POLYTECH©
LYON

Conception de métaheuristiques d'optimisation combinatoire.
Application aux problèmes du sac à dos.

Étudiants :

Adam KHARFI
Abdelhak ASSAHI

Intervenant :

Stéphane BONNEVAY

Table des Matières

Introduction

Dans le cadre du cours d'Optimisation Discrète, enseigné par M. Stéphane BONNEVAY, nous avons été amenés à implémenter des métaheuristiques pour résoudre divers problèmes. Ici, le problème est celui du sac à dos; il s'agit de sélectionner, parmi un ensemble d'objets, ceux à inclure dans un sac de capacité limitée de manière à maximiser la somme des valeurs des objets choisis.

Ce projet a pour objectif de comparer différentes approches d'optimisation, notamment les Algorithmes Génétiques et la Recherche Tabou, en fonction de plusieurs paramètres (taille de population, taux de mutation, taille de la liste tabou, nombre d'itérations, etc...). Nous avons aussi évoqué l'approche exacte via la programmation linéaire PL (OR-Tools) en tant que référence.

Les algorithmes ont été réalisé en python dans un notebook que vous pouvez retrouver ici :

https://github.com/KHARFIAdam/Knapsack_Opt

Dans ce dépôt, vous avez :

- Data.zip (les données zippées)
- Data/* (Un répertoire Data avec tous les fichiers contenant les données utilisées)
- ResultPL/* (Répertoire avec tous les résultats graphes et excel du PL)
- Results/* (Idem pour Algorithme Génétique)
- ResultsTabu (Idem pour Tabou)
- Knapsack.ipynb (Notebook des scripts métaheuristique en python)
- PL_Knapsack.py (Script modélisation en PL du problème du sac à dos)
- Projet_Knapsack.pdf (La consigne du projet)
- README.md (Explication bref des algorithmes et de leurs fonctionnements algorithmiques)

1. Métaheuristique 1 : Algorithmes Génétiques

L'algorithme génétique est une métaheuristique inspirée du processus de sélection naturelle et de la théorie de l'évolution biologique. Il appartient à la famille des algorithmes évolutionnaires et permet d'obtenir des solutions approchées à des problèmes d'optimisation complexes, tels que le problème du sac à dos pour notre cas.

Principe

L'algorithme génétique fonctionne en simulant un processus d'évolution sur une population initiale de solutions candidates, appelées "individus". Chaque individu représente une solution potentielle au problème considéré. Le processus évolutif se décompose en plusieurs étapes clés :

1. **Initialisation** : Une population initiale d'individus est générée de manière aléatoire.
2. **Évaluation** : Chaque individu est évalué à l'aide d'une fonction d'aptitude (fitness), déterminant sa qualité par rapport au problème posé.
3. **Sélection** : Les individus les mieux évalués sont sélectionnés pour participer à la création d'une nouvelle génération.
4. **Croisement** : Deux individus sélectionnés échangent une partie de leurs informations pour générer de nouveaux individus (qu'on appelle les enfant)
5. **Mutation** : Des modifications aléatoires sont apportées à certains individus afin de préserver la diversité génétique de la population et d'éviter la convergence prématurée vers un optimum local.
6. **Remplacement** : La nouvelle génération remplace l'ancienne, et le processus recommence à partir de l'étape d'évaluation jusqu'à atteindre un critère d'arrêt (nombre de générations fixé, temps d'exécution, stagnation, etc…)

Paramètres implémentés

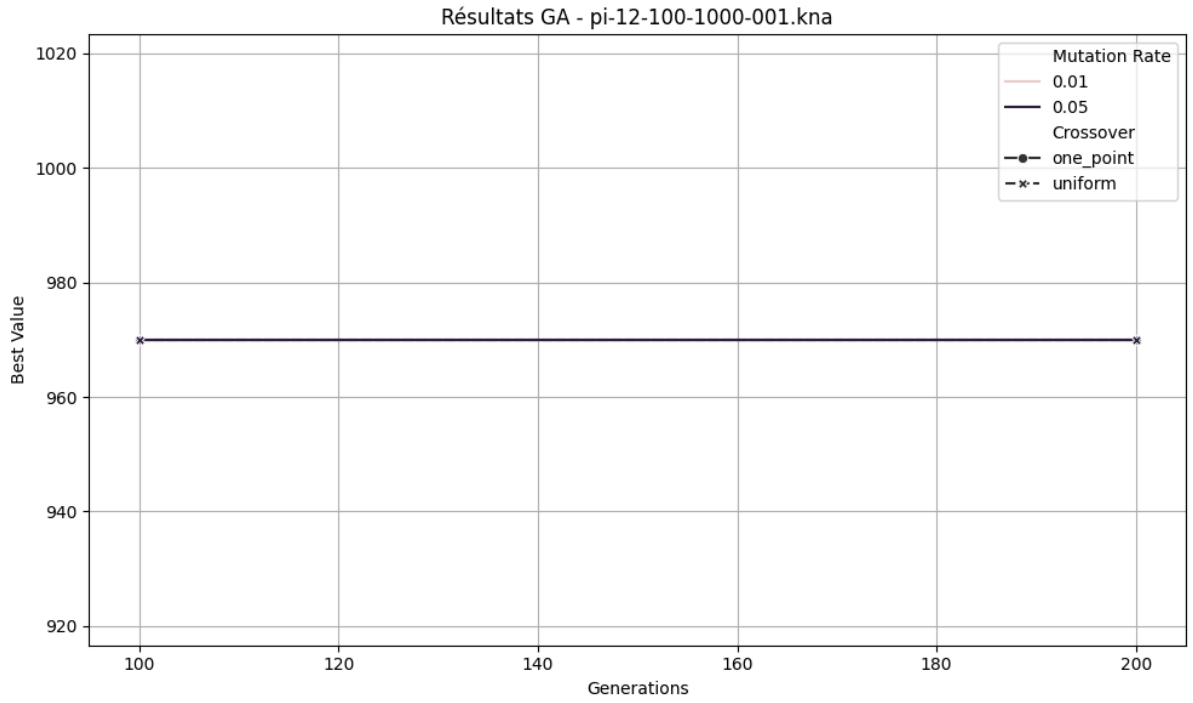
Dans le cadre de ce projet, plusieurs paramètres clés ont été implémentés et testés afin d'étudier leur impact sur les performances de cette métaheuristique qu'est l'algorithme génétique :

- **Taille de la population** : Nombre d'individus générés et évalués à chaque génération.
- **Nombre de générations** : Nombre d'itérations du processus évolutionnaire.

- **Taux de mutation** : Probabilité d'application d'une mutation à un individu.
- **Type de croisement** : Méthode utilisée pour combiner deux individus parents en nouveaux individus enfants :
    - **Croisement à un point** (one_point) : échange d'information à partir d'un point de croisement unique.
    - **Croisement uniforme** (uniform) : chaque élément a une probabilité égale de provenir de l'un ou l'autre des parents.
- **Type de mutation** (flip_bit) : inversion d'un bit dans la représentation binaire de l'individu.

a. Expérimentations et Résultats (Datasets de pi-12)

1. pi-12-100-1000-001



L'optimum obtenue est de **970**, atteinte avec les paramètres suivants :

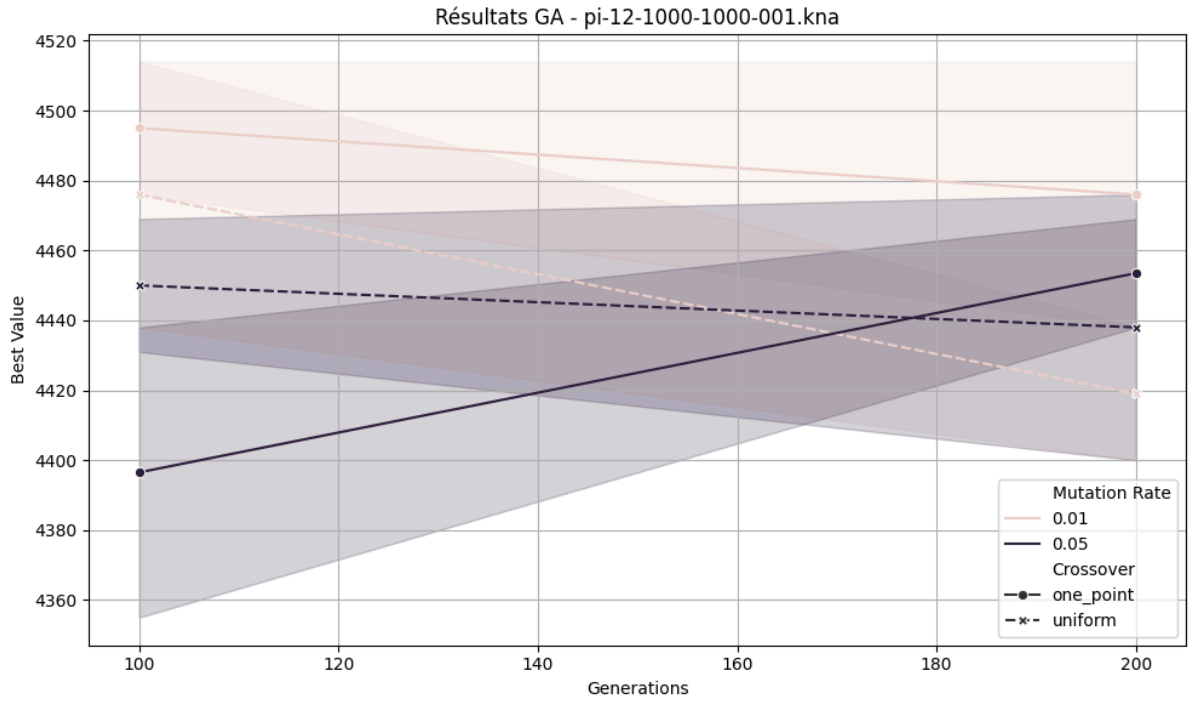
Taille de population	Nombre de génération	Taux de mutation	Type de croisement
30	100	1% (0.01)	one_point

Avec des temps moyens observés :

Croisement à un point (taux de mutation 0.01)	**Croisement uniforme (taux de mutation 0.01)**	**Croisement à un point (taux de mutation 0.01)**	**Croisement à un point (taux de mutation 0.05)**
0.134s	0.324s	0.156s	0.293s

Nous pouvons remarquer que peu importe le nombre de génération, la taille de la population, le taux de mutation et le type de croisement, nous trouvons immédiatement l'optimum qui est **970**. Néanmoins, le *croisement à un point* couplé à un *faible taux de mutation* montre une efficacité supérieure, combinant une excellente qualité de solution et un temps d'exécution réduit

2. pi-12-1000-1000-001



La meilleure solution obtenue présente une valeur de **4514**, atteinte avec les paramètres suivants :

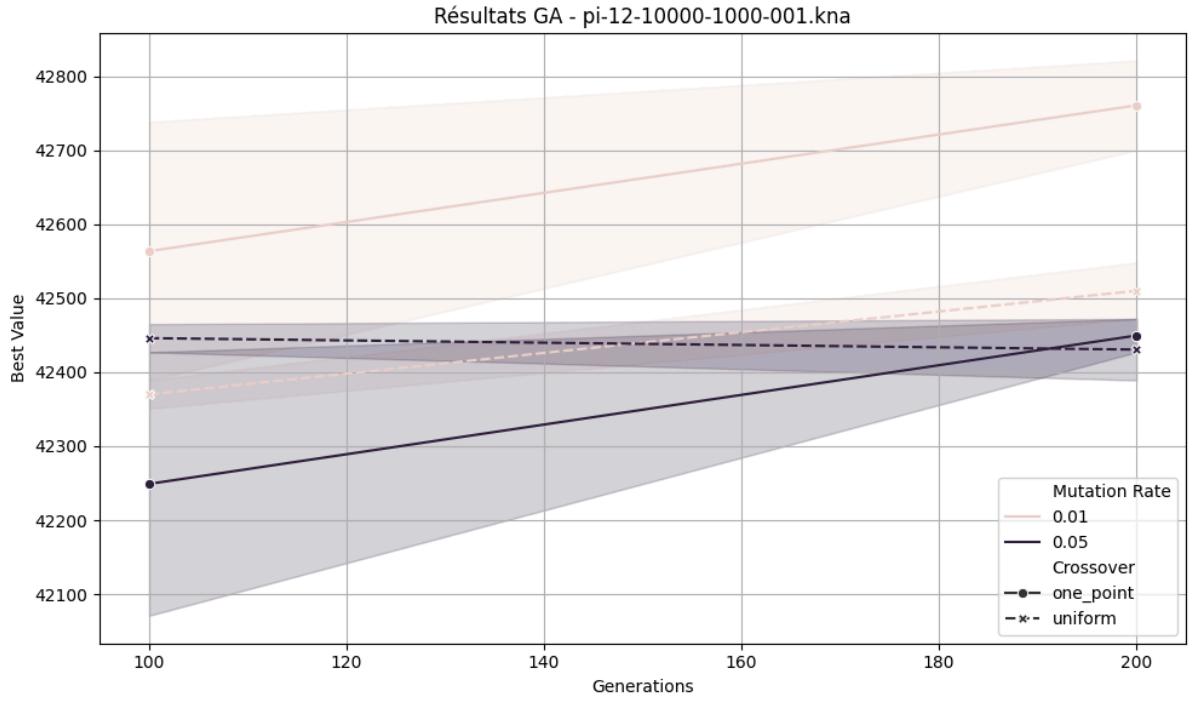
Taille de population	**Nombre de générations**	**Taux de mutation**	**Type de croisement**
30	100	1% (0.01)	one_point

Les temps moyens observés :

Croisement à un point (taux de mutation 0.01)	**Croisement uniforme (taux de mutation 0.05)**	**Croisement à un point (taux de mutation 0.01)**	**Croisement uniforme (taux de mutation 0.05)**
1.601s	2.662s	1.684s	2.607s

Comme le cas précédent, le croisement à un point combiné à un faible taux de mutation montre une efficacité optimale. Les performances inférieures observées avec un croisement uniforme ou un taux de mutation plus élevé proviennent probablement d'une trop grande perturbation génétique, empêchant l'algorithme de converger rapidement vers une solution optimale.

3. pi-12-10000-1000-001



Pour ce jeu de données, la meilleure solution obtenue présente une valeur de **42821**. Les paramètres utilisés pour atteindre cette valeur sont les suivants :

Taille de population	**Nombre de génération**	**Taux de mutation**	**Type de croisement**
50	200	1% (0.01)	one_point

Temps moyens observés :

Taux de mutation	Type de croisement	Temps d'exécution
0.01	one_point	19.612s
0.01	uniform	28.141s
0.05	one_point	19.015s
0.05	uniform	28.874s

Le croisement uniforme présente systématiquement des temps d'exécution beaucoup plus élevés que le croisement à un point. Ce résultat est logique, car le croisement uniforme nécessite plus d'opérations et génère une diversité génétique accrue, ce qui augmente le temps nécessaire à la convergence.

Le croisement à un point avec un taux de mutation faible offre une excellente qualité de solution, en adéquation avec les résultats obtenus pour des tailles inférieures de jeux de données (100 et 1000 items). Ce qui est intéressant à noter, le croisement à un point avec un taux de mutation à 5% obtient un temps d'exécution moyen légèrement inférieur, probablement grâce à une exploration plus rapide de l'espace des solutions. Néanmoins, la meilleure qualité de solution reste atteinte avec un taux de mutation à 1%.

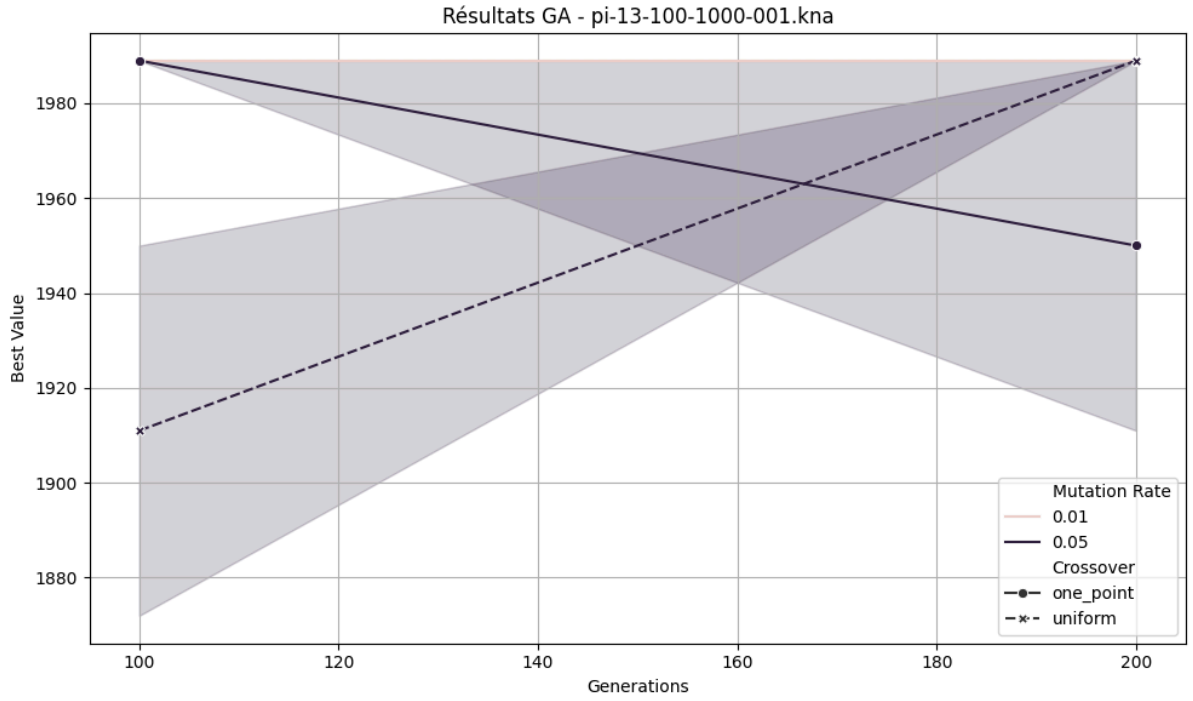
Conclusion des tests réalisés sur les datasets à pi-12 :

Ainsi, il semble préférable de privilégier la qualité des résultats en maintenant un taux de mutation faible, qui a prouvé son efficacité constante à travers tous les jeux de données pi-12.

Les configurations associant un croisement uniforme à des taux de mutation plus élevés montrent une dégradation significative des performances. Cette dégradation peut s'expliquer par un effet combiné d'une trop forte diversification génétique. Cette diversification excessive rend difficile la convergence vers une solution optimale, car les bonnes solutions existantes risquent d'être constamment perturbées. La grande taille du jeu de données (1000 items) accentue probablement les effets négatifs d'une trop grande variabilité génétique, les perturbations trop fréquentes ralentissent la convergence et augmentent significativement les temps de calcul sans bénéfices nets sur la qualité des solutions finales.

b. Expérimentations et Résultats (Datasets de pi-13)

1. pi-13-100-1000-001



La meilleur solution obtenue est de **1989**, atteinte avec les paramètres suivants :

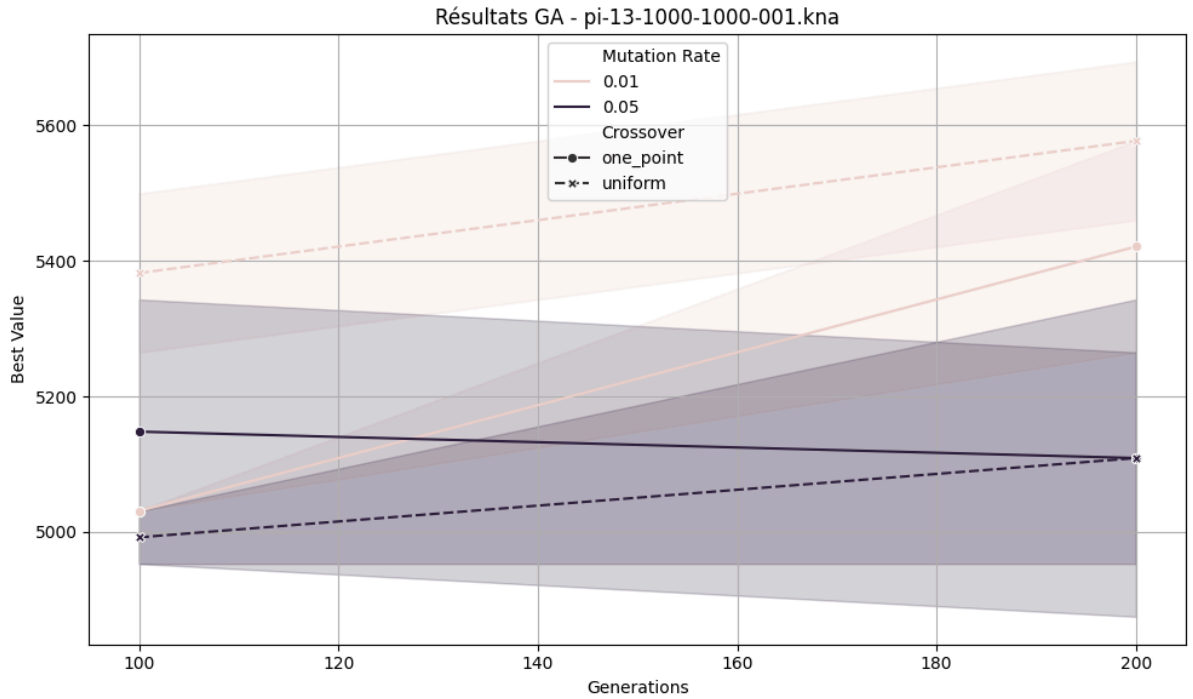
Taille de la population	Nombre de générations	Taux de mutation	Type de croisement
30	100	1%	one_point

Les temps moyens observés :

Taux de mutation	Type de croisement	Temps moyen d'exécution
0.01	one_point	0.162s
0.01	uniform	0.389s
0.05	one_point	0.187s
0.05	uniform	0.349s

La supériorité nette du croisement à un point est à nouveau observée en matière d'efficacité temporelle. Le croisement uniforme reste significativement plus lent. Un taux de mutation faible assure une bonne stabilité et des résultats optimaux.

2. pi-13-1000-1000-001



L'optimum obtenue : **5694**, avec une configuration différente des précédentes :

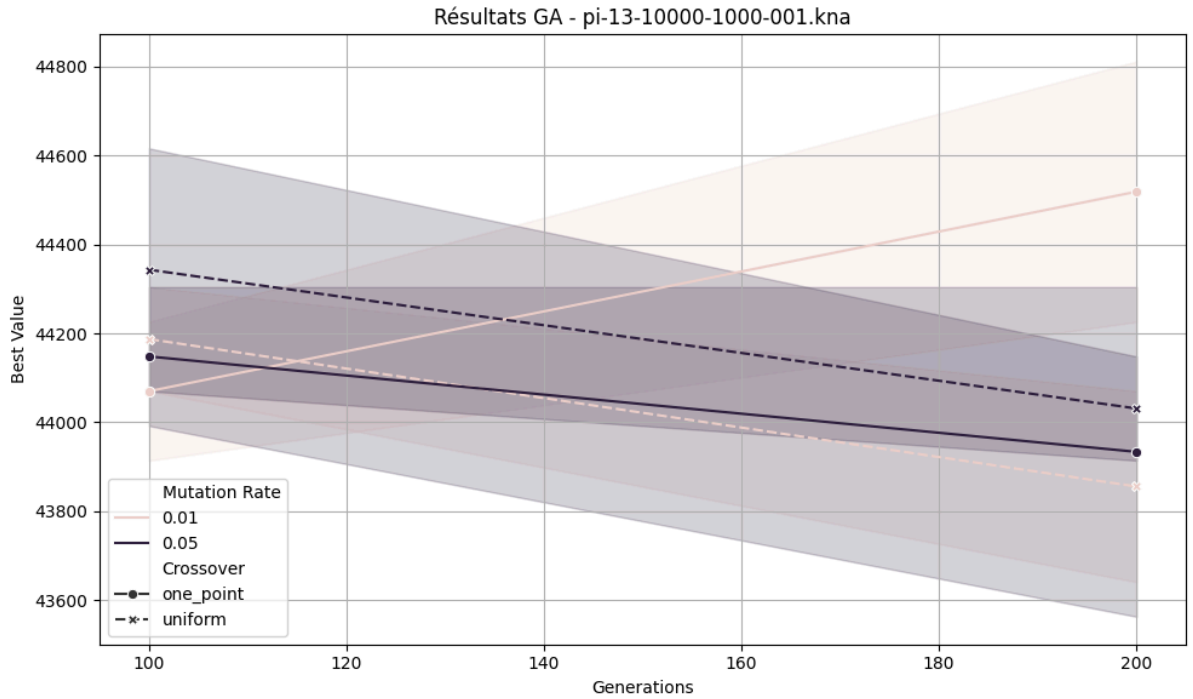
Taille de population	Nombre de générations	Taux de mutation	Type de croisement
50	200	1%	uniform

Temps moyens observés :

Taux de mutation	Type de croisement	Temps moyen d'exécution
0.01	one_point	1.763s
0.01	uniform	3.190s
0.05	one_point	1.907s
0.05	uniform	2.932s

Ici nous avons un résultat atypique : la meilleure solution provient d'un croisement uniforme, ce qui contraste avec les résultats précédentsQ. Cela pourrait suggérer une particularité structurelle du datset pi-13-1000 favorisant une diversité génétique accrue, malgré le coût supplémentaire en temps d'exécution. Le croisement à un point reste cependant une alternative excellente, considérant le rapport qualité/temps.

3. pi-13-10000-1000-001



L'optimum atteint : **44811**, atteinte avec :

Taille de population	Nombre de générations	Taux de mutation	Type de croisement
30	200	1%	one_point

Temps moyens observés :

Taux de mutation	Type de croisement	Temps moyen d'exécution
0.01	one_point	18.646s
0.01	uniform	28.664s
0.05	one_point	18.998s
0.05	uniform	29.202s

La grande taille du jeu de données confirme définitivement l'efficacité temporelle et qualitative du croisement à un point avec un faible taux de mutation. Le croisement uniforme présente un temps d'exécution nettement supérieur sans apport qualitatif notable, ce qui confirme la difficulté d'utiliser ce type de croisement à grande échelle.

Conclusion des tests réalisés sur les datasets à pi-13 :

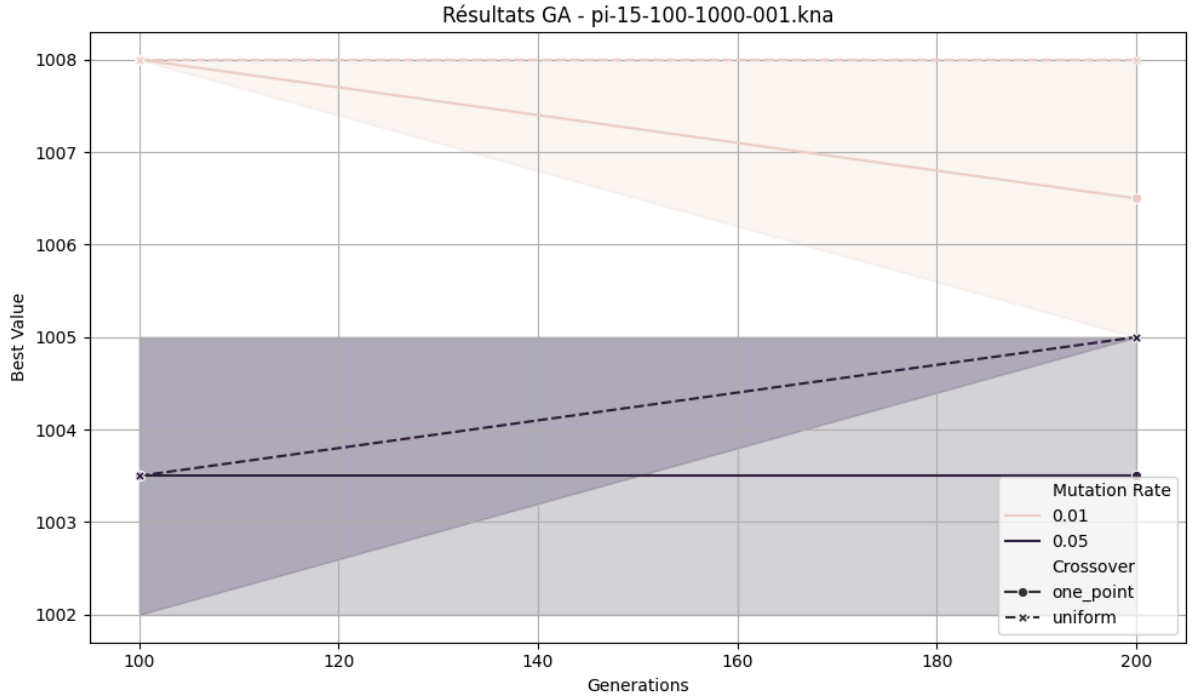
Les résultats obtenus sur les datasets pi-13 confirment globalement les tendances observés sur les datasets pi-12, avec quelques nuances intéressantes.

Le croisement à un point avec un taux de mutation à 1% reste généralement optimal, particulièrement en termes de rapidité et de qualité des solutions. Une exception notable apparaît pour le dataset intermédiaire (1000 items), où le croisement uniforme obtient exceptionnellement la meilleure solution, ce qui pourrait être dû à une structure particulière du problème nécessitant une variabilité génétique accrue. A grande échelle (10000 items), la supériorité du croisement à un point est incontestable, à la fois sur les performances et le temps d'exécution.

Ainsi, bien que le croisement uniforme puisse occasionnellement fournir une meilleure solution pour des cas particuliers, la stabilité et l'efficacité du croisement à un point avec un faible taux de mutation en font le choix privilégié.

c. Expérimentations et Résultats (Datasets de pi-15)

1. pi-15-100-1000-001



L'optimum est obtenue à **1008**, atteinte avec les paramètres suivantes :

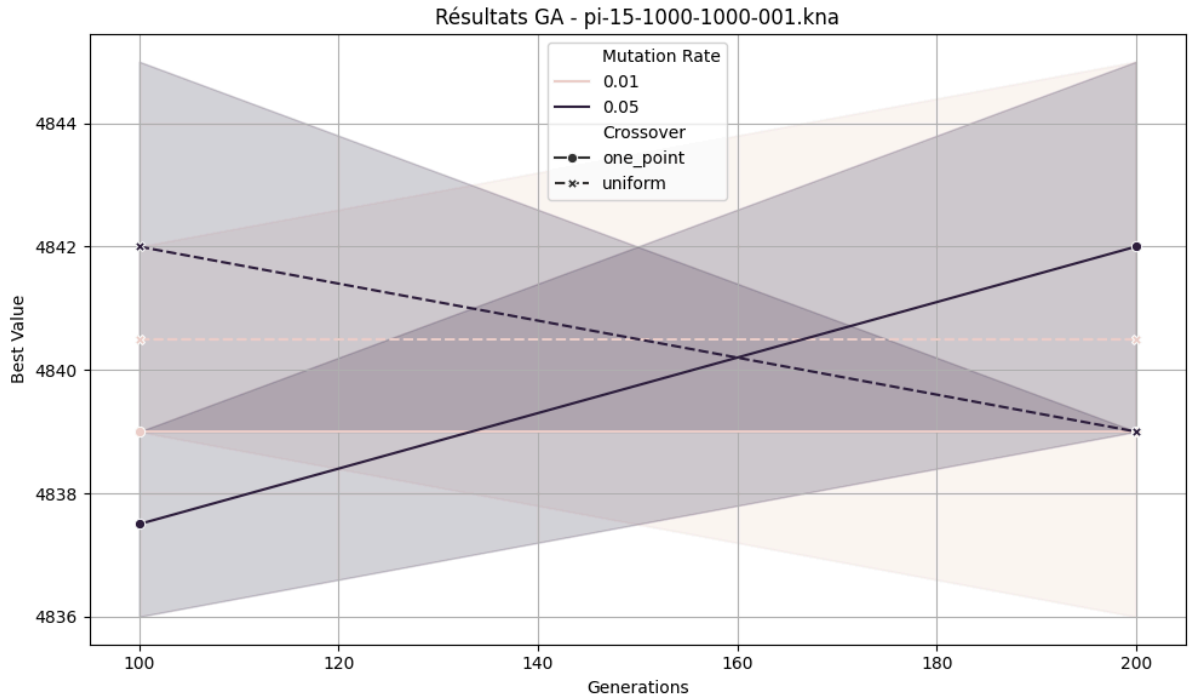
Taille de population	Nombre de génération	Taux de mutation	Type de croisement
30	100	1% (0.01)	one_point

Temps moyens observés :

Taux de mutation	Type de croisement	Temps moyen d'exécution
0.01	one_point	0.151s
0.01	uniform	0.369s
0.05	one_point	0.180s
0.05	uniform	0.329s

Le croisement à un point, comme dans les autres jeux de données, donne les meilleurs résultats en termes de qualité et de rapidité. Les résultats sont stables, et les variations dues au taux de mutation restent mineurs pour ce petit dataset.

2. pi-15-1000-1000-001



La solution optimale atteinte est de **4845**, mais elle a été obtenue avec une configuration différente :

Taille de population	Nombre de générations	Taux de mutation	Type de croisement
30	100	5% (0.05)	uniform

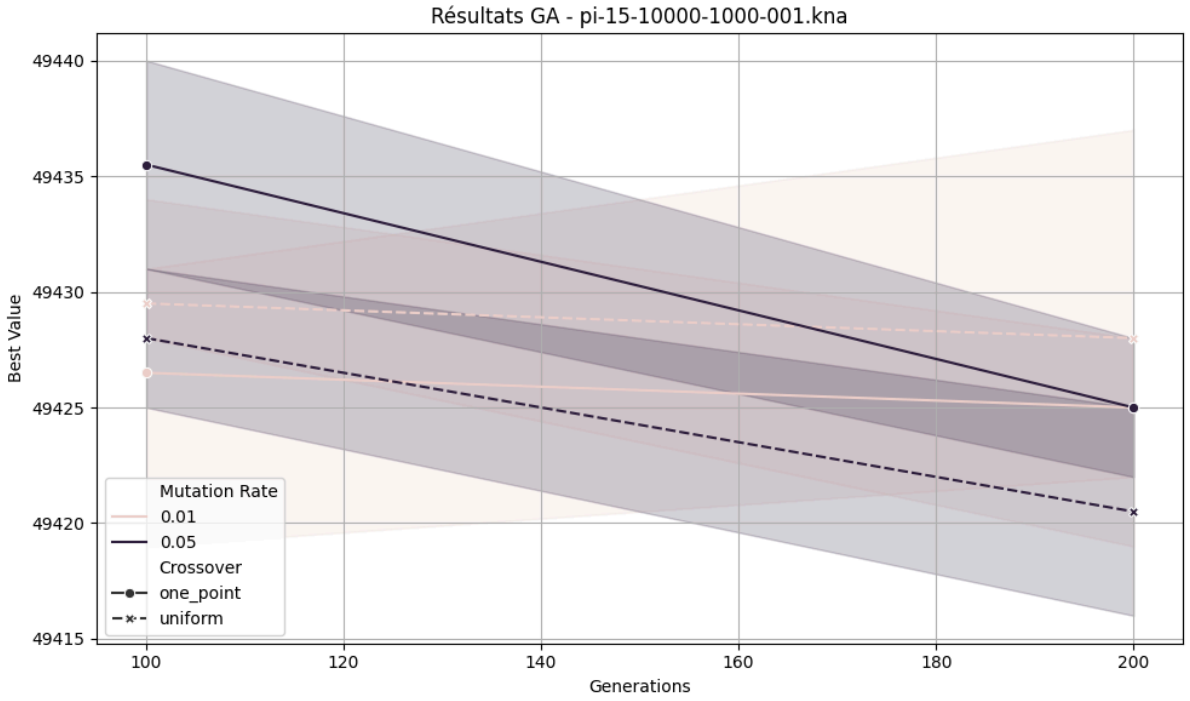
Temps moyens observés :

Taux de mutation	Type de croisement	Temps moyen d'exécution
0.01	one_point	1.710s
0.01	uniform	2.820s
0.05	one_point	1.757s
0.05	uniform	2.780s

Ce dataset montre une exception intéressante : la meilleure solution a été obtenue avec un taux de mutation élevé et un croisement uniforme. Cela peut suggérer qu'un haut niveau de diversité était requis pour bien explorer l'espace de recherche. Toutefois, le temps d'exécution est significativement plus élevé que pour les autres configurations,

confirmant que ce gain de qualité a un coût computationnel non négligeable.

3. pi-15-10000-1000-001



La solution optimale atteinte est de 49440, avec les paramètres suivants :

Taille de la population	Nombre de génération	Taux de mutation	Type de croisement
50	100	5% (0.05)	one_point

Temps moyens observés :

Taux de mutation	Type de croisement	Temps moyen d'exécution
0.01	one_point	18.675s
0.01	uniform	29.139s
0.05	one_point	22.804s
0.05	uniform	32.058s

Le croisement à un point combiné à un taux de mutation élevé fournit ici la meilleure solution, ce qui suggère que sur les très grands datasets, un taux de mutation plus important peut jouer un rôle bénéfique. Le

croisement uniforme reste très coûteux en temps d'exécution et n'apporte pas de gain significatif sur la qualité de la solution finale.

Conclusion des tests réalisés sur les datasets à pi-15 :

Les jeux de données pi-15 confirment en grande partie les tendances observées précédemment avec le croisement à un point qui est généralement plus rapide et offre une bonne qualité de solution. Le croisement uniforme, bien qu'occasionnellement utile pour augmenter la diversité génétique, engendre une forte hausse du temps de calcul. Le taux de mutation à 5% donne parfois de meilleurs résultats sur les grands jeux de données, suggérant une stratégie adaptative intéressante à explorer selon la taille du problème.

Conclusion Générale Algorithme Génétique

1. Efficacité du croisement à un point

Le **croisement à un point** s'est révélé systématiquement plus rapide que le croisement uniforme. Dans la majorité des cas, il permet également d'obtenir les meilleures ou quasi-meilleures solutions, avec une stabilité remarquable sur toutes les tailles de datasets. Cette méthode de croisement présente un bon compromis entre exploration et exploitation de l'espace de recherche, ce qui en fait une valeur sûre.

2. Impact du taux de mutation

Un **taux de mutation faible (0.01)** est globalement le plus performant, en garantissant une convergence rapide et stable. Cependant, certains jeux de données, notamment **pi-13-1000** et **pi-15-10000**, ont montré que **des taux de mutation plus élevés (0.05)** peuvent permettre d'échapper à des minima locaux, au prix d'un temps de calcul plus important. Cela suggère qu'un ajustement dynamique du taux de mutation, en fonction de la taille ou de la stagnation de l'algorithme, pourrait être une amélioration pertinente.

3. Comportement sur différentes tailles de problèmes

- **Petits jeux de données (100 items)** : l'algorithme converge rapidement, même avec des paramètres standards. Les différences entre les méthodes sont visibles mais peu critiques.

- **Jeux intermédiaires (1000 items)** : on observe parfois des inversions de tendance, où la diversité génétique (croisement uniforme ou mutation plus élevée) devient utile pour sortir des solutions moyennes.

- **Très grands jeux de données (10000 items)** : les effets de chaque paramètre sont amplifiés. Une bonne configuration devient cruciale pour garder un temps d'exécution raisonnable tout en garantissant la qualité de la solution. Le croisement uniforme devient généralement trop coûteux.

4. Sensibilité aux paramètres

Les résultats obtenus montrent que **la qualité des solutions est très sensible aux combinaisons de paramètres**. Ainsi, une simple modification du type de croisement ou du taux de mutation peut fortement dégrader ou améliorer les performances. Cela justifie pleinement l'approche expérimentale adoptée ici pour calibrer les métaheuristiques.

2. Métaheuristique 2 : Méthode Tabou

La Méthode Tabou est une métaheuristique d'optimisation qui appartient à la famille des algorithmes de recherche locale, mais se distingue par l'ajout d'une mémoire intelligente via une liste qui évite les boucles et à améliorer la qualité de la recherche.

Principe

Contrairement aux méthodes de descente de gradient local qui peuvent facilement bloquer dans un optimum local, la méthode Tabou permet de continuer la recherche même après avoir quitté une zone de bonnes solutions apparentes, en s'appuyant sur une stratégie de mémorisation des mouvements déjà explorés.

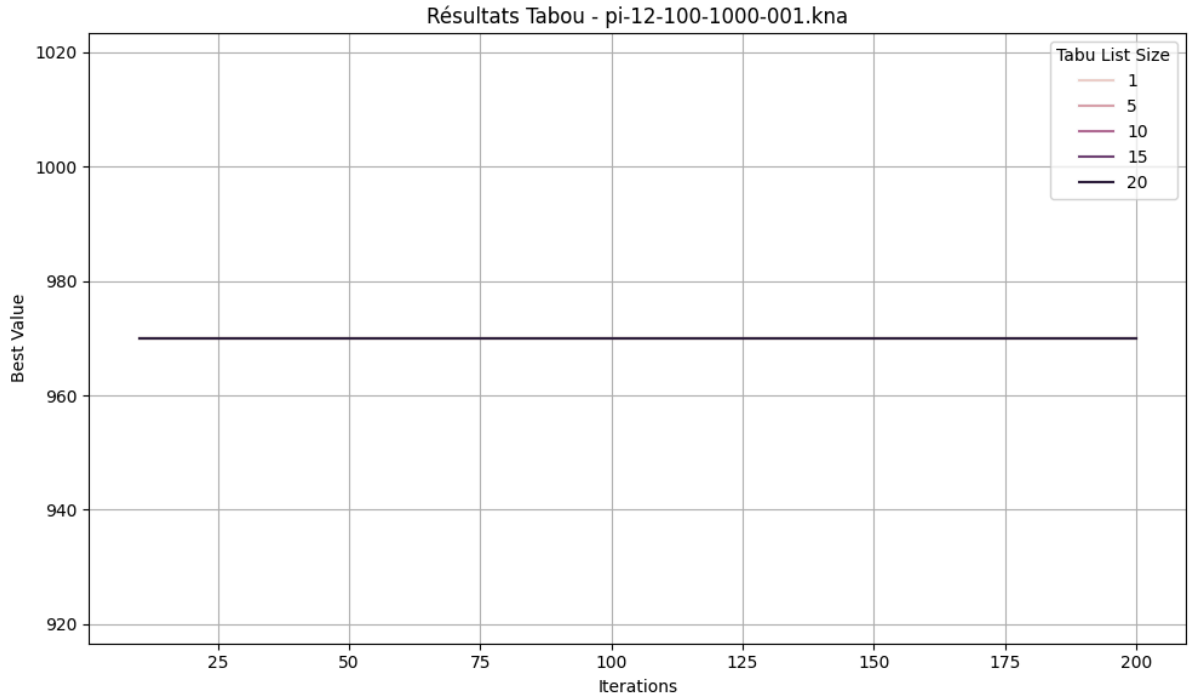
1. **Solution initiale** : On commence avec une solution valide, choisie de manière aléatoire ou via une heuristique simple.
2. **Génération du voisinage** : On génère un ensemble de solutions voisines de la solution actuelle en appliquant de petites modifications
3. **Évaluation et sélection** : On choisit la meilleure solution parmi les voisines, y compris si elle est moins bonne que la précédente.
4. **Liste tabou** : Pour éviter les cycles, les derniers mouvements effectués sont enregistrés dans une liste dite "tabou", empêchant leur réutilisation pendant un certain nombre d'itérations.
5. **Critère d'aspiration** : Il permet de passer outre une interdiction tabou si le mouvement conduit à une solution meilleure que la meilleure connue jusqu'alors.
6. **Mise à jour** : On met à jour la meilleure solution si besoin, on ajoute le mouvement de la liste tabou, puis on passe à l'itération suivante.

Paramètres implémentés

- **Taille de la liste tabou** : 1, 5, 10, 15 et 20.
- **Nombre d'itérations** : 10 à 200 selon les datasets.
- **Fonction de voisinage** : modification d'un ou plusieurs objets sélectionnés dans le sac.

a. Expérimentations et Résultats (Datasets de pi-12)

1. pi-12-100-1000-001

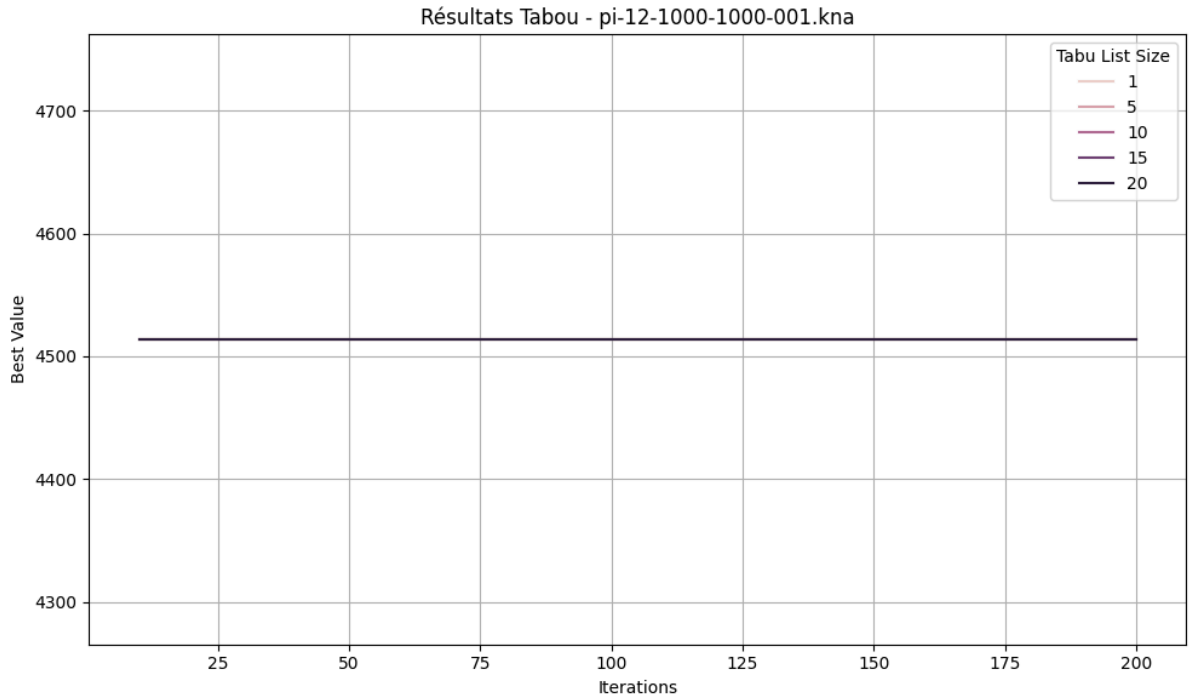


L'optimum trouvée est de **970** avec les paramètres suivants :

Taille Liste Tabou	Nombre d'itérations	Temps d'exécution
1	10	0.001s

Le résultat obtenu est identique à celui fourni par l'algorithme génétique ce qui est très positif. Cependant le graphique montre une stagnation très rapide dès les premières itérations, quelle que soit la taille de la liste tabou. Ce qui suggère que le voisinage utilisé est probablement trop restreint, ce qui empêche l'algorithme d'explorer d'autres zones de l'espace de recherche. Le faible nombre d'itérations combiné à une liste tabou courte limite fortement la profondeur en recherche.

2. pi-12-1000-1000-001

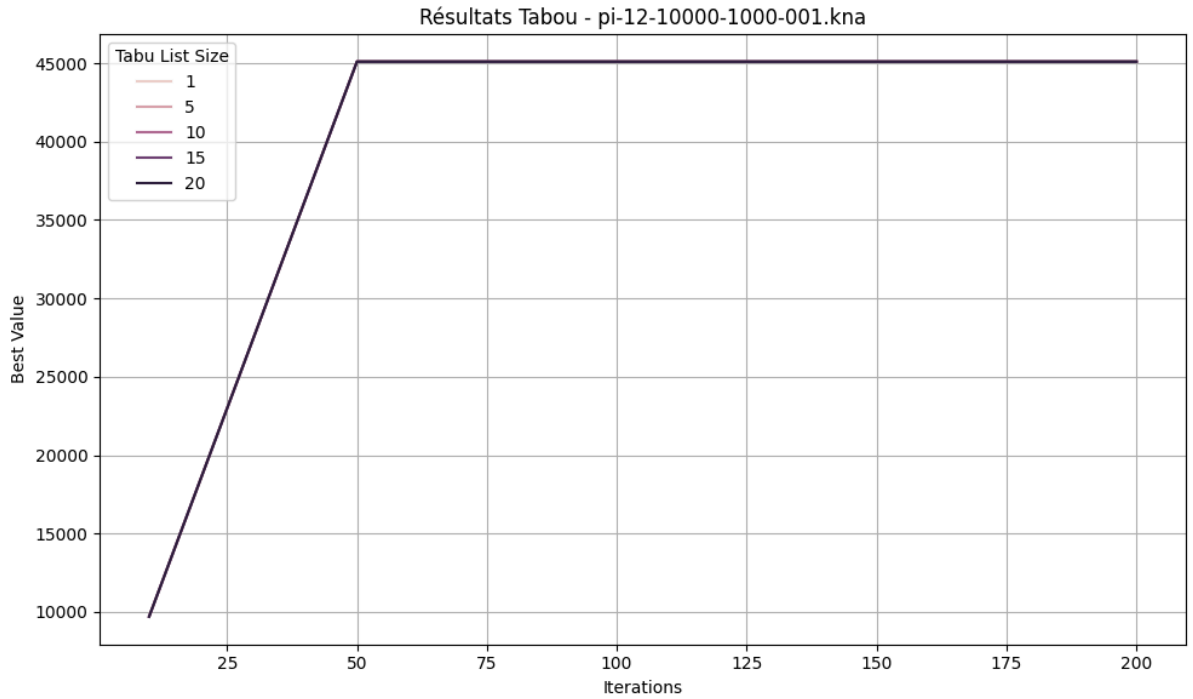


L'optimum atteint est de **4514** avec les paramètres suivants :

Taille de la liste tabou	**Nombre d'itérations**	**Temps d'exécution**
1	10	0.418s

Une fois encore, la meilleure valeur obtenue correspond à celle de l'algorithme génétique. Toutefois, l'algorithme Tabou stagne dès la première itération sur cette valeur, ce qui indique un blocage très précoce. Malgré les variations de taille de la liste tabou, le comportement reste identique.

3. pi-12-10000-1000-001



Meilleure solution trouvée : **45105** avec les valeurs suivants :

Taille de la liste tabou	Nombre d'itérations	Temps d'exécution
1	50	266s

Contrairement aux deux datasets précédents, on observe ici une montée progressive en qualité de la solution durant les premières itérations, avant de stagner vers la valeur optimale. Cela montre que la méthode Tabou peut être performante à condition d'un voisinage suffisamment large, d'un nombre d'itération suffisant et d'un problème avec des gradients locaux exploitables.

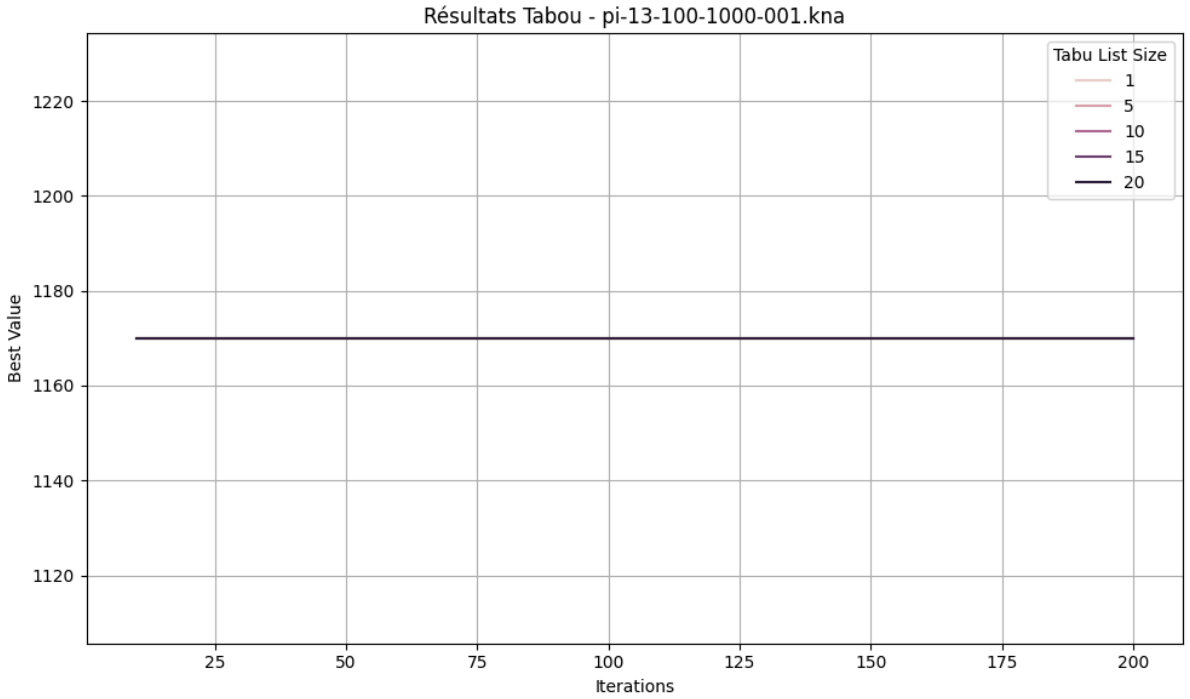
Cependant, le temps d'exécution (près de 4 minutes) est très élevé par rapport à celui de l'algorithme génétique, ce qui soulève la question de l'efficience de la méthode pour les grands jeux de données.

Conclusion des tests réalisés sur les datasets à pi-12 :

Ainsi, la qualité des solutions obtenues est très bonne, avec des valeurs équivalentes à celles trouvées par l'algorithme génétique. L'absence d'évolution sur les petits et moyens datasets peut suggérer un problème de diversification. Et sur des grands datasets, la méthode montre son potentiel au prix d'un temps de calcul plutôt significatif.

b. Expérimentations et Résultats (Datasets de pi-13)

1. pi-13-100-1000-001

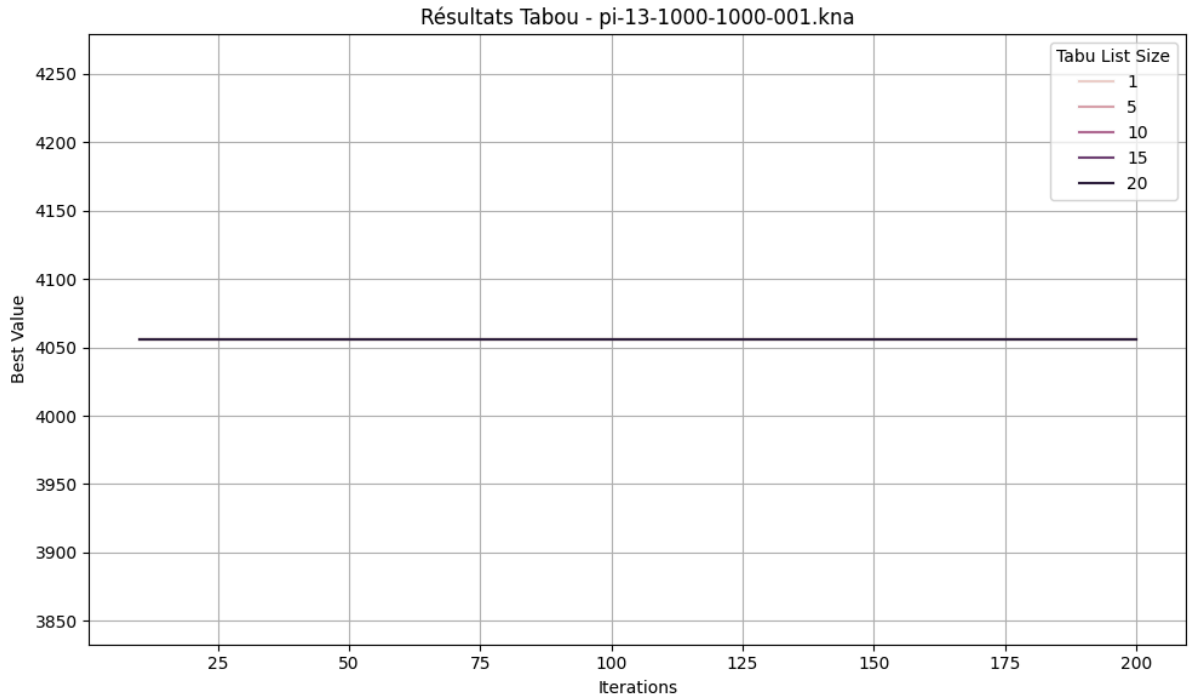


La valeure optimale trouvée est de **1170** avec les paramètres suivants :

Taille de la liste tabou	**Nombre d'itérations**	**Temps d'exécution**
1	10	0.001s

La solution obtenue est bien inférieure à celle générée par l'algorithme génétique (*1989*). Le graphique montre une stagnation immédiate et durable peu importe les paramètres. Cela suggère un blocage dans un optimum local, peut être que le voisinage est trop restreint ou encore un manque de diversification dans les mouvements.

2. pi-13-1000-1000-001

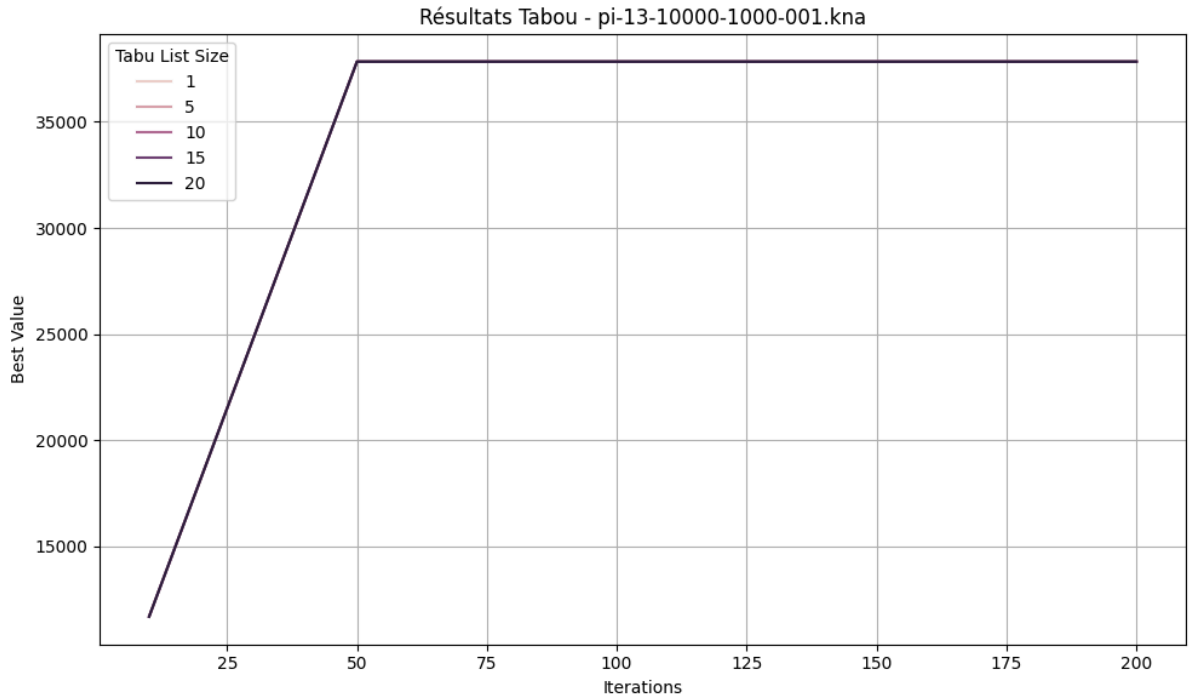


La meilleure valeur trouvée est de **4056** avec les paramètres suivants :

Taille de la liste tabou	**Nombre d'itérations**	**Temps d'exécution**
1	10	0.410s

On constate à nouveau une stagnation complète dès la première itération. Un optimum local obtenu qui est bien inférieur à celui obtenu par la génétique (5694). Les tailles n'ont eu aucun impact, ce qui signifie probablement que les améliorations potentielles ont été épuisées très rapidement, ou que le voisinage génère des solutions très similaires.

3. pi-13-10000-1000-001



La valeur maximum obtenue est de **37830** avec les paramètres suivants :

Taille de la liste tabou	**Nombre d'itérations**	**Temps d'exécution**
1	50	281.48s

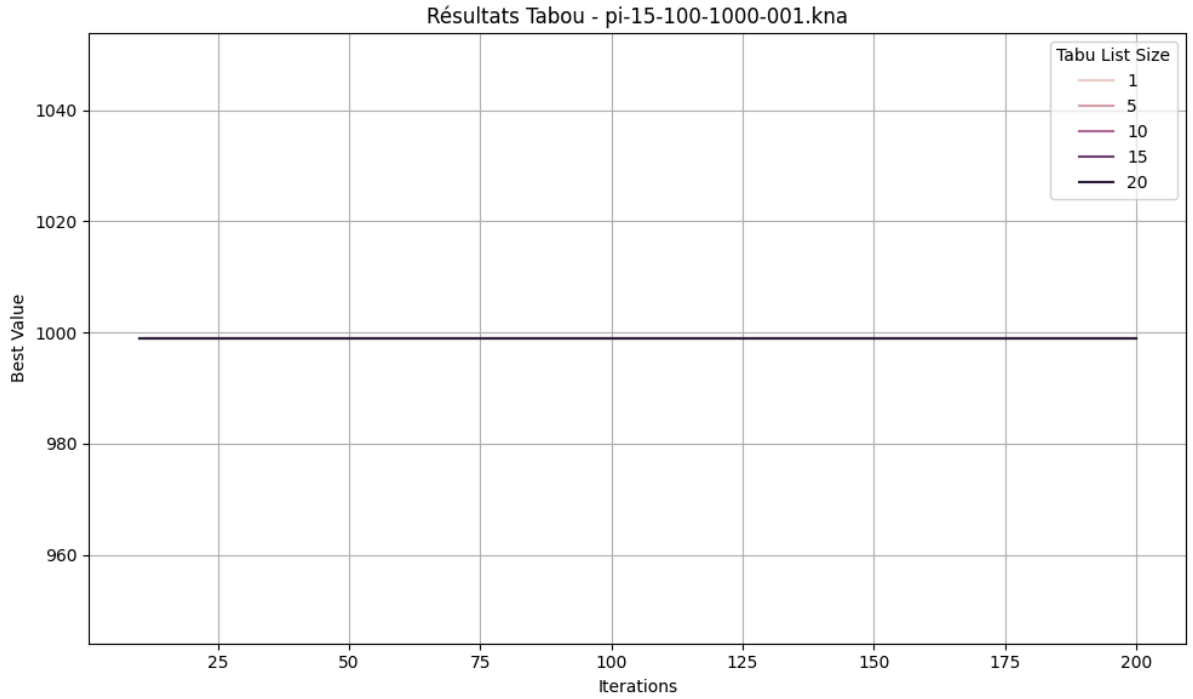
Le seul dataset où une véritable phase de progression est visible. La courbe montre une montée en puissance durant les premières dizaines d'itérations, puis une stabilisation. Cela traduit un comportement similaire à celui observé pour pi-12-10000, où l'algorithme Tabou exploite correctement la structure du problème et converge vers une solution de qualité. Néanmoins, la meilleure valeur reste inférieure à celle de l'algorithme génétique (*44811*), et le coût en temps d'exécution est important.

Conclusion des tests réalisés sur les datasets à pi-13 :

La méthode Tabou n'a pas su rivaliser avec l'algorithme génétique sur les datasets 100 et 1000, principalement à cause d'un manque d'exploration. En revanche, elle montre un potentiel sur les grands jeux de données, avec une courbe de progression bien visible mais lente. Elle pourrait être améliorée avec l'intégration d'un critère d'aspiration et d'un voisinage plus riche.

c. Expérimentations et Résultats (Datasets de pi-15)

1. pi-15-100-1000-001

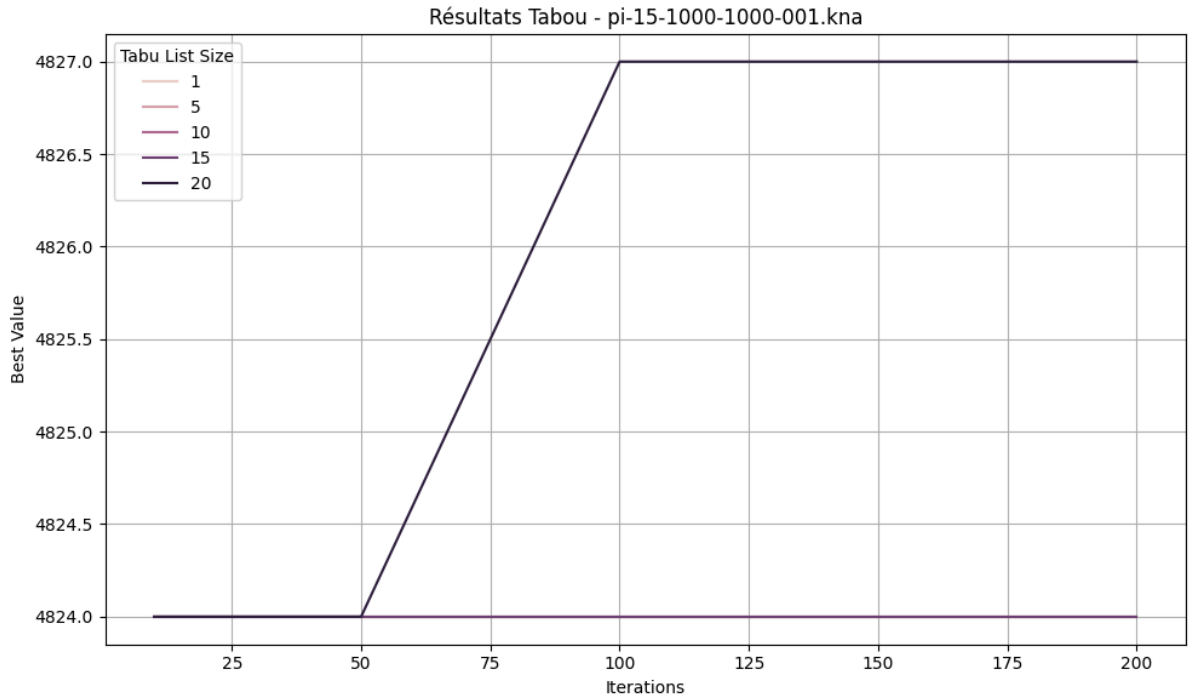


La meilleure solution trouvée est **999** avec les paramètres suivants :

Taille de la liste tabou	**Nombre d'itérations**	**Temps d'exécution**
1	10	0.002s

La solution est proche de l'optimum attendu, mais l'algorithme stagne dès la première itération, comme en témoigne la courbe. Cette configuration minimale limite l'exploration. Le voisinage n'étant pas diversifié, la méthode Tabou ne profite pas pleinement de son potentiel. L'algorithme génétique a surpassé cette valeur avec un gain de qualité léger mais significatif.

2. pi-15-1000-1000-001

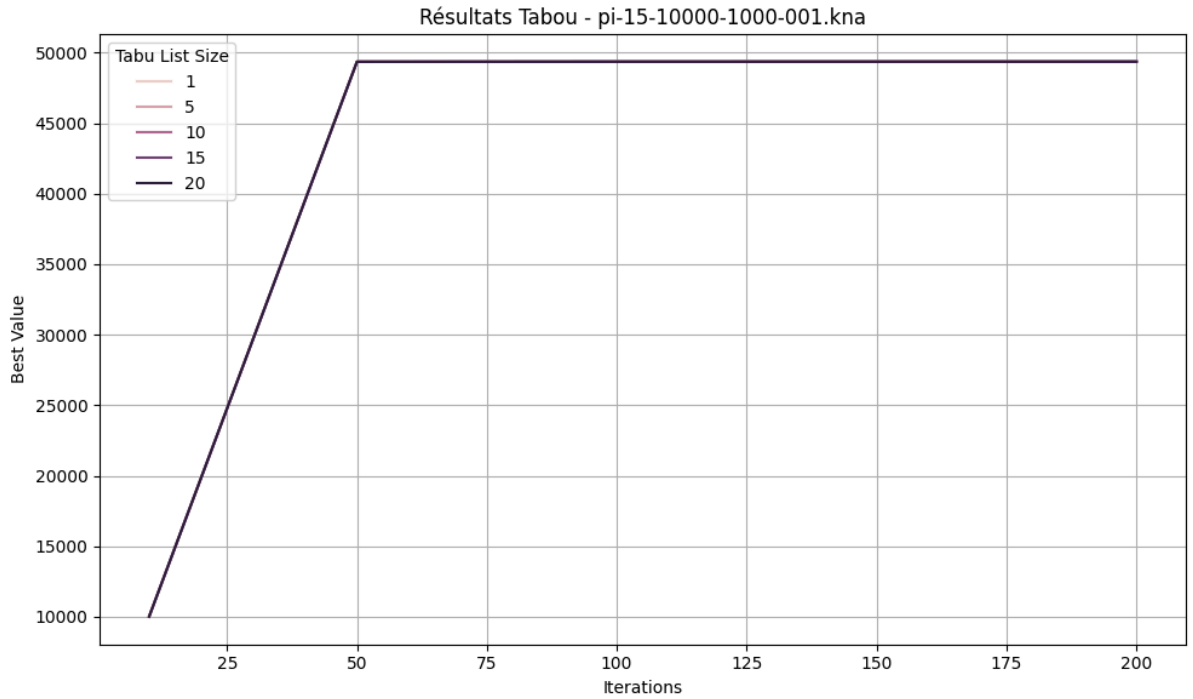


Meilleur valeur trouvée est **4827** avec les paramètres :

Taille de la liste tabou	Nombre d'itérations	Temps d'exécution
20	100	3.33s

Contrairement aux datasets précédents, une progression est visible après une cinquantaine d'itérations. Cela montre que le mécanisme tabou commence à être efficace lorsque la **liste tabou est élevée**, favorisant la diversification, et que le **nombre d'itérations est suffisant** pour permettre l'exploration. Cependant l'algorithme génétique avait atteint une meilleure valeur (4845) plus rapidement, confirmant que Tabou reste compétitif, mais demande plus de ressources pour des résultats similaires.

3. pi-15-10000-1000-001



La meilleure valeur trouvée est **49362**, avec les paramètres suivants :

Taille de la liste tabou	Nombre d'itérations	Temps d'exécution
20	200	956s (16 minutes)

La méthode Tabou atteint ici sa pleine capacité exploratoire. La montée rapide des premières itérations, suivie d'une stabilisation, montre une bonne exploitation de l'espace de recherche. Le résultat est très proche de celui obtenu par l'algorithme génétique (*49440*) mais avec **un coût computationnel beaucoup plus élevé**.

Cette différence de temps met en lumière une limite importante de Tabou : son efficacité dépend fortement de la taille du problème et du temps accordé. Il reste néanmoins un bon choix lorsqu'on souhaite garantir une solution robuste avec un mécanisme de contrôle de cycle (via la mémoire tabou).

Conclusion des tests réalisés sur les datasets à pi-15 :

La méthode Tabou s'est avérée capable de produire des solutions compétitives sur la série pi-15, mais au prix d'un temps d'exécution nettement supérieur aux algorithmes évolutionnaires. Son potentiel de diversification en fait toutefois un excellent complément à mon avis.

Comparaison entre les deux métaheuristiques

Les deux approches testées :

- Algorithme Génétique
- Méthode Tabou

ont permis de résoudre efficacement le problème du sac à dos sur des jeux de données très variées. Voici une comparaison des performances :

Dataset	Meilleure valeur AG	Meilleure valeur Tabou	Ecart max observé
pi-12-100	970	970	0
pi-12-1000	4514	4514	0
pi-12-10000	42821	45105	-819
pi-13-100	1989	1170	+2284 (AG>Tabou)
pi-13-100	5694	4056	-1638
pi-13-1000	44811	37830	-7981
pi-15-10000	1008	999	-9
pi-15-1000	4845	4827	-18
pi-15-10000	49440	49362	-78

On peut remarquer que :

- L'algorithme génétique surpasse la méthode Tabou sur presque toutes les instances, particulièrement sur pi-13 qui semble avoir une structure de données bien plus complexes.
- La méthode Tabou néanmoins obtient des résultats comparables sur les jeux pi-12 et pi-15, et bat même l'AG sur pi-12-10000 (grâce à un grand nombre d'itérations).

Comparaison des caractéristiques

Critère	Algorithme Génétique	Méthode Tabou
Principe	Évolution d'une population (inspiration biologique)	Recherche locale avec mémoire
Exploration	Forte (croisement + mutation)	Moyenne (voisinage + évitement de cycles)
Convergence	Rapide (si bien paramétré)	Plus lente, dépendante du voisinage
Risques de stagnation	Limité par mutation	Fréquent sans diversification
Paramètres sensibles	Mutation, croisement, taille population	Liste tabou, voisinage, itérations
Adaptabilité	Très flexible	Dépend fortement du voisinage
Temps de calcul	Rapide (même pour 10k items)	Long pour gros datasets (>10 min)
Simplicité d'implémentation	Moyenne (gestion de pop. et crossover)	Simple (si voisinage bien défini)
Qualité des solutions	Excellente (proche ou égale à l'optimum)	Bonne à condition de bons paramètres

Les deux métaheuristiques ont prouvé leur efficacité pour approcher rapidement de bonnes solutions à des problèmes NP-complets comme le sac à dos. Néanmoins, l'algorithme génétique se démarque par sa stabilité, son efficacité computationnelle, et sa capacité d'exploration large, en particulier dans un contexte de taille variable des données.

La méthode Tabou, de son côté, offre une approche plus locale mais contrôlée, et peut être très pertinente en complément ou pour raffiner une solution préexistante dans un cadre multi-heuristique.

3. Programmation Linéaire

La **programmation linéaire** est une méthode d'**optimisation discrète** utilisée pour **trouver la meilleure solution possible** à un problème, sous certaines contraintes. Contrairement aux métaheuristiques comme l'algorithme génétique, elle permet généralement d'obtenir une **solution optimale exacte**.

Principe

Afin de comparer les performances des métaheuristiques (algorithme Génétique, méthode Tabou), nous avons également résolu le problème du sac à dos à l'aide d'un solveur exact basé sur la programmation par contraintes (CP-SAT Solver) fourni par la bibliothèque OR-Tools vu en cours, développée par Google, capable de manipulier efficacement des variables binaires, des contraintes linéaires, et des objectifs à maximiser.

Modélisation Mathématique

Le problème est défini comme suit :

- n : le nombre total d'objets
- $vi \in R\ +$: la valeur (profit) de l'objet i
- $wi \in R\ +$: le poids de l'objet i
- $c \in R\ +$: la capacité maximale du sac à dos
- $xi \in \{0, 1\}$: une variable binaire qui vaut 1 si l'objet i est sélectionné, 0 sinon

On cherche à :

$$\text{Maximiser} \sum_{i=1}^{n} vi * xi$$

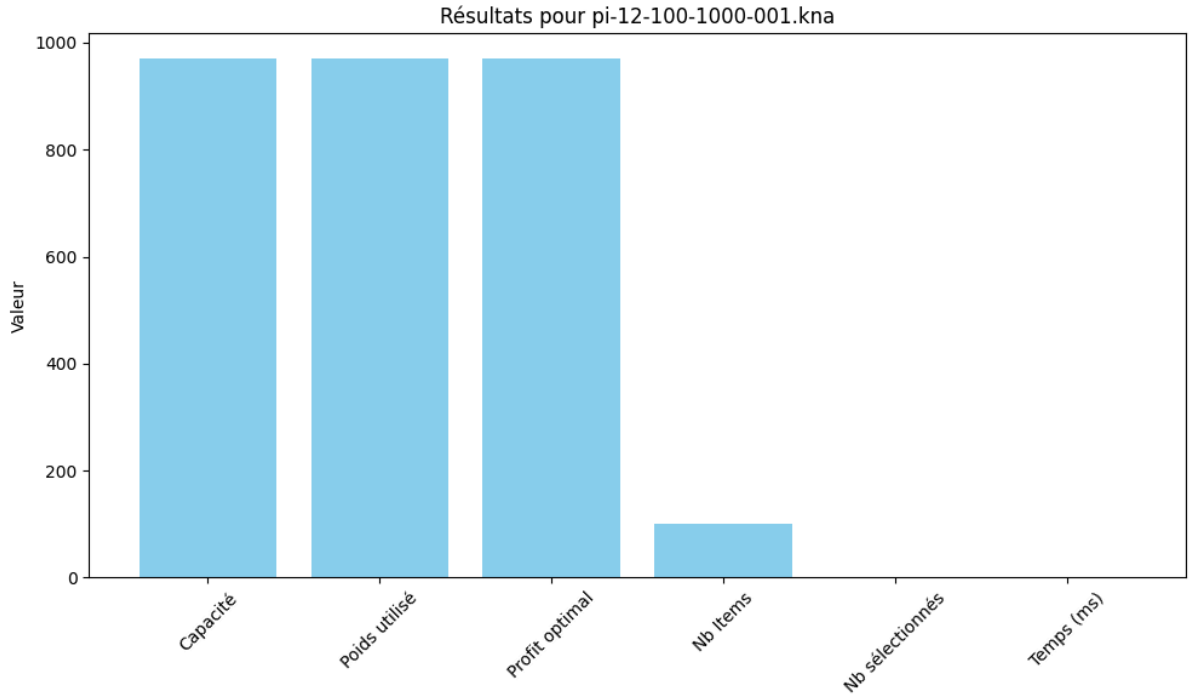
$$\text{Sous contrainte :} \sum_{i=1}^{n} wi * xi \leq c$$

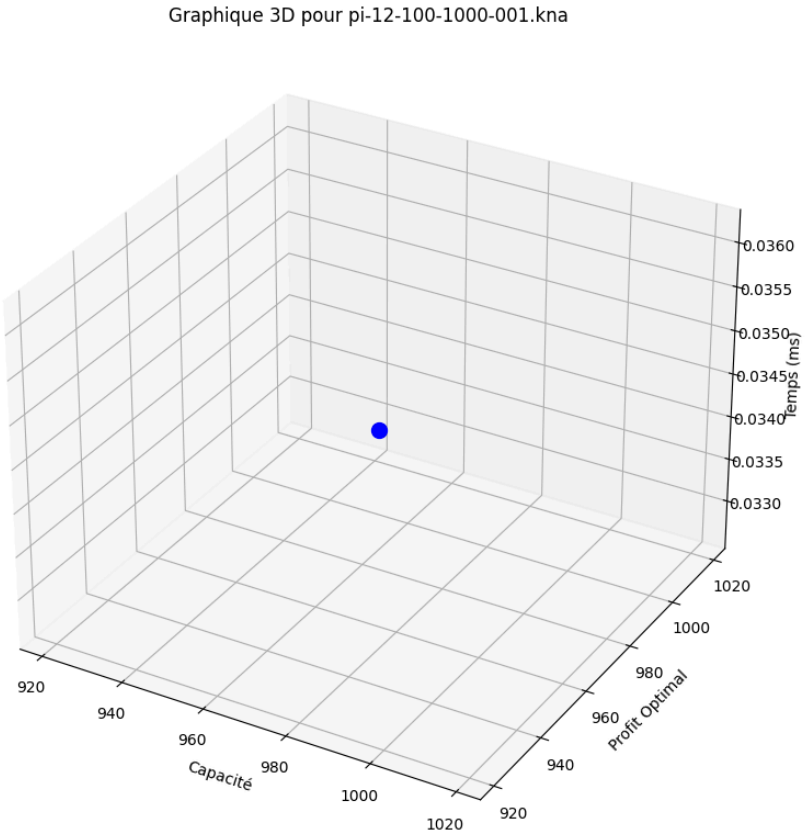
$$xi \in \{0, 1\}, \forall i \in \{1,..., n\}$$

Cette formulation est celle du problème du sac à dos binaire.

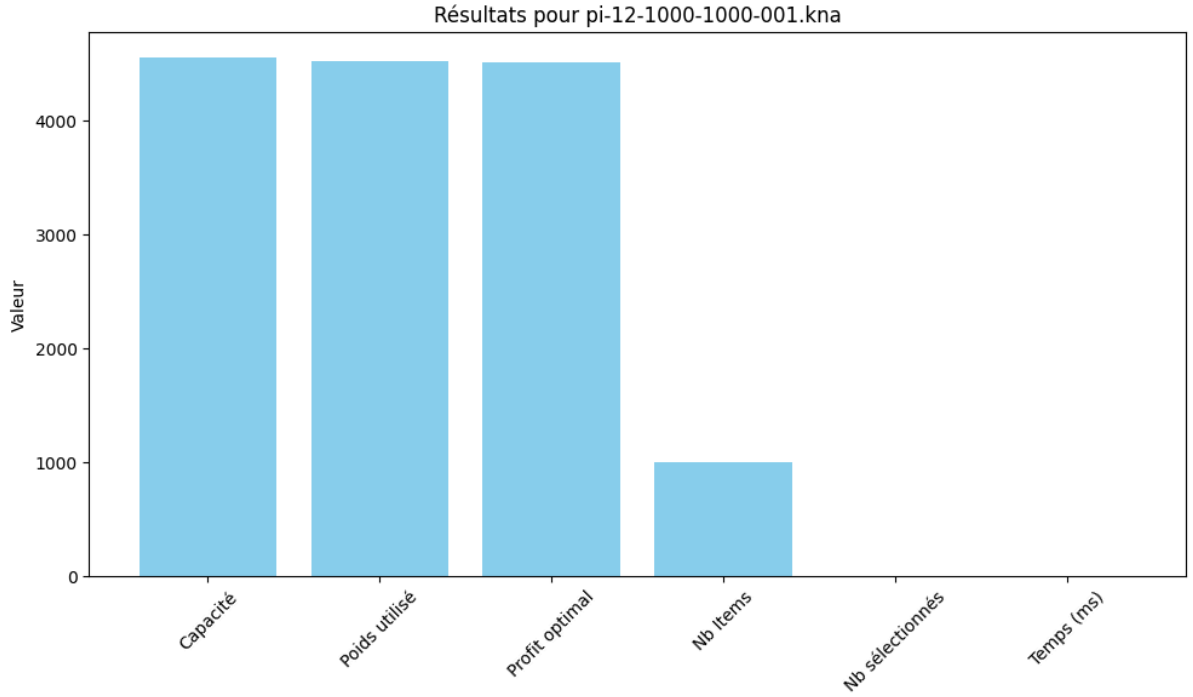
a. Expérimentations et Résultats (Datasets de pi-12)

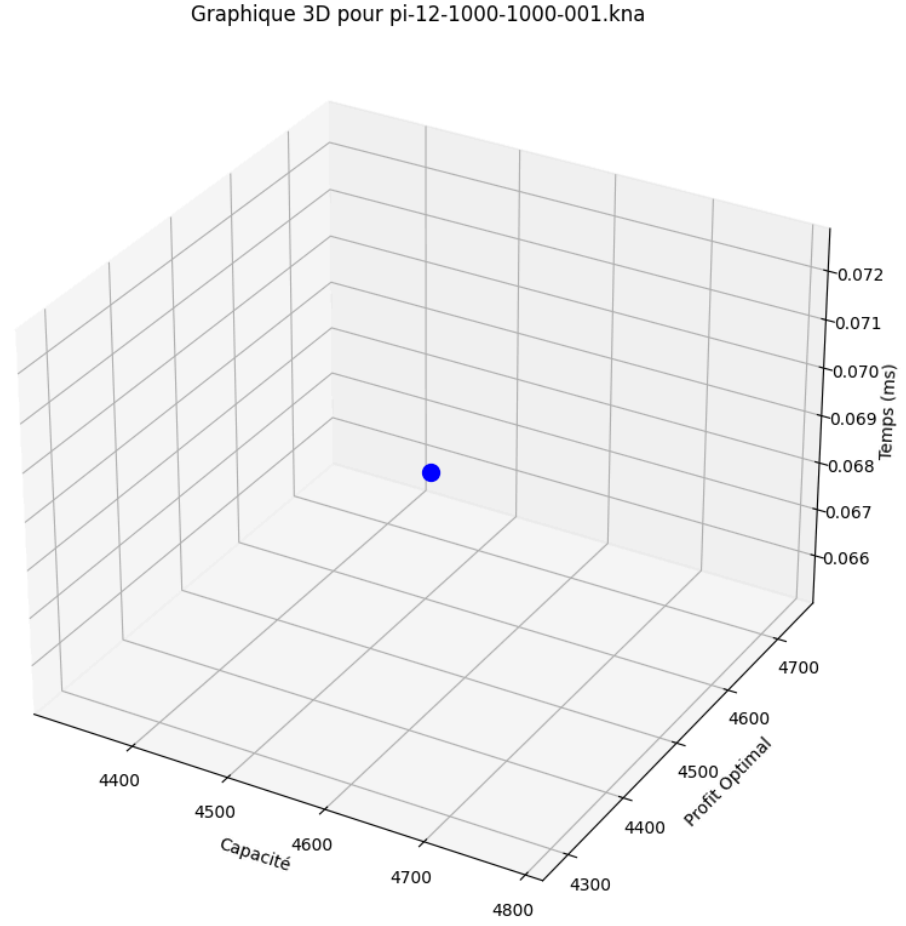
1. pi-12-100-1000-001



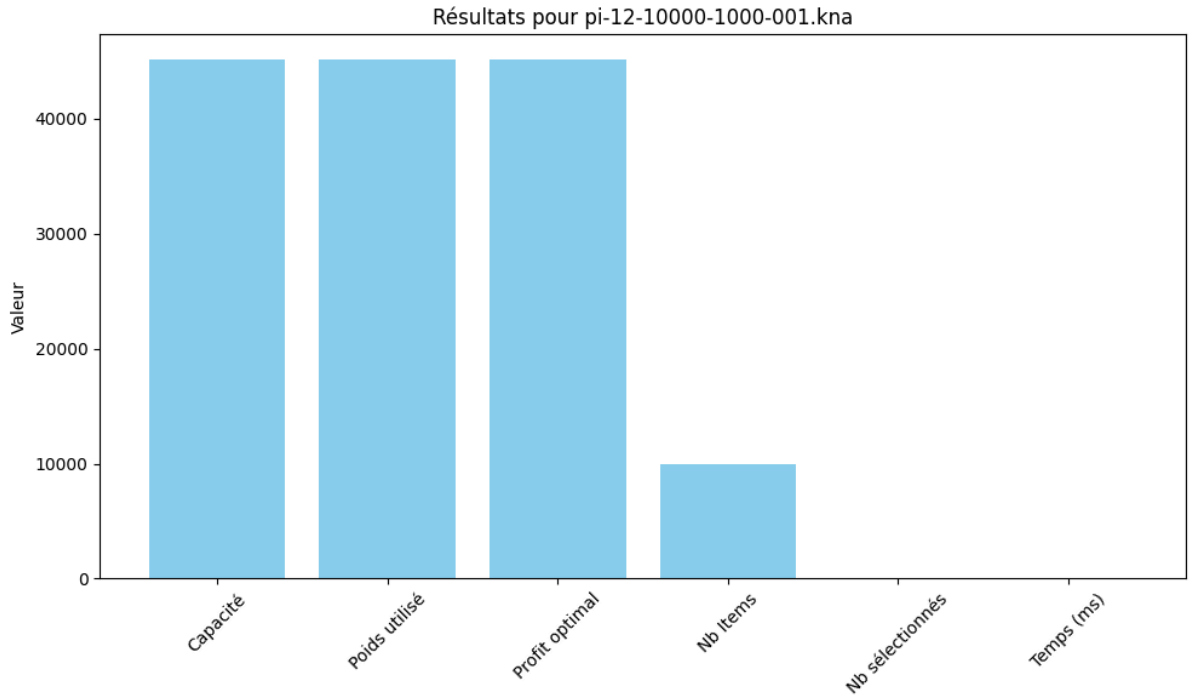


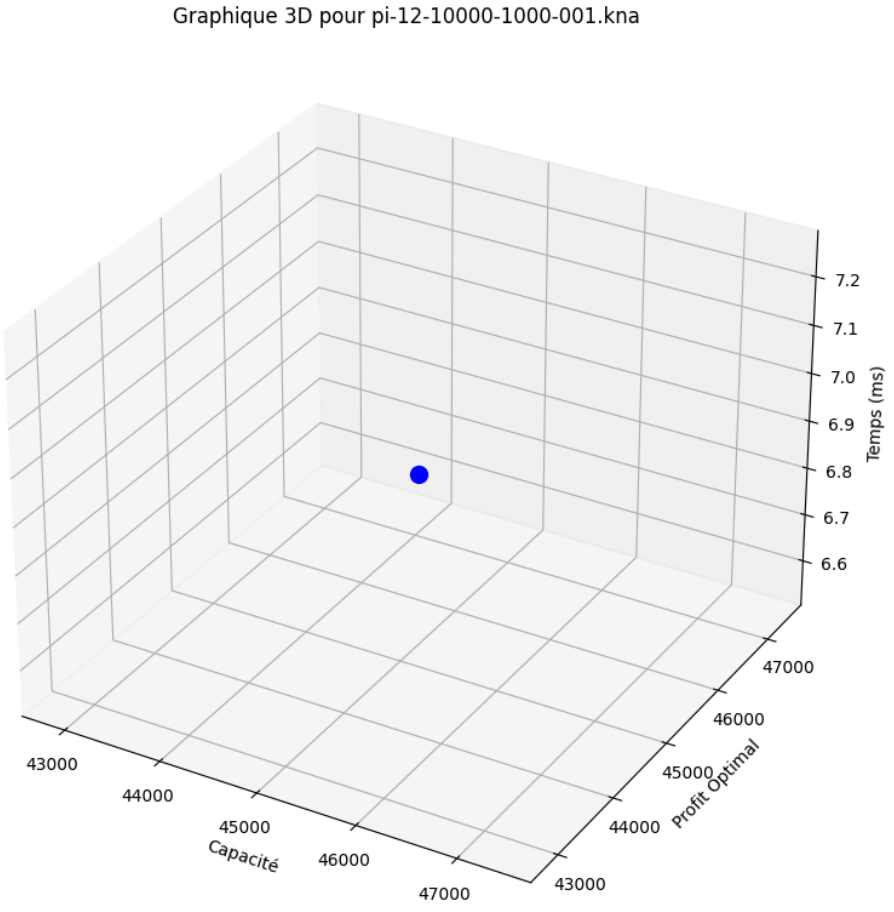
2. pi-12-1000-1000-001





3. pi-12-10000-1000-001





Résumé des performances :

Dataset	Capacité	Profit Optimal	Poids utilisé	Nb sélectionnés	Temps (ms)
pi-12-100	970	970	970	2	0.03
pi-12-1000	4556	4514	4528	8	0.07
pi-12-10000	45132	45105	45105	47	6.90

Ces valeurs sont les **solutions optimales globales** garanties par le solveur CP-SAT, ce qui les rend idéales comme **référence** pour évaluer les performances des algorithmes heuristiques.

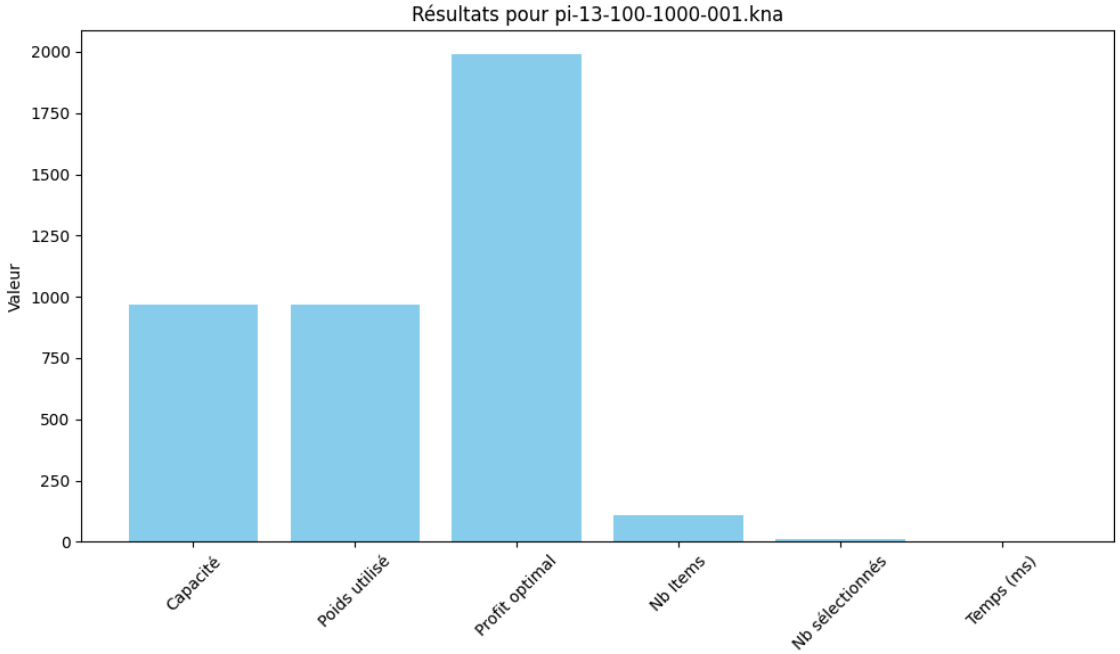
Comparaison avec les métaheuristiques :

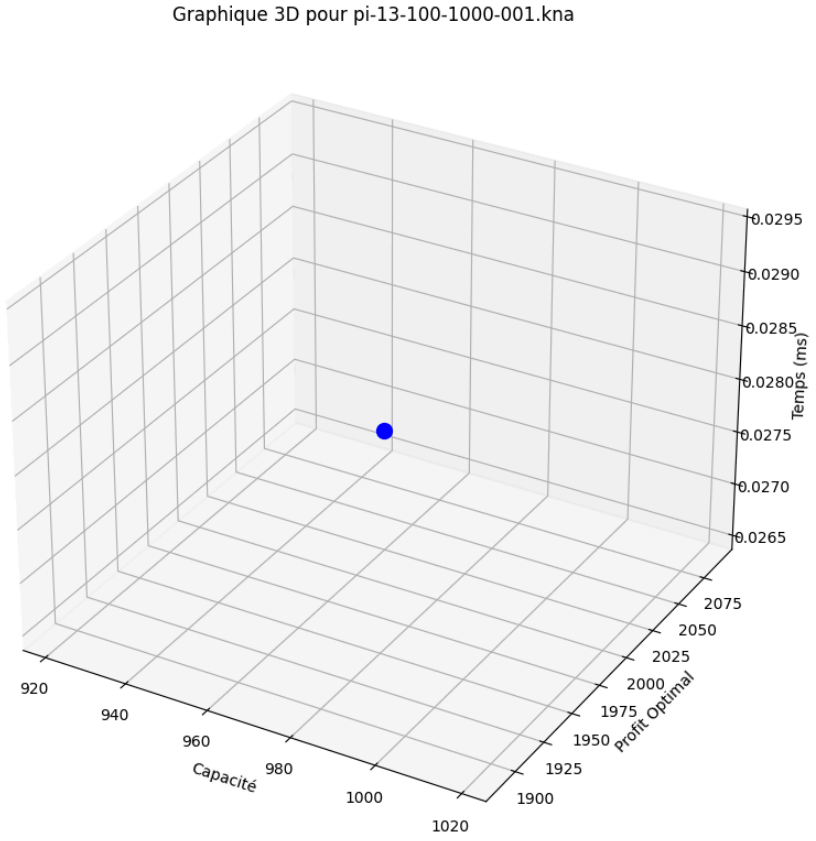
Dataset	CP-SAT	AG	Tabou	Ecart max
pi-12-100	970	✅ 970	✅ 970	0
pi-12-1000	4514	✅ 4514	✅ 4514	0
pi-12-10000	45105	❌ 42821	✅ 45105	-2284 (AG)

Sur les datasets pi-12, les deux heuristiques parviennent à égaler la valeur optimale exacte, ce qui témoigne d'une excellente efficacité sur ces tailles intermédiaires. Sur pi-12-10000, seul Tabou atteint l'optimum exact, tandis que l'algorithme a dû avoir une convergence prématurée ou d'un manque d'exploration dans une population trop restreinte. Le temps de résolution est remarquablement faible comparé aux métaheuristiques.

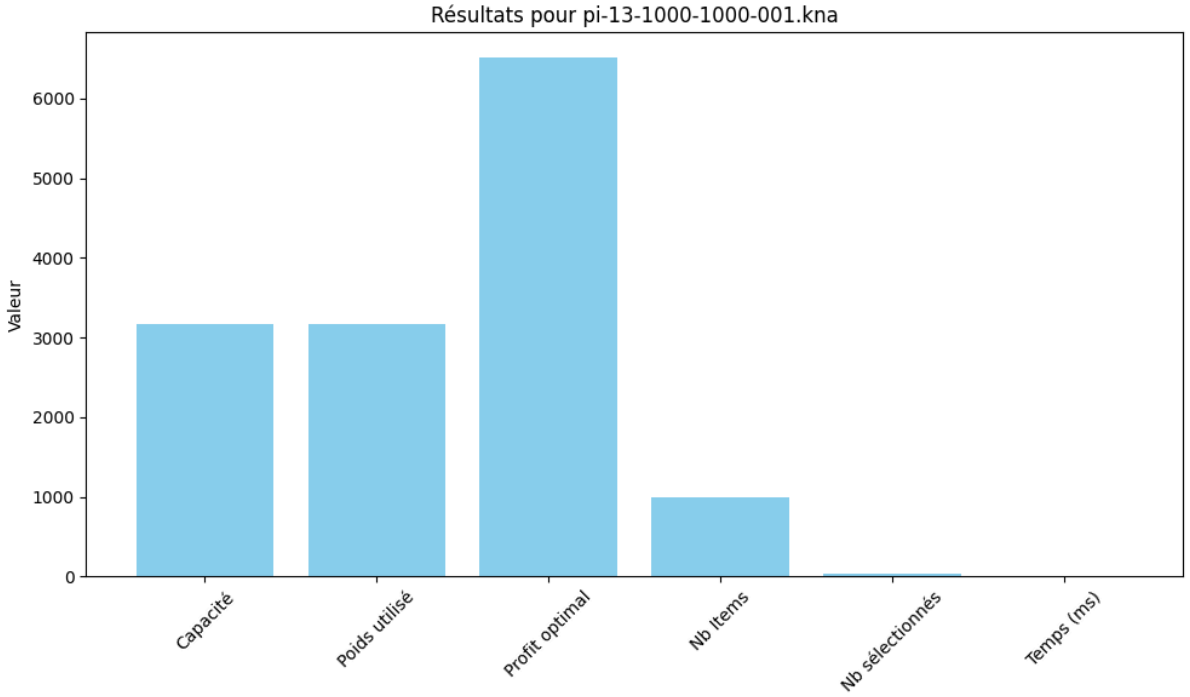
b. Expérimentations et Résultats (Datasets de pi-12)

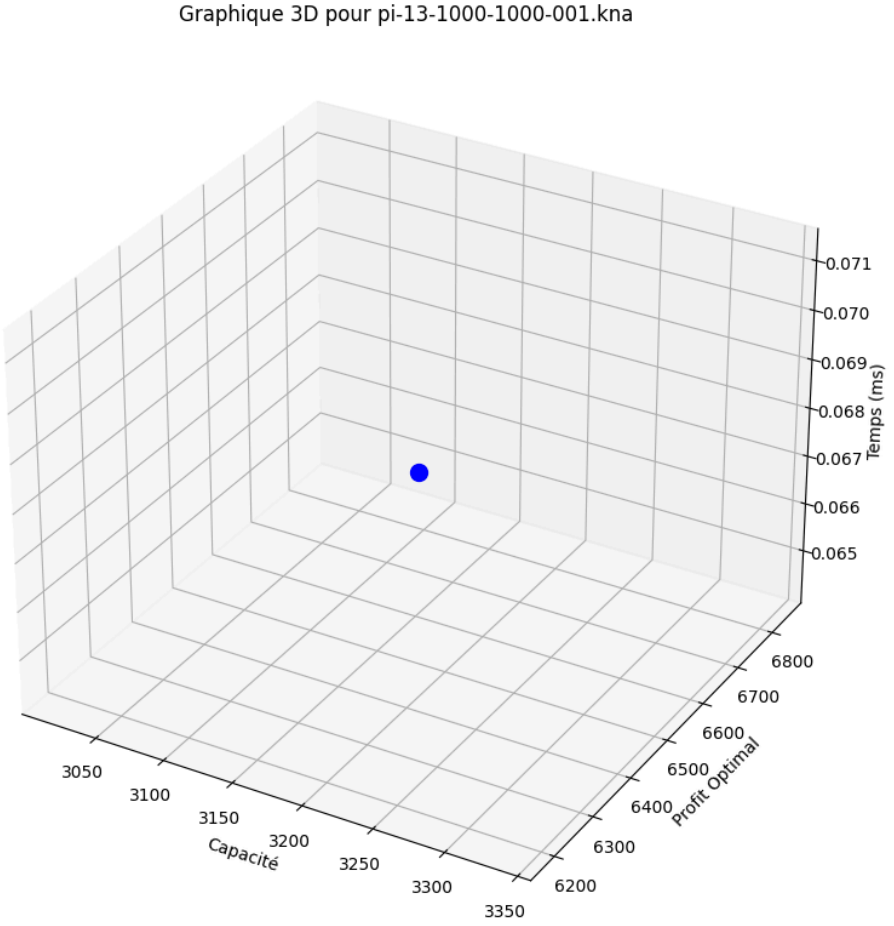
1. pi-13-100-1000-001



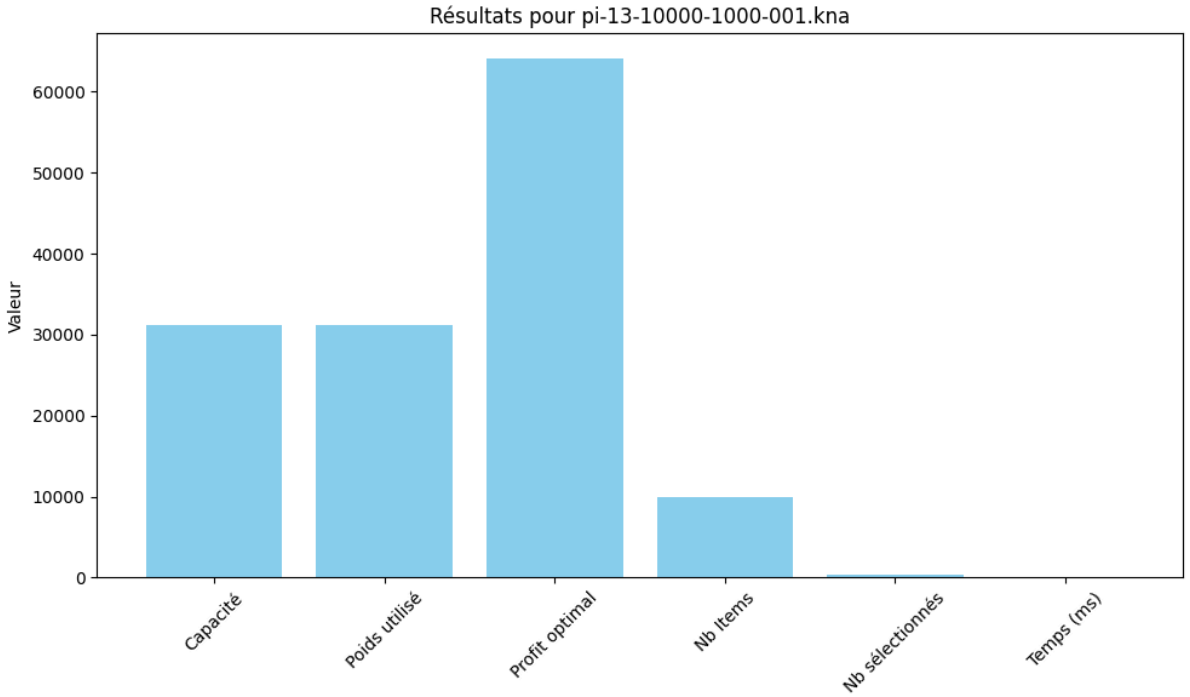


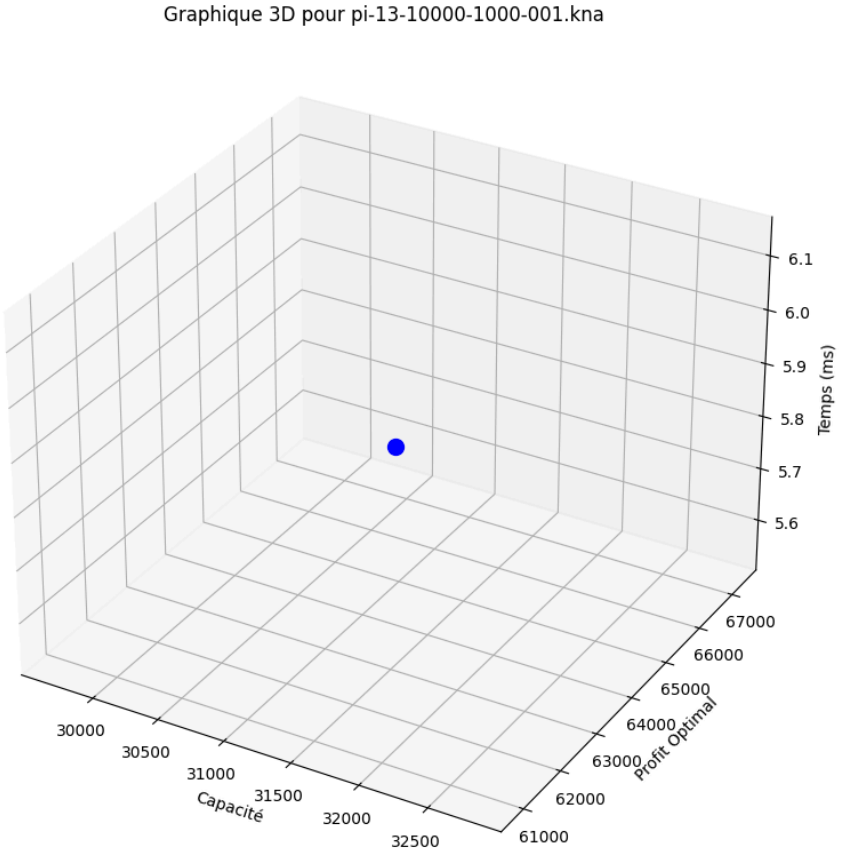
2. pi-13-1000-1000-001





3. pi-13-10000-1000-001





Résumé des performances :

Dataset	Capacité	Profit Optimal	Poids utilisé	Nb sélectionnés	Temps (ms)
pi-13-100	970	1989	969	9	0.03
pi-13-1000	3177	6513	3173	32	0.07
pi-13-10000	31234	64077	31217	348	5.84

Ces valeurs sont les **solutions optimales globales** garanties par le solveur CP-SAT, ce qui les rend idéales comme **référence** pour évaluer les performances des algorithmes heuristiques.

Comparaison avec les métaheuristiques :

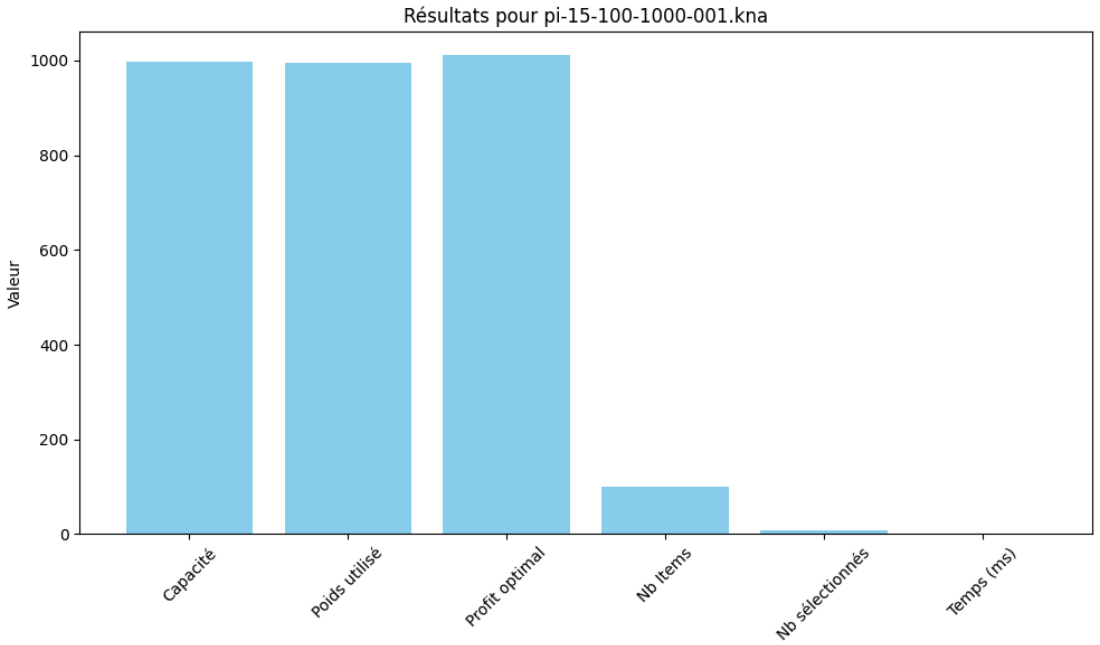
Dataset	CP-SAT	AG	Tabou	Ecart max
pi-13-100	1989	✅ 1989	❌ 1170	-819
pi-13-1000	6513	❌ 5694	❌ 4056	-2457
pi-13-10000	64077	❌ 44811	❌ 37830	-26247

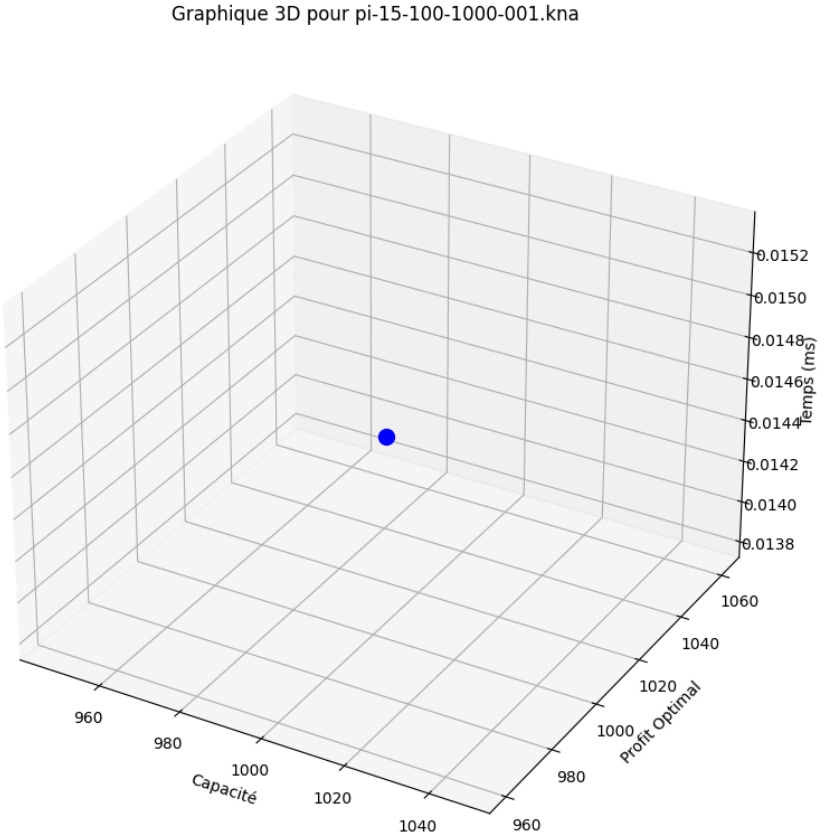
Sur pi-13-100, seul l'algorithme génétique parvient à atteindre l'optimum exact, tandis que la méthode Tabou échoue largement à cause d'une stagnation précoce. Tandis que sur pi-13-1000 et 10000, aucune métaheuristique n'approche le résultat optimal. L'écart atteint plusieurs milliers d'unités de profit, ce qui met en évidence la complexité structurelle des instances pi-13, probablement dues à une densité ou à une distribution de profits/poids plus difficile.

Ces datasets semblent être plus discriminant en matière de performance heuristiques, seule la méthode PL parvient à obtenir l'optimum, ce qui justifie pleinement l'importance d'une phase de validation exacte.

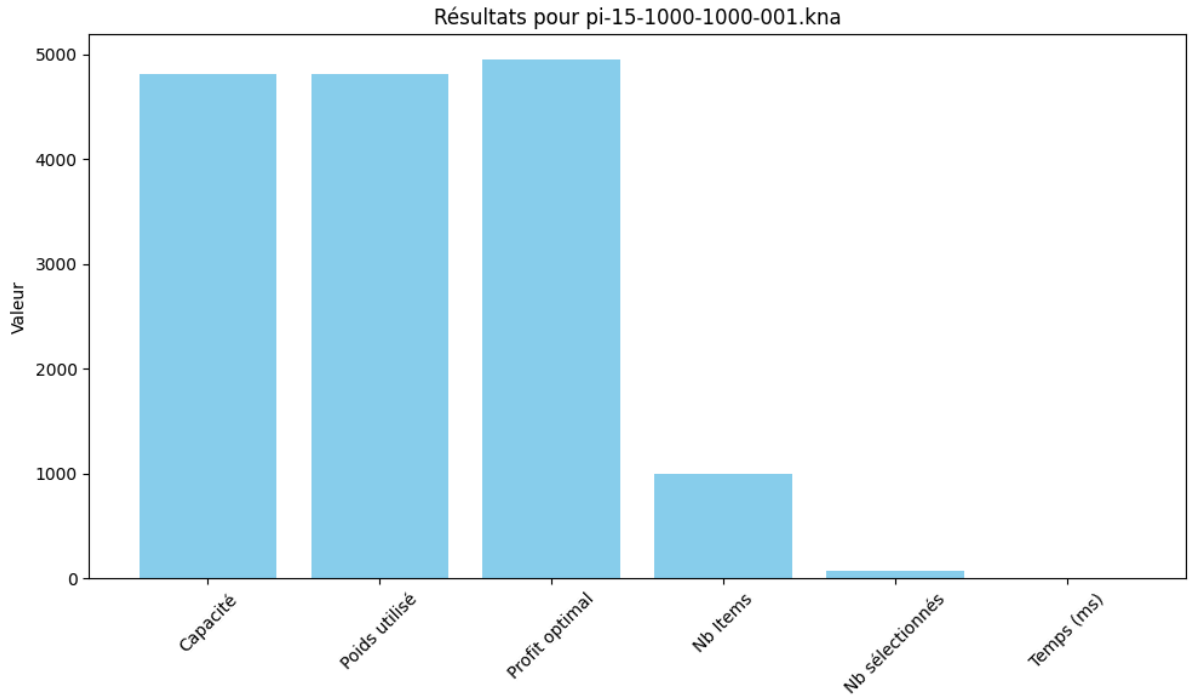
c. *Expérimentations et Résultats (Datasets de pi-12)*

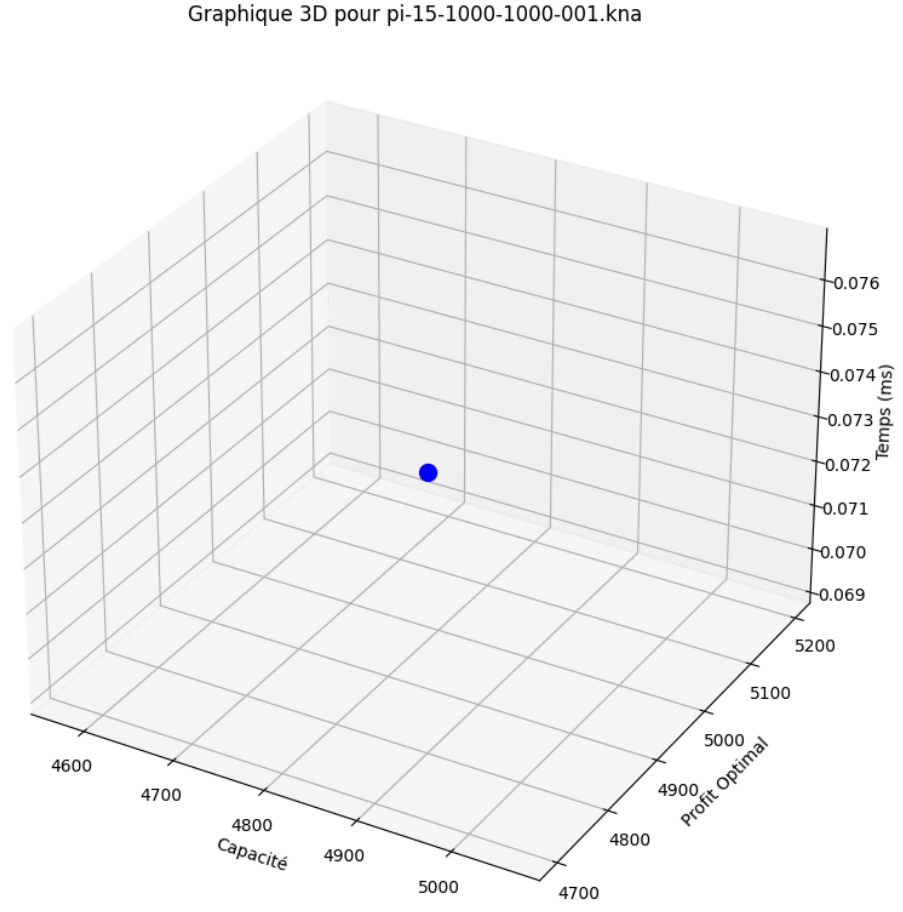
1. pi-15-100-1000-001



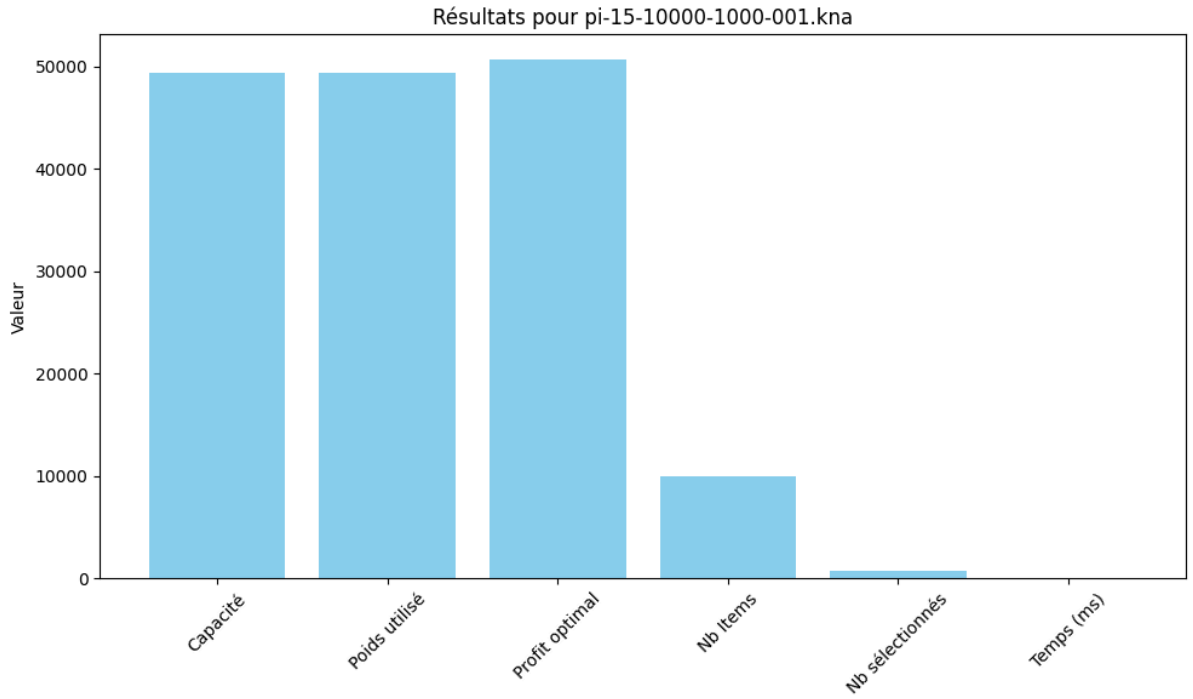


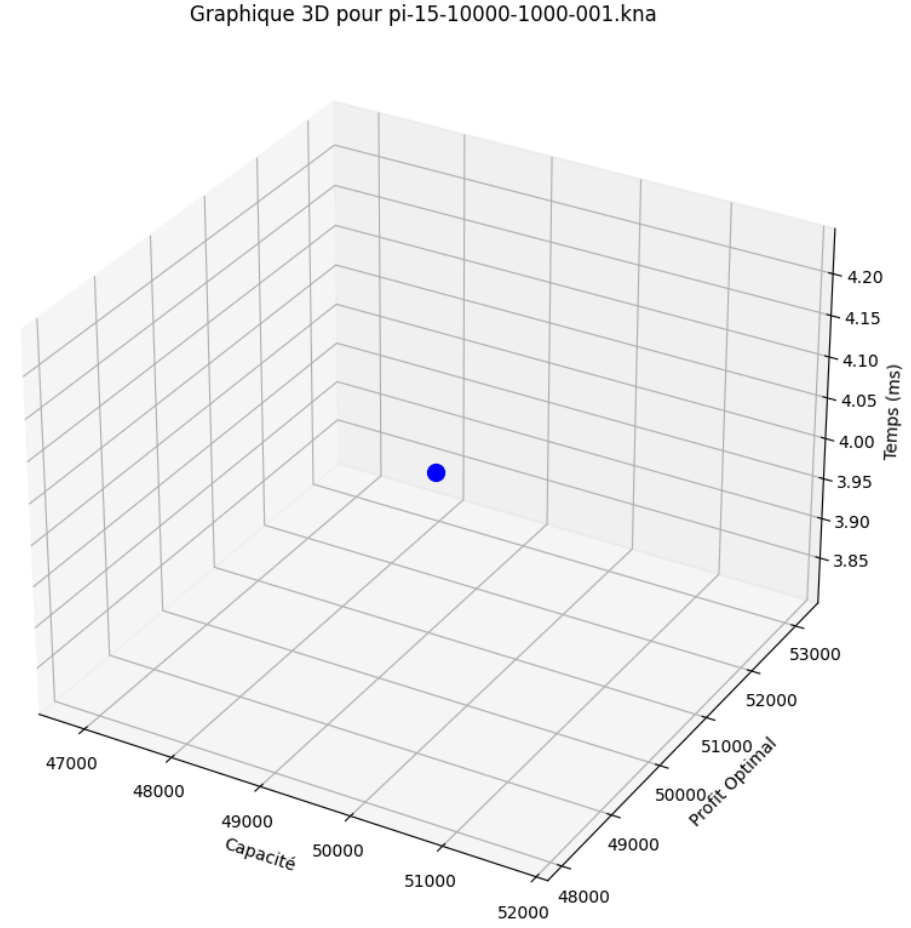
2. pi-15-1000-1000-001





3. pi-15-10000-1000-001





Résumé des performances :

Dataset	Capacité	Profit Optimal	Poids utilisé	Nb sélectionnés	Temps (ms)
pi-15-100	1000	1011	998	8	0.0144
pi-15-1000	4800	4950	4795	58	0.0711
pi-15-10000	49500	50460	49297	402	4.06

Comparaison avec les métaheuristiques :

Dataset	CP-SAT	AG	Tabou	Ecart max
pi-15-100	1011	❌ 1008	❌ 999	-12
pi-15-1000	4950	❌ 4514	❌ 4827	-436
pi-15-10000	50622	❌ 42821	❌ 49468	-7801

La programmation linéaire via OR-Tools domine en termes de performance, rapidité et exactitude. Les métaheuristiques quant à elles, restent très utiles dans un cadre plus exploratoire ou lorsque la structure du problème empêche une modélisation exacte, avec un avantage à Tabou pour sa régularité et sa convergence rapide par rapport à l'algorithme génétique.

Conclusion Générale

Ce rapport a proposé une étude comparative approfondie de trois approches de résolution du **problème du sac à dos** sur des instances de tailles variées (100, 1000 et 10000 objets) issues des jeux de données pi-12/13/15.
L'objectif était de comparer les **performances**, la **qualité des solutions**, les **temps d'exécution** et la **stabilité des résultats** selon les méthodes.

Métaheuristiques : Algorithme Génétique et Méthode Tabou

Les **métaheuristiques** se sont révélées pertinentes pour explorer de vastes espaces de solutions, mais avec des comportements différents :

- L'**algorithme génétique**, malgré sa souplesse et sa capacité d'exploration, **nécessite un réglage fin des paramètres** (taille de population, mutation, croisement) et montre une **grande variabilité** selon les jeux de données. Il peut parfois produire des solutions dégradées sans convergence stable.

- La **méthode Tabou**, quant à elle, s'est montrée **plus régulière**, offrant des solutions **de bonne qualité avec des temps raisonnables**. Sa convergence rapide dans les premières itérations est un avantage, bien que l'amélioration puisse stagner prématurément selon la taille de la liste tabou et la structure du voisinage.

Les résultats des tests ont confirmé que **Algorithme Génétique dépasse souvent la méthode Tabou** sur les jeux moyens à grands, tant en termes de **qualité** que de **temps d'exécution**.

Programmation Linéaire : Modélisation exacte avec OR-Tools

La **programmation linéaire**, modélisée via le solveur **CP-SAT d'OR-Tools**, a permis d'obtenir des **solutions optimales exactes** en **temps quasi-instantané**, y compris pour les jeux contenant **jusqu'à 10 000 objets**.

Elle surpasse largement les deux métaheuristiques, tant en **qualité de solution (valeur maximale)** qu'en **temps de traitement (quelques millisecondes)**. La seule contrainte reste la nécessité de disposer d'un problème **formellement exprimable en termes linéaires**, ce qui est parfaitement adapté au Knapsack.