Problème du voyageur canadien couvrant

Abstract

Dans ce rapport, nous analysons et comparons deux algorithmes pour le problème du *Voyageur Canadien Couvrant* : l'algorithme Cr avec un rapport d'approximation de $O\left(\sqrt{k}\right)$ et l'algorithme Cnn avec un rapport d'approximation de $O(\log k)$, où k est le nombre d'arêtes bloquées. Nous validons empiriquement ces bornes théoriques à l'aide de constructions spécifiques et évaluons leur performance sur diverses classes de graphes pour déterminer quand chaque algorithme est préférable en pratique.

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 [2], 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).

(Thérorème) 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 1000 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.

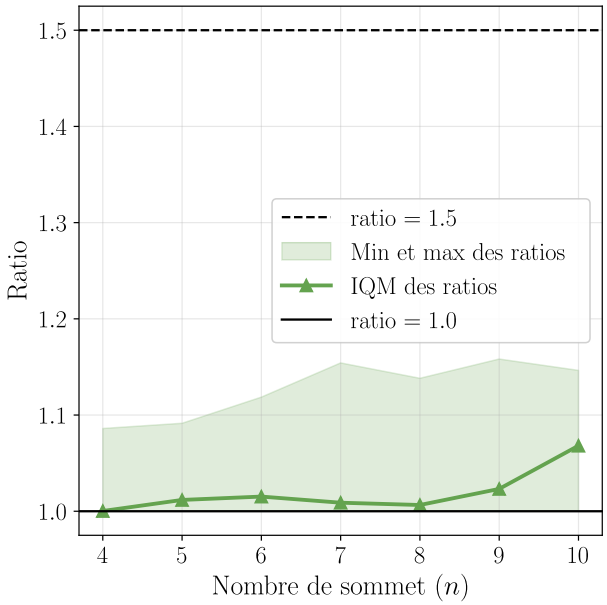

Fig. 1. – Évaluation du rapport d'approximation de l'algorithme de Christofides sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (15 instances par taille). La courbe représente la moyenne interquartile et la zone teintée représente l'intervalle entre le minimum et le maximum.

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.

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.

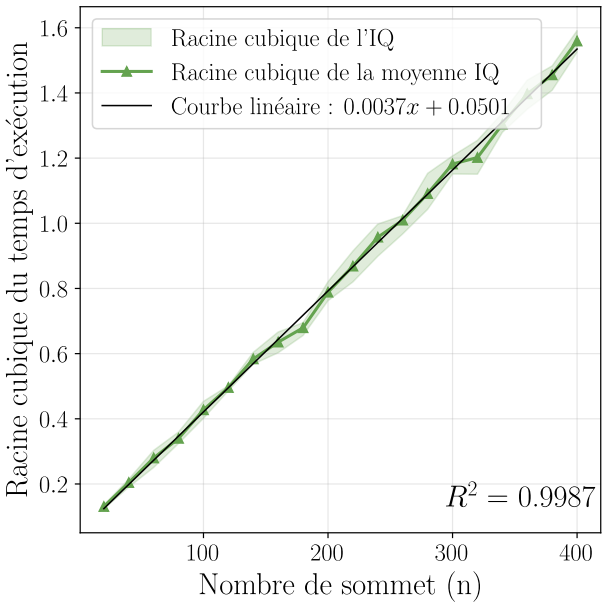

Fig. 2. – Évolution du temps d'exécution de l'algorithme de Christofides sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (15 mesures par taille). La courbe représente la moyenne interquartile et la zone teintée l'intervalle interquartile. La courbe noire représente le 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.

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.

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 [1] (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. L'algorithme Cr repose sur le raisonnement suivant : l'itinéraire complet peut être décomposé en plusieurs tours ; à chaque tour, le voyageur tente de visiter le plus grand nombre possible de sommets en suivant l'ordre de visite de la tournée dérivée de l'algorithme de Christofides.

L'algorithme se déroule en trois phases principales :

1. Calcul d'un tour initial T avec Christofides.
2. Appliquer des opérations de raccourci pour éviter les arêtes bloquées.
3. Parcourir le graphe en alternant les directions lorsque nécessaire.

Plus précisément, soit $T : s = v_1 - v_2 - ... - v_n - s$ le tour calculé par l'algorithme de Christofides. L'algorithme Cr explore les sommets non visités via des raccourcis sur le tour T tout en découvrant des blocages. L'algorithme suit l'ordre de visite de T et parcourt un raccourci vers T à travers autant de sommets non visités que possible dans chaque tour.

Lors de chaque itération m, nous définissons V_m l'ensemble des sommets non visités au début du tour m. L'algorithme tente alors de visiter tous les sommets de V_m en suivant l'ordre du tour T ou l'ordre inverse, selon le résultat du tour précédent. Si lors d'un tour, le voyageur ne parvient pas à réduire le nombre de sommets non visités ou s'arrête avant d'atteindre le dernier sommet non visité, la direction est inversée pour le tour suivant.

La procédure de raccourci Shortcut fonctionne de la manière suivante. Le voyageur tente de visiter chaque sommet non visité en suivant l'ordre du tour (ou l'ordre inverse, comme expliqué précédemment). Lorsque l'algorithme est au sommet u et cherche à aller au sommet v, trois cas sont possibles. Soit (u, v) n'est pas bloquée, ce qui peut être détecté, car on est à un sommet adjacent de cette arête, dans ce cas, on parcourt (u, v) pour aller en v. Si (u, v) est bloqué, alors, comme v est le prochain sommet bloqué à visiter, alors tout sommet entre u et v, notons w dans l'ordre initial de Christofides a été déjà visité à une itération précédente, sans quoi w serait le prochain sommet à visiter. On cherche alors w tel que (u, w) et (w, v) ne sont pas bloqués, ce que l'on sait parce que l'on est déjà passé au sommet w à une itération précédente et que ce sommet est adjacent à ces deux arêtes. Si un tel sommet w n'existe pas, alors, on abandonne et on cherche à aller au prochain sommet v', qui n'a pas encore été visité et est après v dans l'ordre du tour.

L'algorithme répète ce processus jusqu'à ce que tous les sommets soitent visités, puis retourne au point de départ.

Preuve du rapport d'approximation. Cette section présente la preuve de l'article de C.-S. Liao et Y. Huang (2014).

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

(Preuve) Pour démontrer ce rapport d'approximation, nous allons d'abord établir plusieurs propriétés de l'algorithme Cr, puis les combiner pour obtenir le résultat final.

On note M le nombre total d'itérations effectuées par l'algorithme Cr, V_m l'ensemble des sommets non visités au début de l'itération m, avec $(1 \leq m \leq M)$. Finalement, E_m est l'ensemble des arêtes bloquées découvertes lors de l'itération m.

Tout d'abord, nous pouvons montrer que l'algorithme Cr visite au moins un sommet à chaque itération (*Lemme 4.1*). Cela implique que :

$$|V_1| > ... > |V_m| > ... |V_M|$$

Ensuite, nous pouvons démontrer que les ensembles d'arêtes bloquées découvertes à différentes itérations sont disjoints (*Lemme 4.2*). En effet, pour qu'une arête bloquée soit découverte à l'itération j, ses deux extremités doivent être dans V_j. Si l'une des extrémités a été visitée lors d'une itération précédente i, alors l'arete ne peut pas être découverte à l'itération j. On a donc :

$$E_i \cap E_j = \emptyset, \quad \forall 1 \leq i < j \leq M$$

Nous pouvons également établir que le nombre d'arêtes bloquées découvertes à l'itiration m est au moins égal au nombre de sommets qui restent non visités après cette itération (*Lemme 4.3*). En effet, pour chaque sommet v qui reste non visité dans V_{m+1}, il doit exister une arête bloquée découverte durant l'itération m qui empêche d'atteindre v. On a donc :

$$|E_m| \geq |V_{m+1}| \quad \forall i \leq m \leq M$$

Ces propriétés nous permettent de borner le nombre total d'itérations M. Puisque les ensembles E_m sont disjoints et que leur union contient au plus k arrêtes bloquées, nous avons :

$$|E_1| + |E_2| + ... + |E_M| \leq k$$

De plus, en combinant avec le résultat du Lemme 4.3, nous obtenons :

$$|V_2| + |V_3| + ... + |V_{M+1}|$$

$$\leq |E_1| + |E_2| + ... + |E_M| \leq k$$

Dans le pire cas, selon le Lemme 4.1, l'algorithme ne visite qu'un seul sommet par itération, ce qui implique $|V_m \setminus V_{m+1}| = 1$ pour tout m. Cela signifie que $|V_{M+1}| = 0$, $|V_M| = 1$, et ainsi de suite jusqu'à $|V_2| = M - 2$.

En utilisant cette relation, on obtient :

$$\frac{(1 + (M-1))(M-1)}{2} \leq k$$

En résolvant cette inéquation avec M, on obtient :

$$M \leq \left\lfloor \frac{1 + \sqrt{1 + 8k}}{2} \right\rfloor = O\big(\sqrt{k}\big)$$

Finalement, pour calculer le coût total du tour, on établit que le coût de chaque itération est au plus $3 \times \mathrm{OPT}$ (*Lemme 4.4*). En ajoutant le cout du retour final au point de départ (au plus OPT), le coût total est :

$$c(T_{\mathrm{Cr}}) \leq (3M + 1) \times \mathrm{OPT} = O\big(\sqrt{k}\big) \times \mathrm{OPT}$$

Ce qui établit le rapport d'approximation de $O\big(\sqrt{k}\big)$ pour l'algorithme Cr. ■

Implémentation et validation. Notre implémentation de l'algorithme Cr est disponible dans le fichier `cctp/cr.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 1000 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 dans la première partie, cependant pour le tirage aléatoire des

nœuds bloquées, nous effectuons un tirage sans remise de k arêtes parmis les arêtes du graphes.

Analyse du rapport d'approximation. Pour évaluer le rapport d'approximation empirique de CR, nous avons construit une classe de graphes qui atteint la borne de complexité théorique. Cette construction s'inspire directement de l'exemple fourni par C.-S. Liao et Y. Huang démontrant que la borne en $O\left(\sqrt{k}\right)$ est sérrée.

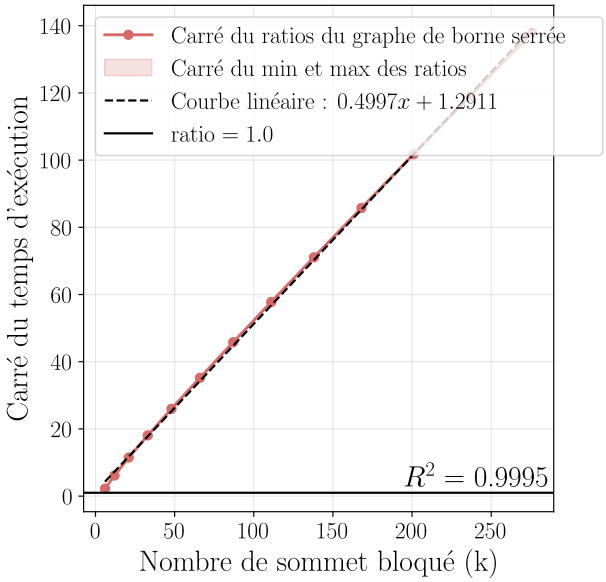

Fig. 3. – Évolution du rapport d'approximation de l'algorithme CR sur des graphes de borne serrée en fonction du nombre de sommets. La courbe avec marqueurs représente les carrés des ratios mesurées tandis que la ligne pointillée noire montre la régression linéaire avec son équation. La ligne horizontale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

Notre implémentation du graphe de borne serrée organise les sommets en une structure circulaire comprenant trois groupes principaux reliés par des sommets intermédiaires. Les sommets au sein de chaque groupe sont positionnés très près les uns des autres, tandis que les groupes eux-mêmes sont placés à distance significative. Pour un paramètre n donné, nous créons un total de $3(\left(\frac{p(p+1)}{2}+1\right)$ sommets et définissons un chemin reliant des sommets de sorte que la première itération ne passe que par les sommets non compris dans les groupes, puis que dans chaque itération suivante, on ne passe que par un sommet de chaque groupe.

Le principle de cette construction est de bloquer toutes les arêtes sauf celles du chemin trouvé. Bien que cela implique de bloquer plus que $k = n - 2$ arêtes, cette approche ne compromet pas la validité de l'algorithme car elle préserve la connectivité du graphe via le chemin. Cette décision permet de se rapprocher du comportement théorique décrit dans la preuve. Si nous avions choisi de bloquer moins d'arêtes, choisies par un tirage aléatoire, le rapport d'approximation aurait été moins proche du rapport théorique de $O\left(\sqrt{k}\right)$ décrit dans la preuve.

La Fig. 3 présente les résultats de cette analyse, et confirment que le rapport d'approximation croît linéairement avec $O\left(\sqrt{k}\right)$, comme le montre la figure. La régression linéaire sur les valeurs mesurées démontre un corrélation presque parfaite ($R^2 = 0.9995$), validant que la borne théorique est effectivement serrée.

Cette figure valide également notre implémentation, puisque nous observons exactement l'évoluation du rapport d'approximation prédite par l'analyse théorique.

Analyse de complexité. La complexité temporelle de l'algorithme CR est dominée par celle de l'algorithme de CHRISTOFIDES, 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 procédure de raccourci SHORTCUT est exécutée à chaque itération et peut examiner jusqu'à $O(n)$ sommets non visités. Pour chaque sommet, dans le pire des cas, l'algorithme doit vérifier jusqu'à $O(n)$ sommets déjà visités pour trouver un chemin alternatif, ce qui donne une complexité de $O(n^2)$ par itération. Comme montré dans la preuve du rapport d'approximation, le nombre d'itération est borné par $O\left(\sqrt{k}\right)$, où k est le nombre d'aretes bloquées.

La complexité totale est donc $O\left(\sqrt{k} \times n^2\right)$. Pour $k \ll n$, cette complexité est dominée par $O(n^2)$. Pour vérifier cette complexité empiriquement, nous avons mesuré le temps d'exécution de l'algorithme sur des graphes de tailles croissantes.

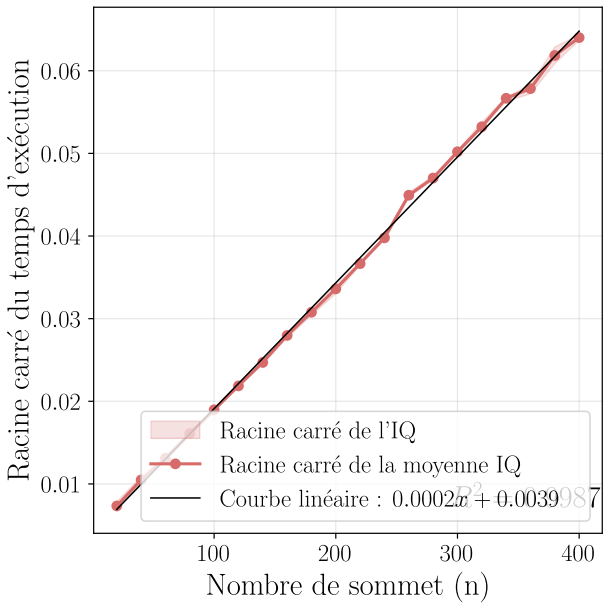

Fig. 4. – Évolution du temps d'exécution de l'algorithme Cr sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (15 mesures par taille). La courbe représente la moyenne interquartile et la zone teintée l'intervalle interquartile. La courbe noire représente le résultat de la regréssion linéaire sur les racines carrées 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. 4 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 carrée du temps d'exécution, obtenant un coefficient de corrélation linéaire R^2 proche de 1, ce qui confirme la complexité en $O(n^2)$. Ces résultats montrent une croissance quadratique du temps d'exécution lorsque la taille du graphe augmente.

La Fig. 5 présente les résultats de cette analyse, avec le temps d'exécution en fonction du nombre d'arêtes bloquées. Nous avons appliqué une régression linéaire sur la racine carrée du temps d'exécution, obtenant un coefficient de corrélation linéaire R^2 d'environ 0.84, ce qui indique une corrélation modérément forte avec la complexité théorique. On note que la variabilité est plus importante dans ce cas, suggérant que la distribution des arêtes bloquées influence davantage la performance de l'algorithme que leur nombre total. On note finalement, critiquement, que la complexité est directement liée au rapport d'approximation. Sur ce graphe, nous prenons des instances aléatoires, on est donc dans des cas bien meilleurs que le pire cas. C'est donc normal que nous ne voyons pas directement la complexité en $O\left(\sqrt{k}\right)$, car ici la complexité peut être bien plus variée.

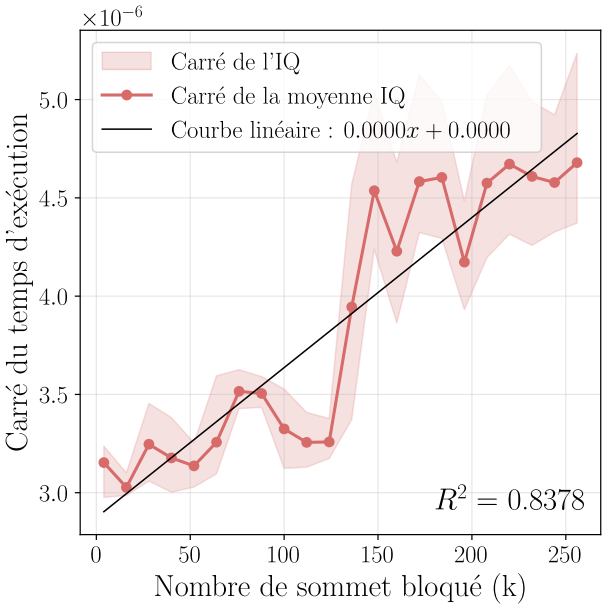

Fig. 5. – Évolution du temps d'exécution de l'algorithme Cr sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (25 mesures par taille). La courbe représente la moyenne interquartile et la zone teintée l'intervalle interquartile. La courbe noire représente le résultat de la regréssion linéaire sur les carrés 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.

Cnn

L'algorithme Cnn [3] (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 T avec Christofides.
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 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 utili-

sant 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 1000 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.

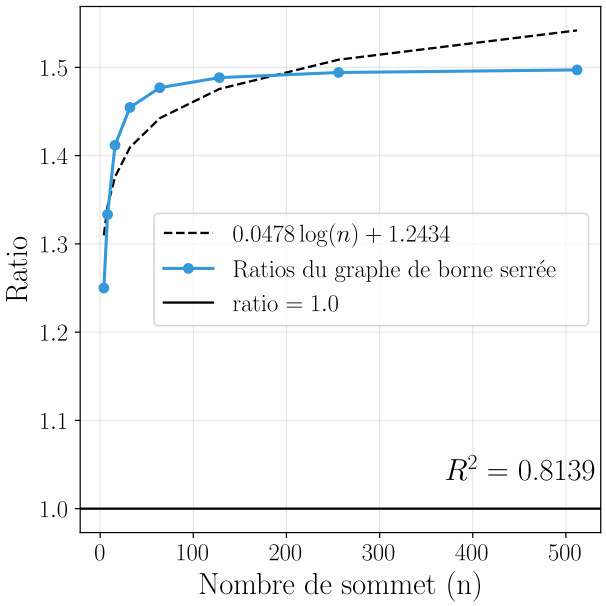

Fig. 6. – Évolution du rapport d'approximation de l'algorithme Cnn sur des graphes de borne serrée en fonction du nombre de sommets. La courbe avec marqueurs représente les ratios mesurés, tandis que la ligne pointillée noire montre la régression logarithmique avec son équation. La ligne horizontale 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. 6 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 de Christofides, 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.

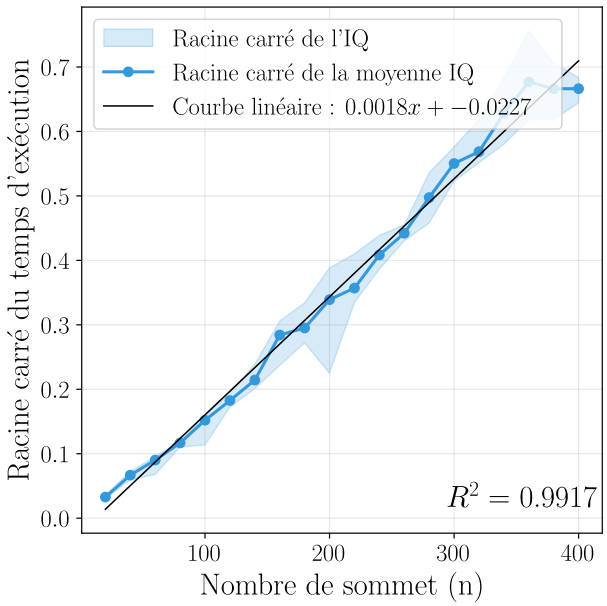

Fig. 7. – Évolution du temps d'exécution de l'algorithme Cnn sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (15 mesures par taille). La courbe représente la moyenne interquartile et la zone teintée l'intervalle interquartile. La courbe noire représente le résultat de la regréssion linéaire sur les racines carrées 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 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 $O(n^2)$. Pour vérifier cette complexité empiriquement, nous avons mesuré le temps d'exécution de l'algorithme sur des graphes de tailles croissantes.

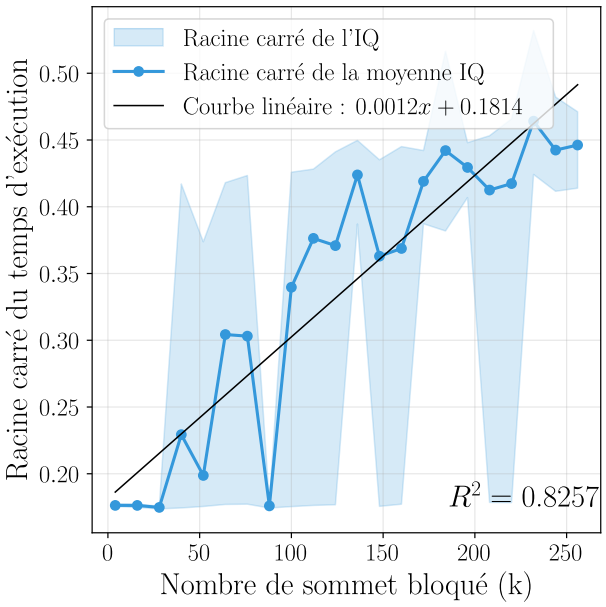

Fig. 8. – Évolution du temps d'exécution de l'algorithme Cnn sur des instances du TSP générées aléatoirement en fonction du nombre de sommet (25 mesures par taille). La courbe représente la moyenne interquartile et la zone teintée l'intervalle interquartile. La courbe noire représente le résultat de la regréssion linéaire sur les racines carrées 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.

La Fig. 7 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 $O(n^2)$.

La Fig. 8 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.

Comparaisons

Afin d'évaluer les perforamnces relatives des algorithmes Cr et Cnn dans différents contextes réels, nous anvons mené une analyse comparative détaillée sur cinq classes distinctes de graphes. Cette comparaison empirique vient compléter l'analyse théorique du rapport d'approximation présentée dans les sections précédentes. Pour chaque classe de graphe, nous étudions l'évolution des performances en fonction de deux paramètres : le nombre de sommets n et le nombre d'arêtes bloquées k.

Notre protocole d'évalution mesure le rapport 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. Ce ratio, calculé sur de multiples instances (15 par configuration), permet de caractériser empiriquement le comportement moyen et la variabilité des performances des algorithmes.

Graphe à poids constant. Les graphes à poids constant représentent un cas particulier où toutes les arêtes ont un poids identique, fixé arbitrairement à 1. Ces graphes constituent une référence théorique importante car ils permettent d'isoler l'impact de la topologie du graphe sur les performances des algorithmes, indépendamment des variations de distance.

Pour construire ces graphes, nous générons un graphe complet à n sommets où chaque paire de sommets est reliée par une arête de poids 1. Cette structure satisfait l'inégalité triangulaire puisque le coût d'un chemin direct entre deux sommets est toujours inférieur à celui d'un chemin indirect.

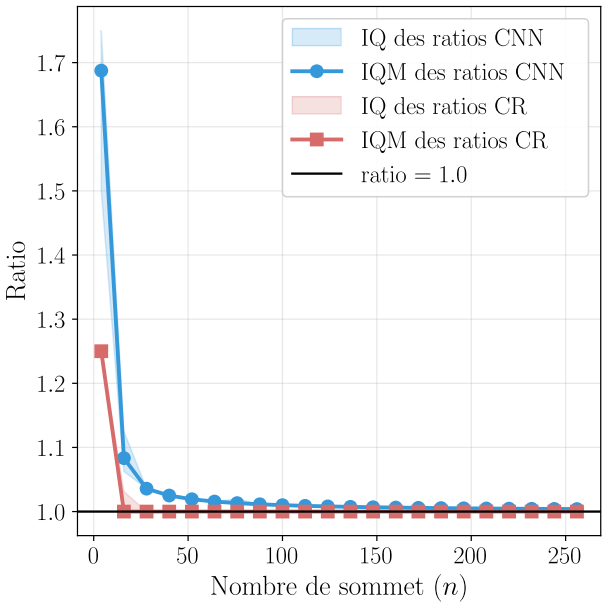

Fig. 9. – Évolution du rapport d'approximation des algorithmes Cr et Cnn sur des graphes à poids constant en fonction du nombre de sommet n. La courbe bleue représente la moyenne interquartile (IQM) des ratios Cnn tandis que la courbe rouge représente celle des ratios Cr. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe d'arêtes bloquées $k = n - 2$ pour chaque valeur de n, sur 15 instances aléatoires par configuration.

Les résultats présentés dans la Fig. 9 montre que pour les petites valeurs de n, les deux algorithmes présentent des rapport d'approximation relativement élevés (1.7 pour Cnn et 1.25 pour Cr), qui décroissent rapidement à mesure que n augmente. Pour $n > 50$, les deux algorthmes convergent vers un rapport très proche de 1, ce qui indique une performance quasi-optimale.

La Fig. 10 montre quant à elle que Cr maintient un rapport d'approximation constant de 1 quelle que soit la valeur de k tandis que Cnn présente quelques fluctuations limitées en fonction de k. On note tout de meme que même dans les pire

des cas, Cnn ne monte que jusqu'à 1.004, ce qui reste très proche de l'optimal et correspond aux observations faites sur le précédent graphe.

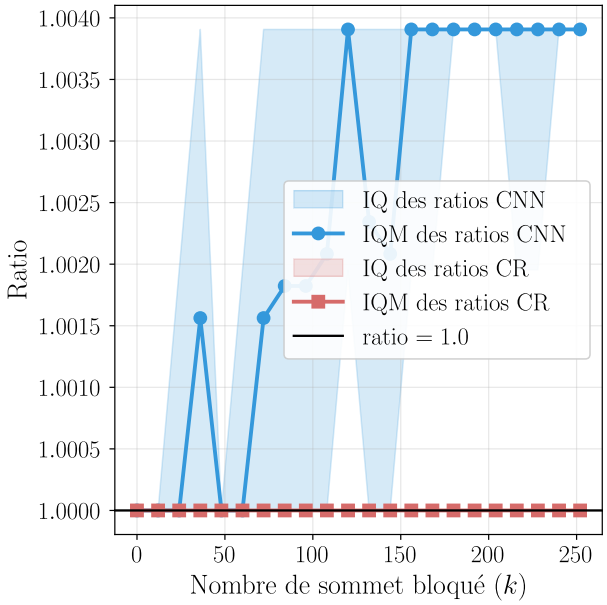

Fig. 10. – Évolution du rapport d'approximation des algorithmes Cr et Cnn sur des graphes à poids constant en fonction du nombre d'arêtes bloquées k. La courbe bleue représente la moyenne interquartile (IQM) des ratios Cnn, tandis que la courbe rouge représente celle des ratios Cr. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe de sommets $n = 256$, sur 15 instances aléatoires par configuration.

Graphes euclidiens. Les graphes euclidiens constituent une classe importante pour les applications réelles de planification d'itinéraire. Dans ces graphes, les sommets représentent des points dans un espace euclidien à deux dimensions, et les poids des arêtes correspondent aux distances euclidiennes entre ces deux points.

Pour construire ces graphes, nous générons n points avec des coordonnées aléatoires uniformément distribuées, dans l'intervalle $[-5, 5]^2$. Nous créons ensuite une graphe complet où le poids de chaque arête est la distance euclidienne entre les sommets correspondants. Cette méthode garantit que l'inégalité triangulaire est respectée, pusique dans un espace euclidien, la distance directe entre deux points est toujours inférieure ou égale à la somme des distances via un point intermédiaire.

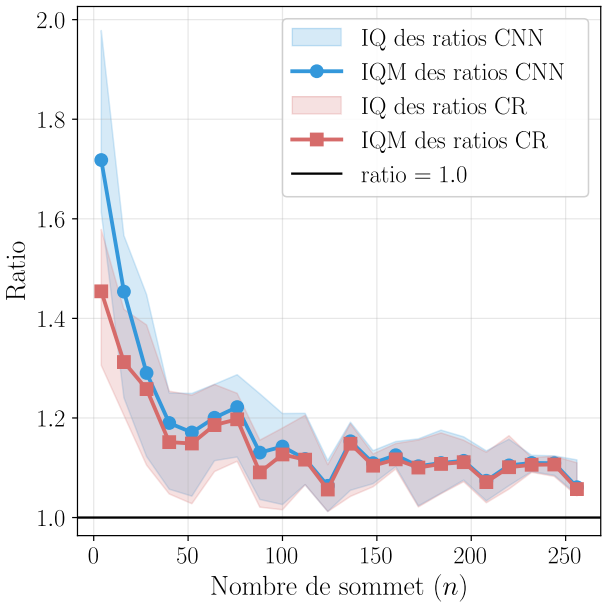

Fig. 11. – Évolution du rapport d'approximation des algorithmes Cr et Cnn sur des graphes euclidien en fonction du nombre de sommet n. La courbe bleue représente la moyenne interquartile (IQM) des ratios Cnn tandis que la courbe rouge représente celle des ratios Cr. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe d'arêtes bloquées $k = n - 2$ pour chaque valeur de n, sur 15 instances aléatoires par configuration.

La Fig. 11 montre que les rapports d'approximation de Cr et Cnn suivent des trajectoirs similaires, diminuant rapidement avec l'augmentation de n jusqu'à atteindre un plateau autour de 1.1 pour les grandes valeurs de n. Cette convergence s'explique par le fait que lorsque le nombre de sommets augument, la probabilité qu'une arete bloquée se trouve sur le chemin optimal diminue, réduisant ainsi l'impact des blocages sur la performance globale.

La Fig. 12 révèle une autre tendance : les performances des deux algorithmes se dégradente progressivement à mesure que k augmente, avec

une variabilité croissante. Cette détérioration progressive a du sens, plus le nombre d'arête bloquées est élevé, plus les détours nécessaires sont fréquents et potentiellement coûteux. On note en particulier que, bien que CR et CNN utilisent des méthodes très différentes, les performances sont très similaires, ce qui suggère des solutions proches de l'optimal.

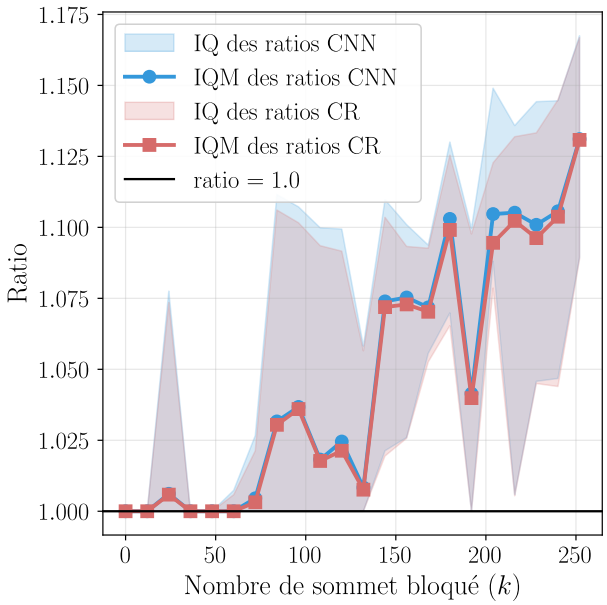

Fig. 12. – Évolution du rapport d'approximation des algorithmes CR et CNN sur des graphes euclidien en fonction du nombre d'arêtes bloquées k. La courbe bleue représente la moyenne interquartile (IQM) des ratios CNN, tandis que la courbe rouge représente celle des ratios CR. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe de sommets $n = 256$, sur 15 instances aléatoires par configuration.

Graphe de Manhattan. Les graphe en grille avec distances de Manhattan représentent un modèle particulièrement pertinent pour la planficiation d'iténéraires urbains, où les déplacements s'effectuent généralement selon une structure en quadrillage.

Pour construire ces graphes, nous générons une grille de n sommets disposé régulièrement dans un espace bidimmensionnel, formant une grille carrée. Chaque sommet est relié à tous les autres sommets, et le poids d'une arête entre deux sommets correspond à leur distance de Manhattan, c'est-à-dire la somme des différences absolues de leur coordonnées. Cette métrique respecte l'inégalité triangulaire.

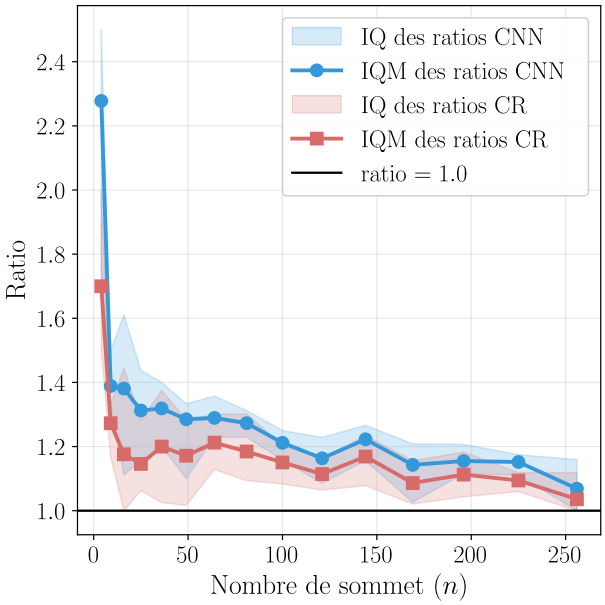

Fig. 13. – Évolution du rapport d'approximation des algorithmes CR et CNN sur des graphes de Manhattan en fonction du nombre de sommet n. La courbe bleue représente la moyenne interquartile (IQM) des ratios CNN tandis que la courbe rouge représente celle des ratios CR. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe d'arêtes bloquées $k = n - 2$ pour chaque valeur de n, sur 15 instances aléatoires par configuration.

La Fig. 13 révèle que les rapports d'approximation sont particlièrement élevés (2.25 pour CNN et 1.7 pour CR), mais diminuent rapidement avec l'augmentation de n pour se stabiliser autour de 1.1. Cette diminution rapide s'explique par la multiplication des chemins alternatifs disponibles lorsque la taille de la grille augumente, offrant plus d'options pour contourner les aretes bloquées.

La Fig. 14 montre une flucturation plus importante des performances par rapport aux autres classes de graphes. Pour les faibles valeurs de

k, le deux algorithmes présentent des performances similaires, proche de l'optimal. Cependant avec l'augmentation de k, les performances se dégradent avec une variabilité accrue, tout en restant globalement sous un ratio de 1.15. Cette variabilité accrue peut s'expliquer par la structure particulière des grilles, où le blocage de certaines arêtes stratégique peut forcer des détours considérables, tandis que le blocage d'autres arêtes peut avoir un impact minimal. On note que ces résultats correspondent à l'intuition des touristes dans des villes comme Manhattan, où des voies bloquées forcent un détour complet du bloc, qui représente un grand détour.

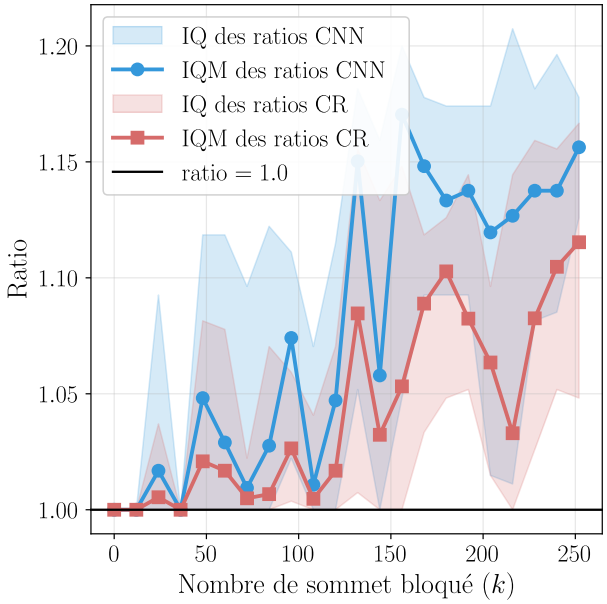

Fig. 14. – Évolution du rapport d'approximation des algorithmes CR et CNN sur des graphes de Manhattan en fonction du nombre d'arêtes bloquées k. La courbe bleue représente la moyenne interquartile (IQM) des ratios CNN, tandis que la courbe rouge représente celle des ratios CR. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe de sommets $n = 256$, sur 15 instances aléatoires par configuration.

Graphe fortement clusterisés. Les graphes fortement clusterisés modélisent des réseaux présentant une structure communautaire prononcée, où des groupes de sommets sont fortement interconnectés, tandis que les connexions entre groupes sont plus rares ou plus coûteuses. Ce type de structure se retrouve dans de nombreux réseaux réels, notamment les réseaux de transport régionaux avec des zones urbaines denses reliées par des axes interurbains.

Pour construire ces graphes, nous générons un ensemble de clusters, chacun contenant un nombre de sommet pour arriver à un total de n sommets. Les centres des clusters sont positionnés aléatoirement dans l'espace, mais à des distances significatives les uns des autres (facteur multiplicatif de 20). Les sommets des chaque cluster sont ensuite positionnés dans un voisinage proche de leur centre de cluster (distance maximale de 1). Cette construction génère naturellement des communcautés distinctes avec des distances intra-cluster faibles et des distances inter-clusters élevées.

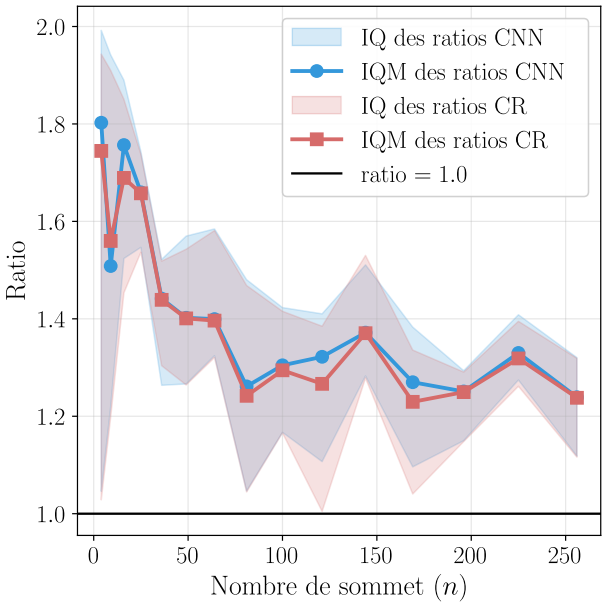

Fig. 15. – Évolution du rapport d'approximation des algorithmes CR et CNN sur des graphes fortement clusterisés en fonction du nombre de sommet n. La courbe bleue représente la moyenne interquartile (IQM) des ratios CNN tandis que la courbe rouge représente celle des ratios CR. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe d'arêtes bloquées $k = n - 2$ pour chaque valeur de n, sur 15 instances aléatoires par configuration.

La Fig. 15 présente le profil d'évolution suivant : pour les petites valeurs de n, les rapport d'approximation sont élevés (jusqu'à 1.8 pour Cnn et 1.7 pour Cr), puis diminuent rapidement avant de se stabiliser autour de 1.3, soit une valeur plus élevée que pour d'autres classes de graphes. Cette stabilisation à un niveau supérieur s'explique par la structure clusterisée : lorsqu'une arête inter-clusters est bloquée, les détours nécessaires sont substantiellement plus coûteux.

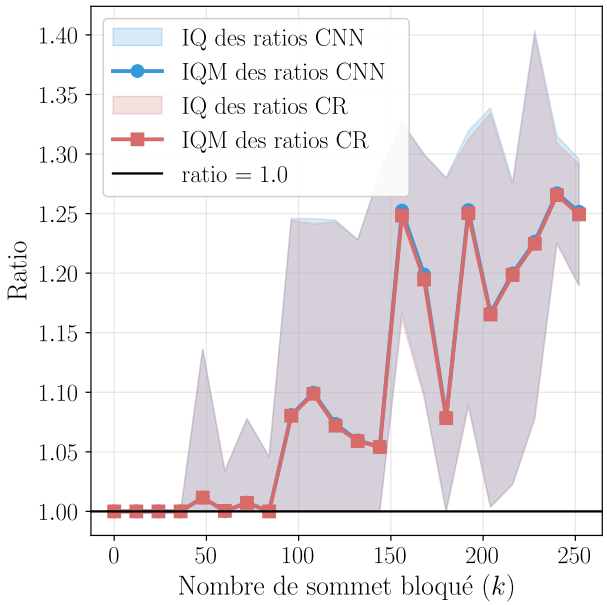

Fig. 16. – Évolution du rapport d'approximation des algorithmes Cr et Cnn sur des graphes fortement clusterisés en fonction du nombre d'arêtes bloquées k. La courbe bleue représente la moyenne interquartile (IQM) des ratios Cnn, tandis que la courbe rouge représente celle des ratios Cr. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe de sommets $n = 256$, sur 15 instances aléatoires par configuration.

La Fig. 16 révèle un comportement encore plus étonnant, avec un seuil critique autour de $k = 100$. En dessous ce ce seuil, les deux algorithmes maitiennent des performances proches de l'optimal. Au-delà, les perforamnces se dégradent rapidement, avec une grande variabilité. Ce phénomène de seuil peut s'expliquer par la transition entre un régime où les blocages affectent principalement des aretes intra-cluster (facilement contournables) et un régime où les blocages commencent à impacter les arêtes inter-clusters critiques, nécessitant des détours considérables.

Graphes basés sur des lois de puissance. Les graphes basés sur des lois de puissance modélisent des réseaux où la distribution des poids suit une loi de puissance, caractéristique de nombreux phénomènes naturels et sociaux. Dans ces graphes, quelques sommets jouent le rôle de *hubs* centraux, tandis que la majorité des sommets sont périphériques.

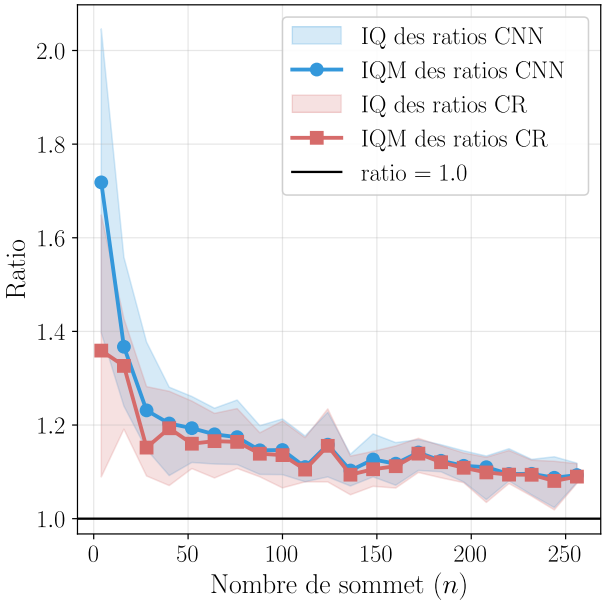

Fig. 17. – Évolution du rapport d'approximation des algorithmes Cr et Cnn sur des graphes basés sur des lois de puissance en fonction du nombre de sommet n. La courbe bleue représente la moyenne interquartile (IQM) des ratios Cnn tandis que la courbe rouge représente celle des ratios Cr. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe d'arêtes bloquées $k = n - 2$ pour chaque valeur de n, sur 15 instances aléatoires par configuration.

Pour construire ces graphes, nous positionnons les sommets selon un distribution radiale où la distance au centre est proportionnelle à

$\left(\frac{i+1}{n}\right)^{\frac{1}{2.5}} \times 10$, où i est l'indice du sommet et n le nombre total de sommets. L'ange est choisi aléatoirement entre 0 et 2π. Les arêtes reliant tous les sommets, avec des poids égaux aux distances euclidiennes, préservant ainsi l'inégalité triangulaire.

La Fig. 17 montre un comportement similaire à celui des graphes euclidiens : un rapport d'approximation initial élevé (environ 1.7 pour Cnn et 1.35 pur Cr) qui diminue rapidement avec l'augmentation de n pour se stabiliser autour de 1.1. Cette convergence reflète la robustesse des deux algorithmes face à la structure en hub des graphes en loi de puissance.

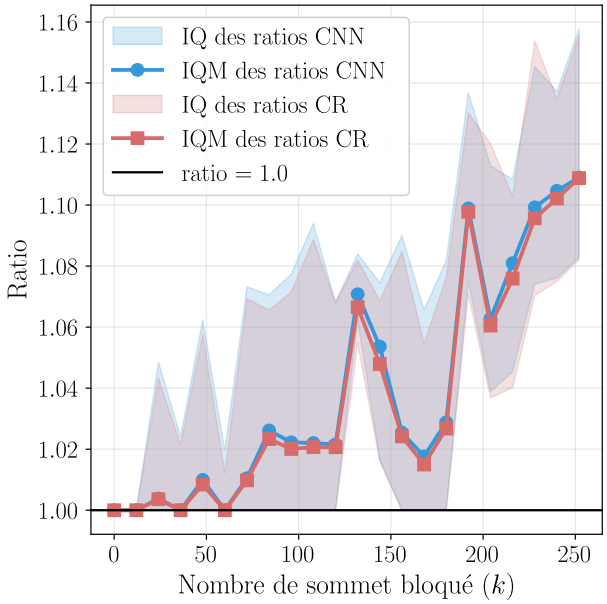

Fig. 18. – Évolution du rapport d'approximation des algorithmes Cr et Cnn sur des graphes basés sur des lois de puissance en fonction du nombre d'arêtes bloquées k. La courbe bleue représente la moyenne interquartile (IQM) des ratios Cnn, tandis que la courbe rouge représente celle des ratios Cr. Les zones colorées correspondent aux intervalles interquartiles (IQ). Les mesures ont été effectuées avec un nombre fixe de sommets $n = 256$, sur 15 instances aléatoires par configuration.

Le Fig. 18 révèle une dégradation progressive mais limitée des performances avec l'augmentation de k. Pour $k < 100$, les deux algorithmes maitiennent des perforamnces proches de l'optimal. Au-delà, les performances se dégradent progressivement, mais de manière plus contenue que pour d'autres classes de graphes, les ratios restant inférieurs à 1.12. Cette robustesse relative s'explique par la présence de hubs qui offrent de multiples chemins alternatifs, limitant l'impact des blocages individuels.

Analyse comparative. Notre analyse empirique sur ces cinq classes de graphes permet de dégager plusieurs remarques sur le choix d'un algorithme en fonction du contexte de l'application.

L'algorithme Cr présente généralement des performances légèrement meilleures que Cnn pour les faibles valeurs de k, particulièrement sur les graphes à poids constnat et les graphes euclidiens. Sa simplicité conceptuelle et sa robustesse en font un excellent choix pour les applications où le nombre de d'obstacle est prévisible et limité.

L'algorithme Cnn, bien qu'ayant généralement une performance légèrement inférieures est plus stable face à l'augmentation du nombre d'arêtes bloquées, grâce a sa garantie en $O(\log k)$.

On note cependant que nous n'avons pas pu observer sur les classes de graphes et les tailles d'instances un cas ou Cnn est meilleur que Cr de façon significatif. Ce résultat peut s'expliquer de différentes façons. D'une part, le type de graphe que nous étudions ne permet pas d'exhiber une telle différence. D'autre part, les tailles d'instances que nous avons fait ne permettent pas de voir une grande différence. Finalement, on note que la performance de Cr et Cnn est très proche, ce qui suggère que les deux algorithmes trouvent en pratique des solutions très proche de l'optimal à atteindre.

Compte tenu de ces observations, il paraît adapté d'utiliser Cr dans les contextes où le nombre d'obstacle est faible par rapport à la taille du réseau et d'opter pour Cnn dans les environnements où les blocages peuvent être nombreux ou imprévisibles. On note en particulier que ces deux algorithmes tournent en temps polynomial,

il est donc tout à fait possible de calculer le tour proposé par les deux algorithmes et de prendre le meilleur rapport d'approximation, dans la plupart des cas, CR sera sans doute utilisé, mais CNN garantit une bien meilleure complexité théorique lorsque k est grand.

En conclusion, bien que la théorie suggère que CNN devrait systématiquement surpasser CR avec l'augmentation de k, notre analyse empirique montre que cette supériorité n'est pas toujours manifeste sur les instances pratiques, et que le choix entre ces deux algorithmes doit tenir compte des caractéristiques spécifiques du problème à résoudre.

Bibliographie

[1] Chung-Shou Liao et Yamming Huang. 2014. The Covering Canadian Traveller Problem. *530*.

[2] Nicos Christofides. 1976. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. *3*.

[3] Niklas Hahn et Michalis Xefteris. 2023. The Covering Canadian Traveller Problem Revisited. Consulté à l'adresse https://arxiv.org/abs/2304.14319