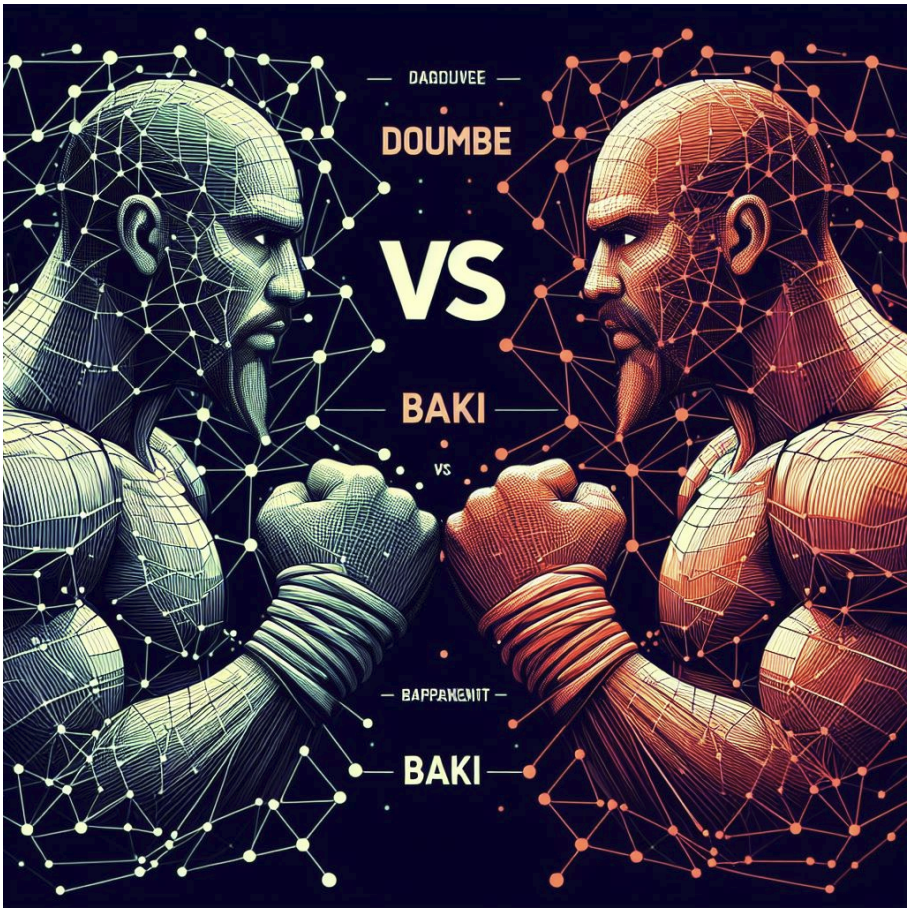
Rapport pour le problème du voyageur de commerce

Alexandre Untereiner Kamelia Slimani

14 Mars 2024



Abstract

Le problème du voyageur de commerce (TSP) est un casse-tête classique en optimisation combinatoire qui a captivé les mathématiciens et les informaticiens depuis des décennies. Il ne s'agit pas seulement d'un défi théorique ; les solutions au TSP ont des applications pratiques dans des domaines variés tels que la logistique, la planification d'itinéraires et même l'organisation de circuits électroniques. Ce rapport explore plusieurs stratégies algorithmiques pour résoudre le TSP, notamment les algorithmes des colonies de fourmis, le recuit simulé, l'algorithme génétique et l'approche 2-OPT. Chaque méthode est évaluée en termes d'efficacité et de praticité, avec une attention particulière portée à leur capacité à trouver des solutions proches de l'optimal dans des temps de calcul raisonnables.

Contents

1 Introduction

Le problème du voyageur de commerce (TSP) consiste à trouver le chemin le plus court qui visite chaque ville d'un ensemble donné une seule fois et retourne à la ville de départ. Malgré sa formulation simple, le TSP est un problème NP-difficile, ce qui signifie qu'il n'existe pas, à ce jour, d'algorithme permettant de résoudre toutes les instances du problème dans un temps polynomial. Ce document présente une étude comparative de plusieurs approches algorithmiques et différente combinaison entre elle pour résoudre le TSP, en mettant en lumière leurs forces et leurs faiblesses.

2 Fonctionnement de l'algorithme des colonies de fourmis

Un des algorithmes qu'on à explorer pour résoudre se problème était le principe des colonies de fourmis, qui s'inspire du comportement des fourmis qui exploitent leur environnement pour trouver la source de nourriture la plus proche. Elles déposent des phéromones sur leur chemin, attirant les autres fourmis vers les solutions prometteuses.

Pour simuler ce comportement, l'algorithme utilise des agents, des phéromones et une heuristique. Les agents sont des "fourmis artificielles" qui explorent l'espace de recherche. Les phéromones sont une information virtuelle qui guide les agents vers les solutions prometteuses, tandis que l'heuristique évalue la qualité d'une solution.

Chaque agent construit une solution en parcourant l'espace de recherche et en choisissant les meilleurs chemins à suivre. La probabilité de choisir un chemin dépend de la quantité de phéromones et de l'heuristique associée à ce chemin. Les agents déposent des phéromones sur les chemins empruntés, indiquant aux autres agents qu'il s'agit d'une solution prometteuse.

Les phéromones s'évaporent progressivement, incitant les agents à explorer de nouvelles solutions. Les chemins des meilleures solutions reçoivent plus de phéromones, les rendant plus attractifs.

La qualité de chaque solution est évaluée en fonction d'un objectif spécifique (distance parcourue, coût, etc.). La meilleure solution est sélectionnée et peut être utilisée comme point de départ pour de nouvelles explorations.

Ce processus de construction, de mise à jour et d'évaluation des solutions est répété jusqu'à ce qu'un critère d'arrêt soit atteint, tel que le temps écoulé ou le nombre d'itérations.

3 Fonctionnement du recuit simulé

Le recuit simulé vise à trouver la configuration présentant une énergie minimale dans un système, similaire à la structure stable d'un métal. Il explore l'espace de recherche en ajustant la configuration actuelle et en évaluant l'énergie de la nouvelle configuration.

Pour démarrer, une configuration aléatoire est définie avec une température élevée. Ensuite, une nouvelle configuration est générée à partir de celle en cours. L'évaluation se fait en calculant la différence d'énergie entre les deux configurations : si la nouvelle configuration a une énergie inférieure, elle est acceptée directement. En revanche, si elle a une énergie supérieure, elle peut être acceptée avec une probabilité qui décroît avec la différence d'énergie et la température.

La température est progressivement réduite selon un plan défini. Ce processus est répété jusqu'à ce que la température atteigne un seuil minimal. Cette répétition permet au recuit simulé d'explorer progressivement l'espace de recherche, même en acceptant des configurations de haute énergie. Cela lui permet d'éviter les minima locaux et de converger vers la solution optimale.

4 Fonctionnement de l'algorithme génétique

Un autre algorithme tester, était basé sur algorithme les génétique, opérant sur une population de solutions potentielles, représentées sous forme de chromosomes. Chaque chromosome correspond à une solution possible au problème donné.

Le processus débute par la création d'une population initiale de chromosomes, généralement de manière aléatoire. Ces chromosomes sont ensuite évalués en fonction d'une fonction objectif, qui mesure leur adéquation à résoudre le problème.

Ensuite, les chromosomes sont soumis à des opérateurs génétiques tels que la sélection, le croisement et la mutation. Ces opérateurs simulent les processus de reproduction et de mutation présents dans la nature.

La sélection favorise les chromosomes les plus performants, en leur permettant de se reproduire et de transmettre leurs caractéristiques à la génération suivante. Le croisement mélange les caractéristiques de deux chromosomes parents pour créer de nouveaux individus. La mutation introduit de petites variations aléatoires dans les chromosomes, permettant d'explorer de nouveaux espaces de recherche.

Ce processus de reproduction, de croisement et de mutation est répété sur plusieurs générations, chaque fois en favorisant les individus les mieux adaptés. Cela conduit à une évolution de la population vers des solutions de plus en plus performantes.

L'algorithme génétique atteint son critère d'arrêt lorsque la population converge vers une solution satisfaisante ou lorsque le nombre maximal de générations est atteint.

5 Fonctionnement de l'algorithme 2-OPT

Une solution courante pour optimiser un problème issu du voyageur de commerce (TSP), est l'algorithme 2-opt, une méthode simple et efficace pour optimiser un itinéraire dans le contexte du problème du voyageur de commerce (TSP). Son objectif est de trouver le chemin le plus court possible pour visiter un ensemble de villes données.

L'algorithme fonctionne en explorant le voisinage d'une solution et en recherchant des améliorations locales. Il échange des paires d'arêtes dans l'itinéraire et conserve les modifications qui réduisent la distance totale parcourue.

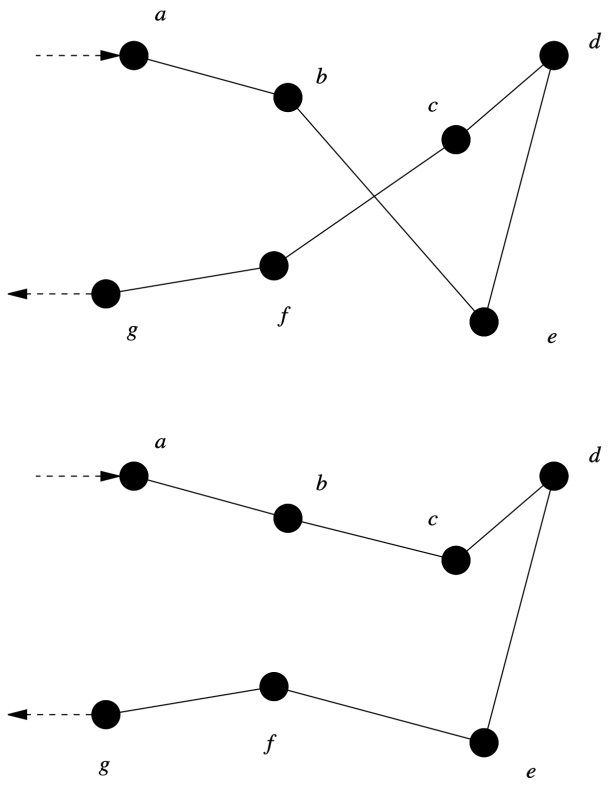


Figure 1: l'algorithme 2-OPT

Mais en plus de 2-opt, nous avons développé l'algorithme 1-opt, inspiré de la méthode 2-opt. Alors que l'algorithme 2-opt échange des paires d'arêtes pour améliorer la solution globale, l'opérateur 1-opt se concentre sur l'échange d'une seule ville à la fois, permettant une exploration plus fine du voisinage de la solution actuelle.

Cette approche est particulièrement utile lorsque la solution est déjà relativement bonne et que des améliorations marginales sont recherchées. 1-opt peut identifier des améliorations locales qui pourraient être négligées par une recherche plus large et moins ciblée comme celle effectuée par l'algorithme 2-opt.

Ceci est le résultat d'une analyse approfondie des forces et des limites de l'algorithme 2-opt. En nous appuyant sur le principe de l'échange d'arêtes, nous avons conçu un opérateur capable de raffiner bien plus les solutions, offrant ainsi une nouvelle dimension d'optimisation pour les problèmes de TSP.

6 Développement

6.1 DoumbeEstropier

Au cours de nos recherches sur les méthodes d'optimisation pour le problème du voyageur de commerce (TSP), nous avons exploré diverses stratégies avant de nous arrêter sur une qui a particulièrement retenu notre attention : l'algorithme Fourmi/2-opt–Génétique–Recuit qu'on va appeler DoumbeEstropier dans notre version. Cette méthode hybride associe les principes de l'algorithme des colonies de fourmis (ACO), de l'algorithme génétique (AG) et du recuit simulé (SA), exploitant ainsi les avantages de chaque approche pour améliorer la recherche de solutions optimales.

Intrigués par les résultats prometteurs rapportés dans sur un article scientifique, nous avons décidé d'essayer mettre en œuvre cette technique au sein de notre propre cadre de recherche. Notre objectif était de comprendre en profondeur les mécanismes et de vérifier si les performances observées pouvaient être reproduites et éventuellement améliorées.

Nous utilisons l'ACO pour explorer l'espace de recherche en simulant le comportement des fourmis, qui déposent des phéromones sur leur chemin pour attirer les autres fourmis vers des solutions prometteuses. Les phéromones sont mises à jour en fonction des chemins parcourus par les fourmis, favorisant ainsi les chemins les plus courts.

En parallèle, nous créons une population initiale de solutions potentielles représentées sous forme de chromosomes dans l'algorithme génétique. Ces chromosomes subissent des opérations de sélection, de croisement et de mutation, simulant ainsi les processus de reproduction et de variation présents dans la nature. Cela nous permet de rechercher des solutions optimales à travers l'évolution de la population.

Enfin, nous utilisons le recuit simulé pour affiner la solution obtenue par l'AG ou l'ACO. Cette méthode explore de nouveaux voisins de la solution actuelle en acceptant parfois des solutions de moins bonne qualité pour éviter de rester bloqué dans un minimum local.

Notre processus global consiste à exécuter l'ACO pour obtenir des chemins initiaux, puis à les améliorer à l'aide de l'AG. Ensuite, nous appliquons le SA pour raffiner davantage la solution finale. Ce processus est répété jusqu'à ce que nous trouvions une solution satisfaisante ou qu'un critère d'arrêt soit atteint.

6.1.1 Explication du code

- initialization : initialise les paramètres de l'algorithme et les phéromones.
- loop : boucle principale qui exécute les étapes de l'algorithme (ACO, GA, SA).
- updatePheromones : met à jour les phéromones en fonction de la qualité des chemins trouvés par les fourmis.
- calculateDistance : calcule la distance entre deux villes.
- calculateHeuristic : calcule une valeur heuristique basée sur la distance (ici, l'inverse de la distance).
- selectNextCity : choisit la prochaine ville à visiter pour une fourmi en tenant compte des phéromones et de l'heuristique.
- constructAntPath : construit un chemin complet pour une fourmi.
- Fonctions de l'algorithme génétique :
  - greedy: génère un chemin initial par construction gloutonne.
  - generateInitialPopulation : crée une population initiale de chemins.
  - two_opt : améliore un chemin en inversant un segment.
  - selection : sélectionne des parents pour la reproduction basée sur un tournoi.
  - crossover : crée un nouvel individu en combinant des parties des parents.
  - mutation : mute aléatoirement des chemins de la population.
- Fonctions du recuit simulé :
  - simulatedAnnealing : recherche un meilleur minimum en explorant le voisinage de la solution courante.
  - getNeighbor : génère une solution voisine en échangeant deux villes.

6.1.2 Résultats

Pour évaluer l'efficacité de notre algorithme DoumbeEstropier, nous avons effectué des tests comparatifs sur différentes instances du problème du voyageur de commerce (TSP). Chaque algorithme a été exécuté 10 fois sur chaque instance pour assurer la fiabilité des résultats. Les critères suivants ont été utilisés pour l'évaluation :

- **Valeur Min :** La plus courte distance trouvée par l'algorithme.
- **Valeur Max :** La plus longue distance trouvée par l'algorithme.
- **Moyenne :** La distance moyenne obtenue sur les 10 essais.
- **Optimum :** La meilleure distance connue pour l'instance, servant de référence pour évaluer l'efficacité.
- **Efficacité :** Calculée en comparant la moyenne des distances obtenues par l'algorithme avec l'optimum connu. Elle est exprimée en pourcentage et reflète la proximité de l'algorithme à l'optimum.

Table 1: Test de DoumbeEstropier face à d'autre algorithme sur différentes instances 10 fois

Algorithme	Instance	Valeur Min	Valeur Max	Moyenne	Optimum	Efficacité
RandomSearch	bier127	546556	559917	553098	≈ 118282	≈ 21%
	bier127corriger	526772	532957	529546	118282	23%
	gr666	53293	54107	53731	3062	5%
	pr8192	2.171553e+7	2.182841e+7	2.177054e+7	16382	0%
greedy	bier127	141057	141057	141057	≈ 118282	≈ 83%
	bier127corriger	133971	133971	133971	118282	88%
	gr666	3835	3835	3835	3062	79%
	pr8192	20476	20476	20476	16382	80%
DoumbeEstropier	bier127	135934	144185	139535	≈ 118282	≈ 86%
	bier127corriger	125130	129760	127209	118282	94%
	gr666	4327	5455	5052	3062	70%
	pr8192	4008307	NA	NA	16382	0%

RandomSearch:

- Pour l'instance *bier127*, il a trouvé des valeurs allant de 546,556 à 559,917 avec une moyenne de 553,098, ce qui représente une efficacité d'environ 21% par rapport à l'optimum estimé.
- Pour *bier127corriger*, les valeurs étaient légèrement meilleures, allant de 526,772 à 532,957 avec une moyenne de 529,546 et une efficacité de 23%.
- L'efficacité chute dramatiquement pour *gr666* à 5%, avec des valeurs moyennes autour de 53,731.
- Pour *pr8192*, cet algorithme n'a pas été efficace, avec une efficacité de 0%.

greedy:

- A obtenu des résultats constants pour *bier127* et *bier127corriger* avec des efficacités d'environ 83% et 88% respectivement.
- Pour *gr666* et *pr8192*, l'efficacité était respectivement de 79% et 80%.

DoumbeEstropier:

- A montré une bonne performance pour *bier127* avec une efficacité d'environ 86%.
- Pour *bier127corriger*, il a atteint une efficacité de 94%.
- Pour *gr666*, l'efficacité était de 70%.
- Aucune donnée n'était disponible pour *pr8192* suite à l'arrêt du programme.

Il est intéressant de constater, d'après les resultat, que le chemin obtenu pour le bier127 par notre méthode est meilleur que celui produit par l'algorithme greedy, lui même bien meilleur que des chemins générer aléatoirement. Cependant, malgré cette amélioration du chemin, le tableau montre que l'évaluation de notre méthode reste en générale inférieure à celle de l'algorithme greedy. Dans le cadre de notre projet sur l'algorithme hybride pour le TSP, il est possible que nous ayons commis certaines erreurs. Voici quelques-unes des erreurs potentielles que nous aurions pu faire :

Nous avons peut-être sous-estimé la complexité d'intégrer trois méthodes différentes, ce qui a pu conduire à des erreurs dans le réglage des paramètres ou dans la logique de l'algorithme.

- Équilibrage des méthodes Il est possible que nous n'ayons pas trouvé le bon équilibre entre l'ACO, l'AG et le SA, ce qui pourrait avoir limité la synergie entre ces approches et affecté les performances globales.
- Coût computationnel Nous aurions pu négliger l'impact du coût computationnel de l'utilisation simultanée de plusieurs algorithmes, ce qui a pu entraîner des temps de calcul plus longs que prévu.
- Sur-optimisation Nous avons pu trop nous concentrer sur les instances testées, ce qui a pu mener à une sur-optimisation et à une généralisation insuffisante de l'algorithme.
- Convergence prématurée Malgré nos efforts, il est possible que nous n'ayons pas complètement évité la convergence vers des optima locaux, limitant ainsi la qualité des solutions trouvées.

Ces erreurs potentielles sont des points d'apprentissage importants pour nous. Elles soulignent la nécessité d'une analyse rigoureuse. C'est pourquoi nous avons décider de tester une autre méthode assez similaire, mais en repartant sur des bases plus saines.

6.2 l'Algorithme Génétique *BakiTKO*

L'algorithme génétique est une méthode de résolution du problème du voyageur de commerce (TSP) basée sur des principes inspirés de la théorie de l'évolution, ici on va appeler cette algorithme *BakiTKO* comme notre objectif et de réussir à battre *DoumbeEstropier*.

6.2.1 Notre Méthodologie de Développement

Début avec un algorithme génétique aléatoire Nous avons commencé par créer une population initiale de chemins de manière totalement aléatoire, sans aucune heuristique pour guider la sélection des villes. Les chemins générés étaient très variés, mais la qualité globale des solutions était médiocre. Les chemins n'étaient pas optimisés et présentaient souvent des détours inutiles, ce qui entraînait des distances totales significativement plus longues que nécessaire.

Croisement aléatoire Nous avons utilisé un croisement aléatoire pour combiner les chemins. Bien que le croisement aléatoire ait introduit une certaine diversité dans la population, il n'a pas conduit à une amélioration notable des chemins. Cela suggère que le croisement aléatoire seul n'est pas suffisant pour guider l'algorithme vers de meilleures solutions.

Introduction des mutations L'introduction des mutations avait pour but d'explorer de nouvelles régions de l'espace de recherche. Cependant, les mutations aléatoires n'ont pas réussi à produire des améliorations significatives, indiquant que sans une stratégie de mutation plus ciblée, les chances de trouver de meilleures solutions restent faibles. Malheureusement, cela n'a pas eu l'impact positif attendu sur la qualité des solutions.

Initialisation avec l'algorithme greedy Pour améliorer les résultats de départ, nous avons décidé d'initialiser la population avec des chemins générés par l'algorithme greedy. L'utilisation de cette l'algorithme glouton pour initialiser les chemins a permis de partir sur de meilleures bases. Les chemins étaient plus courts et plus cohérents, mais ils n'étaient pas nécessairement optimaux. Les algorithme glouton ont tendance à se focaliser sur des gains à court terme, ce qui peut conduire à négliger des solutions meilleures à long terme.

Optimisation des chemins Enfin, nous avons ajouté des méthodes d'optimisation pour affiner les chemins. Ces méthodes, telles que 2-opt, ont permis de corriger les erreurs et d'améliorer la qualité des solutions et de corriger certains des défauts des chemins, en éliminant les croisements et en raccourcissant les distances. Et grâce à toutes les phases d'expérimentation réalisées précédemment, nous avons réussi à diminuer fortement les risques de converger vers des optima locaux plutôt que globaux.

6.2.2 Méthode utilisé

Le développement de notre algorithme génétique, intégrant 2-opt et d'autre algorithme, a suivi une série d'étapes méthodiques pour résoudre le problème du voyageur de commerce (TSP). Ce processus a été guidé par les principes de la théorie de l'évolution et des heuristiques d'optimisation.

- Génération de la Population Initiale: Une population initiale de chemins est générée aléatoirement et améliorer grâce à greedy et 2-opt, utilisé pour améliorer les chemins en échangeant deux arêtes, dans le but de réduire la longueur totale du chemin. Avec un nombre de population de 20 individu, pour maximiser le nombre de génération comme les opérateur de croisement utilisé par la suite peuvent être assez long sur une trop grande taille de la population initial.
- Évaluation de la Qualité: Chaque chemin est évalué selon sa longueur totale, avec une préférence pour les chemins plus courts et trié en fonction.
- Sélection et Croisement: Les meilleurs chemins sont sélectionnés pour le croisement entre les 25% à 50% meilleurs, de façon aléatoires, où des parties de deux chemins parentaux sont échangées pour créer de nouveaux chemins, appelés enfants.
- Mutation: Après avoir appliqué le croisement, une mutation fixer à 50% de manière arbitraire suite différente expérimentation, qui peut être appliquée à certains enfant générer, dont la mutation consiste généralement à apporter de petites modifications aléatoires aux chemins pour explorer de nouvelles solutions potentielles. On à tester diffèrent modèle de mutation comme des inversion entre des villes, des décalage et d'échange avec de multiple combinaison possible entre ces mutation. on a finalement garder deux mutation le décalage et d'échange de ville dans la versions final.
- Diversification et Évitement des Optimums Locaux: La méthode de croisement et de mutation permet de reproduire des chemins existants et d'en créer de nouveaux, augmentant la diversité et évitant la stagnation des solutions dans des optimum locaux.
- Optimisation avec 2-opt et 1-opt: Après les mutation, les chemins enfants sont optimisés en utilisant d'abord 2-opt, puis 1-opt, une nouvelle fonction qui corrige les chemins non optimisés pour chaque ville individuellement.
- Sélection pour la Génération Suivante: Les meilleurs chemins, y compris les parents et les enfants, sont trié pour etre sélectionnés à la génération suivante.
- Répétition du Processus: Les étapes précédentes sont répétées sur plusieurs générations jusqu'à atteindre la fin du temps impartie.

6.2.3 Résultats

La même méthodologie de test à etait appliquée pour évaluer l'efficacité de notre algorithme BakiTKO, elle est identique à celle utilisée pour les autres algorithmes testés.

Table 2: Performance des algorithmes sur différentes instances de test 10 fois

Algorithme	Instance	Valeur Min	Valeur Max	Moyenne	Optimum	Efficacité
HillClimbing	bier127	611658	645852	632724	≈ 118282	≈ 32%
	bier127corriger	589133	633734	609805	118282	5%
	gr666	55945	59407	57973	3062	19%
	pr8192	2.215967e+7	2.244925e+7	2.229529e+7	16382	0%
BakiTKO	bier127	125999	126649	126115	≈ 118282	≈ 94%
	bier127corriger	118294	118294	118294	118282	100%
	gr666	3139	3178	3157	3062	97%
	pr8192	16382	16382	16382	16382	100%
greedy	bier127	141057	141057	141057	≈ 118282	≈ 83%
	bier127corriger	133971	133971	133971	118282	88%
	gr666	3835	3835	3835	3062	79%
	pr8192	20476	20476	20476	16382	80%
RandomWalk	bier127	551954	571108	562049	≈ 118282	≈ 21%
	bier127corriger	495992	546115	527774	118282	23%
	gr666	54262	55020	54636	3062	5%
	pr8192	2.192137e+7	2.228724e+7	2.211248e+7	16382	0%

HillClimbing:

- A montré une légère amélioration pour *bier127* avec des valeurs allant de 611,658 à 645,852 et une moyenne de 632,724, soit une efficacité d'environ 32%.
- Cependant, pour *bier127corriger*, l'efficacité a chuté à 5%.
- Pour *gr666*, l'efficacité était de 19%.
- Comme RandomSearch, HillClimbing n'a pas été efficace pour *pr8192*.

BakiTKO:

- A excellé sur toutes les instances testées, atteignant l'optimum pour *bier127corriger* et *pr8192* avec une efficacité de 100%.
- Pour *bier127*, il a atteint une efficacité d'environ 94% avec des valeurs très proches de l'optimum.
- Pour *gr666*, il a également performé remarquablement bien avec une efficacité de 97%.

RandomWalk:

- A eu des performances comparables à RandomSearch pour *bier127* et *bier127corriger* avec des efficacités d'environ 21% et 23%.
- Pour *gr666* et *pr8192*, comme les autres algorithmes non-optimisés, il a montré une efficacité de 5% et 0% respectivement.

Résumé des résultats

L'algorithme *BakiTKO* s'est avéré être le plus efficace, atteignant ou se rapprochant de l'optimum dans toutes les instances testées. Les autres algorithmes ont montré des efficacités variables, avec *greedy* et *DoumbeEstropier* offrant des performances relativement bonnes pour certaines instances. Mais BakiTKO reste invaincu.

7 Améliorations Futures

Malgré les très bon résultat, l'efficacité de notre algorithme génétique pour le problème du voyageur de commerce (TSP) peut être améliorée de plusieurs manières. Voici les stratégies potentielles pour renforcer la performance de l'algorithme que nous avons pas eux le temps d'explorer.

- Hybridation avec d'autres heuristiques L'intégration d'autres heuristiques ou algorithmes d'optimisation, tels que le recuit simulé ou la recherche tabou, pourrait améliorer significativement les résultats obtenus.
- Optimisation multi-objectif En adaptant l'algorithme pour considérer plusieurs critères de performance simultanément, nous pourrions obtenir des solutions plus complètes et applicables à des problèmes plus complexes.
- Parallélisation Même si dans le cadre de notre projet, il nous à était interdit de mettre en place de la parallélisation. La mise en œuvre d'une version parallèle de l'algorithme pourrait réduire le temps de calcul nécessaire, particulièrement pour les grandes instances du TSP.
- Amélioration des opérateurs génétiques Le développement de meilleur stratégies de croisement et de mutation plus avancées pourrait augmenter la diversité génétique tout en préservant la qualité des solutions.
- Évaluation par apprentissage automatique Une dernier hypothèse est l'incorporation de modèles d'apprentissage automatique pour évaluer la qualité des solutions, cela pourrait guider l'algorithme dans la sélection des chemins les plus prometteurs.

8 Conclusion

L'algorithme génétique *BakiTKO* se distingue comme l'une des approches les plus rentables et optimales pour résoudre le problème du voyageur de commerce (TSP) dans diverses catégories.

L'utilisation de l'algorithme génétique garantit une diversité génétique dans la population de chemins, ce qui permet d'éviter la convergence prématurée vers des optimums locaux et d'explorer efficacement l'espace des solutions.

L'algorithme génétique avec 2-opt combiné avec d'autres méthodes est capable de gérer efficacement des instances de problèmes de différentes tailles, offrant ainsi une solution robuste et adaptable à diverses situations.

L'approche peut être encore améliorée et personnalisée par l'intégration d'autres opérateurs génétiques ou techniques d'optimisation, ce qui la rend applicable à un spectre encore plus large de problématiques et de conditions spécifiques. Les perspectives d'amélioration sont vastes et prometteuses.

Ce projet a été enrichissant à plusieurs égards. Il a stimulé notre capacité à résoudre des problèmes complexes, a renforcé notre compréhension des stratégies d'optimisation et a affûté nos compétences en programmation. Ce travail a également favorisé une approche innovante et créative dans la recherche de solutions optimales, tout en soulignant l'importance de la persévérance face aux défis techniques.

9 Références

- Fourmi/2-opt–Génétique–Recuit : `https://docplayer.fr/2653298-Resolution-du-probleme-du-voyageur-de-commerce-metaheuristique.html`
- Recuit simulé : `https://fr.wikipedia.org/wiki/Recuit_simul`
- Colonies de fourmis : `https://fr.wikipedia.org/wiki/Algorithme_de_colonies_de_fourmis#:~:text=Les%20algorithmes%20de%20colonies%20de,par%20Marco%20Dorigo%20et%20al.`
- Génétique : `https://fr.wikipedia.org/wiki/Algorithme_gntique`
- 2-OPT : `https://fr.wikipedia.org/wiki/2-opt#:~:text=2%2Dopt%20est%20un%20algorithme%20itratif%20%3A%20%20chaque%20tape%2C,voir%20figure%20ci%2Dcontre).`