisées, telles que l'hypervolume des fronts de Pareto [36]. En ce qui concerne les temps CPU, [16],

[18], [20] comparent les versions de recherche locale en termes de temps CPU pour atteindre les solutions cibles. En outre, on peut noter que parmi tous les papiers de la catégorie B, 8 calculent agrègent leur métrique sur 100 exécutions ou moins ([23], [27], [24], [25], [22], [15], [20]).

Enfin, bien que les articles de la catégorie C ne soient pas directement impactés par [1], ils propagent sa conclusion, influençant potentiellement d'autres auteurs sur le choix de la version de recherche locale dans leurs méthodes. En fait, plus de 2000 articles citent les manuscrits des catégories A, B et C selon Google Scholar en mars 2023.

L'étude précédente a révélé que de nombreux articles utilisent la conclusion de [1]. Cependant, cette étude a soulevé plusieurs limites dans l'étude de ces deux stratégies. Tout d'abord, il est légitime de remettre en question l'application de cette règle à d'autres problèmes NP-difficiles. Une comparaison empirique est donc nécessaire. Certains articles ont effectué cette comparaison, mais la plupart se contentent de calculer des métriques moyennes. Cependant, la moyenne est influencée par les valeurs extrêmes, ce qui rend difficile de déterminer si les différences observées sont significatives ou simplement dues à des fluctuations statistiques. Parfois, seules des informations sur la dispersion des résultats sont fournies, ce qui donne une indication de la certitude des résultats. De plus, le nombre d'exécutions utilisées pour calculer les métriques est souvent limité à moins de 100 échantillons.

Ces observations soulignent la nécessité de mener une étude plus approfondie de ces deux stratégies. Il est essentiel de les comparer non seulement pour le TSP, mais aussi pour d'autres problèmes NP-difficiles. De plus, pour assurer la comparabilité, il est nécessaire d'utiliser une métrique moyenne appropriée, même si elle est insuffisante en elle-même. L'écart-type peut fournir une indication de la dispersion des données, mais il peut être difficile de déterminer si une valeur d'écart-type est considérée élevée ou faible pour un problème de taille fixe. Tari, Sara et Ochoa, 2021 [37] ont récemment proposé une approche qui ne mentionne pas la conclusion de [1], mais qui teste les deux stratégies. Ils utilisent le test de Mann-Whitney pour déterminer si, pour une stratégie donnée, le fait de modifier le nombre de perturbations de leur algorithme

entraîne un changement significatif de leur métrique. Les auteurs utilisent la valeur p, valeur qui si elle est inférieure à un seuil ($p < 0.05$) indique qu'un changement significatif est présent. Cette mesure est calculée à partir d'une métrique comparée sur un grand nombre d'échantillons. On s'inspirera de cette méthode pour fournir une valeur p capable d'indiquer s'il existe une différence significative entre les deux stratégies. Cependant, contrairement à [37], on souhaite comparer les performances des stratégies pour une même instance d'un problème. Par conséquent, on utilisera des tests pairés pour calculer chaque valeur p. De plus, après une recherche bibliographique approfondie, il apparaît que fournir la magnitude d'effet peut donner une estimation de l'importance avec laquelle une stratégie est meilleure qu'une autre.

2.2 Analyse des stratégies de recherche locale

La figure 2.2 explicite les stratégies de recherche locale de *Meilleur améliorant* et de *Premier améliorant* sous forme algorithmique. Ces deux stratégies démarrent d'une solution initiale fourni par un algorithme puis cherchent itérativement une amélioration en explorant des modifications possibles. On note en bleu la différence entre les deux stratégies : dans le cas de la stratégie de premier amélioration, la modification est "sauvegardée" mais directement appliquée alors que pour la stratégie de meilleur améliorant, la meilleure modification est sauvegardée et appliquée uniquement après avoir exploré toutes les modifications possibles. On répète la procédure jusqu'à ce qu'aucune modification ne soit possible et on retourne la solution trouvée.

Outre la meilleure et la première amélioration, il existe d'autres variantes de la recherche locale dans la littérature. Elles visent à (i) atteindre de meilleurs optima locaux, et/ou (ii) utiliser le moins de temps de calcul possible. [38] formalisent les stratégies de recherche locale en trois catégories différentes. Dans $S_i^{=}$, la recherche se déplace vers le i-ième meilleur voisin améliorant du nœud courant n. Dans la stratégie $S_i^{\leq}$, la recherche est déplacée aléatoirement vers un nœud améliorant parmi les i meilleurs voisins de n. Enfin, dans la catégorie $S_i^{\geq}$, l'un des nœuds améliorants classés après le i-ième nœud est sélectionné au hasard. La meilleure et la première amélioration sont des cas particuliers des catégories ci-dessus, où la recherche de

Algorithm 1 Meilleur améliorant

Require: Les hyperparamètres du problème
Ensure: La solution finale améliorée
1: Générer la solution initiale
2: *amélioration* ← *Vrai*
3: **while** *amélioration* **do**
4: *amélioration* ← *Faux*
5: **for** Toutes les modifications possibles **do**
6: Tester temporairement la modification
7: **if** améliore la meilleure solution **then**
8: Sauvegarder la modification
9: Stopper la boucle d'exploration
10: **end if**
11: **end for**
12: Appliquer la meilleure possibilité de modification
13: **end while**
14: Retourner la solution finale

Algorithm 2 Premier améliorant

Require: Les hyperparamètres du problème
Ensure: La solution finale améliorée
1: Générer la solution initiale
2: *amélioration* ← *Vrai*
3: **while** *amélioration* **do**
4: *amélioration* ← *Faux*
5: **for** Toutes les modifications possibles **do**
6: Tester temporairement la modification
7: **if** améliore la meilleure solution **then**
8: Sauvegarder la modification
9: **end if**
10: **end for**
11: Appliquer la meilleure possibilité de modification
12: **end while**
13: Retourner la solution finale

FIGURE 2.2 Deux stratégie d'amélioration pour la recherche locale

la meilleure amélioration correspond à $S_1^=$, et la recherche de la première amélioration est équivalente à $S_{2^z}^{\leq}$. Pour éviter les optima locaux trop prématurés, [20] propose la stratégie d'amélioration retardée qui utilise des vérificateurs pour guider la recherche locale vers l'amélioration des solutions avec un nombre minimal de conditions d'optimalité locale satisfaites. De cette manière, la recherche locale peut s'exécuter pendant un plus grand nombre d'itérations, en cherchant dans de plus grandes parties de l'espace de solution. Dans l'amélioration d-best de [19], le meilleur voisin améliorant est sélectionné dès que d voisins améliorants sont trouvés. Une condition plus stricte peut être appliquée pour arrêter l'exploration du voisinage. Dans la stratégie k-sequential improvement de [39], la recherche locale est arrêtée après que k voisins strictement améliorants de la solution en place x ont été trouvés en séquence, c'est-à-dire, les k solutions voisines $x_1, x_2, \ldots, x_k$ sont telles que $c(x) \geq c(x_1) \geq c(x_2) \geq \ldots \geq c(x_k)$. A ce stade, la recherche locale commence à partir de x_k. Le hasard peut également être introduit pour créer de nouvelles stratégies. Par exemple, [1] ont étudié l'impact de l'ordre d'exploration du voisinage dans le cadre de la recherche locale en première amélioration. Enfin, le choix du voisin suivant est modélisé comme un problème d'apprentissage par renforcement dans [40], [41], [42].

En somme, de nombreuses stratégies de recherche locale ont été devéloppées et mettent en évidence que les deux stratégies étudiées dans Hansen et Mladenovic, 2006 [1] ne sont qu'une des première approches pour obtenir un algorithme performant.

CHAPITRE 3 COMPARAISON STATISTIQUE DE RECHERCHES LOCALES : STRATÉGIES DE PREMIÈRE ET MEILLEURE AMÉLIORATION

Après analyse de la littérature utilisant la règle de [1], on met en oeuvre une série d'analyses statistiques approfondies pour vérifier si cette règle concernant les stratégies de *Premier améliorant* et *Meilleur améliorant* est valide. Ensuite, on montre que cette règle ne peut pas être extrapolée à d'autres problèmes NP-difficile en appliquant la même analyse statistique au *Problème de regroupements avec Minimisation de la Somme des Carrés* (MSSC) et au *MAXSAT-pondéré*

Dans cette section, on décrit le dispositif expérimental pour ces expériences. Cela inclut : (i) une description concise des problèmes d'optimisation combinatoire TSP, MSSC et MAX-SAT pondéré et des recherches locales évaluées pour chacun des problèmes approchés ; (ii) les procédures d'initialisation ; (iii) les jeux de données utilisés dans notre étude, (iv) la métrique utilisée pour comparer les versions de recherche locale ; et (v) la description des tests statistiques effectués.

3.0.1 Problèmes d'optimisation combinatoire et recherches locales évaluées

TSP

Le *Problème du Voyageur de Commerce* est un problème NP-difficile utilisé dans de nombreux domaines comme la logistique, la conception de circuits électroniques ou la biologie [43]. Étant donné un ensemble de villes, le TSP consiste à trouver le circuit le plus court passant par chaque ville exactement une fois. Formellement, le TSP est défini sur un graphe $G = (V, E)$ où V correspond à l'ensemble des villes et E à l'ensemble des arêtes les reliant. Une fonction $c : E \to \mathbb{R}$ attribue un coût non négatif à chaque arête dans E. Ainsi, le coût d'une solution TSP réalisable est simplement calculé en additionnant le coût de chacune de ses arêtes.

Comme dans [1], l'évaluation des versions de recherche locale pour le TSP se fait par rapport au voisinage 2-opt, qui comprend toutes les solutions voisines qui peuvent être obtenues à partir d'une solution TSP donnée en échangeant une paire de ses arêtes. La figure 3.1 illustre une solution TSP et l'un de ses voisins 2-opt. Une recherche locale 2-opt recherche des améliorations parmi toutes les paires d'arêtes possibles qui peuvent être échangées dans une solution TSP.

MSSC

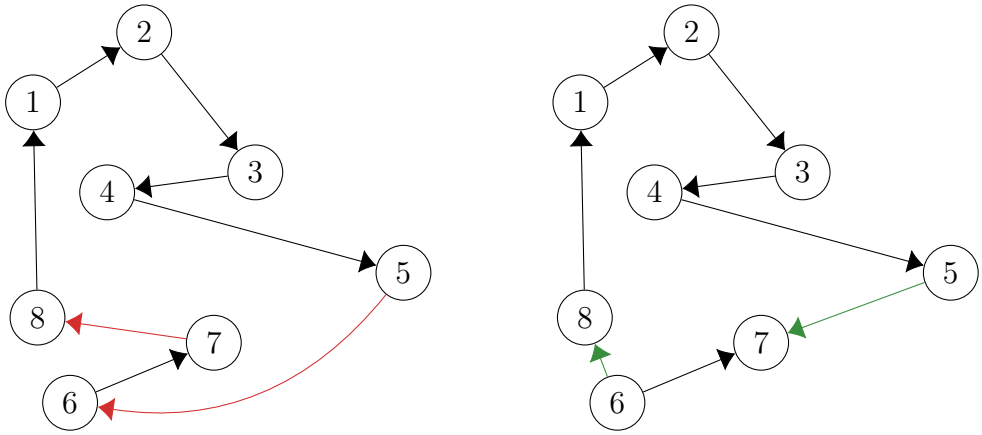
Le *Problème de regroupements avec Minimisation de la Somme des Carrés* (MSSC) est un problème largement étudié qui vise à regrouper un ensemble de données en fonction de leur similarité. Il permet d'extraire des groupes de données similaires, de détecter des anomalies et de réaliser une classification non-supervisée. Les applications du clustering sont nombreuses, notamment dans le domaine de la compression de données et de la classification lorsque peu ou pas d'exemples étiquetés sont disponibles. Ce problème est particulièrement adapté pour le traitement de données massives où il permet de découvrir des structures et des groupes pertinents au sein des données.

Le *Problème de regroupements avec Minimisation de la Somme des Carrés* consiste à diviser n points de données en un nombre donné k de sous-ensembles, appelés clusters. Dans le MSSC, l'objectif est de trouver k clusters qui minimisent la somme des distances euclidiennes quadratiques entre chaque point de données et son centroïde de cluster, ce qui peut être formulé comme suit :

$$\min_{x} \sum_{i=1}^{n} \sum_{j=1}^{k} x_{ij} \left\| p_i - \frac{\sum_{\ell=1}^{n} x_{i\ell} p_i}{\sum_{\ell=1}^{n} x_{i\ell} p_i} \right\|_2^2,$$

où x_{ij} est égal 1 à si le point de données p_i est assigné à la cluster j, et à 0 dans le cas contraire.

L'analyse de première et de meilleure amélioration pour le MSSC de cette étude utilise le voisinage exploité par l'heuristique H-means [8]. À partir d'une solution



(a) Solution de TSP (b) Voisin 2-opt

FIGURE 3.1 Illustration d'un voisinage 2-opt du TSP

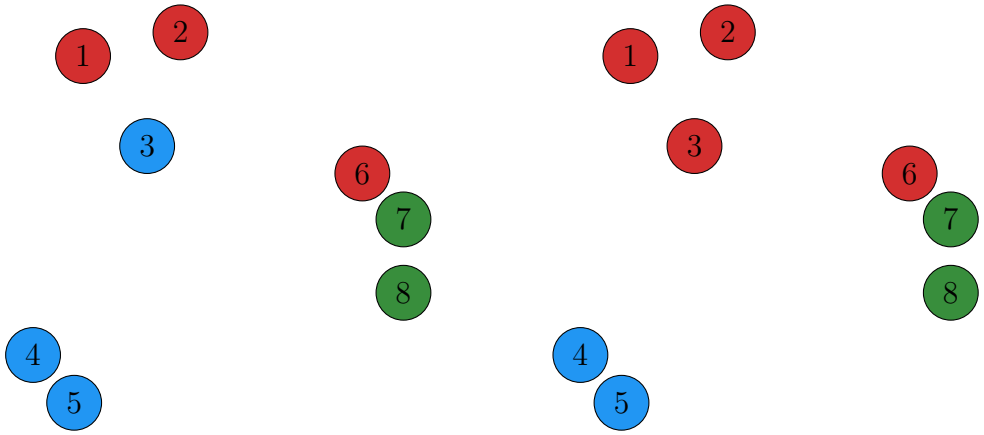
donnée, l'algorithme évalue chaque point p_i et identifie le cluster j qui améliorerait la solution de clustering en déplaçant p_i de son cluster actuel vers j. La figure 3.2 illustre un exemple de solution améliorée dans ce voisinage.

MAXSAT-pondéré

Etant donné un ensemble $V = \{v_1, ..., v_n\}$ de variables booléennes et $C = \{c_1, c_2, \ldots, c_m\}$ de clauses, où chaque clause est une disjonction de littéraux, et un poids w_i est associé à chaque clause, le problème MAXSAT-pondéré consiste à trouver une affectation de valeurs de vérité aux variables dans V de telle sorte que la somme des poids des clauses satisfaites soit maximisée.

Ce problème a des nombreuses applications [44]. Cela inclut des tâches de planification, vérification, sécurité, machine learning et bioinformatique. La deuxième édition du "Handbook of Satisfiability" ([44], [45]) répertorie les approches principales pour résoudre le problème du MAXSAT-pondéré.

Les recherches locales pour le problème MAXSAT-pondéré sont évaluées en fonction du voisinage d'une solution dans laquelle la valeur de vérité d'une seule variable est



(a) Solution de MSSC (b) Voisin H-Means

FIGURE 3.2 Illustration d'une solution de voisins H-Means de MSSC. Chaque couleur de la figure représente un groupe différent.

complémentée, c'est-à-dire de vrai à faux ou de faux à vrai. La figure 3.3 montre une solution voisine possible dans ce voisinage.

3.0.2 Initialisations

La conclusion de [1] indique qu'une version de recherche locale est plus performante que l'autre en fonction de la manière dont elles sont initialisées. Ainsi, les expériences ont été menées en commençant la recherche par des solutions aléatoires et gourmandes, afin d'évaluer si cette décision a réellement un impact sur les performances des méthodes de recherche locale évaluées. Plus précisément, quatre procédures d'initialisation différentes ont été utilisées :

- *aléatoire* : choix aléatoire de la solution initiale
- *gloutonne* : choix itératif glouton n'impliquant pas d'aléatoire
- *top-k* : choix itératif glouton parmi les K meilleures possibilités
- *gloutonne-randomisée* : choix itératif glouton parmi tous les voisins avec des probabilités pondérées en fonction de l'optimalité locale du choix

$$\underset{w_1 = 2}{(x_1 \vee x_2)} \quad \wedge \quad \underset{w_2 = 3}{(x_1 \vee \neg x_2)} \quad \wedge \quad \underset{w_3 = 5}{(x_2 \vee \neg x_3)} \quad \wedge \quad \underset{w_4 = 2}{(\neg x_1 \vee \neg x_2)}$$

(a) Instance MAXSAT-pondéré

$$\underset{\sum_i w_i \times s_i = 5}{\{x_1 = false, x_2 = false, x_3 = true\}}$$

(b) Solution de MAXSAT-pondéré

$$\underset{\sum_i w_i \times s_i = 7}{\{x_1 = true, x_2 = false, x_3 = true\}}$$

(c) Solution voisine obtenue en prenant le complément de x_1

FIGURE 3.3 Illustration d'une solution voisine de MAXSAT-pondéré

TSP

On décrit dans cette partie les quatre procédures d'initialisation pour le TSP, qui sont utilisées par [1], ainsi que dans le cadre des expériences. L'initialisation *aléatoire* mélange les villes n_{TSP} au hasard et produit un tour obtenu en reliant les villes dans cet ordre. La méthode *gloutonne* commence par une ville sélectionnée aléatoirement x et ajoute itérativement la ville non visitée la plus proche x' à x, en mettant à jour $x \leftarrow x'$ jusqu'à ce que toutes les villes soient visitées. Enfin, le tour s'achève par un retour à la ville initiale. Pour l'initialisation *top-k*, la procédure gourmande sélectionne la ville suivante au hasard parmi les K villes non visitées les plus proches de x. L'initialisation *gloutonne-randomisée* utilise une distribution de probabilité pondérée pour sélectionner la ville suivante, où une ville non visitée x' est choisie avec une probabilité inversement proportionnelle à sa distance par rapport à x. Les algorithmes précis de chaque initialisations sont fournis en annexe C

MSSC

En ce qui concerne MSSC, les solutions initiales *aléatoires* sont obtenues en assignant chaque point de données à un groupe au hasard, tandis que les solutions initiales *gloutonne*, *top-K* et *gloutonne-randomisée* sont obtenues par des variations de l'heuristique k-means++ de [46] présentée dans 10. Tous les algorithmes sont présentés

en annexe D.

MAXSAT-pondéré

Pour *MAXSAT-pondéré*, l'initialisation *aléatoire* consiste à assigner une valeur de vérité aléatoire à chaque variable. Cependant, pour les initialisations de type gloutonne, on transforme en stratégie gloutonne l'algorithme "$\frac{3}{4}$-approximative" de [47]. Le détail de cette adaptation et les algorithmes précis sont présentés en annexe E.

3.0.3 Jeux de données

Les expériences ont été menées sur deux types d'instances de données : des instances générées de manière aléatoire et des instances de référence collectées à partir de bases de données benchmark.

Pour générer des instances de données pour le *Problème du Voyageur de Commerce* (TSP), une distribution uniforme pour l'emplacement des villes a été utilisée. Conformément à la méthodologie décrite dans [1], les villes sont uniformément sélectionnées dans un carré de 100×100. Cela a permis de s'assurer que le processus de génération des données reste cohérent avec l'approche établie dans l'étude citée en référence. Les instances TSP couvrent une gamme de nombres de villes $n_{TSP} \in \{20, 30, ..., 150\}$ $\cup\{200, 250, ..., 500\}\cup\{500, 600, ..., 1000\}$. Pour chaque valeur de n_{TSP}, 1000 instances sont générées.

En ce qui concerne le *Problème de regroupements avec Minimisation de la Somme des Carrés* (MSSC), des instances générées de manière aléatoire sont créées par échantillonnage de données à partir d'un carré de 100 × 100. Les instances sont créées comme suit. Pour chaque nombre de points $n_{MSSC} \in \{20, 30, ..., 150\}\cup\{200, 250, ..., 500\}$ $\cup\{500, 600, ..., 1000\}$, des instances avec différents nombres de clusters k dans $\{2^1, 2^2,$ $\ldots, 2^8\}$ sont générées. Ensuite, pour chaque combinaison de n_{MSSC} et k, on génère 2000 instances de données : 1000 par échantillonnage uniforme, et 1000 par échantillonnage à partir de distributions gaussiennes afin de mieux représenter des structures de clusters Dans ce dernier cas, on définit les paramètres des distributions

gaussiennes, $\mathcal{N}(\mu, \sigma^2)$, où μ est échantillonné à partir du carré 100×100. L'écart-type σ est échantillonné uniformément dans l'intervalle $[\frac{100}{10}, \frac{100}{2}]$, qui est choisi pour fournir une gamme variée de distributions gaussiennes. Enfin, on détermine le nombre de clusters souhaité et on distribue les points de données aussi uniformément que possible, en garantissant une différence maximale d'un point entre chaque paire de clusters.

Les instances de référence pour le TSP ont été obtenues à partir de la TSPLIB. Toutes les instances de la TSPLIB pour lesquelles $n_{TSP} \leq 1000$ ont été utilisées. En ce qui concerne le MSSC, on a utilisé 12 instances de référence provenant du référentiel UCI Machine Learning [48]. Elles permettent d'évaluer l'influence du nombre de clusters, du chevauchement des clusters, du nombre de dimensions, etc. sur les regroupements trouvés. Enfin, des instances de référence MAXSAT pondérées sont collectées auprès de [49], comprenant 29 instances de données pour lesquelles le nombre de clauses m_{MAXSAT} et le nombre de variables n_{MAXSAT} se situent dans l'intervalle $[1000, 5000[$. Les détails concernant les instances de référence utilisées dans les expériences se trouvent dans l'annexe A.

3.1 Métrique de performance

Pour vérifier la conclusion de [1], on doit déterminer s'il existe une différence de performance entre les versions de première et de meilleure amélioration d'une procédure de recherche locale pour un problème d'optimisation combinatoire particulier, en fonction de la manière dont elles sont initialisées.

La qualité d'une solution est fournie dans la définition de chaque problème étudié : pour le TSP la longueur du tour, pour MSSC la somme des distances euclidiennes aux carrés des points à leur centroide et pour MAXSAT-pondéré la somme des poids des clauses satisfaites. Ces métriques ne sont pas adaptées si on souhaite comparer des résultats pour deux instances d'un problème donné mais de tailles différentes (par exemple une instance TSP avec 10 villes et une autre avec 1000 villes auront des longueurs de tours très différentes). Cependant, on souhaite déterminer l'amélioration

d'une stratégie sur l'autre sur une grande variétés de tailles de problèmes.

Plusieurs métriques de performances sont possibles : Hansen et Mladenovic, 2006 [1] utilise l'amélioration comme métrique de performance : $amelioration = \frac{cout(MA)-cout(PA)}{cout(PA)}$ avec $cout(MA)$ le coût final obtenu par la stratégie de *Meilleur améliorant* et $cout(PA)$ le coût final obtenu par la stratégie *Premier améliorant*. Cette métrique mesure le pourcentage de réduction du coût entre PA et MA par rapport au coût atteint par PA. Elle permet de comparer deux stratégies entre elles pour des instances de coût spécifique. On note qu'un même écart de $cout(PA) - cout(MA)$ donnera une amélioration beaucoup plus importante pour un jeux de données simple que pour un jeux de données de coût plus faible. Pour un écart absolu fixe entre PA et MA mais des valeurs absolues décalées d'un écart Δ la mesure d'amélioration peut varier grandement. Comparons deux cas : Dans le premier cas PA et MA atteignent les coûts finaux respectifs $cout_1(PA)$ et $cout_1(MA)$. Dans le deuxième cas PA et MA sont séparés d'un même écart $cout_1(MA) - cout_1(PA)$ mais les coût absolus atteint par PA et MA sont décalés de Δ : $cout_2(PA) = cout_1(PA) + \Delta$ et $cout_2(MA) = cout_1(MA) + \Delta$. On peut exprimer alors l'*amelioration* pour les coût 1 et 2 :

$$\begin{aligned} amelioration_1 &= \frac{cout_1(MA) - cout_1(PA)}{cout_1(PA)} \\ amelioration_2 &= \frac{(cout_1(MA) + \Delta) - (cout_1(PA) + \Delta)}{cout_1(PA) + \Delta} \\ &= \frac{cout_1(MA) - cout_1(PA)}{cout_1(PA) + \Delta} \\ &= amelioration_1 \times \frac{1}{1 + \frac{\Delta}{cout_1(PA)}} \end{aligned} \tag{3.1}$$

La figure 3.4 montre l'évolution du facteur multiplicatif en fonction de Δ et de $cout_1(PA)$, qui est le coût de référence. À $cout(PA)$ constant, la valeur d'amélioration sera d'autant plus grande que Δ tend vers $-cout(PA)$. Cela sera plus visible pour les instances de plus faible $cout(PA)$. Soit une instance avec $cout_1(PA) = 10$, $cout_1(MA) = 9$ et une instance où $cout_2(PA) = 10$, $cout_1(MA) = 95$. On a alors

$amelioration_1 = \frac{100-90}{100} = -0.1$ et $amelioration_2 = \frac{105-95}{105} \approx 0.095$. La différence absolue entre les deux algorithmes est inchangée mais la métrique n'est pas la même.

On propose une autre métrique : la différence en amélioration :

$$diffAmelioration = \frac{cout(MA) - cout(PA)}{coutInit} \tag{3.2}$$

où $coutInit$ correspond au coût de la solution de départ de la recherche locale Cette métrique s'interprète comme la différence en amélioration des algorithmes relativement au coût initial : $\frac{cout(MA)}{coutInit}$ représente l'amélioration apportée par MA relativement au coût initial et $\frac{cout(PA)}{coutInit}$ réalise de même pour PA. Contrairement à précédemment, on a :

$$diffAmelioration_2 = \frac{(cout_1(MA) + \Delta) - (cout_1(PA) + \Delta)}{coutInitial} = diffAmelioration_1.$$

Par conséquent, cette métrique donnera le même résultat pour un même écart en valeur absolue quelle que soit la valeur de $cout_1(PA)$.

On utilisera la métrique de différence en amélioration ici : Celle-ci prend en compte la complexité de l'instance avec $coutInit$ pour indiquer quel algorithme permet d'améliorer le plus la qualité de la solution initiale. Elle permet également des comparaisons plus neutres entre instances de différentes complexités

3.2 Analyse statistique

Dans cette étude, l'objectif est de déterminer s'il existe une différence significative entre les algorithmes MA et PA pour les problèmes d'optimisation combinatoire, ainsi que l'ampleur de cette différence. Pour atteindre cet objectif, des tests statistiques approfondis sont utilisés. Tout d'abord, on explique pourquoi deux tests statistiques différents sont nécessaires. Ensuite, on explicite l'utilisation de chacun de ces tests dans cette étude

Les résultats de [1] suggèrent qu'en cas d'initialisation aléatoire, PA produit de meilleurs optima locaux que MA, tandis que pour une initialisation gourmande, MA

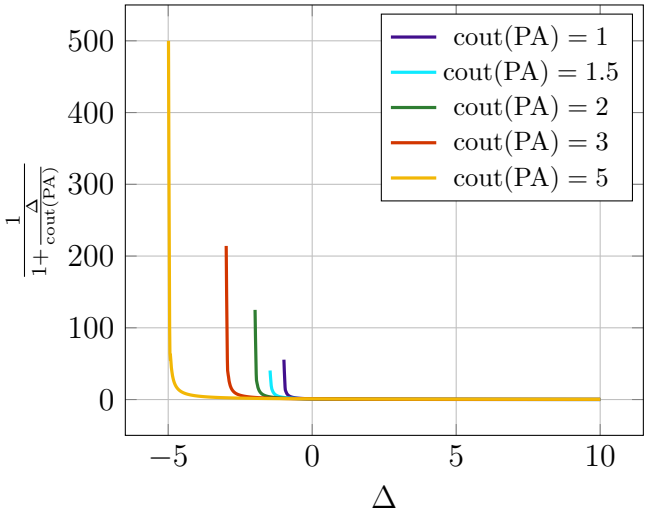


FIGURE 3.4 Evolution du facteur multiplicatif entre deux instances tel que $cout_2(PA) = cout_1(PA) + \Delta$ et $cout_2(MA) = cout_1(MA) + \Delta$

génère des solutions finales supérieures. Afin d'évaluer la présence d'une différence significative entre les optima locaux obtenus par PA et MA, des tests statistiques sont utilisés en utilisant la métrique $diffAmelioration$. Pour chaque type d'instance, la métrique est calculée sur 1000 échantillons non corrélés, cependant, la méthode d'échantillonnage varie en fonction du type d'instance.

Pour les instances de référence, on obtient 1000 échantillons de la métrique $diffAmelioration$ en utilisant des solutions initiales distinctes obtenues par les méthodes *top-K*, *gloutonne-randomisée* ou *aléatoire*, avec des graines différentes à chaque fois. Ainsi, on dispose de 1000 échantillons uniques pour chaque instance de référence.

Dans le cas des instances de référence, chaque échantillon $diffAmelioration$ est calculé en utilisant une solution initiale distincte, obtenue par *top-K*, *gloutonne-randomisée* ou *aléatoire*, en utilisant à chaque fois une graine différente. Ce processus permet d'obtenir un total de 1000 échantillons de données, avec 1000 solutions initiales uniques utilisées pour chaque instance de référence.

Initialement, on a utilisé un t-Test traditionnel pour étudier l'existence d'une dif-

férence significative entre MA et PA, en supposant une hypothèse nulle selon laquelle la différence moyenne d'amélioration est nulle (H0 : $\mu = 0$). Pour évaluer la forme de la distribution, on a utilisé un QQPlot avec des intervalles de confiance à 95%, ce qui a permis d'évaluer le degré de ressemblance entre la distribution observée et une distribution gaussienne. Cependant, on a constaté que la distribution de $diffAmelioration$ ne suit pas systématiquement une distribution normale, ce qui viole les hypothèses du t-Test. Pour résoudre ce problème, on a utilisé le test de Wilcoxon signé, qui est un test non paramétrique adapté aux cas où la distribution de $diffAmelioration$ s'éloigne de la normalité. Ce test examine si la médiane de $diffAmelioration$ est nulle (H0 : $\mu = 0$), sans faire aucune hypothèse sur la forme de la distribution.

Pour chaque test, l'objectif était de calculer la statistique de test pour obtenir une valeur p et la magnitude d'effet. Une valeur p inférieure à 0,05 indique qu'il n'y a pas de différence significative entre les deux algorithmes, tandis qu'une valeur p plus élevée ne fournit pas de preuve concluante. La magnitude d'effet permet d'estimer l'ampleur de la différence entre les deux recherches locales évaluées de manière standardisée, indépendamment de la taille de l'échantillon [50]. Le t-Test utilise le coefficient de Cohen d comme magnitude d'effet [51], tandis que le test de Wilcoxon signé utilise la corrélation rang-bisériale des paires appariées [52]. Cependant, pour le test de Wilcoxon signé, on utilise la stratégie de gestion des zéros de Wilcoxon [53] qui supprime les valeurs nulles du calcul. L'approximation normale est utilisée car on dispose d'un grand nombre d'échantillons.

La figure 3.5 présente la procédure détaillée pour obtenir la valeur p et la magnitude d'effet. Tout d'abord, on identifie le type de distribution suivi par les différences en amélioration à l'aide d'un QQPlot (étape 2). Si la distribution s'éloigne de l'intervalle gaussien, en particulier pour la partie centrale de la distribution, on applique le test de Wilcoxon (B1). Sinon, si les hypothèses du t-Test sont vérifiées, on utilise le t-Test (A1). On calcule ensuite les statistiques pour chaque test. Pour le test de Wilcoxon, on utilise l'approximation normale étant donné que on a 1000 échantillons (B4). On peut ensuite calculer la valeur p pour chaque test. En parallèle, on peut également

calculer la magnitude d'effet à partir de la statistique de chaque test (A2 et B3 pour chaque test). On caractérise ensuite la magnitude d'effet. La littérature ne fournit pas de caractérisation précise des magnitude d'effet pour des valeurs spécifiques telles que “small” pour 0.1, “medium” pour 0.3 et “big” pour 0.5. Cependant, on souhaite classifier les magnitude d'effet en utilisant des intervalles. Par exemple, pour le t-Test, on choisit de nommer les catégories “small-medium” lorsque la magnitude d'effet est comprise entre 0.1 et 0.3.

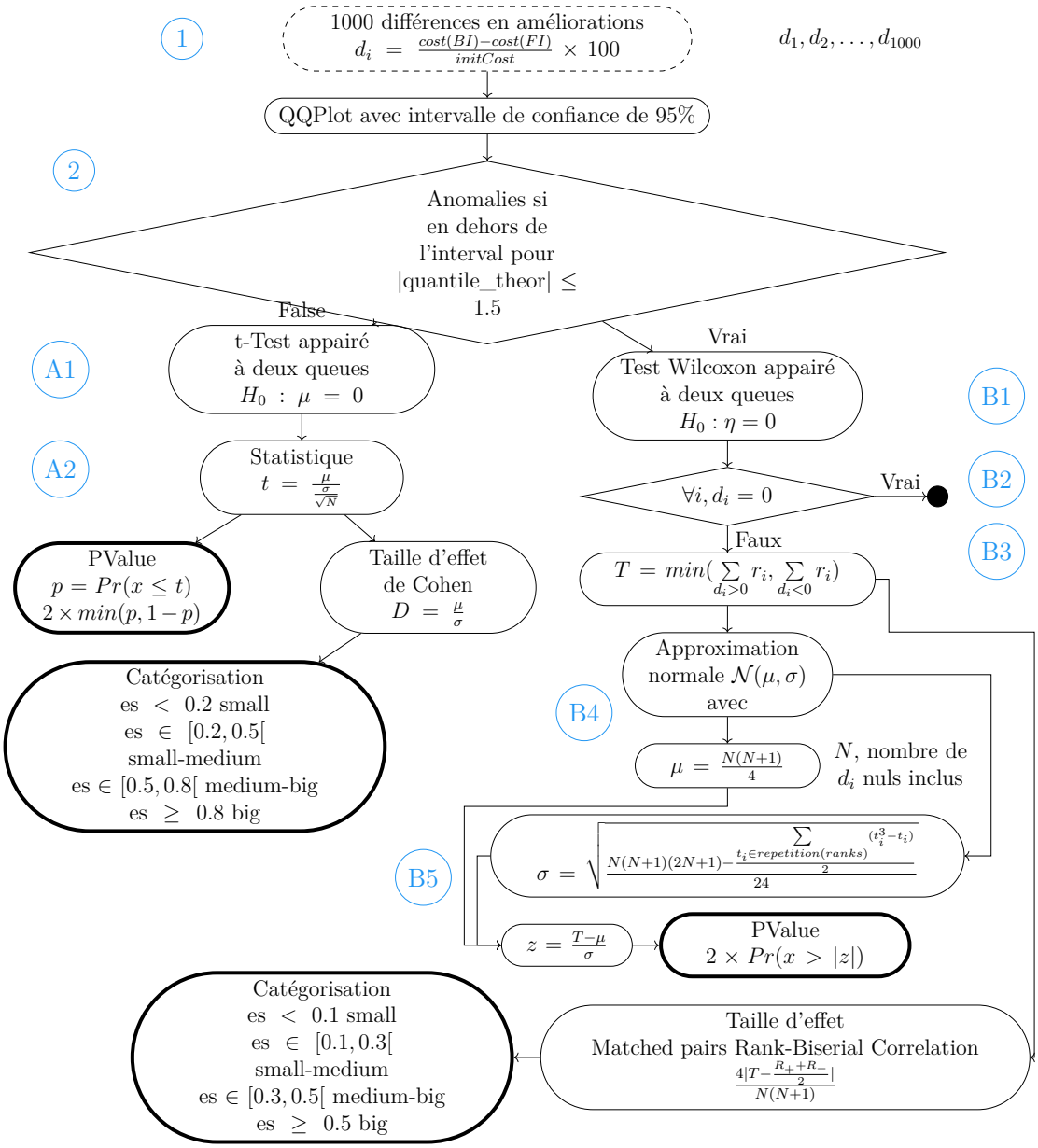


FIGURE 3.5 Procédure statistique utilisée

CHAPITRE 4 ANALYSE DES RÉSULTATS

Dans cette section, on présente les résultats de nos tests statistiques avec les recherches locales de type *Premier améliorant* et *Meilleur améliorant* pour les trois problèmes d'optimisation combinatoire.

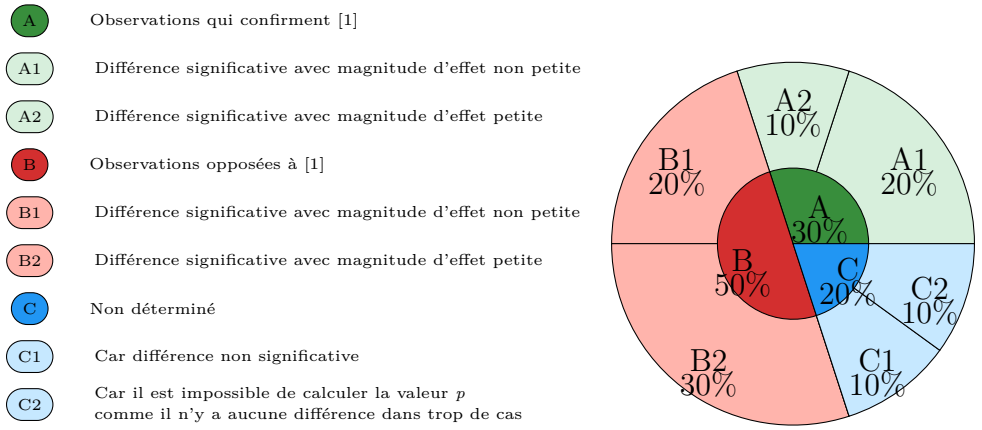
Pour faciliter l'interprétation, on a utilisé des diagrammes en secteurs adaptés (Figure 4.1) pour présenter les résultats. Ces diagrammes servent de représentations visuelles indiquant si les conclusions de l'étude menée par [1] ont été validées ou réfutées par les tests statistiques réalisés. Ils doivent être interprétés comme suit.

Les recherches locales *PA* et *MA* sont évaluées en fonction de : (i) le problème d'optimisation combinatoire abordé, (ii) l'initialisation effectuée, et (iii) la nature de l'ensemble de données. Ainsi, un diagramme en secteurs est créé pour chaque combinaison possible de (i)-(iii).

Prenons l'exemple d'un diagramme en secteurs donné dans la Figure 4.1b pour le problème X sur des instances de type Y pour lesquelles les solutions initiales sont obtenues avec la procédure Z. Dans le diagramme en secteurs, la couleur verte (A) représente la proportion d'instances qui confirment les conclusions de [1], tandis que la couleur rouge (B) est utilisée pour indiquer les cas où la conclusion opposée est en réalité vraie. Enfin, la couleur bleue (C) fait référence aux instances de données à partir desquelles aucune conclusion ne peut être tirée.

En outre, on détaille chacun des cas (A-B-C) représentés dans le diagramme en secteurs. A1 et A2 (vert clair) distinguent les tests confirmant la règle de [1] en fonction de la magnitude d'effet. De même, B1 et B2 distinguent les cas réfutant la règle de [1] de la même manière. En revanche, les cas non concluants sont catégorisés comme C1 si la valeur p du t-Test dépasse 0,05, tandis que C2 fait référence à la proportion d'instances où les deux recherches locales donnent le même optimum local final (c'est-à-dire, $diffAmelioration = 0$).

Examinons la légende de la figure 4.1a et de l'exemple fictif de la figure 4.1b. On étu-



(a) Légende (b) $X :: Y :: Z$

FIGURE 4.1 Exemple de diagramme en secteurs des expérimentations

die le problème du MSSC avec une initialisation aléatoire. On considère ici seulement 5 nombres de points (par exemple, 10, 20, 30, 40 ou 50 points) et deux nombres de clusters (par exemple 2 ou 4 clusters). On dispose de $5 \times 2 = 10$ combinaisons d'hyperparamètres. Pour chaque combinaison, on effectue 1000 exécutions et appliquons les tests statistiques. On détermine ensuite si la différence est significative (valeur p inférieure à 0.05), quelle est la magnitude d'effet (petit, petit-moyen, ...) et quel algorithme donne les meilleurs résultats (avec le signe de la différence en amélioration). 4.1a montre les sous-cas construits avec les informations précédentes. On commence par éliminer les cas de la catégorie C où l'on ne peut pas conclure statistiquement soit en raison d'une valeur p trop élevée (>0,05), soit parce que la statistique ne peut être calculée, les deux stratégies atteignant trop souvent le même coût final. On compte le nombre total de combinaisons pour lesquelles la valeur p est élevée (dans cet exemple 2), on le divise par le nombre total de combinaisons (10 ici) et on le multiplie par 100 pour obtenir le pourcentage de combinaisons pour lesquelles aucune conclusion ne peut être tirée. Pour les 8 combinaisons restantes, il existe une différence significative. On veut ensuite savoir si le meilleur algorithme suit la règle de Hansen et Mladenovic, 2006 [1] adaptée du TSP ou non. On connaît l'initialisation avec la légende : ici l'ini-

tialisation est aléatoire. La règle [1] prévoit que PA donnera de meilleurs résultats. On travaille sur MSSC, un problème de minimisation. La différence d'amélioration est $diffAmelioration = \frac{cost(MA)-cost(PA)}{initCost}$ ce qui signifie que l'on sait que PA est le meilleur algorithme lorsque $cost(MA) > cout(PA) \implies diffAmelioration > 0$. C'est le cas pour 3 combinaisons (30 %). Ainsi, 30 % du diagramme en secteurs intérieur sera pour la zone A. Enfin, on place les combinaisons restantes dans la catégorie B pour les combinaisons d'hyperparamètres qui ne suivent pas [1] (50 % pour 5 combinaisons). Les catégories A, B et C forment le diagramme en secteurs central. Les diagramme en secteurs extérieurs donnent plus de détails sur chaque catégorie. Pour la catégorie A, on distingue les combinaisons d'hyperparamètres où PA est significativement meilleur que MA avec une faible magnitude des cas où il est meilleur avec une magnitude moyenne ou importante. On fait de même pour la catégorie B où MA est significativement meilleur que PA avec une magnitude d'effet grande ou petite. Dans la catégorie B, on ajoute également les combinaisons où la différence moyenne d'amélioration est nulle, ce qui signifie qu'il n'y a aucune différence entre les deux algorithmes. Dans la catégorie C, la catégorie C1 montre la proportion de cas où aucune conclusion ne peut être tirée en raison d'une valeur p trop élevée et la catégorie C2 montre la proportion de cas où aucune conclusion ne peut être tirée en raison d'une taille d'échantillon trop petite sans différence moyenne d'amélioration non nulle.

4.1 TSP

Villes distribuées uniformément

Les principales conclusions de Hansen et Mladenovic, 2006 [1] sont tirées du problème du TSP (*Problème du Voyageur de Commerce*) sur des instances de données uniformément distribuées. Les résultats rapportés dans l'article sont largement en faveur de PA par rapport à MA lorsque la recherche est lancée à partir de solutions *aléatoires*. En revanche, lorsque la recherche locale est initialisée à partir de solutions obtenues par l'heuristique du plus proche voisin (*gloutonne)*, l'initialisation MA est

meilleure que l'initialisation *PA* dans 100% des cas.

Cependant, on peut remarquer dans la Figure 4.2c, Figure 4.2d que la conclusion de [1] concernant le mode d'initialisation est rejetée car la décision est légèrement aléatoire. Pour les initialisations *top-k* et *gloutonne-randomisée*, *PA* est plus performant que *MA* dans la plupart des cas. On observe également que la décision concernant la meilleure recherche locale est moins évidente lorsque l'on utilise *gloutonne-randomisée* (C1 $\approx$ 20%).

TSPLIB

Dans Hansen et Mladenovic, 2006 [1], il a été observé que la stratégie *PA* a montré un pourcentage d'amélioration supérieur pour 21 instances sur 23 (91% des instances) par rapport à la stratégie *MA*, lors de la résolution d'un problème spécifique avec 10 initialisations aléatoires. L'étude reproduite confirme ces résultats : En analysant la Figure 4.3a, on constate que dans la majorité des cas, l'initialisation *aléatoire* donne de meilleurs minima locaux finaux lorsque l'on utilise *PA* par rapport à *MA* (A= 83,67%). En outre, dans A1=75,5% de ces cas, le t-Test présente une magnitude d'effet moyenne à grande. On observe qu'aucune des instances n'a une répartition de villes uniforme dans les cas B et C. 30 instances ont une répartition uniforme de points. Parmi celles-ci aucune n'appartient à la catégorie B ou C. On ne retrouve dans ces catégories que des instances dont les point sont souvent regroupés suivant des bandes horizontales ou verticales ? Cela semble conforter l'hypothèse que pour une répartition uniforme des points, *PA* est la meilleure stratégie. Ces observations concordent bien avec les conclusions de l'étude de [1]. Cependant, *PA* semble également surpasser *MA* avec les initialisations gourmandes *top-3*, *top-5* et *gloutonne-randomisée* comme on peut l'observer dans Figure 4.3(b-d), ce qui n'est pas conforme à l'affirmation de [1].

En outre, on peut remarquer que *PA* semble l'emporter sur *MA* plus souvent lorsque l'on ajoute plus d'aléatoire au processus d'initialisation. C'est ce que l'on peut conclure en observant les figures 4.3(b) et 4.3(c) lorsque K passe de 3 à 5. Enfin, la détermination de la meilleure version de recherche locale devient plus incertaine

lorsque l'on considère l'initialisation *gloutonne-randomisée* ($C1 \approx 22\%$).

4.2 MSSC

Points distribués uniformément

Les tests statistiques concernant les instances générées aléatoirement par *Problème de regroupements avec Minimisation de la Somme des Carrés* ont été effectués pour des points de données uniformément distribués dans un carré de 100×100, et avec des points de données générés à partir d'un nombre défini de distributions gaussiennes, où chaque distribution représente un groupe de données distinct.

Les résultats de la figure 4.4 semblent souvent correspondre aux conclusions de [1] concernant les initialisations *aléatoire* (A = 65.52%), bien que dans 30.46% des cas testés, *MA* soit la meilleure stratégie avec une magnitude d'effet moyenne à grande. Cependant, en utilisant l'initialisation *gloutonne*, l'observation faite par [1] est infirmée dans près de 50% des cas examinés. En outre, il y a environ deux fois plus de cas avec une magnitude d'effet moyenne-grande (B1) que de cas avec une petite magnitude d'effet (B2).

Points distribués suivant des gaussiennes

Les points distribués normalement donnent des résultats plus marqués : la règle est vérifiée pour l'initialisation *aléatoire* (86.21% des cas) mais pas pour l'initialisation *gloutonne* (B = 60.34%). Dans la plupart des cas, une stratégie est largement supérieure à l'autre, les résultats montrant une magnitude d'effet moyenne à grande. Il n'y a qu'une faible proportion de cas où aucune conclusion ne peut être tirée parce que la valeur p est trop grande. On observe à nouveau que pour *gloutonne-randomisée*, la proportion de cas où la règle n'est pas vérifiée est plus importante que pour les initialisations de type *top-k* (68.97% pour *gloutonne-randomisée* contre 60.34% ou moins pour *top-k*)

Benchmark clustering

Les résultats pour le benchmark de clustering utilisé dans ce travail sont présentés dans la figure 4.6. On remarque que la conclusion de [1] est souvent confirmée lorsque l'on démarre la recherche locale à partir d'une initialisation *aléatoire* (A1 = 76.93%). Cependant, la même chose n'est pas observée avec les initialisations plus gourmandes *top-3*, *top-5* et *gloutonne-randomisée*, où *PA* est souvent la meilleure option de recherche locale.

Il convient de noter que, bien que B=76.92% pour les initialisations *top-3* et *top-5*, la valeur de B1 est plus élevée pour ces dernières. Cela indique que on est plus confiants dans la supériorité de *PA* par rapport à *MA* lorsque la recherche locale est initialisée avec *top-5*. Enfin, la conclusion concernant la meilleure recherche locale est moins claire lorsque l'initialisation *gloutonne-randomisée* est employée.

4.3 MAXSAT-pondéré

MAXSAT evaluation benchmark 2021

La figure 4.7 présente les résultats pour les instances MAXSAT pondérées du benchmark d'évaluation MAXSAT 2021.

On observe dans le diagramme en secteurs Figure 4.7(a) que le résultat de [1] est souvent vérifié (A = 82,76%) avec une initialisation *aléatoire*. Cependant, ce n'est plus le cas pour des initialisations plus gourmandes : B1 = 44,83% pour *top-3*, B1 = 55.17 pour *top-3*, et B1 = 72.41% pour *gloutonne-randomisée*.

De plus, il est intéressant de noter que, contrairement aux tests statistiques avec les instances TSPLIB (Figure 4.3), *MA* surpasse clairement *PA* avec l'initialisation *gloutonne-randomisée* (A = 6.90% et C = 6.90%). Enfin, la décision concernant la meilleure recherche locale est moins claire avec les initialisations *top-3* et *top-5* comme le nombre élevé de cas où la meilleure stratégie n'est pas claire (C = 34.48% pour *top-3* et C = 31.03 pour *top-3*). Ceci est principalement dû à une valeur p qui ne permet pas de conclure en faveur d'une stratégie plutôt qu'une autre.

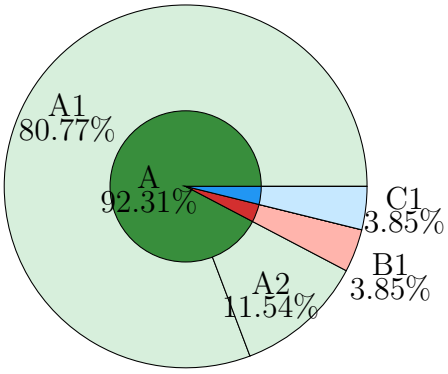
4.4 Analyse globale

En conclusion, les résultats de [1] sont confirmés pour le TSP sur les deux initialisations *aléatoire* et *gloutonne* utilisées dans l'article. On retrouve bien que la stratégie du *Premier améliorant* est la meilleure pour l'initialisation *aléatoire* et la stratégie du *Meilleur améliorant* est la meilleure pour l'initialisation *gloutonne*. Les 2 seuls cas où pour l'initialisation *aléatoire* la bonne stratégie n'est pas trouvée meilleure sont mentionnés comme tel dans le papier. Cependant, en élargissant le concept d'initialisation gloutonne à des initialisations gloutonnes incluant de l'aléatoire (*top-k* et *gloutonne-randomisée*), la conclusion n'est plus vérifiée. Il est intéressant de noter que pour ces initialisations la stratégie de *Premier améliorant* est à nouveau la meilleure stratégie à utiliser. Cette conclusion s'observe sur tous les problèmes et jeux de données même si pour les instances générées aléatoirement du MSSC les résultats sont moins tranchés en faveurs de la stratégie du *Premier améliorant*.

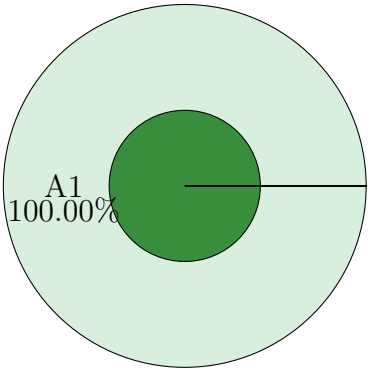
On peut cependant remarquer quelques tendances : pour tous les jeux de données, la stratégie *PA* fonctionne mieux dans une majorité de cas pour l'initialisation *aléatoire*. Un examen plus approfondi est nécessaire pour déterminer quand la règle peut être appliquée et quels sont les facteurs qui expliquent pourquoi la stratégie *PA* donnent de meilleurs résultats pour l'initialisation *aléatoire*. Pour le TSP on peut noter qu'il y a plus de cas où aucune des deux stratégie peut être identifiée comme meilleure (20% des cas contre de l'ordre de 10% pour les initialisations *top-k*). Pour le *Problème de regroupements avec Minimisation de la Somme des Carrés*, on note que les résultats sont moins tranchés en faveur d'une méthode : En générale il y a environ 25% des cas où la règles est vérifiée pour les initialisations gloutonnes. En regardant le détail des valeurs on peut également noter qu'à partir d'environ 16 clusters un effet de seuil apparait : après un certain seuil de nombre de point (dépendant du nombre de clusters) la règle es vérifiée (cas A) mais après ce seuil elle ne l'est plus. Ce phénomène serait à étudier plus en profondeur.

Il est essentiel de souligner que les cas examinés ici concernent exclusivement des algorithmes de recherche locale sans méta-heuristique. En pratique, l'utilisation de

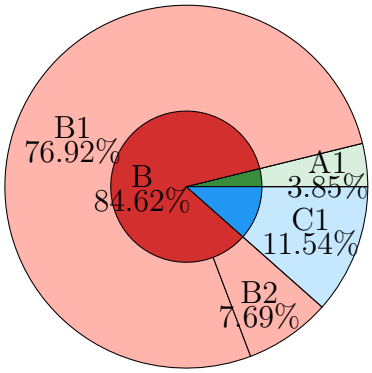
méta-heuristiques permet d'obtenir des solutions de meilleure qualité en évitant les minima locaux. Par conséquent, il serait pertinent d'explorer si les conclusions tirées pour la recherche locale peuvent être étendues aux méta-heuristiques. Les méta-heuristiques pourraient potentiellement surmonter les limitations des stratégies *PA* et *MA*, et trouver une différence entre les deux stratégies notamment pour les cas indéterminés (cas C) de l'étude menée. On a noté en effet pour certaines stratégies(*MAXSAT : :top-k* par exemple) une grande proportion de cas où il n'y a aucune différence significative entre les deux algorithmes.



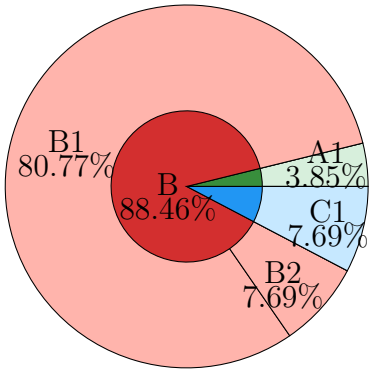
(a) *TSP : :Uniforme : :aléatoire*



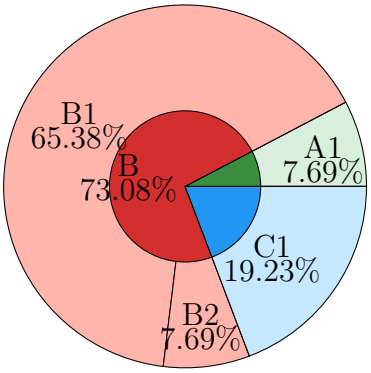
(b) *TSP : :Uniforme : :gloutonne*



(c) *TSP : :Uniforme : :top-3*

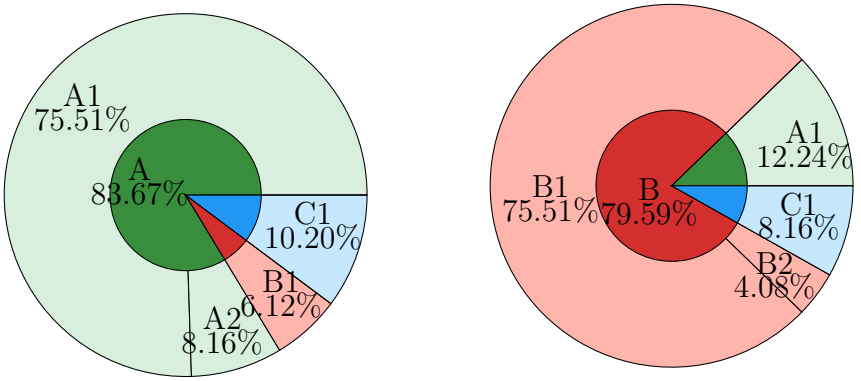


(d) *TSP : :Uniforme : :top-5*



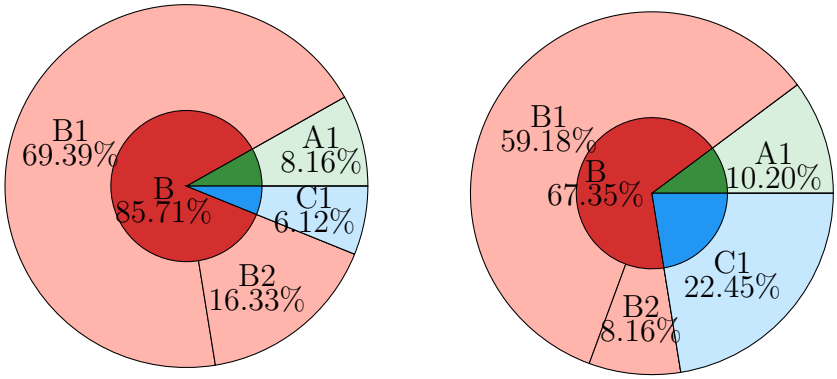
(e) *TSP : :Uniforme : :gloutonne-randomisée*

FIGURE 4.2 Diagramme en secteurs pour les instances TSP avec des villes uniformément distribuées



(a) *TSP : :TSPLIB : :aléatoire*

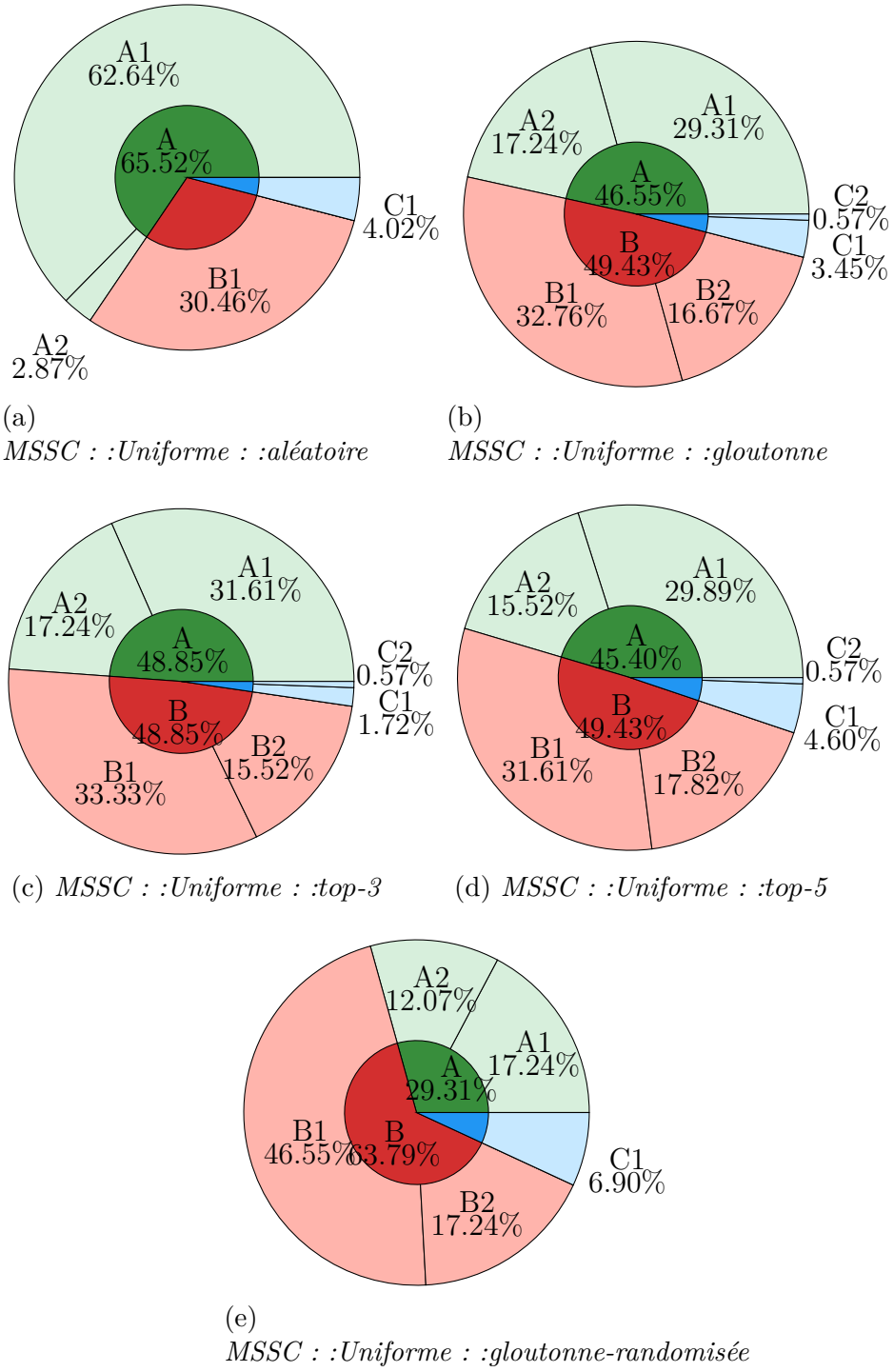
(b) *TSP : :TSPLIB : :top-3*



(c) *TSP : :TSPLIB : :top-5*

(d)
TSP : :TSPLIB : :gloutonne-randomisée

FIGURE 4.3 Diagrammes en secteurs pour les instances TSPLIB



(a)
MSSC : :Uniforme : :aléatoire

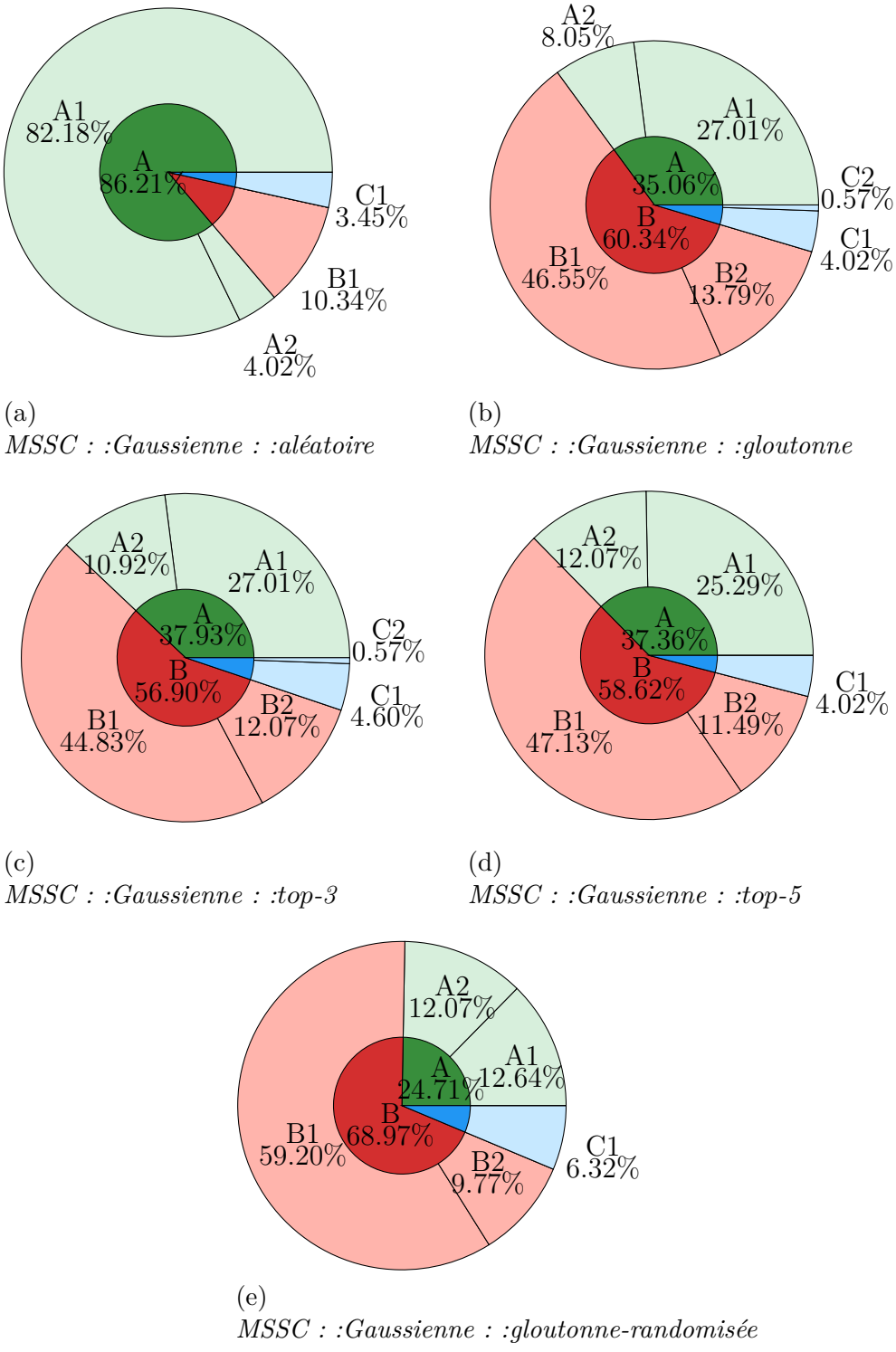
(b)
MSSC : :Uniforme : :gloutonne

(c) *MSSC : :Uniforme : :top-3*

(d) *MSSC : :Uniforme : :top-5*

(e)
MSSC : :Uniforme : :gloutonne-randomisée

FIGURE 4.4 Diagramme en secteur pour le MSSC avec des points distribués uniformément



(a)
MSSC : :Gaussienne : :aléatoire

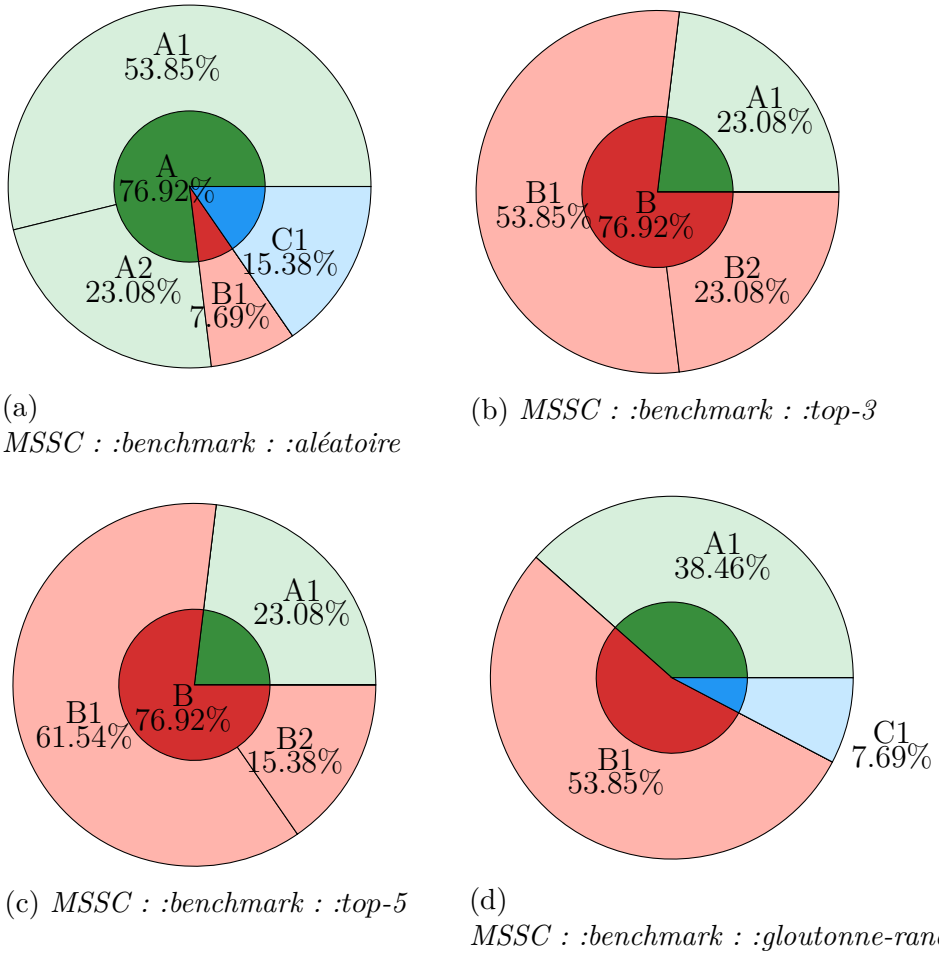
(b)
MSSC : :Gaussienne : :gloutonne

(c)
MSSC : :Gaussienne : :top-3

(d)
MSSC : :Gaussienne : :top-5

(e)
MSSC : :Gaussienne : :gloutonne-randomisée

FIGURE 4.5 Diagramme en secteurs pour le MSSC avec des points distribués suivant une gaussienne



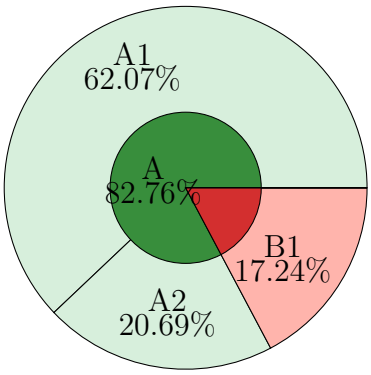
(a) *MSSC : :benchmark : :aléatoire*

(b) *MSSC : :benchmark : :top-3*

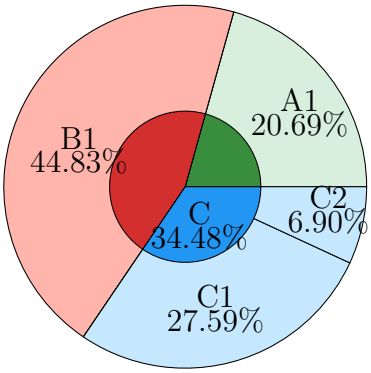
(c) *MSSC : :benchmark : :top-5*

(d) *MSSC : :benchmark : :gloutonne-randomisée*

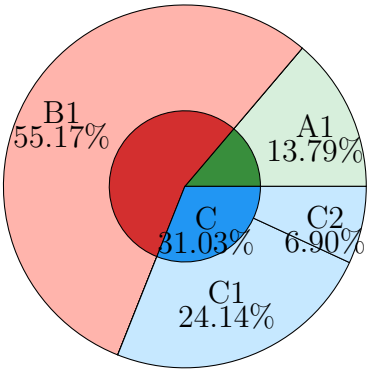
FIGURE 4.6 Diagramme en secteurs pour le MSSC pour le benchmark 1



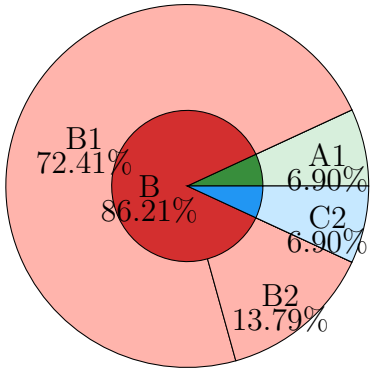
(a)
*MAXSAT : :benchmark : :*aléatoire



(b)
MAXSAT : :benchmark : :top-3



(c)
MAXSAT : :benchmark : :top-5



(d)
*MAXSAT : :benchmark : :*gloutonne-randomisée

FIGURE 4.7 Diagramme en secteur pour le MSSC pour le *MAXSAT Evaluation benchmark 2021*

CHAPITRE 5 CONCLUSION ET RECOMMANDATIONS

5.1 Synthèse des travaux

Une étude statistique plus approfondie des stratégies de première et de meilleure amélioration a été réalisée. Dans Hansen et Mladenovic, 2006 [1] il est observé que pour une initialisation aléatoire la stratégie de première amélioration donne une meilleure solution finale que la stratégie de meilleure amélioration. A l'inverse pour une initialisation améliorée par une heuristique constructive, la meilleure amélioration donne de meilleurs résultats. Ce résultat a été observé pour le TSP par [1]. Cependant leur étude comporte certaines limites : la comparaison des stratégies se base sur des moyennes et le nombre d'échantillons utilisés est faible pour certains cas d'études (instances de grandes tailles et instances TSPLIB). Cette conclusion a été utilisée par 36 papiers dans la littérature dont 13 l'ont utilisée directement sans réaliser à nouveau la comparaison. Une portion majoritaire de ces papiers travaille sur des problèmes et / ou avec des voisinages différents. Cela pose alors la question de savoir s'il est possible d'utiliser la règle dans chacun de ces cas. Afin de vérifier si la règle de [1] s'applique au TSP on a réalisé une analyse statistique pour ce problème. Puis pour montrer que cette règle ne s'applique pas à tout autre problème et algorithme cette étude a été reproduite pour le *Problème de regroupements avec Minimisation de la Somme des Carrés* et le *MAXSAT-pondéré*. La différence en amélioration de la solution initiale moyenne a été choisie comme métrique principale pour déterminer quel algorithme est le plus performant. Pour déterminer si une différence est significative un t-Test ou un test de somme de rangs signés de Wilcoxon a été utilisé en fonction de la distribution des données recueillies. La magnitude d'effet a été adjointe pour déterminer la magnitude de la différence. Il apparaît alors que même si la règle est valide pour le TSP et des villes uniformément réparties, cela n'est plus le cas pour des instances réelles (TSPLIB) ou pour les deux autres problèmes étudiés. Même si parfois la différence entre les deux stratégies est significative, aucune stratégie ne sort clairement "vainqueur" pour chaque initialisation. Il apparait alors que la règle de

Hansen et Mladenovic, 2006 [1] ne peut être utilisée directement pour tout problème, jeux de données et initialisation. Il convient de réaliser une étude statistique au cas par cas pour déterminer pour le problème, le jeu de données, l'initialisation précise quelle est la stratégie qui donne de meilleurs performances. On conseille alors d'utiliser la différence en amélioration comme mesure neutre pour déterminer le meilleur algorithme. Puis on peut utiliser le T-Test ou le test de somme de rangs signés de Wilcoxon pour déterminer si la différence est significative et quelle est sa magnitude. Cela permet alors de déterminer quelle stratégie choisir.

5.2 Limites et Améliorations futures

La conclusion de Hansen et Mladenovic, 2006 [1] n'est pas valable pour le TSP (pour tous les jeux de données et initialisations) et n'est pas valables pour d'autre problèmes. Cependant, une règle plus complexe pourrait exister pour choisir une stratégie en fonction du problème, jeu de données et initialisation. Il pourrait être intéressant de tenter de trouver cette règle et de comprendre pourquoi elle est applicable en étudiant la trajectoire explorée par chaque stratégie. Particulièrement, dans le cas du TSP il serait intéressant d'approfondir l'étude de l'impact de la distribution des villes sur la meilleure stratégie. Pour le clustering il faudrait réaliser une étude similaire en prenant en compte le nombre de clusters des instances.

Hansen et Mladenovic, 2006 [1] comparent les deux stratégies pour le nombre d'itérations. Cette métrique est le deuxième enjeux important lorsque l'on cherche à avoir une solution rapidement. Ces conclusions ont été également utilisées dans d'autres papiers. Il serait alors pertinent de réaliser la même étude pour différents jeux de données pour le nombre d'itérations pour les trois problèmes étudiés.

D'autre part, beaucoup d'algorithmes utilisent ces stratégies dans le câdre de métaheuristiques. Ces algorithmes sont plus couramment utilisés pour des applications concrètes puisqu'ils peuvent s'échapper d'optima locaux. Il serait alors important d'étudier quelle est la meilleure stratégie pour ces types d'algorithmes en étudiant l'impact du changement de voisinage, si certains voisinages sont plus favorables à

une stratégie et déterminer si il est possible d'expliquer pourquoi cela se produit.

Après avoir mené des essais, il semble que l'identification d'une règle simple pour déterminer la meilleure stratégie en fonction du type d'instance étudiée ne soit pas évidente. Cependant, il est pertinent d'envisager l'utilisation d'un algorithme automatisé de recherche d'hyperparamètres, afin de déterminer la meilleure approche. Pour ce faire, il serait nécessaire de fournir des informations telles que le problème et le type d'initialisation, ainsi qu'une mesure représentative de la distribution des points pour chaque instance. En effet, on a remarqué que les résultats peuvent varier considérablement entre des instances uniformément réparties et des instances réelles lorsqu'il s'agit du problème du TSP.

Enfin, si un choix simple n'est pas possible/complexe à réaliser, il serait pertinent d'entraîner un modèle de deep learning pour choisir la meilleure stratégie d'amélioration en lui donnant les paramètres du problème. Cependant, il est nécessaire de fournir un jeux de données suffisamment variées (nombre de points, de clusters, distribution des points pour le *Problème de regroupements avec Minimisation de la Somme des Carrés* et le TSP) pour permettre au modèle de dégager une règle si cela est possible. Une première approche serait de fournir un jeux de donnée très diversifié et avec beaucoup d'exemples pour prédire la meilleure stratégie en fonction de la solution initiale. Une autre approche serait de fournir au modèle la solution initiale et de lui faire prédire après k itérations quelle sera la nouvelle solution et répéter l'opération à partir de cette solution jusqu'à ce qu'aucune modification ne soit détectée ou que le modèle indique qu'il ait terminé. On pourrait alors entrainer deux modèles, le premier capable de prédire la solution finale de PA et le deuxième celle de MA. Il serait possible de déterminer meilleure solution en exécutant ces deux modèles sur la solution initiale et en étudiant le signe de la différence en amélioration.

RÉFÉRENCES

[1] P. Hansen et N. Mladenović, "First vs. best improvement : An empirical study," *Discrete Applied Mathematics*, vol. 154, n°. 5, p. 802–817, 2006, 129. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0166218X05003070

[2] H. Tang, "Efficient Implementation of Improvement Procedures for Vehicle Routing with Time-Dependent Travel Times," http ://dx.doi.org/10.3141/2089-09, jan 1 2009.

[3] A. Dawid, "Entwicklung und Analyse eines Verfahrens zur effizienten Lösung des Tourenplanungs-und Laderaumoptimierungsproblems," jan 1 2008.

[4] R. Lopes, "Location-routing problems of semi-obnoxious facilities : Approaches and decision support," jan 1 2011.

[5] R. Lopes, C. Ferreira et B. Santos, "A simple and effective evolutionary algorithm for the capacitated location–routing problem," http ://dx.doi.org/10.1016/j.cor.2016.01.006, jan 1 2016.

[6] B. Turan, S. Minner et R. F. Hartl, "A VNS approach to multi-location inventory redistribution with vehicle routing," *Computers amp ; Operations Research*, vol. 78, p. 526–536, 2 2017.

[7] M. Abderrahim *et al.*, "Bi-local search based variable neighborhood search for job-shop scheduling problem with transport constraints," *Optimization Letters*, vol. 16, n°. 1, p. 255–280, nov 26 2020.

[8] T. Pereira *et al.*, *Review of Basic Local Searches for Solving the Minimum Sum-of-Squares Clustering Problem*. Springer International Publishing, 2018, p. 249–270.

[9] T. Pereira, "Novas heurísticas para o agrupamento de dados pela soma mínima de distâncias quadráticas," https ://repositorio.ufrn.br/handle/123456789/24010, jan 1 2017.

[10] L. R. Costa, D. Aloise et N. Mladenović, "Less is more : basic variable neighborhood search heuristic for balanced minimum sum-of-squares clustering," *Information Sciences*, vol. 415-416, p. 247–253, 11 2017.

[11] P. Goos *et al.*, "A nonlinear multidimensional knapsack problem in the optimal design of mixture experiments," *European Journal of Operational Research*, vol. 281, nº. 1, p. 201–221, 2 2020.

[12] J. Sánchez-Oro *et al.*, "Improving the performance of embedded systems with variable neighborhood search," *Applied Soft Computing*, vol. 53, p. 217–226, 4 2017.

[13] M. A. Akbay, C. B. Kalayci et O. Polat, "A parallel variable neighborhood search algorithm with quadratic programming for cardinality constrained portfolio optimization," *Knowledge-Based Systems*, vol. 198, p. 105944, 6 2020.

[14] R. Max_Wood, "Modelling activities at a neurological rehabilitation unit," jan 1 2011.

[15] A. Mjirda *et al.*, "Sequential variable neighborhood descent variants : an empirical study on the traveling salesman problem," *International Transactions in Operational Research*, vol. 24, nº. 3, p. 615–633, apr 6 2016.

[16] J. C. Nascimento Silva *et al.*, "Finding the Maximum Multi Improvement on neighborhood exploration," *Optimization Letters*, vol. 16, nº. 1, p. 97–115, feb 22 2020.

[17] G. Babin, S. Deneault et G. Laporte, "Improvements to the Or-opt heuristic for the symmetric travelling salesman problem," *Journal of the Operational Research Society*, vol. 58, nº. 3, p. 402–407, 3 2007.

[18] C. Becker *et al.*, "In-depth analysis of granular local search for capacitated vehicle routing," *Discrete Applied Mathematics*, vol. 329, p. 61–86, 4 2023.

[19] S. Irnich, B. Funke et T. Grünert, "Sequential search and its application to vehicle-routing problems," *Computers amp ; Operations Research*, vol. 33, nº. 8, p. 2405–2429, 8 2006.

[20] H. F. Amaral, S. Urrutia et L. M. Hvattum, "Delayed improvement local search," *Journal of Heuristics*, vol. 27, nº. 5, p. 923–950, jun 29 2021.

[21] N. Mladenović *et al.*, "Less is more approach : basic variable neighborhood search for the obnoxious*p*-median problem," *International Transactions in Operational Research*, vol. 27, nº. 1, p. 480–493, mar 1 2019.

[22] L. Costa, "Abordagem heurística baseada em busca em vizinhança variável para o agrupamento balanceado de dados pelo critério da soma mínima das distâncias quadráticas," https ://repositorio.ufrn.br/handle/123456789/21976, jan 1 22.

[23] N. Mladenović, R. Todosijević et D. Urošević, "Less is more : Basic variable neighborhood search for minimum differential dispersion problem," *Information Sciences*, vol. 326, p. 160–171, 1 2016.

[24] M. M. Drugan et D. Thierens, "Stochastic Pareto local search : Pareto neighbourhood exploration and perturbation strategies," *Journal of Heuristics*, vol. 18, nº. 5, p. 727–766, jul 3 2012.

[25] J. Brimberg *et al.*, "Variable neighborhood search for the heaviest -subgraph," *Computers amp ; Operations Research*, vol. 36, nº. 11, p. 2885–2891, 11 2009.

[26] A. A. Yassine, R. H. Chidiac et I. H. Osman, "Simultaneous optimisation of products, processes, and people in development projects," *Journal of Engineering Design*, vol. 24, nº. 4, p. 272–292, 4 2013.

[27] T. Hackl, "Local Search Methods for the Particle Therapy Patient Scheduling Problem," https ://www.ac.tuwien.ac.at/files/pub/hackl_18.pdf, jan 1 2018.

[28] B. BONTOUX, "Techniques hybrides de recherche exacte et approchée : application à des problèmes de transport," https ://homepages.laas.fr/artigues/theseBorisBontoux.pdf, jan 1 2008.

[29] T. Erdelić et T. Carić, "A Survey on the Electric Vehicle Routing Problem : Variants and Solution Approaches," *Journal of Advanced Transportation*, vol. 2019, p. 1–48, may 9 2019.

[30] S. Almoustafa, "Distance-constrained vehicle routing problem : Exact and approximate solution (mathematical programming)," jan 1 2013.

[31] J. Brimberg et S. Salhi, *A General Framework for Local Search Applied to the Continuous p-Median Problem.* Springer International Publishing, 2019, p. 89–108.

[32] R. Sheikh, “Some applications of continuous variable neighbourhood search metaheuristic (mathematical modelling),” jan 1 2012.

[33] P. Da Costa *et al.*, “Learning 2-opt Local Search from Heuristics as Expert Demonstrations,” dans *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 18 2021.

[34] S. Bougrine, M. A. E. Majdouli et A. A. E. Imrani, “A novel iterative improvement pivoting rule for local search heuristics,” dans *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, jul 15 2017.

[35] S. Tari, M. Basseur et A. Goëffon, “Partial neighborhood local searches,” *International Transactions in Operational Research*, vol. 29, n°. 5, p. 2761–2788, apr 27 2021.

[36] M. Inja *et al.*, *Queued Pareto Local Search for Multi-Objective Optimization*. Springer International Publishing, 2014, p. 589–599.

[37] S. Tari et G. Ochoa, “Local search pivoting rules and the landscape global structure,” dans *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, jun 26 2021.

[38] M. Basseur et A. Goëffon, “On the efficiency of worst improvement for climbing NK-landscapes,” dans *GECCO ’14 : Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA : Association for Computing Machinery, juill. 2014, p. 413–420.

[39] C. Becker *et al.*, “In-depth analysis of granular local search for capacitated vehicle routing,” *Discrete Appl. Math.*, vol. 329, p. 61–86, avr. 2023.

[40] P. Da Costa *et al.*, “Learning 2-opt Local Search from Heuristics as Expert Demonstrations,” dans *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, juill. 2021, p. 1–8.

[41] Y. Wu *et al.*, “Learning Improvement Heuristics for Solving Routing Problems,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 33, n°. 9, p. 5057–5069, avr. 2021.

[42] P. da Costa *et al.*, “Learning 2-Opt Heuristics for Routing Problems via Deep Reinforcement Learning,” *SN Comput. Sci.*, vol. 2, n°. 5, p. 1–16, sept. 2021.

[43] E. Osaba, X.-S. Yang et J. Del Ser, *Traveling salesman problem : a perspective review of recent research and new results with bio-inspired metaheuristics*. Elsevier, p. 135–164. [En ligne]. Disponible : http://dx.doi.org/10.1016/b978-0-12-819714-1.00020-8

[44] F. Bacchus, M. Järvisalo et R. Martins, "Chapter 24. Maximum Satisfiabiliy," dans *Handbook of Satisfiability*. IOS Press, 2021, p. 929–991.

[45] H. A. Kautz, A. Sabharwal et B. Selman, "Incomplete Algorithms," *Frontiers Artif. Intell. Appl.*, vol. 185, n°. 1, janv. 2009.

[46] D. Arthur et S. Vassilvitskii, "k-means++ : The advantages of careful seeding," Stanford, Rapport technique, 2006.

[47] M. Poloczek *et al.*, "Greedy Algorithms for the Maximum Satisfiability Problem : Simple Algorithms and Inapproximability Bounds," *SIAM J. Comput.*, juin 2017. [En ligne]. Disponible : https://epubs.siam.org/doi/10.1137/15M1053369

[48] A. Asuncion et D. Newman, "UCI machine learning repository," 2007.

[49] (2022, nov.) Maxsat evaluation 2021. [Online ; accessed 26. Jan. 2023]. [En ligne]. Disponible : https://maxsat-evaluations.github.io/2021/benchmarks.html

[50] G. M. Sullivan et R. Feinn, "Using Effect Size—or Why the P Value Is Not Enough," *Journal of Graduate Medical Education*, vol. 4, n°. 3, p. 279, sept. 2012.

[51] "Statistical power analysis for the behavioral sciences," mai 2023, [Online ; accessed 23. May 2023]. [En ligne]. Disponible : https://www.worldcat.org/fr/title/Statistical-power-analysis-for-the-behavioral-sciences/oclc/17877467

[52] D. S. Kerby, "The Simple Difference Formula : An Approach to Teaching Nonparametric Correlation," *Comprehensive Psychology*, vol. 3, p. 11.IT.3.1, janv. 2014.

[53] J. W. Pratt, "Remarks on Zeros and Ties in the Wilcoxon Signed Rank Procedures on JSTOR," *J. Am. Stat. Assoc.*, vol. 54, n°. 287, p. 655–667, sept. 1959. [En ligne]. Disponible : https://www.jstor.org/stable/2282543

ANNEXE A DÉTAIL DES INSTANCES BENCHMARK

Nom	#villes	Dans [1]
eil51	51	Yes
berlin52	52	Yes
st70	70	No
eil76	76	No
pr76	76	No
rat99	99	No
kroA100	100	No
kroB100	100	No
kroC100	100	Yes
kroD100	100	Yes
kroE100	100	No
rd100	100	No
eil101	101	No
lin105	105	Yes
pr107	107	No
pr124	124	No
ch130	130	Yes
pr136	136	No
pr144	144	No
ch150	150	Yes
kroA150	150	No
kroB150	150	No
pr152	152	No
u159	159	No

Nom	#villes	Dans [1]
rat195	195	No
d198	198	Yes
kroA200	200	No
kroB200	200	No
ts225	225	No
tsp225	225	Yes
pr226	226	No
gil262	262	No
pr264	264	No
a280	280	No
pr299	299	Yes
lin318	318	No
linhp318	318	Yes
rd400	400	Yes
fl417	417	No
pr439	439	Yes
pcb442	442	Yes
d493	493	No
u574	574	Yes
rat575	575	No
p654	654	Yes
d657	657	No
u724	724	No
rat783	783	No

FIGURE A.1 Instances TSPLIB utilisées

Nom	#points	#clusters	#dimensions
Iris	150	3	4
Thyroid	215	3	5
LibrasMovement	360	15	90
UserKnowledge	403	4	5
WaterTreatment	527	13	38
SyntheticControlChart	600	6	60
Balance	625	3	4
Vowel	871	11	3
Yeast	1484	10	8
MultipleFeatures	2000	7	240
Cardiotocography	2126	10	23
ImageSegmentation	2310	7	19
Optical	3823	10	64

FIGURE A.2 Instances utilisées dans le benchmark de clustering

Nom	#variables	#clauses
BTBNSL/Rounded_BTWBNSL_asia_1000_1_3.scores_TWBound_3	1169	4452
drmx-atmostk/drmx-am20-outof-50-ecardn-w	1039	1534
drmx-atmostk/drmx-am20-outof-50-esortn-w	1137	1681
drmx-atmostk/drmx-am28-outof-60-esortn-w	1147	1691
drmx-atmostk/drmx-am32-outof-70-eseqc-w	2278	4523
drmx-atmostk/drmx-am32-outof-70-esortn-w	3013	4485
MaxSATQueriesinInterpretableClassifiers/wcnf_gz/compas_test_3_DNF_1_10	1065	3755
min-width/MinWidthCB_milan_100_12_1k_2s_1t_2	1458	1816
preference_planning/WCNF_pathways_p01	1004	4855
pseudoBoolean/factor/normalized-factor-size=9-P=107-Q=373.opb	1110	3046
pseudoBoolean/factor/normalized-factor-size=9-P=127-Q=211.opb	1072	3098
pseudoBoolean/factor/normalized-factor-size=9-P=127-Q=359.opb	1114	3205
pseudoBoolean/factor/normalized-factor-size=9-P=149-Q=509.opb	1133	3283
pseudoBoolean/factor/normalized-factor-size=9-P=157-Q=163.opb	1063	2945
pseudoBoolean/factor/normalized-factor-size=9-P=157-Q=373.opb	1110	3220
pseudoBoolean/factor/normalized-factor-size=9-P=191-Q=509.opb	1128	3137
pseudoBoolean/factor/normalized-factor-size=9-P=23-Q=379.opb	1047	2962
pseudoBoolean/factor/normalized-factor-size=9-P=263-Q=367.opb	1132	3288
pseudoBoolean/factor/normalized-factor-size=9-P=307-Q=449.opb	1140	3174
pseudoBoolean/factor/normalized-factor-size=9-P=331-Q=331.opb	1139	3248
pseudoBoolean/factor/normalized-factor-size=9-P=89-Q=487.opb	1122	3092
set-covering/random/scp/scp46_weighted	1000	1200
set-covering/random/scp/scp47_weighted	1000	1200
set-covering/random/scp/scp49_weighted	1000	1200
set-covering/random/scp/scp51_weighted	2000	2200
set-covering/random/scp/scp64_weighted	1000	1200
set-covering/random/scp/scp65_weighted	1000	1200
warehouses/cap71.wcsp	1664	3978
warehouses/cap72.wcsp	1664	3978

FIGURE A.3 MAXSAT evaluation benchmark : instances sélectionnées

ANNEXE B DÉTAIL DES ALGORITHMES D'INITIALISATION DE LA RECHERCHE LOCALE

ANNEXE C TSP

Algorithm 3 TSP initialisation *aléatoire*

Require: n, Le nombre de villes du problème
  function $Initialization_{aléatoire}(n)$
    $tour \leftarrow \{idTown_1, ..., idTown_n\}$
    $Shuffle(tour)$
    return *tour*
  end function

Algorithm 4 TSP initialisation *gloutonne*

Require: n, Nombre de villes du problème
1: **function** $Initialization_{gloutonne}(n)$
2: $initialTown \leftarrow aléatoireTown(remainingTowns)$
3: $tour \leftarrow \{initialTown\}$
4: $remainingTowns \leftarrow \{id_i, \forall i \in [\![1, n]\!], id_i \neq initialTown\}$
5: **while** $remainingTowns.length > 0$ **do**
6: $nextTown \leftarrow$ville la plus proche de la dernière choisie encore dans remainingTowns
7: $tour \leftarrow tour \cup \{nextTown\}$
8: $remainingTowns \leftarrow \{id_i, \forall id_i \in remainingTowns, id_i \neq nextTown\}$
9: **end while**
10: **return** *tour*
11: **end function**

Algorithm 5 TSP initialisation *top-K*

Require: n, Nombre de villes du problème

function $Initialization_{gloutonne}(n)$
    $initialTown \leftarrow aléatoireTown(remainingTowns)$
    $tour \leftarrow \{initialTown\}$
    $remainingTowns \leftarrow \{id_i, \forall i \in [\![1, n]\!], id_i \neq initialTown\}$
    while $remainingTowns.length > 0$ **do**
        $topK \leftarrow k$ villes les plus proches de la dernière choisie encore dans remainingTowns
        $nextTown \leftarrow$ aléatoire town among the *topk*
        $tour \leftarrow tour \cup \{nextTown\}$
        $remainingTowns \leftarrow \{id_i, \forall id_i \in remainingTowns, id_i \neq nextTown\}$
    end while
    return *tour*
end function

Algorithm 6 TSP initialisation *gloutonne-randomisée*

Require: n, Le nombre de villes du problème

function $Initialisation_{gloutonnealéatoireIZED}(n)$
    $initialTown \leftarrow aléatoireTown(remainingTowns)$
    $tour \leftarrow \{initialTown\}$
    $remainingTowns \leftarrow \{id_i, \forall i \in [\![1, n]\!], id_i \neq initialTown\}$
    while $remainingTowns.length > 0$ **do**
        $nextTown \leftarrow id_i$ avec probabilité $\frac{D(t_{id_i}, lastTown(tour))}{\sum_{id_j \in remainingTown} D(id_j, lastTown(tour))}$
        $tour \leftarrow tour \cup \{nextTown\}$
        $remainingTowns \leftarrow \{id_i, \forall id_i \in remainingTowns, id_i \neq nextTown\}$
    end while
    return *tour*
end function

ANNEXE D MSSC

Algorithm 7 Initialisation *aléatoire* MSSC

Entrées :
- n, le nombre de points du problème
- K, le nombre de clusters

Sorties :
- A, le cluster assigné à chaque point : a_i est le cluster assigné au point i

```
1: function Initialization_aléatoire(P,K)
2:     for i = 1 to n do
3:         a_i ← aléatoire(1, k)
4:     end for
5:     return A
6: end function
```

Algorithm 8 Initialisation *gloutonne* MSSC

Entrées :

— P, les points du problème

— K, le nombre de clusters

Sorties :

— C, les centroids des clusters

— A, les assignements de chaque point à un cluster : a_i est le cluster du point i

procedure $Initialization_{gloutonne}(P,K)$
    //Partie 1 : Variation de Kmeans++
    $C \leftarrow \{c_1\}$ avec des coordonnées tirées aléatoirement dans $\mathcal{X} = [0; 100]^2$
    $i \leftarrow 1$
    while $i < K$ **do**
        $C \leftarrow C \cup \{c_i\}$ avec c_i le point le plus éloigné des clusters existants
        $i \leftarrow i + 1$
    end while
    //Partie 2 : Algorithme Kmeans
    while A modifié **do**
        Modifier A pour assigner chaque point à son centroid le plus proche
        Modifier C le centre de gravité de chaque cluster
    end while
    return C, A
end procedure

Algorithm 9 Initialisation *top-K* MSSC

Entrées :

— P, les points du problème

— K, le nombre de clusters

Sorties :

— C, les centroids des clusters

— A, les assignements de chaque point à un cluster : a_i est le cluster du point i

procedure $Initialization_{gloutonneTOPK}(P,K)$
    //Partie 1 : Variation de KMeans++
    $C \leftarrow \{c_1\}$ distribués uniformément dans $\mathcal{X} = [0; 100]^2$
    $i \leftarrow 1$
    while $i < K$ **do**
        // D(p) la distance la plus courte entre le point p et son plus proche centroid dans C
        $topK \leftarrow$ les k points les plus éloignés des centroids de C
        $C \leftarrow C \cup \{p'\}$ avec p' un point aléatoire dans $topK$
        $i \leftarrow i + 1$
    end while
    //Partie 2 : Algorithme Kmeans
    while A modifié **do**
        Modifier A pour assigner chaque point à son centroid le plus proche
        Modifier C le centre de gravité de chaque cluster
    end while
    return C, A
end procedure

Algorithm 10 Initialisation *gloutonne-randomisée* MSSC

Require: P, les points du problème
Require: K, le nombre de clusters
Ensure: C, Les centroides des clusters
Ensure: A, les assignements de chaque points aux clusters : a_i est le cluster du point i

1: **procedure** $Initialization_{gloutonneRandomisée}(P,K)$
2: //Partie 1 : Kmeans++
3: $C \leftarrow \{c_1\}$ tirée uniformément dans $\mathcal{X} = [0;100]^2$
4: $i \leftarrow 1$
5: **while** $i < K$ **do**
6: // D(p) la plus petite distance entre le point p et son plus proche centroid parmi ceux de C
7: $C \leftarrow C \cup \{p'\}$ avec p' choisi avec $\frac{D(p')^2}{\sum_{p \in P} D(p)^2}$
8: $i \leftarrow i + 1$
9: **end while**
10: //Partie 2 : Kmeans
11: **while** A modifié **do**
12: Modifier A pour assigner chaque point à son centroid le plus proche
13: Modifier C avec le centre de gravité de chaque cluster
14: **end while**
15: **return** C, A
16: **end procedure**

ANNEXE E MAXSAT-PONDÉRÉ

On définit la frontière de décision, moyenne entre le poids des clauses saturées et le poids des clauses saturées et saturables. Puis contrairement à [47], on parcourt toutes les variables dans un ordre aléatoire. Pour chaque variable, on calcule avec chaque valeur de vérité à quel point cette valeur améliorerait la frontière de décision (B_i avec l'affectation TRUE et B'_i avec l'affectation FALSE). On a alors 4 cas tels que décrits sur le Tableau E.1. On choisit alors la valeur de vérité qui permet d'améliorer le mieux la frontière de décision : dans les cas 2 et 3 le choix est glouton, le choix en cas 1 est un choix par défaut et pour le cas 4, de la même façon que pour les autres problèmes, on réalise un choix aléatoire en donnant plus de poids au choix qui améliorerait localement la valeur de la frontière de décision (B_i ou B'_i). On explore toutes les variables et garde la meilleure valeur dans le cas *gloutonne*. On choisit une valeur parmi un top-k pour chaque *top-K* initialisation. On choisit une valeur suivant une distribution aléatoire favorisant les voisins améliorant le plus la frontière de décision.

TABLEAU E.1 Choix de la valeur de la variable x_i en fonction de l'amélioration de la frontière (t_i si affectation TRUE, f_i si affectation FALSE)

Cas	TRUE	FALSE	Choix x_i
1	$t_i \leq 0$	$f_i \leq 0$	TRUE
2	$t_i > 0$	$f_i \leq 0$	TRUE
3	$t_i \leq 0$	$f_i > 0$	FALSE
4	$t_i > 0$	$f_i > 0$	$Pr(TRUE) = \frac{t_i}{t_i+f_i}$

Algorithm 11 MAXSAT-pondéré initialisation *aléatoire*

Entrées :
— n, le nombre de variables

Outputs :
— A, l'assignement booléen de chaque variable : $a_i \in \{true, false\}$

1: **function** $Initialization_{aléatoire}(P,K)$
2: **for** $i = 1$ **to** n **do**
3: $a_i \leftarrow$ aléatoire($\{true, false\}$)
4: **end for**
5: **return** A
6: **end function**

Algorithm 12 MAXSAT-pondéré initialisation *top-K*

Require:

U, table de hashage avec U_{var_i} ou $U_{\overline{var_i}}$ les id des clauses non fixées contenant var_i

C, table de hashage avec C_i le nombre de littéraux non fixés dans la clause i

V, une liste des indexes des variables

$SAT_0 \leftarrow 0$, la somme des poids des clauses qui sont satisfaites
$UNSAT_0 \leftarrow 0$, la somme des poids des clauses qui sont non satisfaites
W_{tot}, la somme totale des poids de toutes les clauses
$B_0 \leftarrow \frac{W_{tot}}{2}$
while $td.length > 0$ **do** ▷ td : variables dont la valeur doit être décidée
    $topk \leftarrow \emptyset$
    for $var_i \in nonDecidees$ **do**
        $SAT_i \leftarrow SAT_{i-1} + sum(w_c, \forall c \in U_{var_i})$
        $UNSAT_i \leftarrow UNSAT_{i-1} + sum(w_c, \forall c \in U_{\overline{var_i}}, C_c = 1)$
        $t_i \leftarrow \frac{1}{2} \times (SAT_i + W_{tot} - UNSAT_i) - B_{i-1}$
        $SAT'_i \leftarrow SAT_{i-1} + sum(w_c, \forall c \in U_{\overline{var_i}})$
        $UNSAT'_i \leftarrow UNSAT_{i-1} + sum(w_c, \forall c \in U_{var_i}, C_c = 1)$
        $f_i \leftarrow \frac{1}{2} \times (SAT'_i + W_{tot} - UNSAT'_i) - B_{i-1}$
        if $ti > bestImprovement$ **then**
            insérer {var : var_i, impr : -ti, assign : true, found : true} dans topk
        end if
        if $fi > bestImprovement$ **then**
            insérer {var : var_i, impr : -fi, assign : false, found : true} dans topk
        end if
    end for
    if $\neg imprFound$ **then**
        Réaliser un assignement aléatoire et mettre à jour choice
    end if
    $choice \leftarrow choixAléatoireParmi(topk)$
    Supprimer $choice.variable$ de td et mettre à jour SAT_i $UNSAT$ et U
    Décrémenter les littéraux non fixés des clauses où var_i apparaît
    $B_i \leftarrow \frac{1}{2} \times (SAT_i + W_{tot} - UNSAT_i)$
end while
return assignements des variables

Algorithm 13 MAXSAT-pondéré initialisation *gloutonne-randomisée*

Require:

U, table de hashage avec U_{var_i} ou $U_{\overline{var_i}}$ les id des clauses non fixées contenant var_i

C, table de hashage avec C_i le nombre de littéraux non fixés dans la clause i

V, une liste des indexes des variables

$SAT_0 \leftarrow 0$, la somme des poids des clauses qui sont satisfaites

$UNSAT_0 \leftarrow 0$, la somme des poids des clauses qui sont non satisfaites

W_{tot}, la somme totale des poids de toutes les clauses

$B_0 \leftarrow \frac{W_{tot}}{2}$

while $td.length > 0$ **do** ▷ td : variables dont la valeur doit être décidée

  $topk \leftarrow \emptyset$

  for $var_i \in nonDecidees$ **do**

    $SAT_i \leftarrow SAT_{i-1} + sum(w_c, \forall c \in U_{var_i})$

    $UNSAT_i \leftarrow UNSAT_{i-1} + sum(w_c, \forall c \in U_{\overline{var_i}}, C_c = 1)$

    $t_i \leftarrow \frac{1}{2} \times (SAT_i + W_{tot} - UNSAT_i) - B_{i-1}$

    $SAT'_i \leftarrow SAT_{i-1} + sum(w_c, \forall c \in U_{\overline{var_i}})$

    $UNSAT'_i \leftarrow UNSAT_{i-1} + sum(w_c, \forall c \in U_{var_i}, C_c = 1)$

    $f_i \leftarrow \frac{1}{2} \times (SAT'_i + W_{tot} - UNSAT'_i) - B_{i-1}$

    if $ti > bestImprovement$ **then**

      insérer {var : var_i, impr : -ti, assign : true, found : true}

    end if

    if $fi > bestImprovement$ **then**

      insérer {var : var_i, impr : -fi, assign : false, found : true}

    end if

  end for

  if $\neg imprFound$ **then**

    Réaliser un assignement aléatoire et mettre à jour choice

  end if

  $choice \leftarrow choixAléatoireParmi(conteneur)$ avec probabilité dépendant de improvement

  Supprimer $choice.variable$ de td et mettre à jour SAT_i $UNSAT$ et U

  Décrémenter les littéraux non fixés des clauses où var_i apparaît

  $B_i \leftarrow \frac{1}{2} \times (SAT_i + W_{tot} - UNSAT_i)$

end while

return assignements des variables