

Le problème du voyageur de commerce.

Réalisé par :

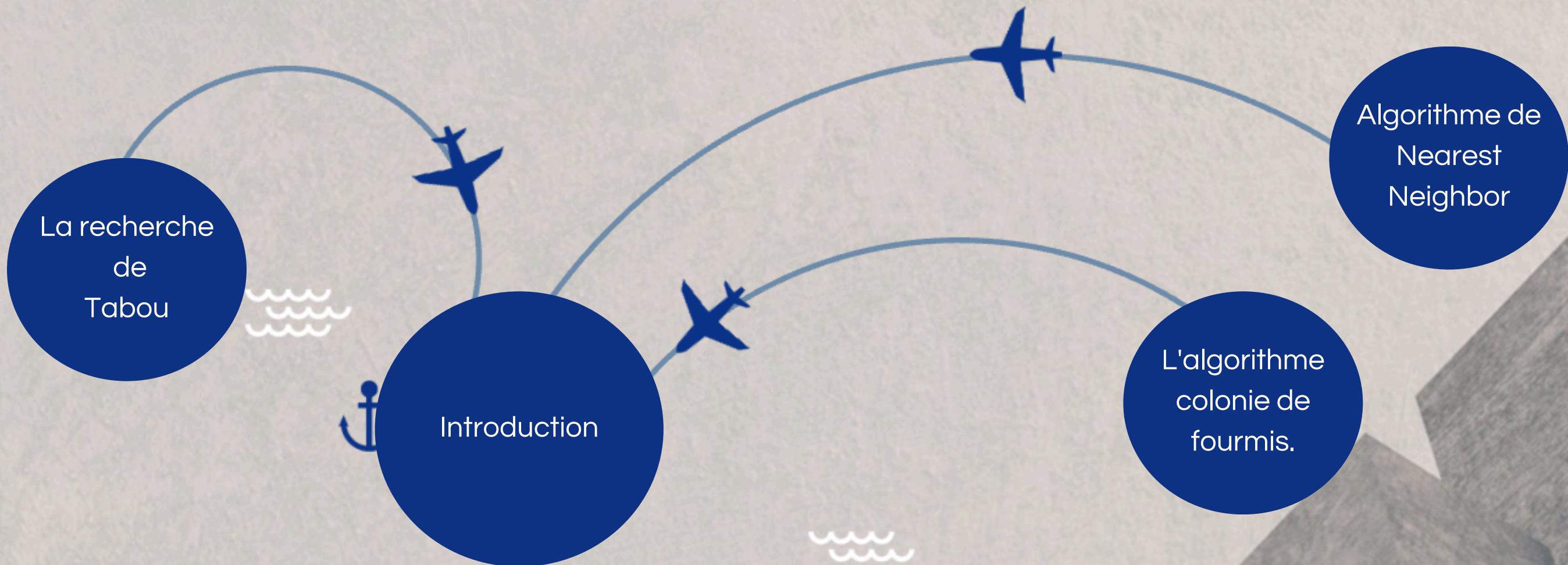
Trabelsi Siwar

Bouzekri Yassine

Cherif Omar

Introduction

- Le problème du voyageur de commerce (TSP) , posé en 1859 par William Rowan Hamilton est l'un des problèmes les plus classiques d'optimisation combinatoire NP-complet.
- Un problème d'optimisation combinatoire est un problème d'optimisation dans lequel l'espace de recherche S est dénombrable



Le problème du voyageur de commerce.

Réalisé par :
Trabelsi Siwar
Bouzekri Yassine
Cherif Omar

L'algorithme colonie de fourmis.

- Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation de sources de nourriture.

Parametres

Pseudo-code

Résolution

Exemple

Complexité

Pseudo-code

- *Tant que le critère d'arrêt n'est pas atteint faire*
- *Pour k=1 à m faire*
- *Choisir une ville au hasard*
- *Pour chaque ville non visitée i faire*
- *Choisir une ville j, dans la liste des villes restantes selon (F-1)*
- *Fin Pour*
- *Déposer une piste sur le trajet (t) conformément à (F-2)*
- *Fin Pour*
- *Évaporer les pistes selon (F-3)*
- *Fin Tant que*

L'algorithme colonie de fourmis.

- Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation de sources de nourriture.

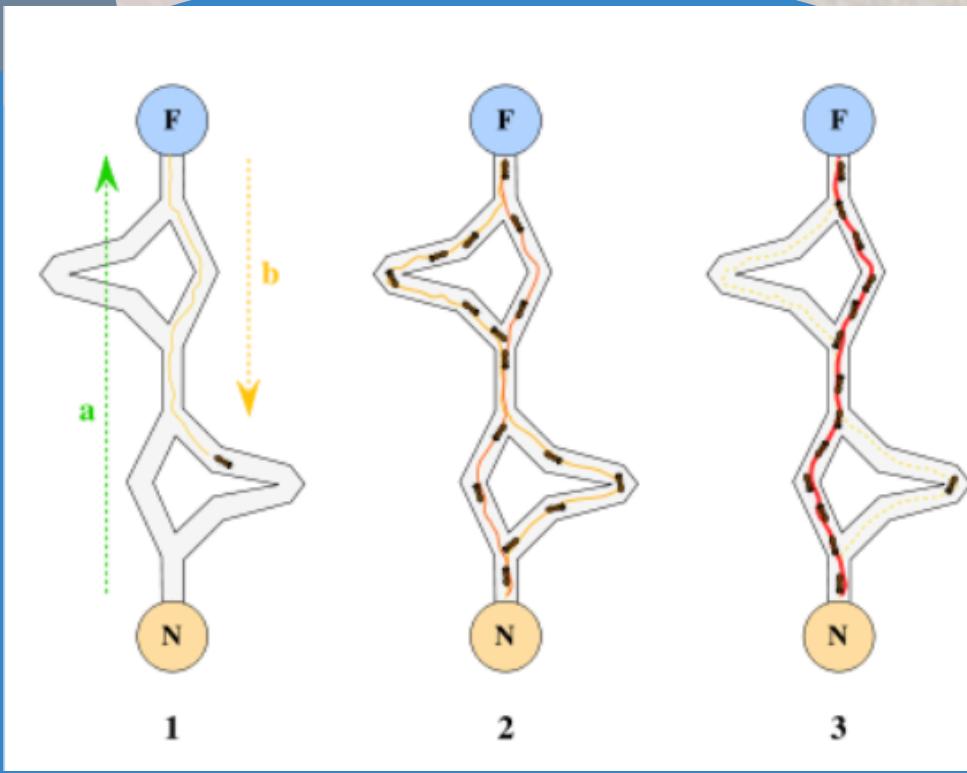
Parametres

Pseudo-code

Résolution

Exemple

Complexité



- 1) la première fourmi trouve la source de nourriture (F), via un chemin quelconque (a), puis revient au nid (N) en laissant derrière elle une piste de phéromone (b).
- 2) les fourmis empruntent indifféremment les quatre chemins possibles, mais le renforcement de la piste rend plus attractif le chemin le plus court.
- 3) les fourmis empruntent le chemin le plus court, les portions longues des autres chemins perdent leur piste de phéromones.

L'algorithme colonie de fourmis.

- Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation de sources de nourriture.

Parametres

Pseudo-code

Résolution

Exemple

Complexité

Paramètres

a et β : Ces deux paramètres contrôlent l'importance relative de l'intensité des pheromones (a) et de la visibilité des arcs (β). Ils influencent la manière dont les fourmis choisissent leur prochaine ville en fonction des phéromones déposées et des distances à parcourir

2 . p (p) : Ce paramètre représente le taux d'évaporation des phéromones sur les arcs entre les villes. Il permet de maintenir un équilibre entre l'exploration et l'exploitation en évitant la stagnation et en favorisant la découverte de nouveaux chemins

3 . Q : La quantité de phéromones déposée par une fourmi lorsqu'elle parcourt un arc. Cette valeur influence la rapidité avec laquelle les chemins sont renforcés par les pheromones, impactant ainsi les choix futurs des fourmis.

4.m : Nombre de fourmis dans la colonie, déterminant la diversité des chemins explorés et l'efficacité globale de l'algorithme

5. NCmax : Nombre maximum de cycles d'exécution de l'algorithme, influençant la convergence vers une solution optimale.

L'algorithme colonie de fourmis.

- Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation de sources de nourriture.

Parametres

Pseudo-code

Résolution

Exemple

Complexité

Resolution

Pour appliquer l'algorithme de colonies de fourmis de manière approchée pour résoudre le problème du voyageur de commerce, voici les étapes clés à suivre :

Construction des solutions : Chaque fourmi parcourt le graphe en construisant un trajet reliant les villes. Le choix des villes dépend des mouvements possibles à chaque étape, de l'inverse de la distance entre les villes, et des villes déjà visitées

1. Dépôt de phéromones : Les fourmis déposent des phéromones sur les chemins empruntés. Les quantités déposées influencent le choix des chemins par les autres fourmis, favorisant ceux avec plus de phéromones. L'atténuation progressive des phéromones simule la diminution de leur attractivité avec le temps
- 2.Optimisation du trajet : En se basant sur les phéromones déposées et l'information heuristique locale, les fourmis ajustent leurs trajets pour trouver une solution optimale au problème du voyageur de commerce.
- 3.Applications diverses : Outre le voyageur de commerce, l'algorithme des colonies de fourmis est utilisé dans divers domaines tels que l'ordonnancement séquentiel, l'affectation quadratique, les tournées des véhicules, l'établissement d'horaires, la coloration de graphes, le partitionnement et les réseaux de télécommunications

L'algorithme colonie de fourmis.

- Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation de sources de nourriture.

Parametres

Pseudo-code

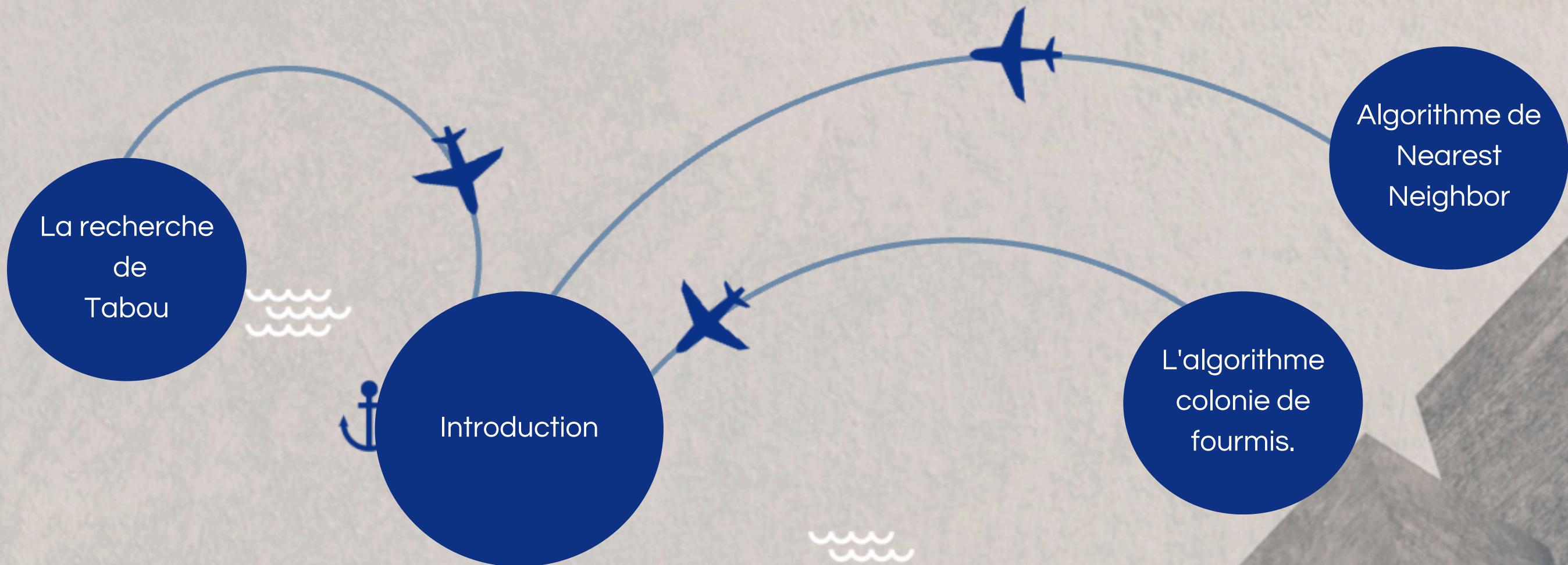
Résolution

Exemple

Complexité

Complexité

- 1) Initialisation : La complexité de l'initialisation est généralement linéaire par rapport au nombre de villes (n), soit $O(n)$
 - 2) Cycle de l'algorithme : La complexité du cycle de l'algorithme dépend du nombre de villes et du nombre de fourmis (m). Elle est donc quadratique, soit $O(n^2 \cdot m)$
 - 3) Fin du cycle et calcul des dépôts de phéromone : Cette étape a une complexité similaire au cycle de l'algorithme, soit $O(n^2 \cdot m)$
 - 4) Evaporation des phéromones : La complexité de l'évaporation des phéromones est linéaire par rapport au nombre de villes, soit $O(n)$
- En combinant ces différentes étapes, la complexité globale de l'algorithme de colonies de fourmis peut être exprimée comme suit :
- $$O(n^2 + m + NC_{\max} \cdot n^2 \cdot m)$$



Le problème du voyageur de commerce.

Réalisé par :
Trabelsi Siwar
Bouzekri Yassine
Cherif Omar

La recherche de Tabou

- Il est essentiel de noter que cette opération peut conduire à augmenter la valeur de la fonction (dans un problème de minimisation) : c'est le cas lorsque tous les points du voisinage ont une valeur plus élevée.

Application de l'algo

Pseudo-code

Complexité

Paramètres

Pseudo-code

1) Initialisation:

- Générer une solution aléatoire S .
- Créer une liste taboue vide L .

2) Itération:

- Explorer le voisinage de S :
 - Pour chaque ville non visitée dans S , construire une nouvelle solution en l'insérant à la meilleure position possible.
- Sélectionner la meilleure solution non taboue:
 - Choisir la solution avec le coût minimal parmi les solutions du voisinage qui ne sont pas dans la liste taboue.
- Mettre à jour la liste taboue:
 - Ajouter la solution sélectionnée à la liste taboue.
 - Supprimer de la liste taboue les solutions qui ne sont plus taboues (selon un critère de durée taboue).

3) Répéter les étapes 2 et 3 jusqu'à atteindre un critère d'arrêt (temps écoulé, nombre d'itérations, etc.).

La recherche de Tabou

- Il est essentiel de noter que cette opération peut conduire à augmenter la valeur de la fonction (dans un problème de minimisation) : c'est le cas lorsque tous les points du voisinage ont une valeur plus élevée.

Application de l'algo

Pseudo-code

Complexité

Paramètres

Paramètres

- n : Le nombre de villes.
- distances : Matrice des distances entre les villes. Cette matrice est carrée et symétrique, avec des distances positives sur la diagonale.
- tour : Tableau qui stocke l'ordre de visite des villes.
- tenure : Le nombre d'itérations pendant lesquelles un mouvement est tabou après avoir été effectué.
- taille_memoire : La taille de la liste tabou.

La recherche de Tabou

- Il est essentiel de noter que cette opération peut conduire à augmenter la valeur de la fonction (dans un problème de minimisation) : c'est le cas lorsque tous les points du voisinage ont une valeur plus élevée.

Application de l'algo

Pseudo-code

Complexité

Paramètres

Application de l'algo

L'algorithme de recherche tabou peut être appliqué à tout problème d'optimisation où il faut trouver une solution optimale ou approchée à un problème combinatoire.

Exemple:

Supposons que vous devez planifier la livraison de colis à plusieurs adresses. Vous pouvez utiliser l'algorithme de recherche tabou pour trouver l'itinéraire le plus court qui visite toutes les adresses.

- Définir les paramètres de l'algorithme.
- Initialiser une solution courante.
- Itérer jusqu'à ce qu'un critère d'arrêt soit atteint.
- A chaque itération:
 - Trouver le meilleur mouvement non tabou.
 - Appliquer le mouvement à la solution courante.
 - Mettre à jour la liste tabou.
 - Evaluer la qualité de la solution courante.

La recherche de Tabou

- Il est essentiel de noter que cette opération peut conduire à augmenter la valeur de la fonction (dans un problème de minimisation) : c'est le cas lorsque tous les points du voisinage ont une valeur plus élevée.

Application de l'algo

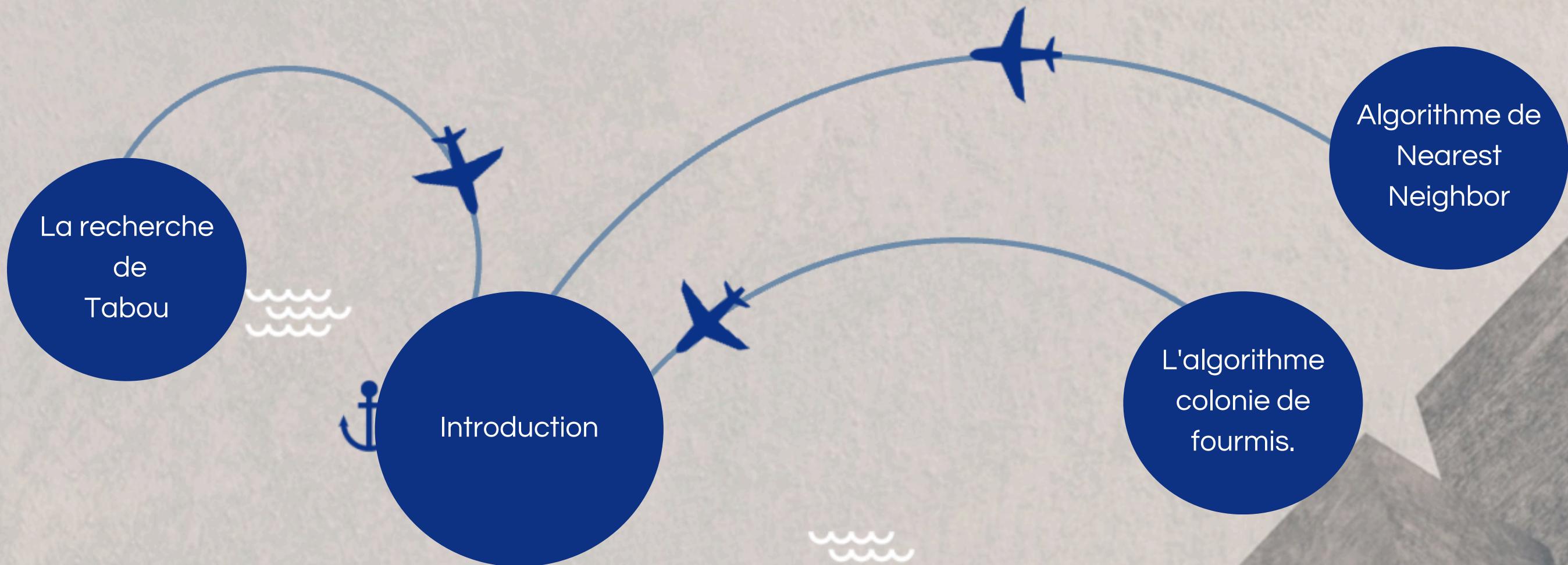
Pseudo-code

Complexité

Paramètres

Complexité

- La complexité dépend du nombre de villes (n) dans l'instance du TSP.
- Dans le pire des cas, l'algorithme effectue n itérations pour explorer l'espace des solutions.
- La complexité est donc $O(n)$.
- Cependant, la taille de la liste taboue et d'autres paramètres peuvent influencer la performance réelle de l'algorithme.



Le problème du voyageur de commerce.

Réalisé par :
Trabelsi Siwar
Bouzekri Yassine
Cherif Omar

Algorithme de Nearest Neighbor

- L'algorithme du voisin le plus proche a été l'un des premiers algorithmes utilisés pour résoudre approximativement le problème du voyageur de commerce. Dans ce problème, le vendeur commence dans une ville au hasard et visite à plusieurs reprises la ville la plus proche jusqu'à ce que toutes les visites aient été effectuées.
L'algorithme donne rapidement une courte tournée, mais généralement pas la plus optimale.

Application de
l'algo

Pseudo-code

Complexité

Paramètres

Conclusion

Pseudo-code

```
void nearest_neighbor(int n, float distances[][n], int *tour) {  
    // Initialisation  
    tour[0] = 0;  
    int villes_non_visitées[n - 1];  
    for (int i = 1; i < n; i++) {  
        villes_non_visitées[i - 1] = i;  
    }  
  
    // Itérer sur les villes non visitées  
    for (int i = 0; i < n - 1; i++) {  
        // Trouver la ville non visitée la plus proche de la dernière ville visitée  
        int ville_suivante = -1;  
        float distance_min = INFINITY;  
        for (int j = 0; j < n - 1; j++) {  
            if (villes_non_visitées[j] != -1) {  
                float distance = distances[tour[i]][villes_non_visitées[j]];  
                if (distance < distance_min) {  
                    distance_min = distance;  
                    ville_suivante = villes_non_visitées[j];  
                }  
            }  
        }  
  
        // Ajouter la ville suivante au tour  
        tour[i + 1] = ville_suivante;  
  
        // Marquer la ville suivante comme visitée  
        villes_non_visitées[ville_suivante] = -1;  
    }  
}
```

1. Choisir un point de départ aléatoire.
2. Tant que tous les points n'ont pas été visités :
 - 2.1 Trouver le point non visité le plus proche du point actuel
 - 2.2 Ajouter ce point à la liste des points visités.
 - 2.3 Mettre à jour le point actuel avec le point ajouté.
3. Ajouter le point de départ à la liste des points visités.
4. Retourner la liste des points visités.

Algorithme de Nearest Neighbor

- L'algorithme du voisin le plus proche a été l'un des premiers algorithmes utilisés pour résoudre approximativement le problème du voyageur de commerce. Dans ce problème, le vendeur commence dans une ville au hasard et visite à plusieurs reprises la ville la plus proche jusqu'à ce que toutes les visites aient été effectuées.
L'algorithme donne rapidement une courte tournée, mais généralement pas la plus optimale.

Application de
l'algo

Pseudo-code

Complexité

Paramètres

Conclusion

Paramètres

- L'algorithme nearest_neighbor a les paramètres suivants :
- n: Le nombre de villes.
- distances: Matrice des distances entre les villes. Cette matrice est carrée et symétrique, avec des distances positives sur la diagonale.
- tour: Tableau qui stockera l'ordre de visite des villes.

Algorithme de Nearest Neighbor

- L'algorithme du voisin le plus proche a été l'un des premiers algorithmes utilisés pour résoudre approximativement le problème du voyageur de commerce. Dans ce problème, le vendeur commence dans une ville au hasard et visite à plusieurs reprises la ville la plus proche jusqu'à ce que toutes les visites aient été effectuées.
L'algorithme donne rapidement une courte tournée, mais généralement pas la plus optimale.

Application de
l'algo

Pseudo-code

Complexité

Paramètres

Conclusion

Application de l'algo

Cet algorithme commence par choisir un noeud au hasard comme la solution initiale. Il calcule ensuite la distance entre ce noeud et chacun de ses voisins et remplace la solution actuelle par le voisin le plus proche si une amélioration est trouvée.

Cet algorithme examine chaque ville une par une et trouve la ville la plus proche non encore visitée. Il ajoute cette ville à la tournée en construction et répète ce processus jusqu'à ce que toutes les villes soient visitées. A chaque étape, il faut calculer la distance entre la ville actuelle et chacune des villes restantes pour déterminer la plus proche

Algorithme de Nearest Neighbor

- L'algorithme du voisin le plus proche a été l'un des premiers algorithmes utilisés pour résoudre approximativement le problème du voyageur de commerce. Dans ce problème, le vendeur commence dans une ville au hasard et visite à plusieurs reprises la ville la plus proche jusqu'à ce que toutes les visites aient été effectuées.
L'algorithme donne rapidement une courte tournée, mais généralement pas la plus optimale.

Application de
l'algo

Pseudo-code

Complexité

Paramètres

Conclusion

Complexité

L'algorithme nearest_neighbor a une complexité temporelle de $O(n^2)$.

Justification:

Comme il y a n villes au total et que $n-1$ villes ont déjà été insérées dans la tournée à chaque étape

- La boucle for principale est exécutée $n-1$ fois, où n est le nombre de villes.
- A chaque itération de la boucle, la boucle for imbriquée est exécutée $n-1$ fois.
- La comparaison de deux voisins a une complexité temporelle constante.

Algorithme de Nearest Neighbor

- L'algorithme du voisin le plus proche a été l'un des premiers algorithmes utilisés pour résoudre approximativement le problème du voyageur de commerce. Dans ce problème, le vendeur commence dans une ville au hasard et visite à plusieurs reprises la ville la plus proche jusqu'à ce que toutes les visites aient été effectuées.
L'algorithme donne rapidement une courte tournée, mais généralement pas la plus optimale.

Application de
l'algo

Pseudo-code

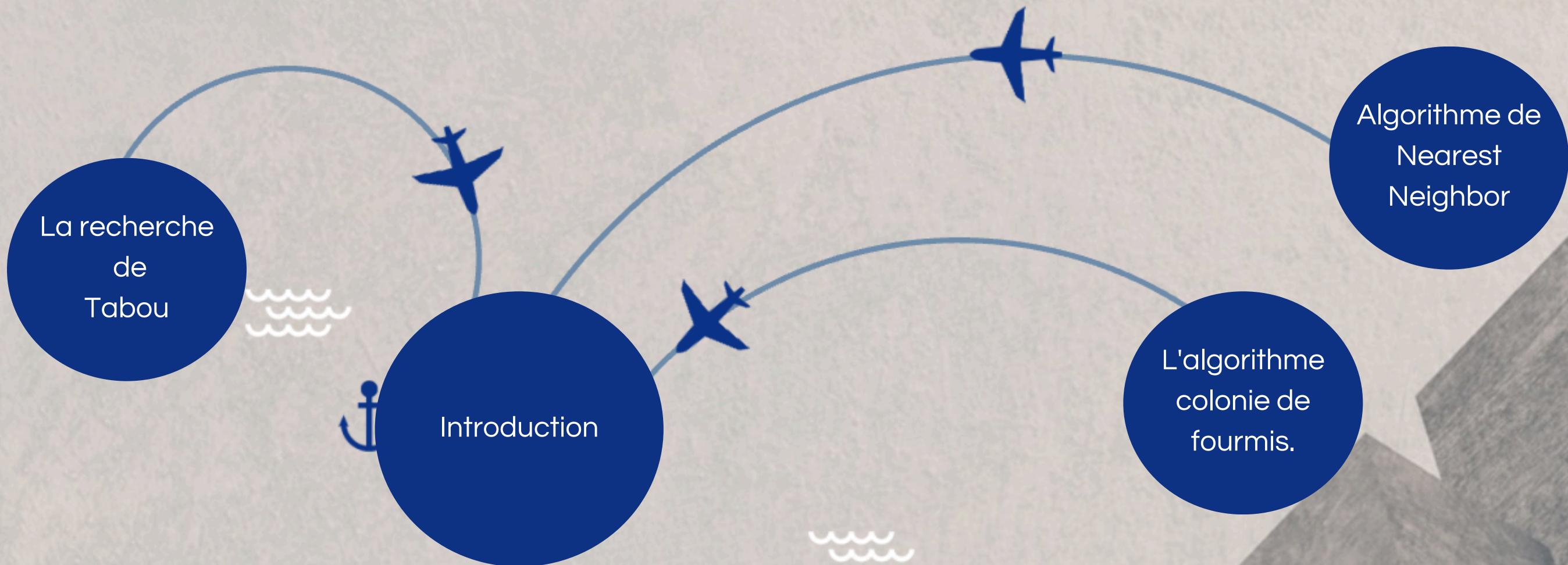
Complexité

Paramètres

Conclusion

Conclusion

- L'algorithme nearest_neighbor est un algorithme simple et efficace pour résoudre le problème du voyageur de commerce. Il est facile à implémenter et a une complexité temporelle raisonnable. Cependant, il ne garantit pas de trouver la solution optimale.



Le problème du voyageur de commerce.

Réalisé par :
Trabelsi Siwar
Bouzekri Yassine
Cherif Omar