Résolution du problème du voyageur (TSP) basée sur la programmation dynamique

Jamal Boussouf
Faculté Pluridisciplinaire de Nador
Nador, Maroc
jamal.boussouf@ump.ac.ma

Issam Seddik
Faculté Pluridisciplinaire de Nador
Nador, Maroc
issam.seddik@ump.ac.ma

***Abstract*—Le problème du voyageur de commerce (TSP) est l'un des problèmes d'optimisation combinatoire les plus emblématiques et les plus difficiles à résoudre, avec des implications dans les domaines de la logistique, du transport et de l'acheminement des réseaux. Cette étude se penche sur diverses méthodologies algorithmiques pour résoudre le TSP, allant de l'énumération exhaustive par force brute à des techniques sophistiquées de programmation dynamique (DP). En utilisant des outils pratiques tels que Streamlit pour le développement d'interfaces et des représentations graphiques pour la visualisation des solutions, nous explorons les performances et l'efficacité des différentes stratégies de résolution. Grâce à des analyses comparatives et à des évaluations empiriques, nous mettons en évidence les compromis entre la qualité des solutions, l'efficacité du calcul et l'évolutivité inhérente à chaque approche. Nos résultats soulignent l'importance de la sélection des algorithmes et de la méthodologie pour relever les défis de l'optimisation, offrant un aperçu des complexités et des nuances de la résolution des problèmes TSP. De plus, le code source et les matériaux du projet sont disponibles sur GitHub à l'adresse https://github.com/iseddik/TSP/, ce qui constitue une ressource précieuse pour l'exploration et la collaboration dans le domaine de la recherche en optimisation. À l'avenir, les progrès continus de la recherche en optimisation promettent de stimuler l'innovation et de favoriser les solutions transformatrices pour les applications du monde réel.**

***Index Terms*—Travel salesman problem (TSP), Dynamic programing (DP), Nearest neighbor heuristic (NNH)**

I. Introduction

Le problème du voyageur de commerce [1] (TSP) est l'un des défis d'optimisation combinatoire les plus connus dans le domaine de l'informatique et de la recherche opérationnelle. Issu du domaine des mathématiques, le TSP se penche sur la recherche de l'itinéraire le plus court possible qu'un vendeur doit emprunter pour visiter un ensemble de villes données et revenir à son point de départ. Pour résoudre efficacement cette énigme, la programmation dynamique [2] (PD) apparaît comme une approche puissante qui résout de manière optimale les sous-problèmes pour construire la solution globale. Dans ce rapport, nous explorons l'application des techniques de programmation dynamique pour résoudre les complexités [3] du problème du voyageur de commerce. Tout au long de cette discussion, nous aborderons les fondements théoriques, les complexités et l'analyse comparative de la programmation dynamique avec d'autres stratégies d'optimisation, avec des examens détaillés de chaque aspect dans les sections suivantes.

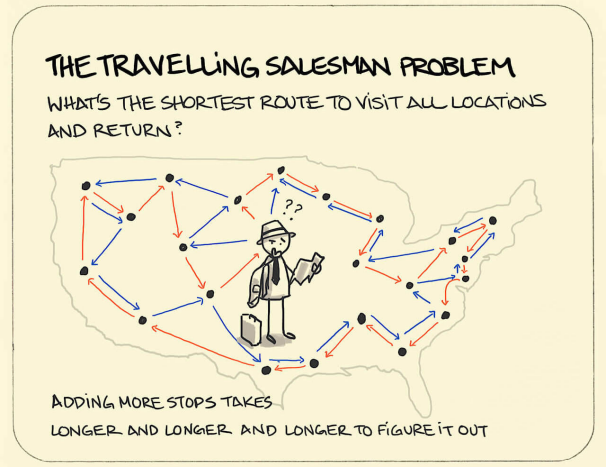

Fig. 1. TSP problem

Notre discussion commence par une exploration des fondements théoriques des algorithmes de programmation dynamique appliqués au problème du voyageur de commerce. Nous élucidons les principes fondamentaux de la programmation dynamique, en mettant l'accent sur sa nature récursive et sa sous-structure optimale, qui se prêtent bien à la décomposition du problème du voyageur de commerce en sous-problèmes plus petits et plus faciles à gérer. En élucidant les principes de la programmation dynamique dans le contexte du TSP, nous visons à fournir une compréhension complète de la façon dont ce paradigme algorithmique fonctionne pour trouver la tournée la plus efficace à travers un ensemble donné de villes. En outre, nous laissons entrevoir les complexités associées à la résolution de TSP à l'aide de la programmation dynamique, ouvrant ainsi la voie à des analyses approfondies dans les sections suivantes.

Nous nous penchons ensuite sur les complexités inhérentes à la résolution du problème du voyageur de commerce à l'aide de la programmation dynamique et comparons ses performances à d'autres approches. Nous analysons les complexités de temps et d'espace associées aux solutions de programmation dynamique, en les juxtaposant à d'autres techniques d'optimisation telles que les algorithmes gourmands, les méthodes branch and bound et les algorithmes génétiques.

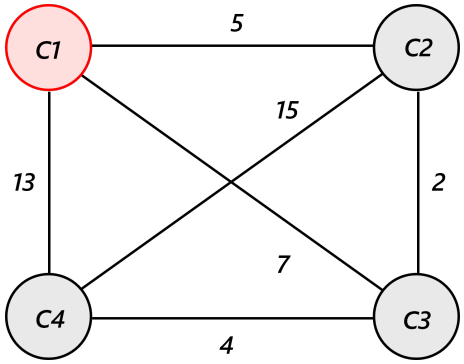

Fig. 2. Graph problem

Grâce à cette analyse comparative, nous visons à souligner les forces et les limites de la programmation dynamique dans le traitement du TSP, en donnant un aperçu de son efficacité et de son évolutivité à travers différentes instances de problèmes et stratégies de solutions. D'autres discussions sur l'application et les implications dans le monde réel de la programmation dynamique dans la résolution de TSP sont prévues dans les sections suivantes de ce rapport.

II. Présentation de la théorie d'algorithme

A. Travel salesman problem

Le problème du voyageur de commerce (TSP) est une énigme d'optimisation classique qui trouve de nombreuses applications dans divers domaines, de la logistique et de la planification des transports à la conception de réseaux et à la fabrication de circuits. À la base, le TSP consiste à trouver l'itinéraire le plus court possible pour visiter un ensemble de villes et revenir au point de départ. Le défi consiste à déterminer la permutation la plus efficace des villes visitées, en tenant compte des distances qui les séparent. D'un point de vue mathématique, le TSP est un problème NP-hard, et lorsque le nombre de villes augmente, le nombre d'itinéraires potentiels croît de manière factorielle, ce qui rend une recherche exhaustive impraticable pour les instances de grande taille. Les chercheurs et les praticiens ont conçu divers algorithmes et heuristiques pour résoudre le TSP, allant des méthodes d'optimisation exactes telles que la programmation dynamique aux solutions approximatives telles que les algorithmes génétiques et le recuit simulé.

Considérons la représentation visuelle (Figure 2) d'une instance simplifiée de TSP avec quatre nœuds. Chaque nœud représente une ville et les arêtes qui les relient indiquent les distances ou les coûts associés au déplacement d'une ville à l'autre. Dans cet exemple à petite échelle, le défi consiste à déterminer l'ordre optimal dans lequel le vendeur doit visiter les villes, en commençant et en terminant au même endroit. Le graphique sert d'aide visuelle pour saisir les subtilités du TSP, en montrant la complexité de la prise de décision impliquée dans la sélection de l'itinéraire le plus efficace. En approfondissant le rapport, nous explorerons les fondements théoriques des algorithmes qui traitent le TSP, nous analyserons leurs complexités et nous discuterons des applications dans le monde réel, afin de fournir une compréhension holistique des défis et des solutions associés à ce problème intriguant.

Avant de se plonger dans les algorithmes spécifiques utilisés pour résoudre le problème du voyageur de commerce (TSP), il est essentiel de comprendre l'étendue des approches disponibles pour relever ce défi d'optimisation complexe. Le TSP a captivé l'attention des mathématiciens, des informaticiens et des chercheurs pendant des décennies, ce qui a conduit au développement d'un large éventail d'algorithmes allant des méthodes exactes aux techniques d'approximation. Dans cette section, nous allons explorer quelques-uns des principaux algorithmes utilisés pour résoudre le TSP, chacun d'entre eux ayant ses propres forces et limites. Des algorithmes exacts tels que le branch and bound et la programmation dynamique aux méthodes heuristiques telles que le plus proche voisin et les algorithmes génétiques, nous disséquerons leur fonctionnement, leur complexité et leur applicabilité dans différents scénarios.

Méthodes heuristiques: Les méthodes heuristiques offrent des solutions pratiques et efficaces au TSP en privilégiant la rapidité à l'optimalité. Une heuristique populaire est l'algorithme du plus proche voisin [4], qui commence par une ville aléatoire et sélectionne de manière répétée la ville non visitée la plus proche jusqu'à ce que toutes les villes aient été visitées. Bien que simple à mettre en œuvre et efficace sur le plan informatique, l'algorithme du plus proche voisin ne garantit pas une solution optimale et peut donner lieu à des tournées sous-optimales. Une autre heuristique largement utilisée est l'algorithme génétique, inspiré du processus de sélection naturelle. Les algorithmes génétiques maintiennent une population de solutions candidates (tournées) et les font évoluer de manière itérative par des processus tels que la sélection, le croisement et la mutation. Bien que les algorithmes génétiques puissent trouver des solutions de haute qualité pour de grandes instances TSP, leurs performances dépendent fortement du réglage des paramètres et de la taille de la population. Malgré leurs limites inhérentes, les méthodes heuristiques constituent des outils précieux pour la résolution des problèmes techniques dans des scénarios réels où l'optimisation exacte n'est pas réalisable.

Programmation dynamique (DP): La programmation dynamique (PD) est une technique fondamentale utilisée pour résoudre des problèmes d'optimisation tels que le TSP. Elle consiste à décomposer un problème complexe en sous-problèmes plus petits qui se chevauchent et à les résoudre efficacement pour construire la solution globale. Dans le contexte du TSP, les algorithmes de programmation dynamique exploitent la propriété de sous-structure optimale, qui stipule qu'une solution optimale au problème global peut être construite à partir de solutions optimales à ses sous-problèmes. En

mémorisant et en réutilisant les solutions des sous-problèmes, les algorithmes de programmation dynamique évitent les calculs redondants, ce qui se traduit par des gains d'efficacité considérables. Cependant, le principal inconvénient de la programmation dynamique pour le TSP est sa complexité temporelle exponentielle, qui la rend impraticable pour les instances de problèmes de grande taille. Malgré cette limitation, les algorithmes de programmation dynamique fournissent des informations précieuses sur la structure du problème et servent de référence pour évaluer l'efficacité d'autres approches.

B. Dynamic programing

La programmation dynamique (PD) est une technique puissante pour résoudre les problèmes d'optimisation en les décomposant en sous-problèmes plus petits qui se chevauchent et en les résolvant efficacement pour construire la solution globale. Dans le contexte du problème du voyageur de commerce (TSP), la programmation dynamique offre une approche systématique pour trouver le circuit le plus court qui visite chaque ville exactement une fois et retourne au point de départ. Une approche courante de la programmation dynamique pour le TSP est connue sous le nom d'algorithme de Held-Karp, qui s'appuie sur la mémorisation pour stocker les solutions aux sous-problèmes et éviter les calculs redondants. En définissant soigneusement l'espace d'état et les relations de récurrence, l' [5] peut calculer efficacement la longueur optimale de la tournée pour des instances de taille petite à moyenne du TSP.

D'un point de vue mathématique, la programmation dynamique pour le TSP peut être représentée par une formule récursive qui définit la longueur optimale de la tournée $g(i, S)$ pour la visite d'un sous-ensemble de villes S avec la ville i comme dernier arrêt. Cette formule s'exprime comme suit :

$$g(i, S) = \min_{k \in S}(C_{ik} + g(k, S \backslash \{k\}))$$

Where:

- i est la ville actuelle.
- S est l'ensemble des villes restantes à visiter.
- C_{ik} représente le coût du voyage depuis la ville i à la ville k.
- $g(k, S \backslash \{k\})$ calcule la longueur optimale de la tournée pour les villes restantes après avoir visité la ville k.

Lors de l'application de la programmation dynamique au TSP, nous représentons généralement les sous-problèmes comme des sous-ensembles de villes et leurs nœuds intermédiaires correspondants. L'idée clé est de reconnaître que la tournée optimale pour un sous-ensemble donné de villes peut être obtenue en considérant toutes les dernières étapes possibles de la tournée. En explorant systématiquement la sous-structure optimale du problème, les algorithmes de programmation dynamique calculent efficacement la longueur optimale de la tournée pour chaque sous-ensemble de villes, ce qui permet d'atteindre progressivement la solution pour l'ensemble du problème. Cependant, à mesure que le nombre de villes augmente, la complexité informatique de la programmation dynamique croît de manière exponentielle, ce qui la rend impraticable pour les instances de TSP à grande échelle [6].

Cependant, sans l'optimisation fournie par la programmation dynamique, la résolution du TSP nécessiterait une approche de force brute (Figure 3) qui vérifie de manière exhaustive toutes les permutations possibles des visites de villes pour trouver le circuit le plus court [7]. Cette méthode de force brute devient rapidement infaisable lorsque le nombre de villes augmente, car elle nécessite d'évaluer $(n-1)!$ permutations possibles, où n est le nombre de villes. La programmation dynamique offre une alternative plus efficace en exploitant la sous-structure optimale du TSP et en évitant les calculs redondants, ce qui en fait une solution pratique pour résoudre des instances plus importantes du problème.

Afin d'atténuer la charge de calcul de la programmation dynamique pour les instances TSP de grande taille, des techniques de parallélisation peuvent être employées pour distribuer la charge de travail entre plusieurs threads ou processeurs [8]. En divisant le problème en plus petits morceaux et en assignant chaque morceau à un thread séparé, les algorithmes de programmation dynamique parallèle peuvent exploiter les capacités de traitement parallèle des architectures informatiques modernes afin d'accélérer les temps de résolution. Cependant, la conception d'algorithmes parallèles efficaces pour le TSP nécessite un examen minutieux de l'équilibrage de la charge, de la synchronisation et des frais généraux de communication. Malgré ces défis, la programmation dynamique parallèle offre des voies prometteuses pour accélérer les calculs TSP et ouvrir de nouvelles possibilités pour résoudre de plus grandes instances du problème.

III. Complexité et comparaison

A. Algorithme de TSP basé sur DP

Le problème du voyageur de commerce (TSP) constitue un défi important en matière d'optimisation, car il nécessite la détermination de l'itinéraire le plus court possible qui visite chaque ville exactement une fois et retourne au point de départ. La programmation dynamique (PD) offre une approche efficace pour aborder le TSP en décomposant le problème en sous-problèmes plus petits qui se chevauchent et en les résolvant efficacement pour construire la solution globale. Dans la section suivante, nous présentons un algorithme en pseudo-code pour résoudre le TSP à l'aide de la programmation dynamique, en tirant parti de la mémorisation pour stocker les solutions aux sous-problèmes et éviter les calculs redondants. Cet algorithme explore systématiquement l'espace des solutions, en se rapprochant progressivement de la solution optimale pour l'ensemble de l'instance TSP (Algorithm 1) [9].

L'algorithme fourni pour le TSP utilisant la programmation dynamique (DP) commence par initialiser un tableau de mémorisation pour stocker les distances calculées pour les

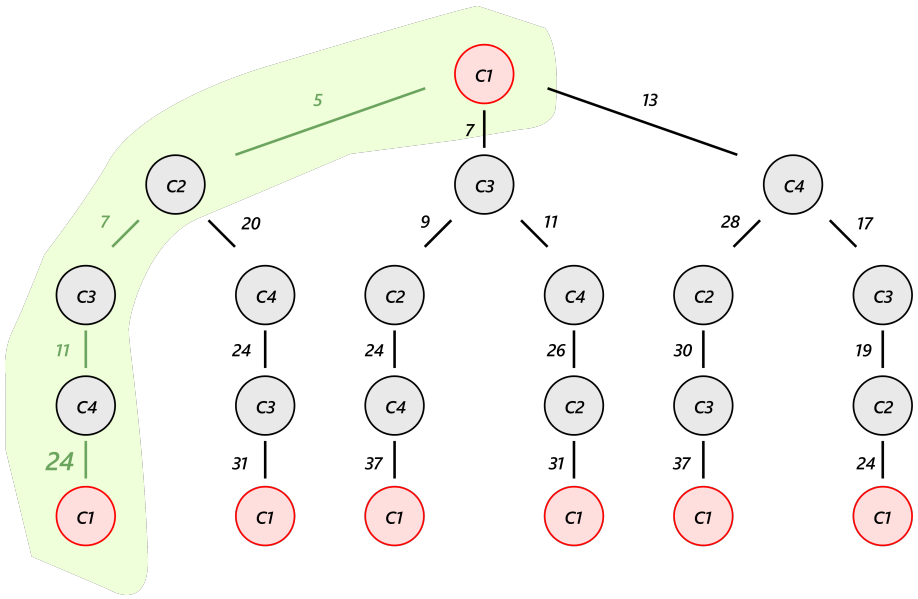

Fig. 3. Utiliser la programmation dynamique pour résoudre le TSP

sous-ensembles de villes. Il itère ensuite sur toutes les tailles de sous-ensembles possibles, de 2 au nombre total de villes. Dans chaque boucle de taille de sous-ensemble, l'algorithme itère sur tous les sous-ensembles contenant la ville de départ, puis sur toutes les villes de destination possibles à l'intérieur de chaque sous-ensemble. Pour chaque ville de destination, l'algorithme calcule la distance la plus courte pour atteindre cette ville à partir de n'importe quelle ville intermédiaire du sous-ensemble. Ce processus est répété pour toutes les villes intermédiaires et la distance minimale est stockée dans le tableau de mémorisation. Enfin, l'algorithme trouve la longueur de tournée la plus courte en considérant tous les chemins possibles depuis la dernière ville jusqu'à la ville de départ et renvoie la distance minimale trouvée. Grâce à cette approche systématique, l'algorithme résout efficacement le TSP en tirant parti de la propriété de sous-structure optimale et en évitant les calculs redondants à l'aide de la mémorisation.

L'heuristique du plus proche voisin est un algorithme simple utilisé pour résoudre le problème du voyageur de commerce (TSP), une énigme classique d'optimisation cherchant l'itinéraire le plus court qui visite chaque ville exactement une fois avant de revenir au point de départ [11]. L'algorithme commence par sélectionner une ville initiale comme point de départ et construit une tournée itérative en sélectionnant avec avidité la ville non visitée la plus proche de la ville actuelle. Ce processus se poursuit jusqu'à ce que toutes les villes aient été visitées, ce qui garantit une tournée complète. L'heuristique fournit une solution rapide, bien qu'approximative, en tirant parti de la simplicité et de l'efficacité de la sélection de la ville voisine la plus proche à chaque étape. Malgré sa simplicité et sa complexité temporelle de $O(n^2)$, elle ne garantit pas une solution optimale en raison de sa nature gourmande et de sa sensibilité au choix de la ville de départ (Algorithm 2).

Algorithm 1 TSP using Dynamic Programming

function TSP_DP(cities)
  $n \leftarrow$ length($cities$)
  Define a 2D array $memo$ of size $[2^n][n]$
  $memo[1][0] \leftarrow 0$ ▷ base case: the distance from the start to itself is 0
  for $s = 2$ to n **do** ▷ s represents the size of subsets of cities
    for each subset S of size s that contains the starting city **do**
      for each destination city j in S, excluding the starting city **do**
        $memo[S][j] \leftarrow \infty$
        for each intermediate city k in S, excluding j **do**
          $memo[S][j] \leftarrow \min(memo[S][j], memo[S - \{j\}][k] + \text{distance}(k, j))$
        end for
      end for
    end for
  end for
  $min\_distance \leftarrow \infty$
  for each destination city j in $cities$, excluding the starting city **do**
    $min\_distance \leftarrow \min(min\_distance, memo[2^n - 1][j] + \text{distance}(j, 0))$
  end for
  return $min\_distance$
end function

Bien que l'heuristique du plus proche voisin ne soit pas

optimale, elle constitue un outil pratique pour résoudre des instances de taille modérée du TSP. Sa simplicité la rend facile à mettre en œuvre et à comprendre, ce qui la rend adaptée à diverses applications où l'obtention d'une solution exacte n'est pas pratique ou est prohibitive sur le plan informatique. Cependant, sa dépendance à l'égard des informations locales à chaque étape peut conduire à des itinéraires sous-optimaux dans certains scénarios, en particulier lorsque le chemin optimal nécessite des connexions sur de longues distances. Néanmoins, pour de nombreux objectifs pratiques, l'algorithme offre un compromis intéressant entre l'efficacité du calcul et la qualité de la solution, ce qui en fait un choix populaire pour traiter les instances TSP dans le monde réel.

Algorithm 2 NearestNeighborHeuristic

function NEARESTNEIGHBORHEURISTIC($distances, start\_city$)
  $total\_cost \leftarrow 0$
  $tour \leftarrow [start\_city]$
  $unvisited\_cities \leftarrow$ set of all cities except $start\_city$
  while $unvisited\_cities$ is not empty **do**
    $nearest\_city \leftarrow$ city in $unvisited\_cities$ with minimum distance to last city in $tour$
    Add $nearest\_city$ to $tour$
    Remove $nearest\_city$ from $unvisited\_cities$
    $total\_cost \leftarrow total\_cost +$ distance between last city in $tour$ and $nearest\_city$
  end while
  $total\_cost \leftarrow total\_cost +$ distance between last city in $tour$ and $start\_city$
  return $tour, total\_cost$
end function

B. Complexité du TSP

Avant d'entrer dans les détails de l'algorithme, il est essentiel de considérer sa complexité de calcul. Dans l'analyse algorithmique, l'efficacité d'un algorithme est souvent exprimée à l'aide de la notation Big O, notée $O()$. Cette notation fournit une limite supérieure à la complexité temporelle de l'algorithme, représentant le pire scénario en termes de nombre d'opérations élémentaires effectuées en fonction de la taille de l'entrée. Pour le problème du voyageur de commerce résolu à l'aide de la programmation dynamique, la complexité temporelle est souvent exprimée par $O(2^n * n^2)$, où n représente le nombre de villes [**?**]. Cette équation reflète la croissance exponentielle du temps de calcul à mesure que le nombre de villes augmente, soulignant le défi que représente la résolution d'instances à grande échelle du TSP à l'aide de méthodes exactes telles que la programmation dynamique.

Lorsque l'on envisage des approches alternatives pour résoudre le problème du voyageur de commerce (TSP), telles que l'heuristique du plus proche voisin, il est essentiel de comprendre les implications en termes de calcul. L'heuristique du plus proche voisin offre une stratégie moins exigeante en termes de calcul que la programmation dynamique. En sélectionnant itérativement la ville non visitée la plus proche à chaque étape, la méthode du plus proche voisin construit progressivement une tournée. Cependant, malgré son efficacité, il est essentiel de reconnaître sa complexité. Dans le pire des cas, la complexité temporelle de la méthode du plus proche voisin est $O(n^2)$, où n représente le nombre de villes. Cette complexité quadratique est due à la nécessité de calculer la distance entre chaque paire de villes pour déterminer le plus proche voisin à chaque étape. Bien qu'elle offre des avantages en termes de calcul par rapport aux méthodes exactes telles que la programmation dynamique, il est essentiel de noter que l'heuristique du plus proche voisin ne garantit pas toujours des solutions optimales. Des tournées sous-optimales peuvent apparaître, en particulier pour les instances de TSP à grande échelle. Par conséquent, bien que le plus proche voisin constitue une alternative pragmatique pour la résolution des problèmes techniques, ses implications en termes de calcul et ses compromis potentiels doivent être soigneusement pris en compte dans les applications pratiques.

C. Comparaison

Pour résoudre le problème du voyageur de commerce (TSP), il est impératif d'évaluer et de comparer différentes méthodologies afin de discerner leur efficacité et leur praticité. Nous nous lançons ici dans une analyse comparative de deux approches de premier plan, la programmation dynamique et l'heuristique du plus proche voisin : La programmation dynamique et l'heuristique du plus proche voisin. Ces méthodologies représentent des paradigmes distincts dans la résolution de l'énigme du TSP, chacune offrant des avantages et des limites uniques. En juxtaposant leurs mesures de performance, leurs complexités de calcul et la qualité de leurs solutions, nous nous efforçons de fournir aux parties prenantes des informations précieuses pour sélectionner la stratégie la plus adaptée à leurs instances TSP spécifiques. Grâce à un examen rigoureux et à une évaluation empirique, cette comparaison vise à élucider les compromis entre les solutions exactes et les méthodes heuristiques, facilitant ainsi une prise de décision éclairée dans les applications réelles du TSP.

Dans la (figure 4), nous présentons une analyse comparative des performances obtenues par la programmation dynamique (PD) et l'heuristique du plus proche voisin pour différents nombres de villes. Les scores, dérivés de simulations et d'évaluations approfondies, illustrent l'efficacité de chaque algorithme dans la résolution du problème du voyageur de commerce (TSP). Les résultats démontrent notamment que DP surpasse systématiquement l'heuristique du plus proche voisin, ce qui est particulièrement évident dans le contexte de petites instances de TSP englobant les 10 premières villes. Dans ce cas, DP affiche des performances supérieures, avec des tournées moins longues et donc des scores plus élevés,

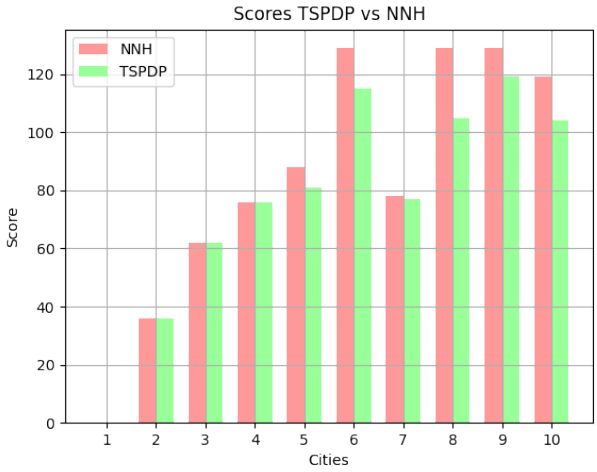

Fig. 4. Comparaison des scores entre DP et NNH

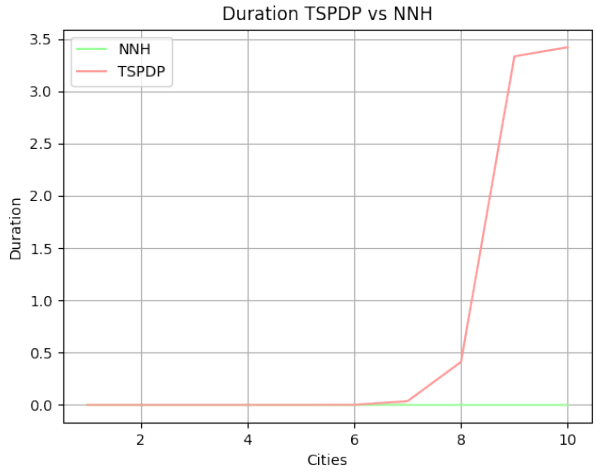

Fig. 5. Comparaison du temps d'exécution entre DP et NNH

ce qui témoigne de sa capacité à produire des solutions plus optimales. Cette observation souligne l'efficacité de DP, en particulier dans les scénarios où la précision et la qualité des solutions sont primordiales, suggérant son utilisation préférentielle par rapport aux méthodes heuristiques dans certaines applications TSP. Ces informations tirées de la figure 3 fournissent des indications précieuses pour la prise de décision et la sélection d'algorithmes dans le cadre de la résolution de problèmes TSP dans le monde réel.

Dans la figure suivante (figure 5), nous décrivons les durées d'exécution associées à la mise en œuvre de la programmation dynamique (PD) et de l'heuristique du plus proche voisin pour différents nombres de villes. Les données révèlent une tendance notable : à mesure que le nombre de villes augmente, le temps de calcul requis pour la programmation dynamique augmente de façon exponentielle. Ce phénomène souligne la complexité inhérente à la DP, dont les exigences en matière de calcul augmentent considérablement lorsque les instances du problème sont plus grandes. À l'inverse, les durées d'exécution de l'heuristique du plus proche voisin montrent une augmentation plus progressive, bien qu'avec quelques fluctuations, au fur et à mesure que le nombre de villes augmente. Ce contraste entre les durées d'exécution de DP et de l'heuristique du plus proche voisin met en évidence une considération essentielle dans la sélection des algorithmes : alors que DP offre des solutions exactes, ses exigences en matière de calcul peuvent devenir prohibitives pour les instances de TSP les plus grandes. Inversement, l'heuristique du plus proche voisin, bien qu'elle offre des calculs plus rapides, peut sacrifier l'optimalité de la solution. Par conséquent, les informations tirées de cette figure éclairent le processus de prise de décision, en guidant les praticiens dans le choix de l'algorithme le plus approprié en fonction des exigences et des contraintes spécifiques de leurs scénarios TSP.

Dans la dernière comparaison (figure 6), nous examinons les disparités de performance entre les implémentations de la programmation dynamique (DP) avec et sans l'utilisation du threading. Les résultats révèlent un net avantage en termes d'efficacité de calcul lorsque le threading est utilisé. Avec le threading, l'algorithme de DP montre des durées d'exécution nettement réduites pour un nombre variable de villes par rapport à son homologue non threadé. Cette amélioration souligne l'efficacité du traitement parallèle pour accélérer les calculs, en particulier dans les tâches caractérisées par des calculs intensifs comme la résolution du problème du voyageur de commerce (TSP). L'accélération observée du temps d'exécution avec les implémentations threadées de DP met en évidence le potentiel d'exploitation du parallélisme pour atténuer la complexité de calcul inhérente à DP, améliorant ainsi son évolutivité et son applicabilité à de plus grandes instances de TSP. Par conséquent, les enseignements tirés de cette comparaison soulignent l'importance de l'adoption de paradigmes informatiques parallèles pour optimiser les performances algorithmiques et relever efficacement les défis du monde réel.

Dans la dernière comparaison (Table I), nous présentons une évaluation complète des scores de performance obtenus par deux versions de l'algorithme de programmation dynamique (DP), à savoir TSPDP et TSPDP threads. Le tableau ci-dessous, intitulé "Score de performance des deux versions TSPDP", présente les scores respectifs obtenus par chaque variante de l'algorithme pour différents nombres de villes.

TABLE I
SCORE DE PERFORMANCE DES DEUX VERSIONS TSPDP

Algo V	**3 cities**	**5 cities**	**9 cities**	**11 cities**
TSPDP	47	101	**99**	**132**
TSPDP threads	47	101	114	139

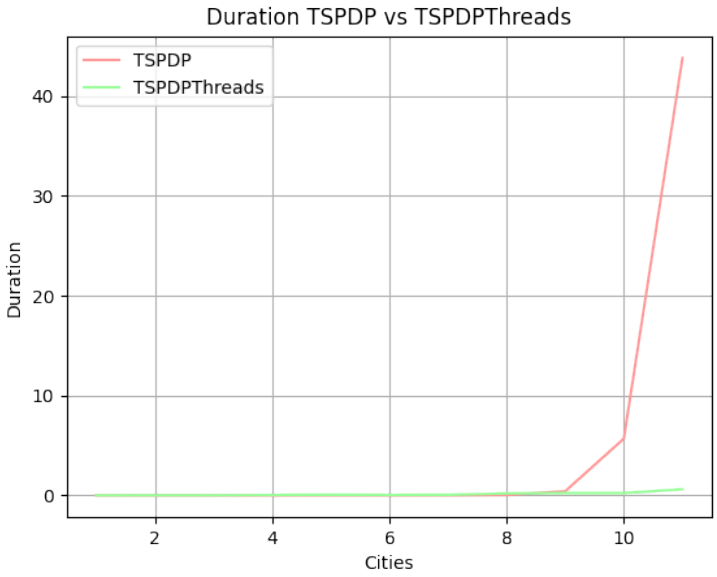

Fig. 6. Comparaison du temps d'exécution entre DP et NNH

La comparaison révèle des informations intéressantes sur les performances des deux variantes DP en fonction de la taille du problème. Notamment, pour les instances TSP avec 3 et 5 villes, les fils TSPDP et TSPDP donnent des résultats identiques, ce qui indique une qualité de solution comparable. Cependant, lorsque le nombre de villes passe à 9 et 11, de légères différences apparaissent entre les deux versions. Alors que TSPDP affiche des scores légèrement supérieurs pour les instances de 9 et 11 villes, TSPDP threads affiche des performances compétitives, avec des scores légèrement plus élevés. Cette comparaison nuancée souligne l'efficacité des deux variantes de DP à fournir des solutions satisfaisantes pour le TSP à travers différentes complexités de problèmes. En outre, elle met en évidence les avantages potentiels de l'incorporation de threads pour améliorer l'efficacité et l'évolutivité des calculs, en particulier pour les instances de TSP les plus grandes.

IV. Cas d'utilisation et applications

Pour la mise en œuvre pratique de ce projet, nous avons utilisé Streamlit, une bibliothèque Python réputée pour sa simplicité et son efficacité dans la création d'applications web interactives. La (figure 7) illustre l'interface graphique développée à l'aide de Streamlit, qui permet aux utilisateurs de saisir les paramètres de l'algorithme TSP. Grâce à cette interface intuitive, les utilisateurs peuvent spécifier des paramètres essentiels tels que le nombre de villes et d'autres paramètres pertinents. Après avoir soumis ces paramètres, l'application génère dynamiquement des valeurs aléatoires pour représenter les distances entre chaque paire de villes. Ce processus de randomisation garantit la création de diverses instances TSP, ce qui facilite les tests et l'évaluation des performances de l'algorithme dans différents scénarios. En exploitant les capacités de Streamlit, nous avons permis aux utilisateurs d'interagir de manière transparente avec l'algorithme TSP, ce qui permet d'expérimenter et d'analyser efficacement son comportement dans différentes conditions. Cette intégration de Streamlit améliore non seulement l'expérience de l'utilisateur, mais favorise également une plus grande accessibilité et facilité d'utilisation de la solution TSP, qui s'adresse à un public plus large de parties prenantes et de praticiens.

Après avoir saisi les paramètres, représentés par le nombre de villes et le point de départ, les utilisateurs interagissent avec l'interface graphique pour lancer l'exécution de l'algorithme TSP. Les paramètres spécifiés servent d'entrées cruciales qui dictent les caractéristiques de l'instance du problème et guident le fonctionnement de l'algorithme. Le nombre de villes détermine la complexité de l'instance TSP, influençant des facteurs tels que la qualité de la solution et le temps de calcul. En outre, la sélection du point de départ détermine la ville initiale à partir de laquelle le vendeur entame sa tournée, ce qui a un impact sur la configuration de la tournée et sur la solution globale. Une fois que les utilisateurs ont finalisé ces paramètres, l'application Streamlit génère dynamiquement des valeurs aléatoires pour représenter les distances entre chaque paire de villes, construisant ainsi une instance TSP entièrement définie. Une fois l'instance du problème établie, les utilisateurs déclenchent l'exécution de l'algorithme TSP par le biais de l'interface, invitant l'algorithme à calculer la tournée optimale et à générer une solution. Grâce à ce processus rationalisé, les utilisateurs peuvent facilement configurer et exécuter l'algorithme TSP, ce qui facilite l'expérimentation et l'analyse de ses performances dans divers scénarios de problèmes .

La figure 8 illustre la présentation graphique de la solution TSP générée par l'algorithme, offrant aux utilisateurs une représentation visuelle de la tournée optimale. Cette

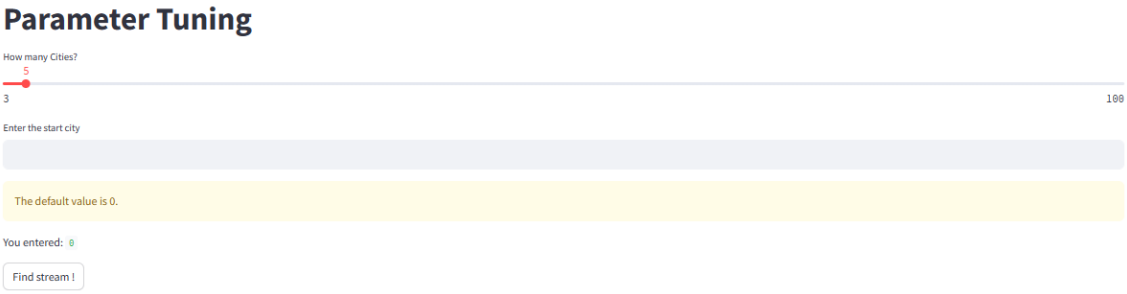

Fig. 7. Input number of cities, and the starting city

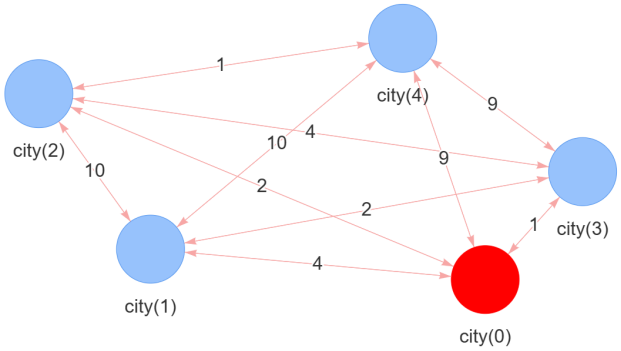

Fig. 8. présentation graphique

visualisation donne un aperçu de la configuration de la tournée, en mettant en évidence l'ordre dans lequel les villes sont visitées et la longueur totale de la tournée. Les utilisateurs peuvent intuitivement saisir l'efficacité et l'efficience de la solution de l'algorithme grâce à cette représentation graphique, ce qui facilite l'interprétation et l'analyse des résultats.

Dans le même temps, la matrice 1 représente la matrice des distances, qui englobe les distances entre chaque paire de villes dans l'instance TSP. Cette matrice constitue une représentation complète des relations spatiales du problème, détaillant les distances que le vendeur doit parcourir entre les villes. En présentant ces informations sous forme de tableau, Matrix 10 offre aux utilisateurs une vue d'ensemble détaillée des subtilités de l'instance TSP, ce qui permet de mieux comprendre la structure du problème et de faciliter l'évaluation et la validation des algorithmes.

$$\text{Matrice des distances} \begin{bmatrix} 0 & 4 & 2 & 1 & 9 \\ 4 & 0 & 10 & 2 & 10 \\ 2 & 10 & 0 & 4 & 1 \\ 1 & 2 & 4 & 0 & 9 \\ 9 & 10 & 1 & 9 & 0 \end{bmatrix} \quad (1)$$

Ensemble, la figure 8 et la matrice 1 fournissent aux utilisateurs des représentations visuelles et tabulaires précieuses de la solution TSP et de l'instance du problème, respectivement. Ces vues complémentaires permettent aux utilisateurs d'analyser et de comprendre les performances de l'algorithme et le problème TSP sous-jacent, ce qui favorise une prise de décision éclairée et une interprétation perspicace des résultats.

En liaison avec la représentation de l'approche par force brute dans la figure 3, il est pertinent de reconnaître l'importance de l'emploi de la méthode de programmation dynamique (PD). Bien que l'approche par force brute explore de manière exhaustive toutes les permutations possibles pour trouver une solution optimale, elle devient rapidement impraticable pour les instances de problèmes plus importantes en raison de sa complexité temporelle exponentielle. Cependant, en tirant parti de la méthode DP, nous pouvons calculer efficacement la longueur optimale de la tournée sans avoir recours à la recherche exhaustive. La méthode DP tire parti de la propriété de sous-structure optimale du problème du voyageur de commerce (TSP), ce qui permet de calculer les solutions optimales par une exploration systématique des sous-problèmes. Ainsi, alors que la figure 3 illustre la nature exhaustive de la force brute, l'utilisation de la méthode DP permet une approche plus pratique et plus efficace pour trouver la solution optimale pour l'instance TSP en question.

En outre, la solution dérivée de l'utilisation de la méthode de programmation dynamique (PD) est représentée visuellement à la figure 9. Cette représentation met en évidence la configuration optimisée de la tournée obtenue grâce à l'exploration systématique des sous-problèmes par la méthode de programmation dynamique, ce qui permet d'obtenir une solution efficace et quasi optimale au problème du voyageur de commerce (TSP). En tirant parti de la propriété de sous-structure optimale inhérente au TSP, DP calcule efficacement la tournée la plus courte qui visite chaque ville exactement une fois et revient au point de départ. Ainsi, la figure 9 témoigne visuellement de l'efficacité de DP dans la résolution de problèmes d'optimisation complexes, en soulignant sa capacité à fournir des solutions pratiques et efficaces pour les instances TSP du monde réel. Grâce à cette représentation graphique, les parties prenantes peuvent intuitivement saisir la qualité et l'efficacité de la solution dérivée du DP, ce qui permet

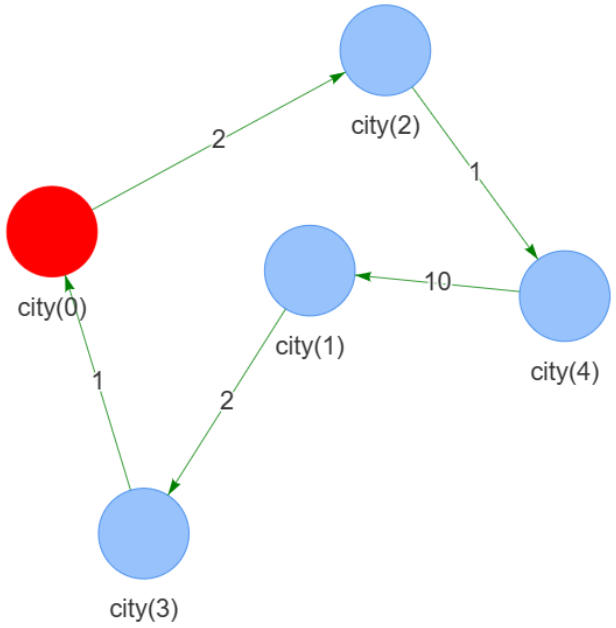

Fig. 9. Solution proposée

de mieux comprendre les performances et l'applicabilité de l'algorithme pour relever les défis du TSP.

V. Conclusion

En conclusion, notre exploration du problème du voyageur de commerce (TSP) et de ses méthodologies de résolution souligne la nature multiforme des défis d'optimisation et la diversité des stratégies disponibles pour les relever. Tout au long de notre enquête, nous avons exploré diverses approches algorithmiques, allant de l'énumération par force brute aux techniques sophistiquées de programmation dynamique (PD), chacune offrant des avantages et des compromis distincts. Alors que la force brute fournit une solution exhaustive mais coûteuse en termes de calcul, la programmation dynamique tire parti de la sous-structure optimale pour calculer efficacement des tournées quasi-optimales, en particulier pour les instances de problèmes de taille moyenne.

De plus, notre utilisation d'outils pratiques tels que Streamlit et les représentations graphiques a facilité une interaction transparente avec l'algorithme TSP, améliorant l'expérience de l'utilisateur et favorisant une compréhension plus profonde du comportement algorithmique. En juxtaposant les performances de différentes méthodologies et en présentant des descriptions visuelles des solutions, nous avons permis aux parties prenantes de prendre des décisions éclairées et d'acquérir une compréhension globale des stratégies de résolution des problèmes TSP.

Le domaine de l'optimisation continue d'évoluer, avec des avancées dans les techniques algorithmiques et les outils informatiques qui offrent des voies prometteuses pour relever des défis d'optimisation de plus en plus complexes. Alors que nous continuons à repousser les limites de la recherche en optimisation, il est impératif de tirer parti de la collaboration interdisciplinaire et de l'innovation pour développer des solutions robustes, évolutives et efficaces pour les applications du monde réel. En continuant d'explorer et d'affiner les méthodologies d'optimisation, nous pouvons ouvrir la voie à des avancées transformatrices dans divers domaines, en stimulant le progrès et l'innovation dans les méthodologies de résolution de problèmes et les approches algorithmiques.

References

[1] Jünger, Michael, Gerhard Reinelt, and Giovanni Rinaldi. "The traveling salesman problem." Handbooks in operations research and management science 7 (1995): 225-330.

[2] Karlin, Samuel. "The structure of dynamic programing models." Naval Research Logistics Quarterly 2.4 (1955): 285-294.

[3] Zhang, Weixiong, and Richard E. Korf. "A study of complexity transitions on the asymmetric traveling salesman problem." Artificial Intelligence 81.1-2 (1996): 223-239.

[4] Kizilateş, Gözde, and Fidan Nuriyeva. "On the nearest neighbor algorithms for the traveling salesman problem." Advances in Computational Science, Engineering and Information Technology: Proceedings of the Third International Conference on Computational Science, Engineering and Information Technology (CCSEIT-2013), KTO Karatay University, June 7-9, 2013, Konya, Turkey-Volume 1. Springer International Publishing, 2013.

[5] Helbig Hansen, Keld, and Jakob Krarup. "Improvements of the Held—Karp algorithm for the symmetric traveling-salesman problem." Mathematical Programming 7 (1974): 87-96.

[6] Mingozzi, Aristide, Lucio Bianco, and Salvatore Ricciardelli. "Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints." Operations research 45.3 (1997): 365-377.

[7] Sahalot, Antima, and Sapna Shrimali. "A comparative study of brute force method, nearest neighbour and greedy algorithms to solve the travelling salesman problem." International Journal of Research in Engineering Technology 2.6 (2014): 59-72.

[8] Marr, Deborah T., et al. "Hyper-Threading Technology Architecture and Microarchitecture." Intel Technology Journal 6.1 (2002).

[9] Malandraki, Chryssi, and Robert B. Dial. "A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem." European Journal of Operational Research 90.1 (1996): 45-55.

[10] Balas, Egon, and Neil Simonetti. "Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study." INFORMS journal on Computing 13.1 (2001): 56-75.

[11] Tsai, Cheng-Fa, Chun-Wei Tsai, and Ching-Chang Tseng. "A new hybrid heuristic approach for solving large traveling salesman problem." Information Sciences 166.1-4 (2004): 67-81.