Problème du voyageur canadien couvrant

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Introduction

Le problème du voyageur de commerce (*Traveling Salesman Problem* ou *TSP*) constitue l'un des défis classiques en optimisation combinatoire. Dans sa formulation standard, on cherche à déterminer le parcours de longueur minimale permettant de visiter une seule fois chaque ville d'un ensemble donné avant de revenir au point de départ. Malgré sa simplicité d'énoncé, ce problème est $\mathcal{NP}$-difficile.

Une variante moins étudiée mais particulièrement pertinente pour les applications pratiques est le problème du voyageur canadien couvrant (*Covering Canadian Travaller Problem* ou *CCTP*). Contrairement au TSP classique où l'ensemble des arêtes est connu à l'avance, le CCTP introduit un élément d'incertitude : certaines arêtes peuvent être bloquées, et un voyageur ne découvre qu'une arête est inaccessibles qu'en atteignant l'un de ses sommets adjacents. Cette contrainte reflète des situtations réelles comme la planification d'itinéraires dans des réseaux routiers où des blocages imprévus (accident, travaux) ne sont découverts qu'en arrivant à proximité.

Une spécialisation importante de ce problème est le k-CCTP, où le nombre d'arêtes bloquées est limité à k. Cette restriction permet de développer des algorithmes avec des garanties de performance théoriques. Dans ce contexte, on s'intéresse au rapport d'approximation, qui quantifie le ratio entre la distance parcourue par l'algorithme et celle qu'aurait parcourue un voyageur omniscient connaissant à l'avance l'emplacement de toutes les arêtes bloquées.

Dans ce document, nous présentons deux algorithmes qui ont été proposés pour résoudre le k-CCTP. L'algorithme CR et l'algorithme CNN, qui possède un rapport d'aproximation de $O\left(\sqrt{k}\right)$ et de $O(\log k)$ respectivement. Nous analysons les garanties théoriques et évaluons empiriquement leurs performances sur diverses classes de graphes.

Christofides

L'algorithme de Christofides, proposé en 1976, est une méthode d'approximation pour le problème du voyageur de commerce dans les graphes métriques, c'est-à-dire où les distances satisfont l'inégalité triangulaire. Cet algorithme garantit une solution dont le coût ne dépasse par 1.5 fois celui de la solution optimale, ce qui en fait l'un des algorithmes d'approximation les plus performants pour le TSP métrique.

Description de l'algorithme. L'algorithme de Christofides se déroule en cinq étapes principales :

1. Calcul d'un arbre couvrant de poids minimal T du graphe G.
2. Identification des sommets de degré impair dans T.
3. Calcul d'un couplage parfait de poids minimal M pour les sommets de degré impair.
4. Combinaison de T et M pour formet un multigraphe eulérien H.

5. Construction d'un circuit eulérien dans H, puis conversion en cycle hamiltonien par *raccourcissement* (élimination des sommets répétés).

Cette approche exploite le fait que tout graphe possédant uniquement des sommets de degré pair admet un circuit eulérien, c'est-à-dire un circuit qui emprunte chaque arête exactement une fois. En ajoutant le couplage minimal M à l'arbre couvrant T, on obtient un tel graphe.

Preuve du rapport d'approximation. Cette section présente la preuve de l'article de N. Chrisofides (1976).

(Théorème) L'algorithme de Christofides garantit un rapport d'aprpoximation de 1.5 pour le problème du voyageur de commerce dans les graphes métriques.

(Preuve) Soit $G = (V, E)$ un graphe complet métrique et OPT le coût de la solution optimale au TSP sur G. Nous voulons montrer que le coût du cycle hamiltonien produit par l'algorithme de Christofides est au plus $\frac{3}{2} \cdot \text{OPT}$.

L'algorithme de Christofides fonctionne par construction intérmédiaires, nous analyserons leurs coûts.

Montrons que $c(T) \leq \text{OPT}$. Soit C^* un cycle hamiltonien optimal. En supprimant une arête quelconque de C^*, nous obtenons un arbre couvrant T'. Puisque T est un arbre couvrant de poids minimal, nous avons : $c(T) \leq c(T') \leq \text{OPT}$.

Montrons que $c(M) \leq \frac{\text{OPT}}{2}$. Soit $O \subset V$ l'ensemble des sommets de degré impair dans T. D'après le lemme des poignées de mains, qui stipule « chaque graphe non orienté fini a un nombre pair de sommets de degré impair », $|O|$ est pair. Considérons le cycle hamiltonien optimal C^*. En parcourant C^* de manière alternée, nous pouvons partitionner ses arêtes en deux ensembles E_1 et E_2 tels que chaque ensemble forme une collection de chemins dont les extremités sont exactement les sommets de O et $c(E_1) + c(E_2) = \text{OPT}$. Par conséquent, l'un de ces ensembles, disons E_1, a un coût au plus égal à $\frac{\text{OPT}}{2}$. Les chemins dans E_1 induisent un couplage valide M' entre les sommets de O. En vertu de l'inégalité triangulaire, le coût de ce couplage M' est au plus égal au coût des chemins correspodnants dans E_1. Pusique M est un couplage parfait de poids minimal entre les sommets de O, nous avons : $c(M) \leq c(M') \leq c(E_1) \leq \frac{\text{OPT}}{2}$.

Le coût du multigraphe eulérien H est donc : $c(H) = c(T) + c(M) \leq \text{OPT} + \frac{\text{OPT}}{2} = \frac{3}{2} \cdot \text{OPT}$.

Le circuit eulérien parcourt exactement les arêtes de H, donc son coût est égal à $c(H)$.

Le cycle hamiltonien est obtenu en *raccourcissant* le circuit eulérien, c'est-à-dire en prenant des raccourcis directs lorsqu'un sommet est revisité. Grâce à l'inégalité triangulaire, ces raccourcis ne peuvent qu'améliorer (ou au pire maintenir) le coût. Donc le coût du cycle hamiltonien final est au plus $c(H) \leq \frac{3}{2} \cdot \text{OPT}$. ■

Implémentation et validation. Notre implémentation de l'algorithme de Christofides utilise *Python* et la bibliothèque *NetworkX* pour représenter les graphes. Cette implémentation est disponible dans le fichier `cctp/christofides.py`. Pour valider la correction de l'implémentation, nous avons développé une suite de tests unitaires vérifiant que :

– le tour commence et se termine au même sommet ;
– le tour généré visite tous les sommets du graphe ;
– le tour généré passe au plus une fois par chaque sommet.

Pour assurer la robustesse de notre implémentation face à différentes configurations, nous avons également effectué 200 tests *fuzzy* sur des graphes générés aléatoirement avec un nombre de sommet variant entre 4 et 256.

Cadre expérimental. Pour évaluer empiriquement les performances de l'algorithmes, nous avons généré des graphes complets aléatoires respectant l'inégalité triangulaire. Chaque graphe est construit en plaçant n points aléatoirement

dans un espace euclidien à deux dimensions, avec des coordonnées tirées uniformément dans $[-5.0, 5.0]$. Les distances entre les sommets corresponent aux distances euclidiennes, garantissant ainsi l'inégalité triangulaire.

Pour chaque taille de graphe, nous avons effectué 15 mesures indépendantes afin d'obtenir des résultats statistiquement significatifs pour des comparaisons. Notre analyse reporte les statistiques suivantes : la valeur minimale et maximale, le premier et troisième quartile et la moyenne inter-quartile.

Analyse du rapport d'approximation. Pour évaluer le rapport d'approximation empirique, nous avons comparé le coût des solutions générées par l'algorithme de Christofides au coût optimal obtenu par une recherche exhaustive, nous limitons la taille de ces instances à 12 nœuds pour se ramener à des instances calculables en un temps raisonnable.

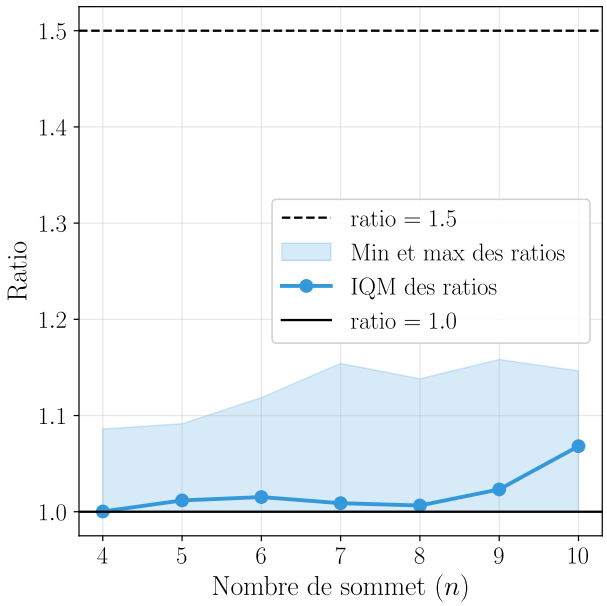

Fig. 1. – Évaluation du rapport d'approximation de l'algorithme de Christofides sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (15 instances par taille). La courbe bleue représente la moyenne interquartile et la zone bleutée représente l'intervalle entre le minimum et le maximum.

La Fig. 1 présente l'évolution du rapport d'approximation en fonction du nombre de sommets. Les résultats montrent que, bien que le rapport théorique soit de 1.5, le rapport observé en pratique est généralement meilleur, se situant autout de 1.1 pour les graphes euclidiens aléatoires. Ce résultat est cohérent avec la borne pessimiste de 1.5 donnée par l'algorithme de Christofides.

Analyse de complexité. La complexité théorique de l'algorithme de Christofides est dominée par le calcul du couplage parfait de poids minimal, qui peut être résolu en $O(n^3)$ où n est le nombre de sommets. Pour vérifier cette complexité empiriquement, nous avons mesuré le temps d'exécution de l'algorithme sur des graphes de tailles croissantes.

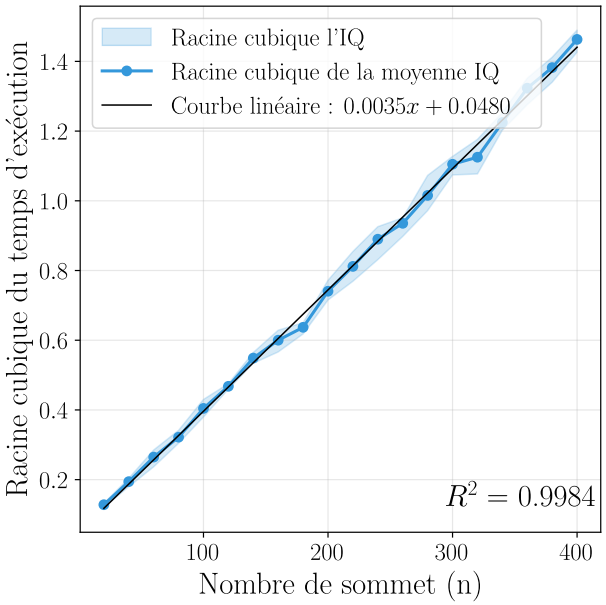

Fig. 2. – Évolution du temps d'exécution de l'algorithme de Christofides sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (15 mesures par taille). La courbe bleue représente la moyenne interquartile et la zone bleutée l'intervalle interquartile. La courbe noire représente le résultat de la regréssion linéaire sur les racines cubiques des IQM, R^2 représente le cohéfficient de linéarité, avec des valeur proches de 1 indiquant une forte corrélation linéaire.

La Fig. 2 présente les résultats de cette analyse, avec le temps d'exécution en fonction du nombre de sommets. Nous avons appliqué une régression linéaire sur la racine cubique du temps d'exécution, obtenant un coefficient de corrélation linéaire R^2 proche de 1, ce qui confrime la complexité théorique en $O(n^3)$. Cette analyse empirique valide l'analyse théorique et donne également une estimation pratique des constantes impliquées, permettant de prédire le temps d'execution pour de plus grandes instances.

Cr

L'algorithme CR (Cyclic Routing) de C.-S. Liao et Y. Huang permet de résoudre le problème du voyageur candien avec un rapport d'approximation de $O\left(\sqrt{k}\right)$.

Description de l'algorithme.

1. Calcul d'un tour initial par l'algorithme de Christofides P
2. Tant qu'il restent des noeuds non-visités : parcourir le graphe P en appliquant la procédure `shortcut` dans le sens de visite de P, changer le sens de visite si nécessaire

Après l'application de `shortcut`, le sens de visite change si :

– le nombre de noeuds non-visités n'a pas deminiué
– l'itération précédente de `shortcut` s'est arrêtée avant d'atteindre le dernier noeud non-visité

L'algorithme `shortcut` se déroule en deux phases principales :

1. assumer que toutes les noeuds non-visités sont accessibles
2. tant qu'il restent des noeuds non visités et accessibles : essayer d'atteindre le noeud non-visité le plus proche, si nécessaire par des noeud intermédiaires (toujours dans le sens de visite)

(Théorème) Le ratio d'approximation de CR est de $O\left(\sqrt{k}\right)$.

(Preuve) Soit E_i une liste des blockages découvertes dans l'itération i de l'algorithme CR, et soit m^{cr} le nombre total d'itérations. Comme le voyageur candien ne redécourvre jamais un blockage qu'il connaît déjà le nombre total de blockages k est borné par :

$$|E_1| + |E_2| + ... + |E_{m^{\text{cr}}}| \leq k$$

Soit V_m une liste de noeuds non-visités lors de l'itération m. Pour chaque blockage découvert l'lors de l'itération m, au moins un noeud est supprimé de la liste des noeuds non-visités V_{m+1}, d'où :

$$|V_2| + |V_3| + ... + |V_{m^{\text{cr}+1}}| \leq$$
$$|E_1| + |E_2| + ... + |E_{m^{\text{cr}}}|$$

avec dans le pire cas $|V_m \setminus V_{m+1}| = 1$, $|V_2| = m^{\text{cr}} - 1$ et $|V_{m^{\text{cr}+1}}| = 0$. En utilisant la formule de la Somme de Gauss, on obtient :

$$\frac{(1 + (m_{\text{cr}} - 1))(m_{\text{cr}} - 1)}{2} \leq k$$
$$\Rightarrow m_{\text{cr}} \leq \left\lfloor \frac{1 + \sqrt{1 + 8k}}{2} \right\rfloor$$

Soit $d(P^{\text{cr}})$ le tour final retourné par l'algorithme CR. Le coût total de ce tour est borné par :

$$d(P^{\text{cr}}) \leq (3m^{\text{cr}} + 1)\text{OPT}, \text{ avec } m^{\text{cr}} = O\left(\sqrt{k}\right)$$

Par conséquent, le ratio d'approximation de l'algorithme CR est de $O\left(\sqrt{k}\right)$. ■

Implémentation et validation. L'implémentation de l'algorithme CR est disponible dans le fichier `cstp/cr.py` et consiste de trois fonctions principales :

– `canadian_traveller_cyclic_routing` : En utilisant l'algorithme de Christofides, cette fonction calcule un tour T et renomme ensuite tous les sommets et blocages du graphe initial pour que leur ordre reflète celui du parcours, de façon croissante
– `cyclic_routing` et `shortcut` implémentent la fonctionnalité décrite ci-dessus

Pour valider la correction de l'implémentation, nous procédons de la même manière que pour

l'algorithme de Christofides. En plus, nous vérifions que le graphe retourné par l'algorithme CR ne contient pas d'arrêts bloqués.

Cadre expérimental. Nous réutilisons ici les mêmes méthodes déjà utilisée durant les autres parties.

Cnn

L'algorithme CNN (Christofides Nearest Neighbor) représente une amélioration par rapport à l'algorithme CR, offrant un rapport d'approximation de $O(\log k)$ et découvert par N. Hahn et M. Xefteris.

Description de l'algorithme. L'algorithme CNN se déroule en quatre phases principales.

1. Calcul d'un tour initial par Christofides T.
2. Parcours du tour en utilisant la procédure Shortcut utilisé par CR. Suite à ce tour, si on ne termine pas au nœud initial, on revient sur ses pas jusqu'à ce qu'on y revienne. Lors de cette phase, à chaque passage de sommet, on note les arêtes qui sont bloquées.
3. On construit un graphe de connaissance H, ce graphe contient tous les sommets du graphe et toutes les arêtes adjacentes à des arêtes visitées lors du tour initial et non bloquées. C'est avec ce graphe de connaissance que l'on peut construire un multigraphe d'exploration G', qui contient tous les nœuds qui n'ont pas pu être exploré lors du tour initial, et le sommet initial. Ce graphe contient entre tous deux sommets u et v deux chemins : d'une part un chemin risqué, qui correspond à l'arête (u, v) de G, mais dont on ne peut pas être sur de si il est accessible ou non, et d'autres part, un chemin alternatif, calculé dans H, mais plus long. On a donc deux options pour chaque passage, soit, si il est accessible, prendre le chemin court, sinon, prendre le chemin long, garanti d'être accessible car calculé dans le graphe de connaissance, mais qui fait faire un détour.
4. La dernière phase utilise l'algorithme du plus proche voisin (*Nearest Neighbor*) pour compléter le tour en visitant tous les nœuds de G'. L'algorithme sélectionne itérativement le nœud le plus proche accessible par un chemin non bloqué. Une fois tous les nœuds visitiés, l'algorithme revient au point de départ pour compléter le tour.

Preuve du rapport d'approximation. Cette section présente la preuve de l'article de N. Hahn et M. Xefteris (2023).

(Théorème) L'algorithme CNN a un rapport d'approximation de $O(\log k)$ pour le problème k-CCTP, où k est le nombre maximal d'arêtes bloquées.

(Preuve) Soit G un graphe complet métrique et T le tour initial calculé par l'algorithme de Christofides. Puisque Christofides garantit une $\frac{3}{2}$ -approximation du TSP, nous avons :

$$c(T) \leq \frac{3}{2} \cdot \mathrm{OPT}$$

Lors de la phase de raccourci, l'algorithme suit T jusqu'à rencontrer des arêtes bloquées, puis retourne au point de départ. Soit T_S le tour obtenue à la suite de la phase de raccourci. Dans le pire cas l'algorithme parcourt le tour puis revient sur ses pas donc :

$$c(T_S) \leq 3 \cdot \mathrm{OPT}$$

Après cette phase, il reste un ensemble de sommets non visités U. Remarquons que chaque sommet non visité correspond à au moins une arête bloqué découverte pendant la phase de raccourci. En effet, pour qu'un sommet reste non visité, l'arête qui aurait permis d'y accéder dans le tour T devait être bloquée. Puisqu'il y a au plus k arêtes bloquée, nous avons $|U| \leq k$. Par conséquent, le graphe d'exploration G' contient au plus $k + 1$ sommets (les sommets non visités plus le sommet de départ).

Pour compléter le tour, l'algorithme utilise l'heuristique du plus proche voisin sur G'. Pour chaque paire de sommets dans G', il existe toujours au moins un chemin sûr dans le graphe de connaissance. En effet, le graphe original G

est complet avec un degré $n-1$ pour chaque sommet. Pour isoler complètement un sommet dans le graphe de connaissance, il faudrait que toutes ses arêtes soient bloquées, ce qui nécessiterait $n-1$ arêtes bloquées. Comme $k < n-1$, cela est impossible. Ainsi, l'algorithme du plus proche voisin peut toujours progresser en utilisant soit le chemin direct (s'il n'est pas bloqué), soit le chemin sûr par le graphe de connaissance.

Il est établit que sur un graphe arbitraire de n sommets, l'algorithme du plus proche voisin a un rapport d'aproximation de $O(\log n)$. Appliqué au graphe d'exploration de taille au plus $k+1$, cela donne un rapport $O(\log(k+1)) = O(\log k)$.

Le cout optimal pour visiter tous les sommets de G' dans le graphe original ne peut pas être inférieur à la solution optimale OPT. Soit T_{NN} le tour obtenu après la phase d'exploration, nous avons :

$$c(T_{\text{NN}}) = O(\log k) \cdot \text{OPT}$$

En combinant les coûts des deux phases, nous obtenons un coût total d'au plus :

$$\begin{aligned} c(T_{\text{CNN}}) &= c(T_S) + c(T_{\text{NN}}) \\ &= 3 \cdot \text{OPT} + O(\log k) \cdot \text{OPT} = O(\log k) \end{aligned}$$

Ce qui établit le rapport d'approximation de $O(\log k)$ pour l'algorithme CNN.

Implémentation et validation. Notre implémentation de l'algorithme CNN est disponible dans le fichier `cctp/cnn.py`. Pour valider la correction de l'implémentation, nous avons développé une suite de tests unitaires vérifiant que :

- le tour commence et se termine au même sommet ;
- le tour généré visite tous les sommets du graphe ;
- le tour généré passe au plus une fois par chaque sommet.

Pour assurer la robustesse de notre implémentation face à différentes configurations, nous avons également effectué 200 tests *fuzzy* sur des graphes générés aléatoirement avec un nombre de sommet variant entre 4 et 256.

Cadre expérimental. Nous réutilisons ici les mêmes méthodes déjà utilisée durant les autres parties.

Analyse du rapport d'approximation. Pour évaluons le rapport d'approximation empirique de CNN, on cherche une classe de graphe qui atteint la borne de complexité. Cela est le cas pour la famille de graphe qui sert d'exemple au fait que la borne en $O(\log k)$ est serrée proposée par N. Hahn et M. Xefteris.

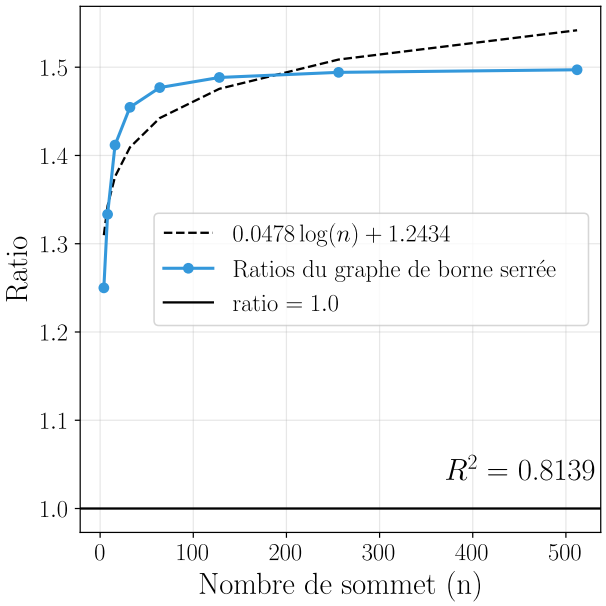

Fig. 3. – Évolution du rapport d'approximation de l'algorithme CNN sur des graphes de borne serrée en fonction du nombre de sommets. La courbe bleue avec marqueurs représente les ratios mesurés, tandis que la ligne pointillée noire montre la régression logarithmique avec son équation. La ligne horizontale noire indique le ratio idéal de 1.0.

Le graphe consiste en une chaîne de triangles commençant en v_0 avec un sommet supplémentaire u connecté uniquement au premier sommet. Les arêtes reliant u au reste du graphe sont bloquée, ce qui fait que le tour après la phase de raccourci ne contient que v_0 et u. Contrairement à l'article, nous avons fait le choix de laisser les autres sommets pour avoir $k < n-1$ et que l'algorithme puisse tourner.

La Fig. 3 présente les résultats de cette analyse. Comme prévu par la théorie, le rapport d'approximation croît logairthmiquement avec la taille du graphe, confirmant la borne en $O(\log k)$. L'ajustement d'une courbe logarithmique aux données mesurées montre une forte corrélation, avec un coefficient de corrélation logarithmique R^2 de 0.81. Ce résultat confirme que la borne supérieure théorique est effectivement serrée, car il existe des instances pour lesquelles l'algorithme CNN atteint un rapport d'approximation de $\Omega(\log k)$.

Cette figure confirme aussi notre implémentation, on observe effectivement l'évolution attenue par l'analyse.

Analyse de complexité. La complexité temporelle de l'algorithme CNN est dominée par celle de l'algorithme CNN, c'est pourquoi nous retirons la phase de calcul du tour de Christofides de la mesure de performance temporelle. On s'intéresse alors à la complexité du reste de l'exécution de l'algorithme.

La phase de raccourci a une complexité de $O(n)$ due à l'algorithme de Christofides. La phase de calcul du graphe de connaissance et du graphe d'exploration implique le calcul de $O(k^2)$ plus courts chemins dans un graphe de taille n, pour une complexité de $O(k^2 \cdot n^2)$. La phase d'exploration a une complexité de $O(k^2)$ car elle applique l'algorithme du plus proche voisin sur un graphe de $k+1$ sommets.

La complexité totale est donc $O(k^2 + n^2)$. Pour $k \ll n$, cette complexité est dominée par $O(n^2)$. On note en particulier, que l'observation faite durant l'évaluation du rapport d'approximation tient toujours ici : pour faire plus que le passage initial, il faut qu'au moins un nœud bloqué soit dans le tour de Christofides initial, mais cette probabilité réduit à mesure que n grandit. Pour vérifier cette complexité empiriquement, nous avons mesuré le temps d'exécution de l'algorithme sur des graphes de tailles croissantes.

La Fig. 4 présente les résultats de cette analyse, avec le temps d'exécution en fonction du nombre de sommets. Nous avons appliqué une régression linéaire, obtenant un coefficient de corrélation linéaire R^2 proche de 1, ce qui confirme la complexité théorique en $O(n^2)$.

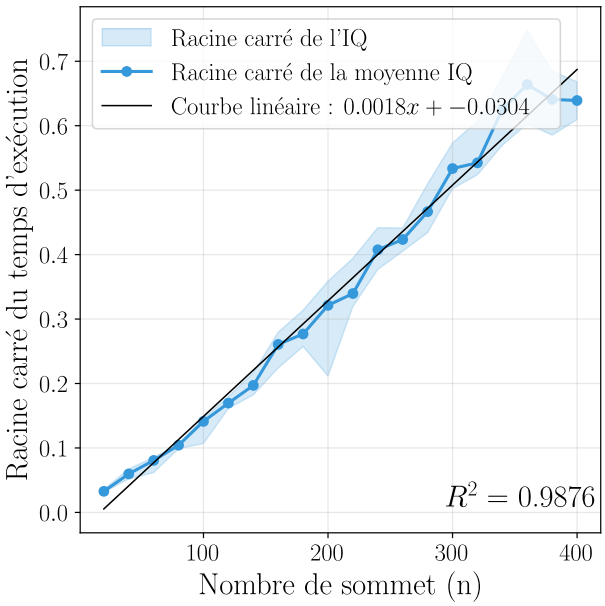

Fig. 4. – Évolution du temps d'exécution de l'algorithme CNN sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (15 mesures par taille). La courbe bleue représente la moyenne interquartile et la zone bleutée l'intervalle interquartile. La courbe noire représente le résultat de la regréssion linéaire sur les racines cubiques des IQM, R^2 représente le cohéfficient de linéarité, avec des valeur proches de 1 indiquant une forte corrélation linéaire. On fixe k à $n-2$.

La Fig. 5 présente les résultats de cette analyse, avec le temps d'exécution en fonction du nombre de sommets bloqué. Nous avons appliqué une régression linéaire, obtenant un coefficient de corrélation linéaire R^2 proche de 1, ce qui confirme la complexité théorique en $O(k^2)$. On note tout de même que la variance est grande sur cette mesure en comparaison à celle observée sur l'évolution en fonction de n.

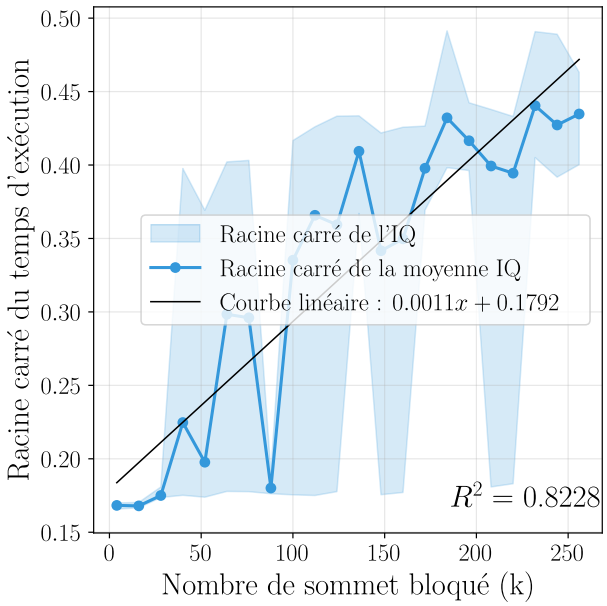

Fig. 5. – Évolution du temps d'exécution de l'algorithme CNN sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (25 mesures par taille). La courbe bleue représente la moyenne interquartile et la zone bleutée l'intervalle interquartile. La courbe noire représente le résultat de la regréssion linéaire sur les racines cubiques des IQM, R^2 représente le cohéfficient de linéarité, avec des valeur proches de 1 indiquant une forte corrélation linéaire. On fixe n à 258.

Comparaisons

Conclusion