Département de génie informatique et de génie logiciel

INF8175

Intelligence artif.: Méthodes et algorithmes

Projet Abalone

Présenté par:

7 Decembre 2023

Table des matières

Liste des tableaux

Listes des figures

Introduction

Ce rapport décrit le développement d'un agent intelligent jouant à Abalone, en présentant la méthodologie adoptée, l'évolution de l'agent au fil des itérations successives, et une discussion complète de ses forces et de ses limites. Les choix de conception de l'agent sont basés sur des principes fondamentaux, l'accent étant mis sur les algorithmes MiniMax, les améliorations heuristiques et les considérations stratégiques du jeu. Les résultats montrent la progression de l'agent à travers des tests contre des versions antérieures et des implémentations alternatives. La discussion qui s'ensuit fournit une analyse nuancée des capacités de l'agent, offrant un aperçu de son efficacité et des domaines susceptibles d'être améliorés. Les sections suivantes offrent une vue d'ensemble du parcours de développement de notre agent.

Méthodologie

Pour développer notre agent, nous avons adopté une approche incrémentale, en commençant par l'implémentation d'un agent simple utilisant l'algorithme de recherche MiniMax de base, jusqu'à ce que nous arrivions à notre agent le plus performant.

1. MiniMax

Lors de la première étape, nous avons implémenté l'algorithme MiniMax de base. Cet algorithme explore l'ensemble de l'arbre de jeu pour trouver le coup optimal en évaluant récursivement le résultat de chaque coup possible. Cependant, sans aucune optimisation, il est très coûteux en termes de calcul, en particulier pour un jeu comme Abalone avec un facteur de branchement important.

2. MiniMax avec élagage Alpha-Bêta

Pour remédier à la complexité de calcul de l'algorithme MiniMax de base, nous avons incorporé l'élagage alpha-bêta. L'élagage alpha-bêta est une technique qui réduit le nombre de nœuds évalués dans l'arbre de recherche en éliminant les branches qui ne peuvent pas affecter la décision finale. Cette optimisation réduit considérablement l'espace de recherche, ce qui rend l'algorithme plus efficace..

3. Heuristique MiniMax (stratégie de type A)

3.1. Objectif

Étant donné qu'il n'est pas toujours possible d'explorer l'ensemble de l'arbre de jeu, nous avons introduit une fonction d'évaluation heuristique pour estimer l'état du plateau de jeu à une profondeur fixe. Ce type de stratégie permet à notre agent de prendre des décisions basées sur une anticipation partielle, ce qui améliore l'efficacité de l'algorithme tout en maintenant des performances raisonnables.

3.2. Principes de l'heuristique utilisée

Dans le jeu Abalone, un agent intelligent a besoin d'une fonction d'évaluation robuste pour évaluer la désirabilité des différents états du jeu. L'heuristique implémentée, dénommée "***abalone_heuristic_version_1***", sert de composant clé dans ce processus d'évaluation. L'heuristique est conçue pour attribuer une note numérique à un état de jeu donné, en tenant compte de divers éléments stratégiques.

a. Score initial

Le point de départ de l'heuristique est le score du joueur actuel, obtenu par la fonction ***"current_state.get_player_score(player)"***. Ce score initial reflète la position actuelle du joueur en fonction du nombre de billes sur le plateau.

b. Bonus et Pénalités

i. La fonction ***"self.get_bonus_for_being_part_of_a_cluster(current_state, player)"*** calcule un bonus pour le joueur en fonction de sa participation à des groupes de billes. Cela incite à placer les billes de façon stratégique pour former des groupes cohérents.

ii. La pénalité pour être sur les bords est déterminée par ***"self.get_penalty_for_being_on_edges(current_state, player)"***. Cela pénalise les billes positionnées sur les bords du plateau de jeu, encourageant les joueurs à maintenir une présence plus centrale.

iii. La fonction ***"self.get_bonus_for_being_close_to_center(current_state, player)"*** calcule un bonus pour les billes positionnées près du centre du plateau. Cela encourage les joueurs à contrôler les régions centrales pour obtenir des avantages stratégiques.

iv. La mobilité d'un joueur est récompensée par ***"self.get_bonus_for_mobility(current_state)"***. Cette composante reflète la prise en compte par l'heuristique de la capacité du joueur à exécuter une variété de mouvements, améliorant ainsi la flexibilité stratégique globale.

c. Scénarios Max et Min

L'heuristique adapte son évaluation en fonction du type de joueur (maximisateur ou minimisateur). Pour le joueur qui maximise ("Max"), les bonus et les pénalités calculés sont ajoutés au score initial. Inversement, pour le joueur minimisateur ('Min'), les mêmes valeurs sont soustraites. Cette approche adaptative garantit que l'heuristique s'aligne sur les objectifs et stratégies distincts de chaque type de joueur.

d. Score final

Le point culminant du processus heuristique est le score final, qui représente l'évaluation complète de l'état du jeu. Ce score englobe les éléments stratégiques de regroupement, de positionnement sur le plateau et de mobilité, fournissant à l'agent une compréhension nuancée de l'opportunité de la configuration actuelle du jeu.

4. Heuristique MiniMax (stratégie de type A) avec tables de transposition.

Pour améliorer encore les performances, nous avons introduit des tables de transposition. Les tables de transposition stockent les positions précédemment calculées et leurs évaluations, évitant ainsi les calculs redondants. Cet ajout permet de gérer les situations dans lesquelles différentes séquences de mouvements conduisent au même état du plateau, réduisant ainsi la charge de calcul.

5. Heuristique MiniMax (stratégie de type B)

5.1. Objectif

Dans la dernière étape, nous avons étendu notre agent en introduisant une heuristique à largeur fixe (stratégie de type B). Cette stratégie consiste à évaluer un nombre fixe de meilleurs coups à chaque niveau de l'arbre de jeu, plutôt que d'évaluer tous les coups possibles à une profondeur fixe. Cette optimisation peut être bénéfique dans les situations où l'heuristique basée sur la profondeur n'est pas suffisante, car elle prend en compte un éventail plus large de possibilités.

5.2. Principes de l'heuristique utilisée

Nous avons utilisé une heuristique améliorée, dénommée ***"abalone_heuristic_version_2"***. Elle s'appuie sur les fondations de son prédécesseur (***balone_heuristic_version_1***) et introduit plusieurs raffinements pour mieux capturer les subtilités de la stratégie de jeu.

a. Critères d'évaluation améliorés

i. *Version 1:* L'heuristique originale se concentrait sur des éléments essentiels tels que le regroupement, le positionnement des bords, la proximité du centre et la mobilité, fournissant ainsi une évaluation holistique de l'état du jeu.

ii. *Version 2:* S'appuyant sur cette base, la nouvelle heuristique introduit un ensemble élargi de critères d'évaluation. En particulier, elle prend désormais en compte la situation dans laquelle le joueur actuel a la possibilité de déloger la bille de son adversaire. Cet ajout permet une meilleure attaque qui peut avoir un impact significatif sur l'issue du jeu. Ce critère remplace le critère de mobilité utilisé précédemment, car ce dernier est très coûteux en temps de calcul, puisqu'il fait appel à la méthode "***get_possible_actions***", ce qui ralentissait fortement notre agent de recherche.

b. Évaluation centrée sur l'adversaire

i. *Version 1:* La version initiale prenait principalement en compte les éléments de score et de stratégie liés au joueur actuel.

ii. *Version 2:* La version 2 incorpore une évaluation centrée sur l'adversaire. Elle applique des bonus et des pénalités à l'état de l'adversaire tels que le regroupement, le positionnement des bords et la proximité du centre, reconnaissant l'importance de comprendre et de répondre à la position de l'adversaire sur le plateau.

c. Impact pondéré de l'expulsion d'une bille de l'adversaire

i. *Version 1:* L'heuristique originale ne considérait pas le déplacement de l'adversaire.

ii. *Version 2:* La version 2 accorde un poids substantiel aux scénarios dans lesquels le joueur actuel peut déplacer la bille de son adversaire. La pénalité pondérée (****50***) reflète l'impact potentiel sur le jeu de l'élimination stratégique des pièces de l'adversaire, ajoutant une couche de profondeur à l'heuristique.

Évolution de l’agent et résultats

1. Évolution de l’agent

1.1. MiniMax

L'implémentation initiale s'appuyait sur l'algorithme fondamental MiniMax pour explorer tous les mouvements possibles, garantissant une recherche exhaustive du mouvement optimal dans le cadre des contraintes de profondeur données. Cependant, la nature complète de l'algorithme s'est avérée dépasser les limites supérieures du jeu.

1.2. MiniMax avec élagage Alpha-Bêta

Reconnaissant le besoin d'efficacité de calcul, l'algorithme a été amélioré avec l'alpha-bêta, une optimisation critique qui réduit théoriquement le nombre de nœuds évalués dans l'arbre de jeu. Cependant, même avec cette réduction, le résultat dépassait encore la limite supérieure de 15 minutes.

1.3. Heuristique MiniMax (stratégie de type A)

Le processus d'affinage s'est poursuivi par l'incorporation d'une fonction d'évaluation heuristique itérative à une profondeur fixe (stratégie de type A). Cette approche itérative a permis à l'heuristique de s'adapter et d'améliorer ses approximations au cours des itérations successives, améliorant ainsi le processus de prise de décision. Notamment, dans la version finale, l'heuristique a été perfectionnée par l'ajout et aussi la suppression de certains bonus et pénalités. Ces optimisations des conditions heuristiques prennent leur source des tests rigoureux pour extrapoler des interprétations telles que notre agent est trop "défensif" ou trop "agressif" pour s’aligner le plus avec un joueur décrit scientifiquement comme “optimal”.

1.4. Heuristique MiniMax (stratégie de type A) avec tables de transposition.

Afin d'éliminer la redondance dans le processus de recherche et d'augmenter les performances, des tables de transposition ont été introduites. Ces tables stockent les positions précédemment évaluées et les scores associés, ce qui évite de réévaluer des états de jeu identiques et améliore l'efficacité globale du calcul. Les résultats de cette mise en œuvre ont été très concluants compte tenu de l'augmentation pure et simple des performances.

1.5. Heuristique MiniMax (stratégie de type B)

L'ultime évolution de la méthodologie a incorporé une approche différente par l'introduction d'une heuristique à largeur limitée (stratégie de type B). Cette stratégie s'écarte des limitations du strategies A et se concentre sur l'exploration d'un nombre fixe de branches les plus intéressantes. Cette approche a été inspirée par le constat que le jeu humain consiste souvent à limiter les branches initiales envisagées plutôt que d'évaluer toutes les possibilités imaginables. Dans notre cas, en utilisant la stratégie A, une profondeur maximale de 3 était un seuil que nous cherchions à dépasser. Avec cette nouvelle implémentation, les résultats ont permis d'atteindre une nouvelle profondeur de 5 en ajustant les conditions heuristiques et en limitant le nombre de branches à 15.

2. Résultats

La phase d'évaluation de notre projet se décompose en deux volets distincts : la précision évaluée en termes de victoires et la performance évaluée en temps et en nombre de cas évalués. Il est important de noter qu'étant donné les résultats dépassant les limites supérieures pour le premier et le deuxième algorithme, leur précision et leur performance ne peuvent pas être considérées dans le même contexte que les autres.

2.1. Précision

En ce qui concerne la précision, nous mesurons l'efficacité de notre bot en analysant son taux de victoire dans le contexte du jeu Abalone. Nous évaluons sa capacité à prendre des décisions stratégiques et à s'adapter aux scénarios de jeu variés, en mettant l'accent sur la maximisation des résultats positifs.

Dans cette comparaison, l'agent affiche une égalité lorsqu'il est confronté à lui-même, mais se démarque nettement lorsqu'il est mis en compétition avec ses prédécesseurs. Comme illustré dans le tableau ci-dessous, le cinquième algorithme se distingue par des performances supérieures à celles de ses prédécesseurs, s'érigeant en vainqueur et consolidant sa position en tant qu'agent le plus précis parmi ceux que nous avons développés.

	Algorithme #3	Algorithme #4	Algorithme #5
Algorithme #3	Égalité	Algorithme #4 Plus performant	Algorithme #5 Vainqueur
Algorithme #4	Algorithme #4 Plus performant	Égalité	Algorithme #5 Vainqueur
Algorithme #5	Algorithme #5 Vainqueur	Algorithme #5 Vainqueur	Égalité

Table 1. Comparaison de la précision des algorithmes

Cette représentation tabulaire met en lumière la nette supériorité de l'Algorithme #5, tant en termes de performances que de succès face à ses prédécesseurs.

2.2. Performance

Parallèlement, la performance de notre agent est évaluée en termes de temps d'exécution et du nombre de coups explorés. Cette mesure nous offre un aperçu de l'efficacité opérationnelle de notre algorithme, mettant en lumière sa rapidité d'analyse et sa capacité à traiter des situations complexes dans un laps de temps défini.

Il existe plusieurs façons d'évaluer les performances des algorithmes. Une perspective initiale consistait à évaluer le jeu en fonction du temps moyen par tour. Cependant, même en utilisant une logique clairement optimisée à cet égard, nous constatons peu ou pas de changement sur cet aspect entre les algorithmes 3 et 4, comme le montre le tableau à droite.

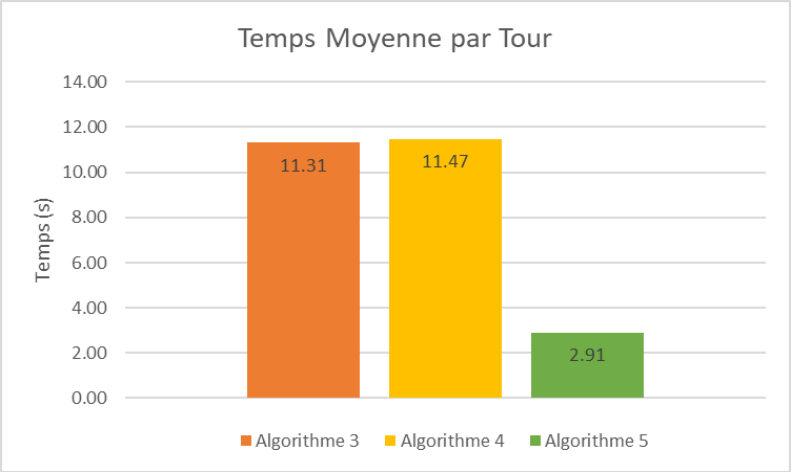

Figure 1. Comparaison des temps moyennes par tour

La véritable optimisation devient évidente lorsqu'on examine la métrique du temps par tour, comme le montre le graphique ci-dessous. Bien que des sommets soient associés au temps d'exécution du hachage Zobrist, les périodes en dehors de ces pics deviennent incroyablement efficaces.

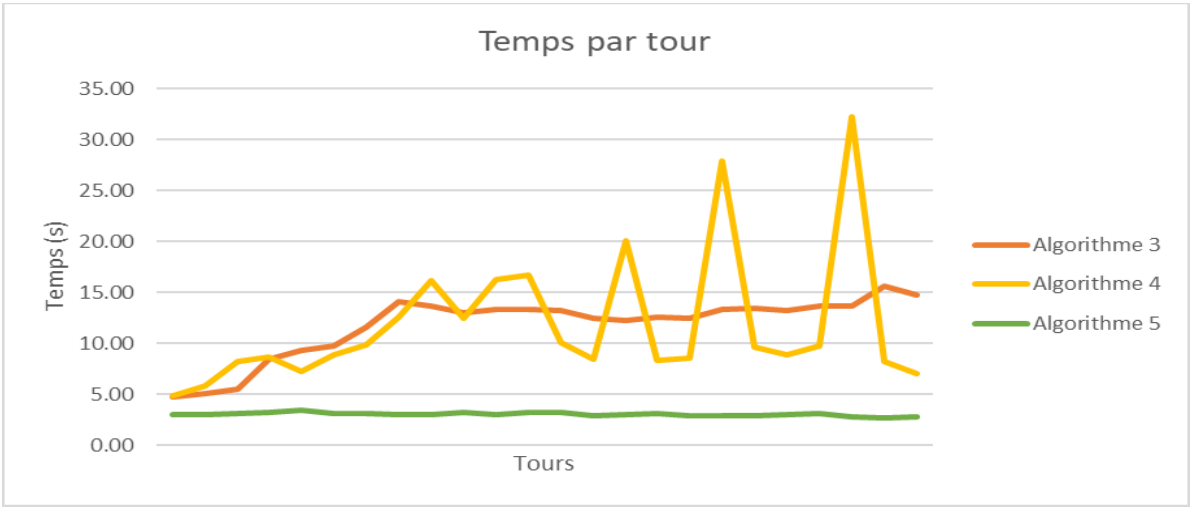

Figure 2. Comparaison des temps d'exécution par tour

Ce qui devient le plus apparent, c'est l'implémentation du cinquième algorithme. Toutes les métriques de performance sont optimisées, et ce qui est également intéressant, c'est la tendance de l'algorithme à devenir plus rapide à mesure que le jeu progresse, tandis que l'algorithme 4 ralentit simplement l'augmentation du temps en comparaison avec le troisième. Cette observation met en évidence une caractéristique notable de l'algorithme 5 : sa capacité à s'adapter et à accélérer ses performances à mesure que le jeu évolue

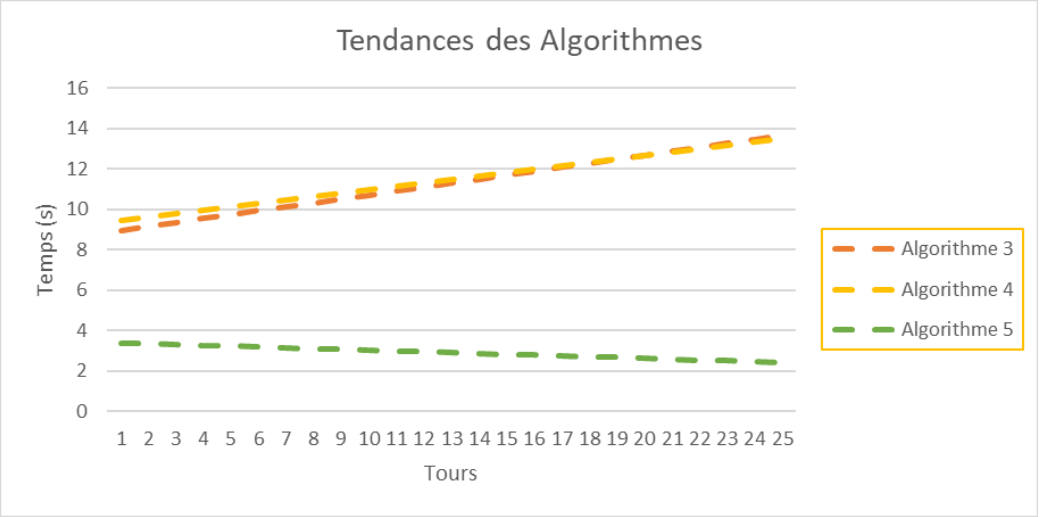

Figure 3. Comparaison des tendances linéaires des différents algorithmes

Discussion

L'itération finale de l'agent de jeu Abalone, incorporant la version 2 de l'heuristique, MiniMax avec des stratégies de type A et de type B, et l'élagage alpha-bêta, démontre des forces notables dans la prise de décision stratégique et l'efficacité de calcul. L'adaptabilité de l'agent est renforcée par la version 2 de l'heuristique, qui introduit un ensemble enrichi de critères d'évaluation, y compris des considérations centrées sur l'adversaire et des bonus pondérés pour le déplacement des billes de l'adversaire. La combinaison de MiniMax avec des stratégies à profondeur fixe (type A) et à largeur fixe (type B) permet à l'agent d'équilibrer la profondeur de l'exploration et l'étendue des possibilités, ce qui contribue à une sélection nuancée et bien informée des mouvements. L'intégration de l'élagage alpha-bêta améliore considérablement l'efficacité des calculs, permettant à l'agent d'explorer l'arbre de jeu plus efficacement.

Toutefois, des domaines d'amélioration méritent d'être pris en considération. Les performances de l'agent dépendent de la précision de la fonction heuristique qui peut introduire des biais ou des oublis. D'après nos tests, nous avons remarqué que notre agent est très souvent sur la défensive et n'attaque que si les billes de l'adversaire sont au bord du plateau. Un raffinement de l'heuristique et de ses paramètres pourrait améliorer et diversifier les stratégies d'attaque de l'agent. En outre, l'efficacité informatique de l'agent, bien qu'optimisée par l'élagage alpha-bêta, et les stratégies de type A et B, peut encore se heurter à des difficultés dans des états de jeu plus complexes ou avec des ressources informatiques limitées. L'exploration d'optimisations supplémentaires ou de techniques de parallélisation pourrait remédier à ces limitations.

Références

Chorus, Pascal. “Implementing a Computer Player for Abalone Using Alpha-Beta and Monte-Carlo Search.” Maastricht University, 2009.

Annexes

# ALGORITHM	# ROUND	EXECUTION TIME	NUMBER OF EXPLORED CELLS
1	1	> 15 mn	N/A
Moyenne		**N/A**	**N/A**
2	1	> 15 mn	N/A
Moyenne		**N/A**	**N/A**
3	1	4.78	5310
	2	5.09	5327
	3	5.53	5593
	4	8.39	7976
	5	9.35	8808
	6	9.70	8639
	7	11.63	10140
	8	14.10	12065
	9	13.67	11556
	10	12.95	11112
	11	13.37	11980
	12	13.36	12502
	13	13.23	12174
	14	12.49	11654
	15	12.21	11588
	16	12.56	11196
	17	12.47	11063
	18	13.36	11636
	19	13.47	11912
	20	13.16	11435
	21	13.62	11951

	22	13.62	12190
	23	15.64	12796
	24	14.71	12760
	25	0.17	204
Moyenne		**11.31**	**10143**
4	1	4.89	5280
	2	5.79	6021
	3	8.25	7207
	4	8.66	7794
	5	7.24	7837
	6	8.91	8342
	7	9.88	8404
	8	12.57	10149
	9	16.20	12103
	10	12.50	12657
	11	16.25	11642
	12	16.68	11222
	13	10.09	10942
	14	8.40	9316
	15	20.07	10783
	16	8.27	9057
	17	8.58	9354
	18	27.88	10713
	19	9.58	10040
	20	8.82	9655
	21	9.72	10882
	22	32.22	10893

	23	8.21	9650
	24	6.97	10053
	25	0.13	178
Moyenne		**11.47**	**9207**
5	1	3.00	733
	2	3.02	733
	3	3.15	733
	4	3.25	733
	5	3.42	733
	6	3.12	733
	7	3.10	733
	8	3.01	733
	9	3.01	733
	10	3.20	733
	11	3.00	733
	12	3.21	733
	13	3.19	733
	14	2.94	733
	15	2.96	733
	16	3.05	733
	17	2.86	733
	18	2.94	733
	19	2.92	733
	20	3.03	733
	21	3.06	733
	22	2.75	733
	23	2.67	733

	24	2.77	733
	25	0.15	45
Moyenne		**2.91**	**705**

Table 2. Données pour comparaison d'algorithmes

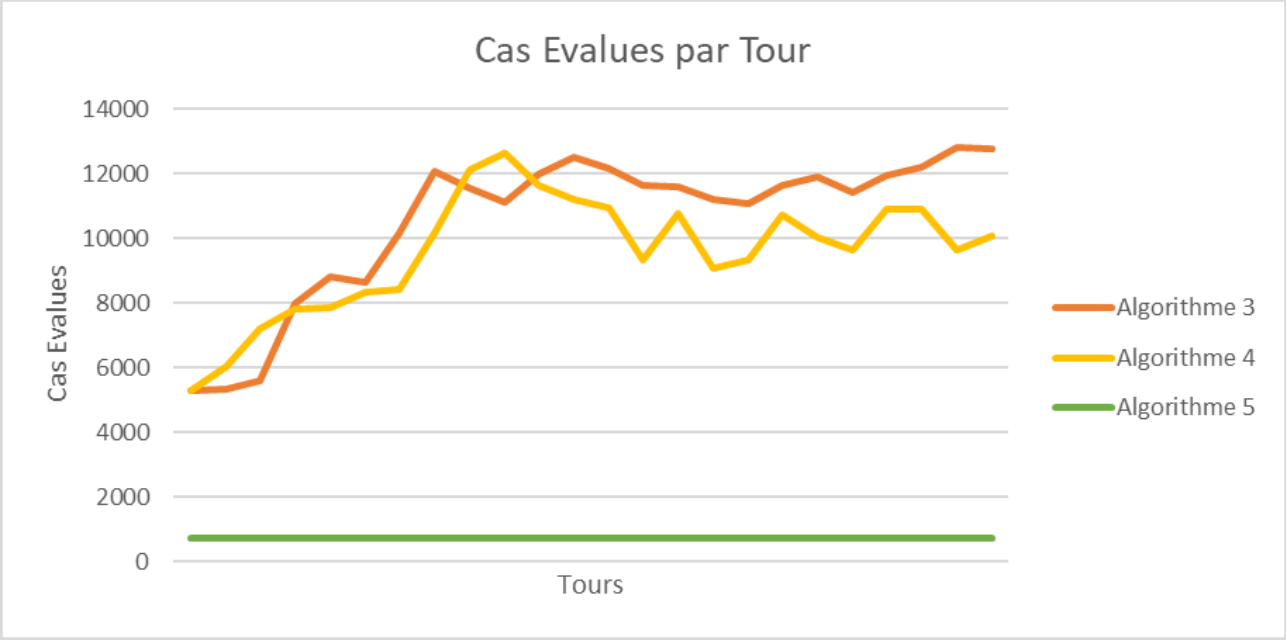

Figure 4. Comparaison des cas évalués par tour pour chaque algorithm

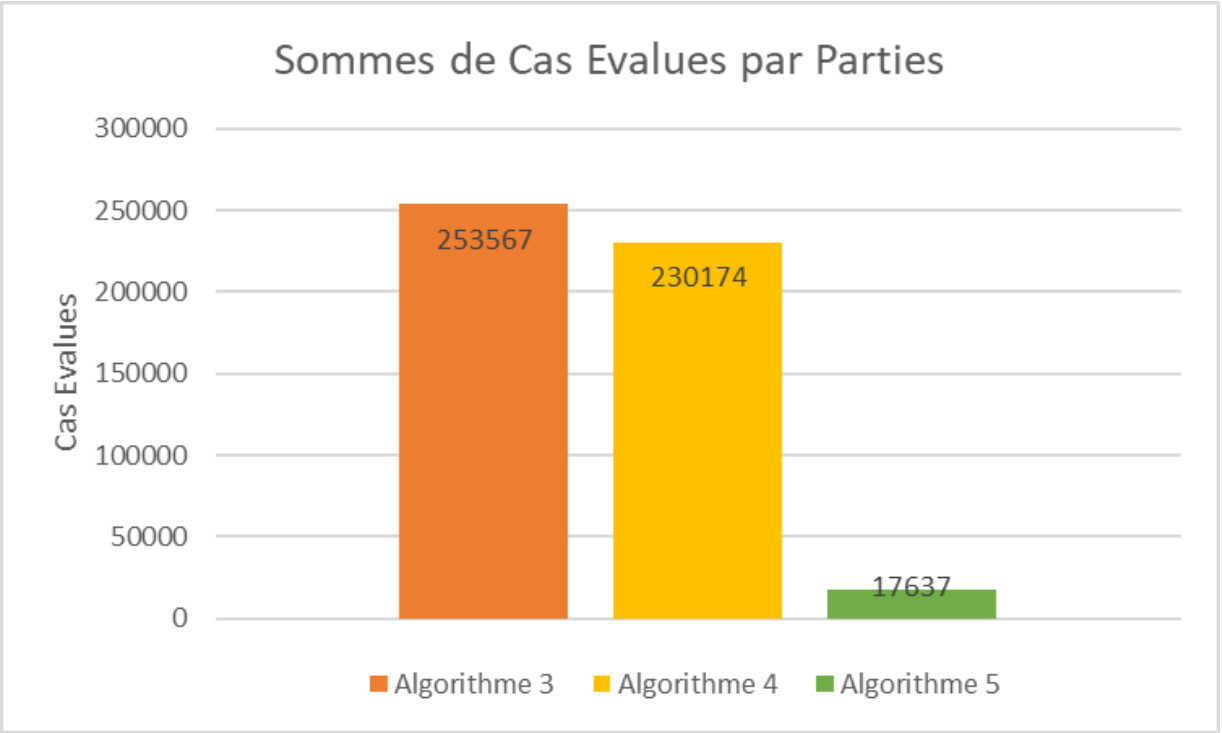

Figure 5. Comparaison des sommes de cas évalués par chaque algorithm

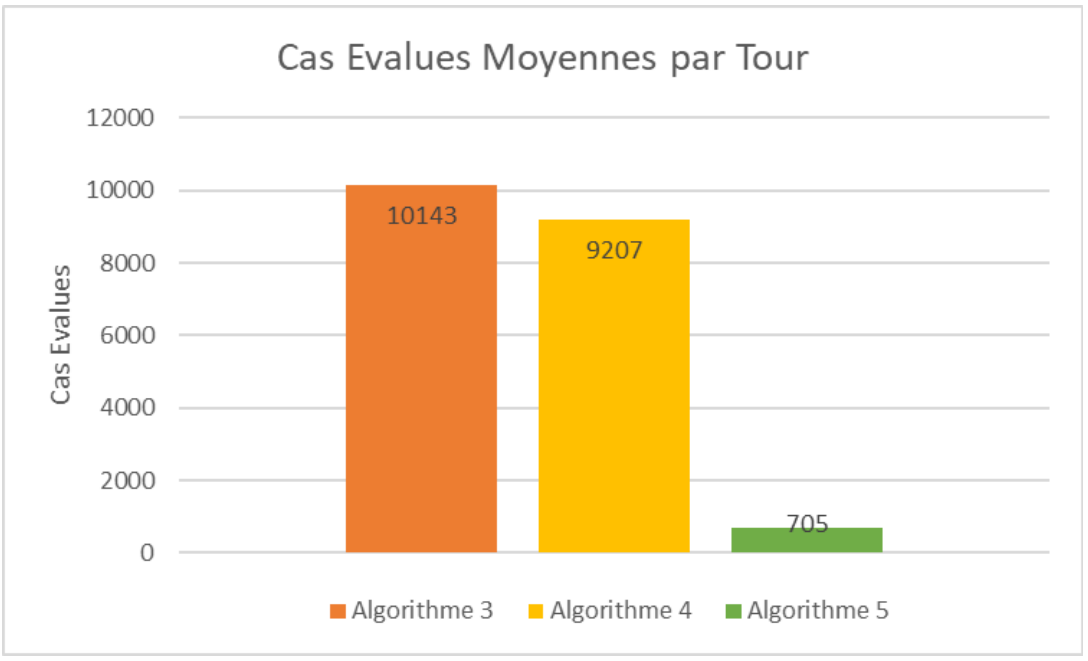

Figure 6. Comparaison du nombre de cas évalués moyennes par tour