République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique

Département Informatique

Spécialité : Bio-Informatique

Projet Module Eléments d'Optimisation Combinatoir

Thème

Problème des sous-séquences commune les plus longue

Réalisé par :

LAIB Ayoub

Table des matières

Table des figures

1. Introduction

1.1. Contexte du Projet

Le présent projet s'inscrit dans le domaine de l'informatique algorithmique et de l'optimisation, où la résolution efficace de problèmes complexes revêt une importance.

L'un de ces problèmes, la plus longue sous-séquence commune (LCS), joue un rôle essentiel dans divers domaines, de la bioinformatique à la compression de données. Par exemple, en bioinformatique, le LCS est utilisé pour aligner des séquences génétiques et identifier des similitudes entre elles, ce qui est vital pour comprendre l'évolution génétique et concevoir des thérapies ciblées en médecine. De même, dans le domaine de la recherche pharmaceutique, la découverte de médicaments repose souvent sur l'analyse comparative de séquences de gènes. Enfin, dans le domaine de la compression de données, le LCS est utilisé pour identifier et éliminer les redondances, ce qui permet de compresser efficacement les données tout en préservant leur intégrité.

1.2. Objectifs du Projet

Ce projet vise à explorer et à comparer différentes approches algorithmiques pour résoudre le problème LCS, en mettant en œuvre à la fois des méthodes exactes et des métaheuristiques. Les objectifs spécifiques sont les suivants :

- Comprendre en profondeur le problème LCS et son importance dans des domaines tels que la bioinformatique, la recherche pharmaceutique et la compression de données.
- Modéliser le problème et développer un générateur de données pour créer des ensembles de chaînes variés, représentatifs des situations réelles.
- Implémenter une méthode exacte, telle que DFS ou BFS, pour obtenir des solutions optimales.
- Utiliser une métaheuristique, comme une affectation aléatoire, pour explorer efficacement l'espace de recherche et trouver des solutions de haute qualité dans un délai raisonnable.
- Expérimenter sur des ensembles de données variés pour évaluer les performances des approches en termes de temps d'exécution et de qualité de la solution.
- Comparer les résultats obtenus et tirer des conclusions sur les avantages et les limites des différentes approches, offrant ainsi des perspectives pour de futures recherches et applications.

1.3. Importance du Problème LCS

Le problème de la plus longue sous-séquence commune (LCS) revêt une importance significative dans plusieurs domaines, notamment :

- Bioinformatique : Utilisé pour aligner des séquences génétiques, identifier des similitudes entre des gènes et étudier l'évolution génétique.
- Recherche Pharmaceutique : Essentiel pour la découverte de médicaments en analysant les séquences de gènes et en identifiant des cibles thérapeutiques.
- Compression de Données : Utilisé pour identifier les redondances et optimiser la compression des données tout en préservant leur intégrité.

1.4. Aperçu des Méthodes de Résolution (Exacte et Heuristique)

Le problème LCS peut être résolu à l'aide de deux principales approches : les méthodes exactes et les méthodes heuristiques.

Les méthodes exactes, telles que DFS et BFS, visent à trouver la solution optimale en examinant toutes les possibilités de manière exhaustive. Cela garantit la précision de la solution, mais peut devenir impraticable pour de grandes instances du problème en raison de la complexité algorithmique.

En revanche, les méthodes heuristiques adoptent une approche plus rapide mais moins précise pour trouver des solutions de haute qualité dans un délai raisonnable. Ces approches exploitent des stratégies de recherche guidée par des règles heuristiques pour explorer l'espace de recherche de manière efficace, en acceptant parfois des solutions sous-optimales pour gagner en temps de calcul.

2. Définition et Modélisation du Problème

2.1. Description Formelle du Problème LCS

La plus longue sous-séquence commune (LCS) d'une chaîne de caractères est un ensemble de caractères qui apparaissent dans l'ordre de gauche à droite, mais pas nécessairement de manière consécutive.

ACTTGCG

"ACTTGCG", "ACT", "ATTC", "T", "ACTTGC" sont toutes des sous-séquences

"TTA" n'est pas une sous-séquence

Une sous-séquence commune de deux chaînes est une séquence qui apparaît dans les deux chaînes. Une plus longue sous-séquence commune est une sous-séquence commune de longueur maximale.

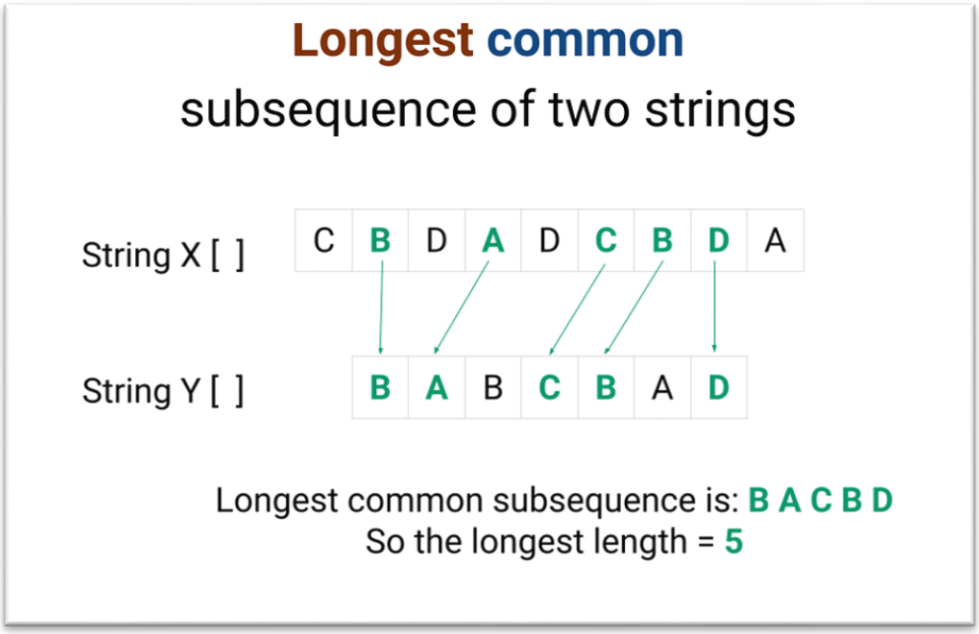

Figure 1: Exemple de la plus longue sous-séquence commune

2.2. Caractéristiques des Données d'Entrée et de Sortie

Le problème de la plus longue sous-séquence commune (LCS) prend en entrée deux chaînes de caractères, que nous nommerons "chaîne A" et "chaîne B". Ces deux chaînes représentent les données sur lesquelles l'algorithme LCS opère pour trouver la plus longue sous-séquence commune.

Les données d'entrée sont généralement des chaînes de caractères composées de lettres de l'alphabet, mais elles peuvent également inclure d'autres types de caractères, tels que des

chiffres ou des caractères spéciaux. La longueur de ces chaînes peut varier selon le contexte de l'application, allant de quelques caractères à plusieurs milliers.

La sortie attendue de l'algorithme LCS est une sous-séquence commune de longueur maximale entre les deux chaînes d'entrée. Cette sous-séquence commune peut être vide si aucune correspondance n'est trouvée entre les deux chaînes. Dans le cas où plusieurs sous-séquences communes de longueur maximale existent, l'algorithme peut en trouver une seule ou toutes, en fonction de l'approche de résolution utilisée.

Il est à noter que les données d'entrée peuvent varier en fonction du contexte de l'application. Par exemple, dans le domaine de la bioinformatique, les chaînes de caractères peuvent représenter des séquences génétiques, tandis que dans le traitement du langage naturel, elles peuvent représenter des phrases ou des documents textuels. Les caractéristiques des données d'entrée peuvent également influencer les choix d'implémentation et les performances de l'algorithme LCS.

3. Génération des Données

La génération des données constitue une étape importante dans notre projet sur le problème de la plus longue sous-séquence commune (LCS). Cette section vise à détailler les objectifs, la conception et les paramètres de variation du générateur de données que nous avons développé.

3.1. Objectifs de la Génération des Données

Les objectifs de la génération des données sont multiples :

- Créer des ensembles de données représentatifs de situations réelles pour évaluer les performances de nos algorithmes LCS.
- Permettre la variation de plusieurs paramètres, tels que la longueur des chaînes, le nombre de chaînes et la complexité de l'alphabet, afin d'explorer différents scénarios.
- Fournir des données diversifiées pour tester la robustesse de nos solutions face à des conditions variées.

3.2. Conception du Générateur de Données

Nous avons conçu un générateur de données capable de créer des ensembles de chaînes de caractères aléatoires en fonction des paramètres spécifiés. Le générateur est implémenté de manière à garantir la représentativité et la variabilité des données générées.

3.3. Paramètres de Variation des Données Générées :

Les paramètres de variation des données générées comprennent :

- **Longueur des Chaînes** : Nous permettons la spécification de la longueur des chaînes de caractères, ce qui nous permet d'évaluer les performances de nos algorithmes pour des chaînes de différentes tailles.
- **Nombre de Chaînes** : Nous pouvons générer des ensembles de données avec un nombre variable de chaînes, ce qui nous permet d'explorer les performances de nos algorithmes dans des situations avec différentes quantités de données.
- **Complexité de l'Alphabet** : Nous avons la possibilité de spécifier la complexité de l'alphabet, permettant ainsi de tester nos solutions dans des contextes avec des ensembles de caractères plus ou moins variés.

En combinant ces paramètres, nous sommes en mesure de générer une large gamme de données d'entrée pour évaluer nos algorithmes LCS dans des conditions diverses et représentatives.

3.4. Présentation des Ensembles de Données Utilisés

Les ensembles de données ont été générés en respectant les paramètres définis. L'alphabet utilisé pour représenter les séquences est {A, C, G, T}, correspondant aux quatre nucléotides de l'ADN.

Voici les ensembles de données générés avec les tailles de séquences spécifiées et représentés avec les nucléotides de l'ADN :

Numéro	Taille	Séquence
1	5	ACTAC
2	5	GCGCA
3	10	GCAGGCCATT
4	10	TATGCTGGAC
5	15	GTCTCTTACTCACAT
6	20	GTGGGATCGGCCAATGGGAT
7	25	GGCGGTATGTTTAGCCGCTAATCCT
8	30	TGTGATTGATGTCTTTGAAGACACTCCCTC
9	35	TGAAACTGTGGCGTGAAGGCTAACCATGCGCCCAC
10	40	CGCAATCCGCCGAAGCATTTATTTATTGTGTACTAGGATT

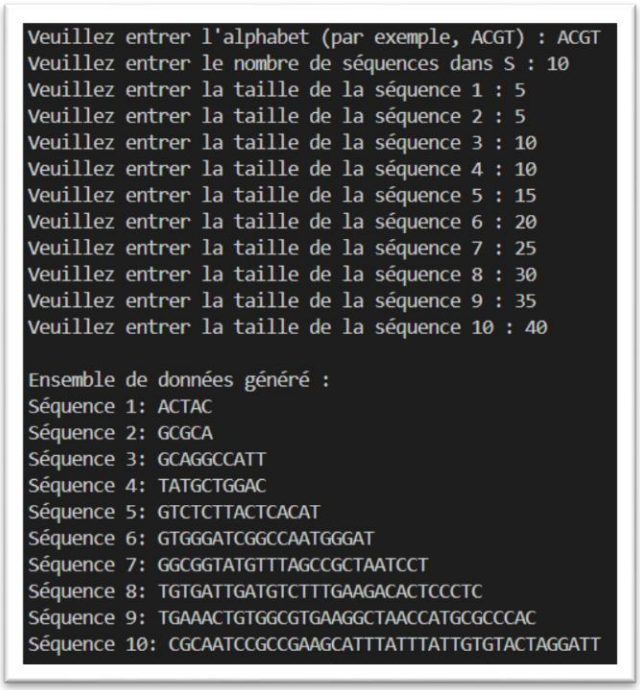

Figure 2: Génération de données avec un programme Python

4. Résolution par Méthode Exacte Utilisant DFS (Depth-First Search)

4.1. Choix de l'Approche

Pour résoudre exactement le problème de la plus longue sous-séquence commune (LCS), nous avons opté pour une approche basée sur la recherche en profondeur (DFS - Depth-First Search). Cette décision découle de la nature du problème, où nous devons explorer systématiquement toutes les combinaisons possibles de caractères des deux séquences.

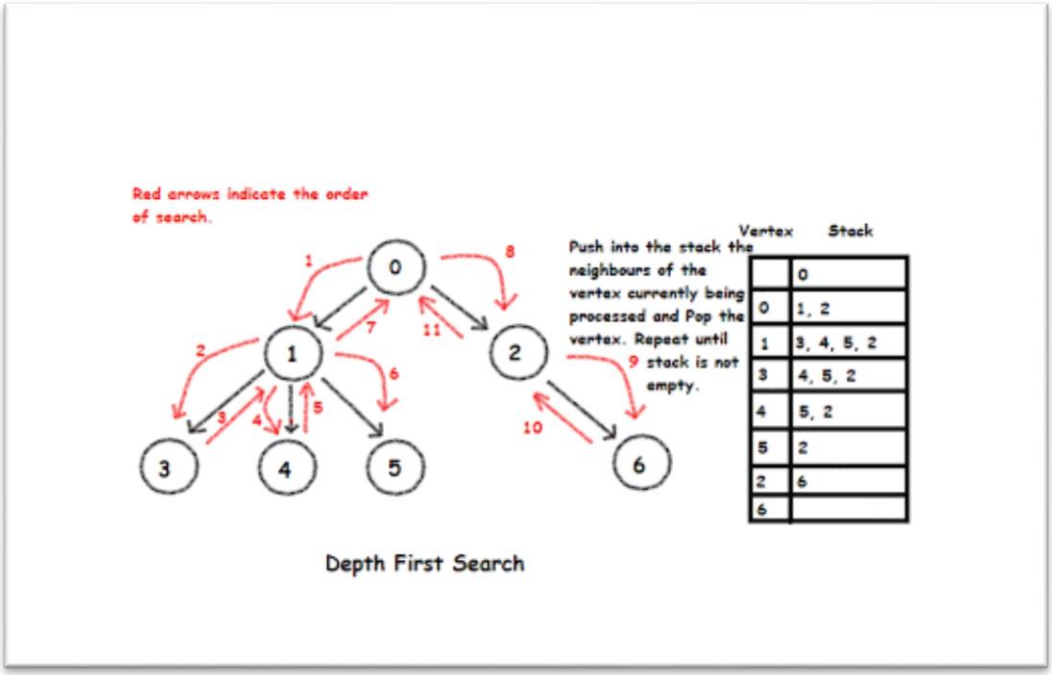

Figure 3: Exemple du principe de fonctionnement de la DFS

4.2. Principe de Fonctionnement de l'Approche

L'algorithme DFS commence par comparer les caractères au début des deux séquences et avance récursivement jusqu'à la fin de l'une ou des deux séquences. Lorsqu'il trouve un caractère commun, il l'ajoute à la sous-séquence courante et poursuit l'exploration récursive. En cas de caractères non correspondants, l'algorithme explore deux branches alternatives, une avec le caractère de la première séquence et l'autre avec le caractère de la deuxième séquence. Cette exploration continue jusqu'à ce que toutes les possibilités soient examinées ou que la fin de l'une des séquences soit atteinte.

4.3. Implémentation de l'Algorithme Exact

Notre implémentation de l'algorithme DFS commence par explorer une branche de la séquence en vérifiant si les caractères correspondants sont présents dans les deux séquences à la même position. Si tel est le cas, nous ajoutons ce caractère à la sous-séquence courante et continuons l'exploration récursive. Sinon, nous explorons deux branches alternatives, l'une avec le caractère de la première séquence et l'autre avec le caractère de la deuxième séquence. Nous

répétons ce processus jusqu'à ce que nous ayons parcouru toutes les possibilités ou atteint la fin de l'une des séquences.

4.4. Analyse des Résultats Obtenu

Nous présentons une synthèse de nos résultats à travers le tableau ci-dessous:

Séquence 1	Séquence 2	Taille globale	Temps d'exécution(s)	Nombre de Comparaisons	Sous-séquences LCS
1	2	10	0.0010018349	137	CC, CA
1	4	15	0.0010261536	369	ACTAC
1	10	45	0.0030357838	685	ACTAC
2	3	15	0.0000000000	46	GCGCA
2	6	25	0.0020024776	372	GCGCA
2	10	45	0.0010032654	144	GCGCA
3	5	25	0.1155760288	47829	GCACCAT
3	6	30	0.2083368301	90553	GAGGCCATT, GCGGCCATT
3	7	35	0.3441855907	145775	GCAGGCCATT
4	5	25	0.1451642513	61197	TTCTAC, TACTAC, TATCAC
4	6	30	0.2798552513	119582	TATGCTGGA
4	7	35	1.0861537457	470920	TATGTGGAC
4	10	50	22.8201949596	5658851	ATGCTGGAC, TAGCTGGAC, TATGCTGGA
5	6	35	518.23564897	525658	GTTCCCAAT
5	7	40	Trop	Trop	Pas de résultat

4.4.1. Le temps d'exécution vs la taille de séquence

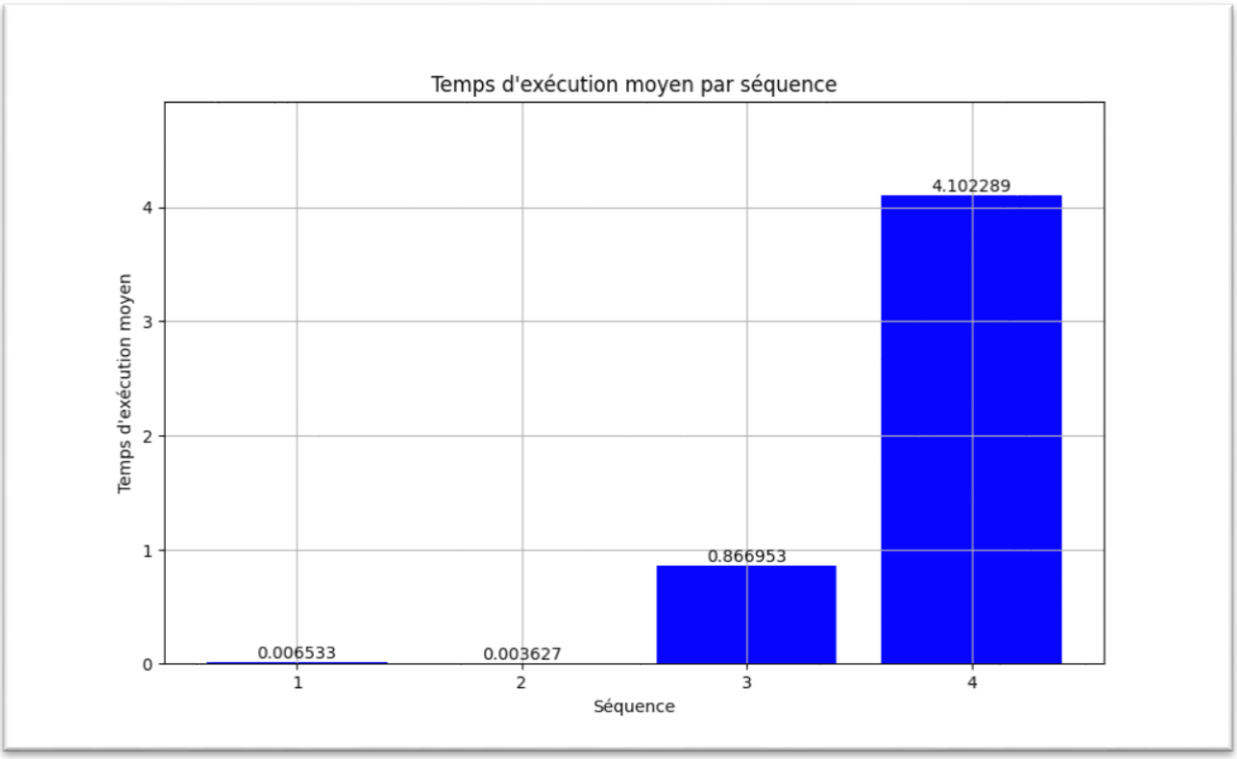

Figure 4: Histogramme comparant les temps d'exécution en fonction de la taille des séquences

L'analyse des temps d'exécution moyens pour les paires de séquences fournies révèle une tendance intéressante. La première observation pertinente est que les moyennes de temps d'exécution pour les paires de séquences 1 et 2 sont relativement faibles, avec des valeurs de 0.006533 et 0.003627 respectivement. En revanche, les moyennes de temps d'exécution pour les paires de séquences 3 et 4 sont nettement plus élevées, avec des valeurs de 0.866953 et 4.102289 respectivement. Cependant, la comparaison des séquences de taille 5 avec des séquences de taille 30, comme cela est mentionné, est plus facile et moins coûteuse que de comparer des séquences de tailles similaires, telles que 15 et 20.

Ainsi, pour que le temps d'exécution augmente de manière significative, il est nécessaire que la taille des deux séquences comparées augmente, plutôt que seulement celle d'une seule séquence.

4.4.2. Nombre de comparaison vs la taille globale :

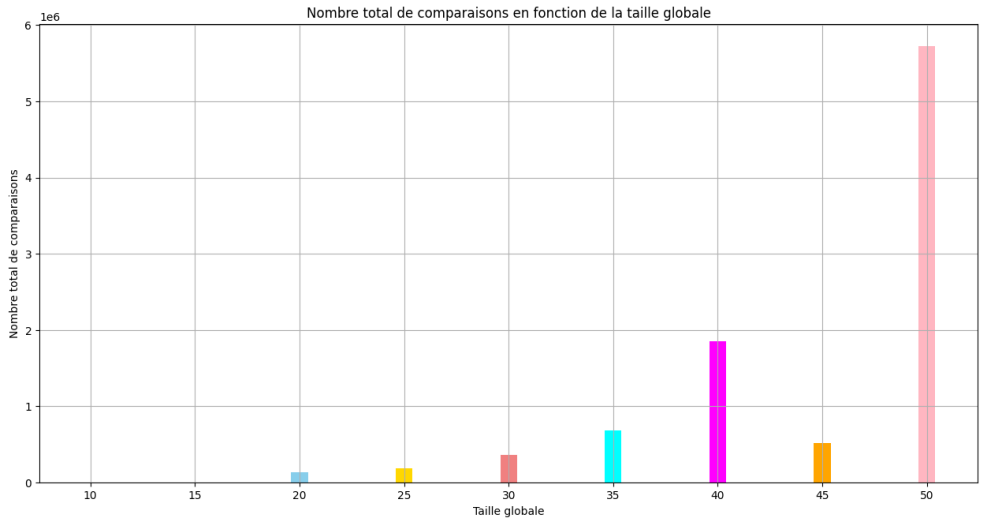

Figure 5: Diagramme à barres montrant le nombre de comparaisons par taille globale

On observe une tendance générale selon laquelle le nombre total de comparaisons augmente à mesure que la taille globale des séquences augmente. Cela est cohérent avec l'intuition, car des ensembles de séquences plus grands nécessitent généralement plus de comparaisons pour identifier la sous-séquence commune la plus longue. Cette relation reflète l'aspect fondamental du problème LCS où la complexité algorithmique est souvent liée à la taille des données.

Bien que le nombre de comparaisons augmente globalement avec la taille globale, il existe une variabilité significative dans les nombres de comparaisons pour chaque taille globale. Cette variabilité peut être due à divers facteurs, tels que la complexité structurelle des séquences elles-mêmes, les propriétés spécifiques des algorithmes utilisés pour résoudre le problème, ou des caractéristiques particulières des jeux de données.

4.5. Conformité avec la Complexité Théorique

En se basant sur la complexité temporelle de l'algorithme DFS, qui est exponentielle en fonction de la somme des longueurs des deux séquences, les résultats obtenus montrent une augmentation significative du temps d'exécution avec la taille des séquences, confirmant ainsi la nature exponentielle de la complexité temporelle de DFS pour le problème LCS.

De plus, la relation observée entre le nombre de comparaisons et la taille globale des séquences est cohérente avec la complexité théorique de DFS. L'augmentation du nombre de comparaisons avec la taille globale des séquences reflète l'effet attendu de la complexité exponentielle de DFS sur ce problème.

5. Résolution par Métaheuristique Utilisant Tabu Search

5.1. Principe de Fonctionnement de la Métaheuristique

La méthode Tabu Search est une métaheuristique largement utilisée pour résoudre des problèmes d'optimisation combinatoire. Son principe de fonctionnement repose sur une exploration intelligente de l'espace des solutions en combinant des mécanismes d'exploration globale et de recherche locale. Contrairement aux algorithmes stochastiques, Tabu Search utilise une mémoire, souvent appelée liste tabou, pour éviter les cycles et guider la recherche vers des solutions prometteuses.

L'idée centrale de Tabu Search est de naviguer dans l'espace des solutions en suivant une série de mouvements, appelés opérateurs de voisinage, tout en maintenant une liste tabou des mouvements interdits. Cette liste tabou est importante pour empêcher l'algorithme de revisiter les mêmes solutions et pour encourager une exploration plus diversifiée de l'espace des solutions. De plus, Tabu Search utilise des critères de sélection intelligents pour choisir les mouvements à effectuer, donnant la priorité aux solutions non visitées récemment ou à celles qui améliorent la fonction objectif.

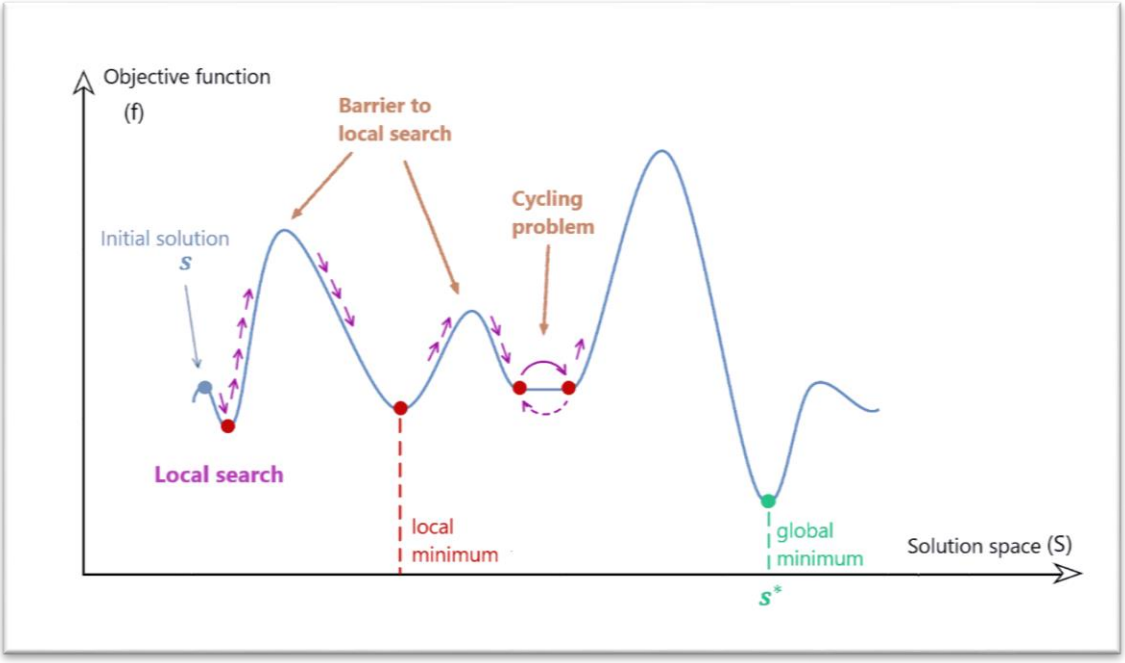

Figure 6: Principe de fonctionnement de Tabu Search

5.2. Adaptation de la Métaheuristique au Problème LCS

Dans le contexte du Problème de la Plus Longue Sous-Séquence Commune (LCS), l'application de Tabu Search nécessite une adaptation spécifique pour exploiter les caractéristiques uniques du problème. Pour ce faire, les opérateurs de voisinage doivent être conçus de manière à générer des solutions voisines perturbant les séquences tout en préservant leur structure. Par exemple, les opérateurs de voisinage peuvent inclure des mouvements tels que l'insertion, la suppression ou le remplacement de caractères dans les séquences.

De plus, la fonction objectif doit être définie pour évaluer la qualité des solutions candidates en termes de longueur de sous-séquence commune maximale. Enfin, la liste tabou devrait être configurée pour interdire les mouvements qui conduisent à des solutions déjà explorées ou qui ont peu d'impact sur l'amélioration de la solution actuelle.

5.3. Implémentation de la Métaheuristique

Le processus de Tabu Search consiste à explorer l'espace des solutions en modifiant la première chaîne de caractères tout en maintenant une liste tabou des mouvements interdits. À chaque itération, une solution candidate est générée en perturbant les caractères de la première chaîne de caractères. Cette solution est ensuite évaluée en termes de longueur de sous-séquence commune maximale avec la deuxième chaîne de caractères. Si la longueur de cette solution est meilleure que la meilleure solution actuelle, elle est mise à jour. Les solutions ayant la même longueur que la meilleure solution sont également enregistrées. Le processus continue jusqu'à atteindre une condition de stagnation ou le nombre maximum d'itérations.

L'objectif de cette implémentation est d'explorer efficacement l'espace des solutions tout en évitant les cycles grâce à la liste tabou, afin de trouver des sous-séquences communes maximales entre les deux chaînes de caractères fournies.

5.4. Analyse des Résultats Obtenu

Nous présentons une synthèse de nos résultats à travers le tableau ci-dessous:

Séquence 1	Séquence 2	Taille globale	Temps d'exécution	Sous-séquences LCS
1	2	10	0.0640306473	C
1	4	15	0.0710370541	TC
1	10	45	0.0715503693	CA
2	3	15	0.0836482048	GCA
2	6	25	0.0750343800	GG
2	10	45	0.0654933453	CA
3	5	25	0.0936565399	GAT
3	6	30	0.0945336819	GGG
3	7	35	0.0896096230	GGGT
5	6	35	0.1109330654	GTTCAT
6	10	60	0.1257965565	CCGCAAT
6	7	45	0.1731417179	GGGGAG
7	8	55	0.1441054344	GGTTTGACT
7	10	65	0.1992521286	CGCAATCCGCCACTAT
8	9	65	0.1946854591	TGAAACTGGTAC
8	10	70	0.2022590637	CGCAATCCAT

5.4.1. Le temps d'exécution vs la taille globale des séquences

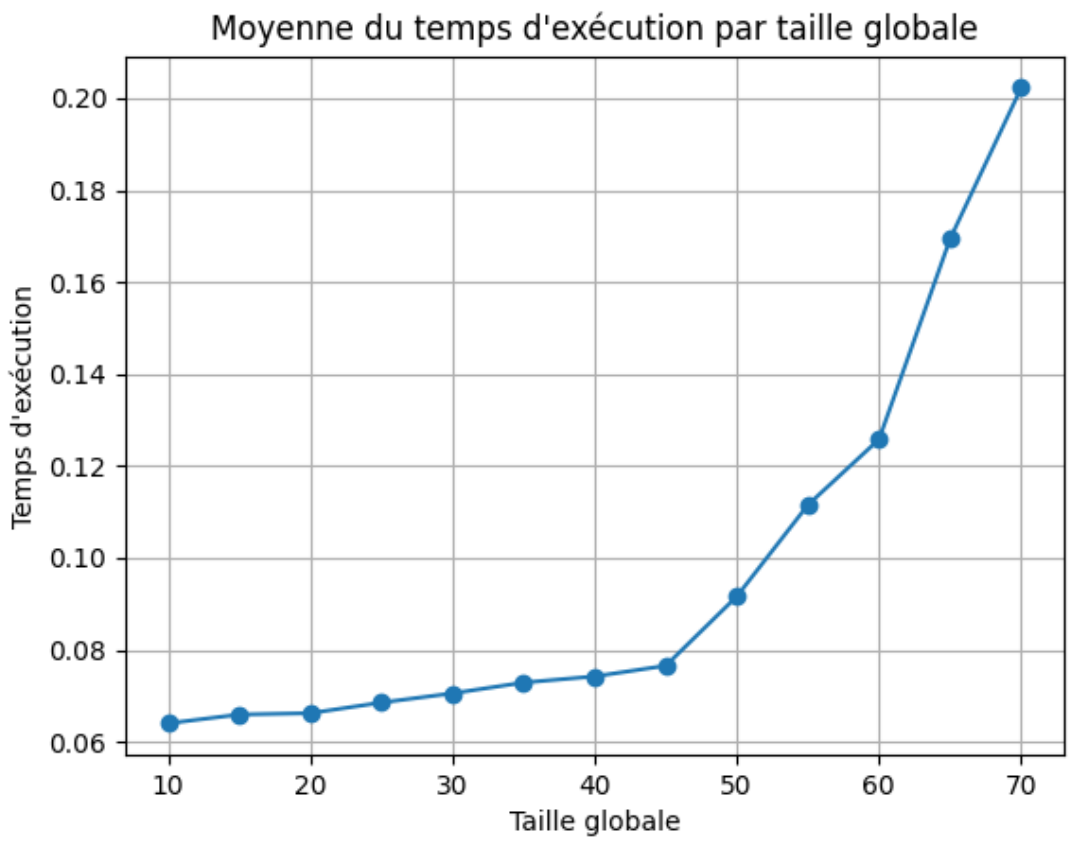

Figure 7: Courbe du temps d'exécution en fonction de la taille globale des séquences

La courbe ci-dessus représente la relation entre la taille globale des séquences et le temps d'exécution moyen de l'algorithme Tabu Search.

On observe une tendance générale à l'augmentation du temps d'exécution avec l'augmentation de la taille globale des séquences. Cette tendance est intuitive, car des séquences plus grandes nécessitent généralement plus de calculs pour trouver une solution optimale ou proche de l'optimale.

Cependant, la croissance du temps d'exécution semble être non linéaire. Jusqu'à une taille globale d'environ 45, le temps d'exécution augmente de manière relativement stable, avec des variations mineures. Cependant, au-delà de cette taille, le temps d'exécution semble augmenter de manière plus significative pour chaque incrémentation de la taille globale.

5.5. Conformité avec la Complexité Théorique

La complexité temporelle théorique d'un problème de Tabu Search, souvent exprimée comme O(2^n), où n est la taille de l'espace de recherche, peut être exponentielle dans la magnitude du problème. Cependant, il est important de noter que cette complexité théorique ne représente pas nécessairement la performance réelle de l'algorithme dans des cas spécifiques.

En examinant les résultats obtenus à partir de l'exécution de l'algorithme Tabu Search sur différentes paires de séquences, on observe une tendance générale où le temps d'exécution augmente avec la taille globale des séquences. Cette observation est conforme à la nature de l'algorithme, où une plus grande taille de l'espace de recherche peut entraîner une exploration plus intensive et donc des temps d'exécution plus longs.

Cependant, il est important de noter que la croissance du temps d'exécution n'est pas nécessairement linéaire avec l'augmentation de la taille globale des séquences. Des variations peuvent être observées en fonction de la nature spécifique des séquences, de la configuration de la liste tabou, du nombre maximum d'itérations et d'autres facteurs.

6. Exploration et Évaluation

Nous examinerons en détail le temps d'exécution ainsi que la qualité des solutions obtenues par chaque méthode.

6.1. Comparaison des Performances entre la méthode Exacte DFS et la Meta-heuristique Tabu search

6.1.1. Le temps d'exécution

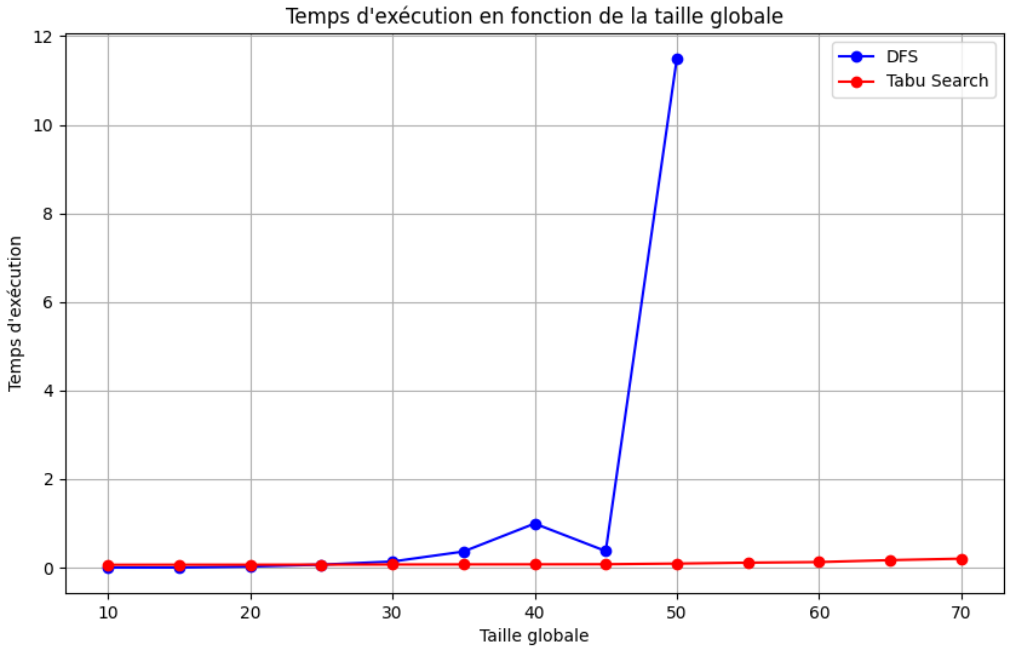

Figure 8: Comparaison des courbes de temps d'exécution entre les méthodes DFS et Tabu Search

La figure 8 présente deux courbes comparant les temps d'exécution moyens de DFS et de Tabu Search en fonction de la taille globale des données. Pour les petites tailles globales, DFS affiche des temps d'exécution rapides, mais ceux-ci augmentent considérablement avec la taille globale, probablement en raison de l'explosion combinatoire. En revanche, Tabu Search offre un temps d'exécution plus stable et généralement plus court, même avec des ensembles de données plus volumineux. Cette différence peut s'expliquer par la nature exploratoire de DFS, tandis que Tabu Search utilise une approche plus intelligente pour naviguer dans l'espace des solutions.

6.1.2. La qualité des sous-séquences obtenues

L'évaluation des sous-séquences LCS révèle que DFS tend à fournir des solutions optimales ou très proches, mais avec un temps d'exécution significatif. En revanche, Tabu Search fournit des solutions dans des délais raisonnables, bien que parfois elles ne soient pas optimales en raison de problèmes liés aux maxima locaux. Cela montre que Tabu Search privilégie l'efficacité temporelle tout en fournissant des solutions de qualité acceptable.

7. Identification des Forces et des Faiblesses de Chaque Approche

7.1. Recherche en Profondeur (DFS) :

Avantages :

- Exhaustivité : DFS explore toutes les possibilités de sous-séquences, garantissant qu'aucune solution n'est manquée.
- Facilité de mise en œuvre : L'algorithme DFS est relativement simple à mettre en œuvre et à comprendre, ce qui en fait une première approche pour aborder le problème.
- Pas de convergence locale : Étant donné qu'il explore toutes les possibilités, DFS n'est pas sujet à la convergence locale, ce qui signifie qu'il peut potentiellement trouver la solution optimale.

Inconvénients :

- Complexité exponentielle : La complexité de DFS peut devenir exponentielle pour les grandes instances du problème, rendant son exécution impraticable.
- Pas de garantie d'optimalité : DFS ne fournit pas de garantie que la solution trouvée est la meilleure possible. Il peut trouver une solution rapidement mais pas nécessairement optimale.
- Utilisation de mémoire : Pour les grands ensembles de données, DFS peut utiliser beaucoup de mémoire en raison de la récursivité et de la nécessité de stocker des informations sur chaque nœud visité.

7.2. Recherche Tabou (Tabu Search) :

Avantages :

- Recherche guidée : Tabu Search utilise une recherche guidée dans l'espace des solutions, ce qui peut lui permettre de converger rapidement vers des solutions de bonne qualité.
- Adaptabilité : L'algorithme peut s'adapter à différentes instances du problème en modifiant les critères de recherche et les règles de voisinage.
- Évite les optimaux locaux : Tabu Search peut échapper aux optimaux locaux en utilisant des mécanismes de diversification et de stratégies de mouvement intelligents.

Inconvénients :

- Dépendance des paramètres : Tabu Search dépend de la sélection appropriée des paramètres et des critères tabous, ce qui peut être difficile à déterminer.
- Temps d'exécution : Bien que généralement plus rapide que DFS, Tabu Search peut nécessiter un temps d'exécution significatif pour converger vers une solution de qualité.
- Sensibilité aux conditions initiales : Comme beaucoup de métaheuristiques, Tabu Search peut être sensible aux conditions initiales, ce qui signifie que différentes exécutions peuvent conduire à des résultats différents.

8. Discussion sur les Limitations et les Perspectives d'Amélioration

DFS est limité par son explosion combinatoire, mais pourrait être amélioré en explorant des stratégies de parcours plus intelligentes telles que le branch and bound. Pour Tabu Search, l'amélioration des mécanismes de gestion des maxima locaux et l'exploration de variantes de l'algorithme pourraient améliorer la qualité des solutions obtenues.

DFS et Tabu Search offrent des approches complémentaires pour résoudre le problème LCS, chacune avec ses forces et ses faiblesses. L'optimisation des paramètres et l'exploration de nouvelles heuristiques pourraient permettre d'améliorer encore les performances des deux méthodes dans la résolution de ce problème complexe.

9. Conclusion

Notre étude comparative entre la méthode exacte DFS et la méta-heuristique Tabu Search pour résoudre le problème de la Plus Longue Sous-Séquence Commune (LCS) a révélé des résultats significatifs et instructifs.

Tout d'abord, nous avons mis en évidence l'importance du problème LCS dans divers domaines, tels que la bioinformatique, la recherche pharmaceutique et la compression de données, soulignant ainsi la nécessité de développer des méthodes efficaces pour sa résolution.

Ensuite, nous avons présenté en détail les principes de fonctionnement, les implémentations et les résultats des deux approches. DFS, en tant que méthode exacte, a démontré sa capacité à fournir des solutions optimales, mais avec des temps d'exécution prohibitifs pour des ensembles de données de grande taille en raison de son explosion combinatoire. D'autre part, Tabu Search, en tant que méta-heuristique, a offert des temps d'exécution plus courts tout en fournissant des solutions de qualité acceptable, bien que parfois non optimales en raison de maxima locaux.

En comparant les performances des deux méthodes, nous avons constaté que DFS excelle dans la précision des résultats, tandis que Tabu Search privilégie l'efficacité temporelle. Cependant, chacune des deux approches présente des limites, notamment en termes de temps d'exécution pour DFS et de qualité des solutions pour Tabu Search.

Enfin, nous avons discuté des perspectives d'amélioration pour chaque approche, telles que l'exploration de stratégies plus intelligentes pour DFS et l'amélioration des mécanismes de gestion des maxima locaux pour Tabu Search.

10. Annexe

Résultats finaux pour DFS :

Séquence 1	Séquence 2	Taille globale	Temps d'exécution	Comparaisons	Sous-séquences LCS
1	2	10	0.0010018349	137	CC, CA
1	3	15	0.0020463467	453	CAC, ACC, ACA, ACT
1	4	15	0.0010261536	369	ACTAC
1	5	20	0.0009996891	367	ACTAC
1	6	25	0.0070030689	2719	ACTA
1	7	30	0.0180418491	4670	ACTAC
1	8	35	0.0130681992	4464	ACTAC
1	9	40	0.0125718117	2418	ACTAC
1	10	45	0.0030357838	685	ACTAC
2	3	15	0.0000000000	46	GCGCA
2	4	15	0.0040001869	656	GCGA, GCGC
2	5	20	0.0020167828	296	GCCA
2	6	25	0.0020024776	372	GCGCA
2	7	30	0.0010097027	103	GCGCA
2	8	35	0.0110294819	3612	GCGCA
2	9	40	0.0079538822	1773	GCGCA
2	10	45	0.0010032654	144	GCGCA
3	4	20	0.0700786114	26523	GCGGA, GCGGC
3	5	25	0.1155760288	47829	GCACCAT
3	6	30	0.2083368301	90553	GAGGCCATT, GCGGCCATT
3	7	35	0.3441855907	145775	GCAGGCCATT
3	8	40	3.8240048885	1736644	GAGGCCATT
3	9	45	1.3444173336	458705	GCAGGCCAT
3	10	50	0.1620724201	67412	GCAGGCCATT
4	5	25	0.1451642513	61197	TTCTAC, TACTAC, TATCAC
4	6	30	0.2798552513	119582	TATGCTGGA
4	7	35	1.0861537457	470920	TATGTGGAC
4	8	40	0.1362750530	56395	TATGCTGGAC
4	9	45	0.1460883617	59125	TATGCTGGAC
4	10	50	22.8201949596	5658851	ATGCTGGAC, TAGCTGGAC, TATGCTGGA

Figure 9: Résultats du tableau de la méthode DFS avec programme Python

Séquence 1	Séquence 2	Taille globale	Temps d'exécution	Comparaisons	Sous-séquence LCS
1	2	10	0.0640306473	10000	C
1	3	15	0.0705192089	10000	C
1	4	15	0.0710370541	10000	TC
1	5	20	0.0575876236	10000	C
1	6	25	0.0550394058	10000	
1	7	30	0.0615065098	10000	
1	8	35	0.0575509071	10000	T
1	9	40	0.0745520592	10000	A
1	10	45	0.0715503693	10000	CA
2	3	15	0.0836482048	10000	GCA
2	4	15	0.1145639420	10000	
2	5	20	0.0675079823	10000	G
2	6	25	0.0750343800	10000	GG
2	7	30	0.0706238747	10000	G
2	8	35	0.0625336170	10000	A
2	9	40	0.0695447922	10000	A
2	10	45	0.0654933453	10000	CA
3	4	20	0.0736188889	10000	G
3	5	25	0.0936565399	10000	GAT
3	6	30	0.0945336819	10000	GGG
3	7	35	0.0896096230	10000	GGGT
3	8	40	0.0846233368	10000	GT
3	9	45	0.0866305828	10000	ACT
3	10	50	0.0916337967	10000	CC
4	5	25	0.0755641460	10000	CT
5	6	35	0.1109330654	10000	GTTCAT
4	6	30	0.0926606655	10000	G
5	9	50	0.0926373005	10000	T
5	10	55	0.1115362644	10000	CCTCA
6	7	45	0.1731417179	10000	GGGGAG
6	8	50	0.1178405285	10000	GTTGA
6	9	55	0.1221396923	10000	TGCG
6	10	60	0.1257965565	10000	CCGCAAT
5	9	50	0.0926373005	10000	T
5	10	55	0.1115362644	10000	CCTCA
6	7	45	0.1731417179	10000	GGGGAG
6	8	50	0.1178405285	10000	GTTGA
6	9	55	0.1221396923	10000	TGCG

Figure 10: Résultats du tableau de la méthode Tabu Search avec programme Python