Titre: Title:	DistrictNet : Optimisation des districts avec l'apprentissage automatique pour une planification efficace à long terme
Auteur: Author:	Cheikh Abdallahi Ahmed
Date:	2024
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Ahmed, C. A. (2024). DistrictNet : Optimisation des districts avec l'apprentissage automatique pour une planification efficace à long terme [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/59886/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/59886/
Directeurs de recherche: Advisors:	Thibaut Vidal
Programme: Program:	Maîtrise recherche en mathématiques appliquées

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

DistrictNet : optimisation des districts avec l'apprentissage automatique pour une planification efficace à long terme

CHEIKH ABDALLAHI AHMED
Département de mathématiques et de génie industriel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Mathématiques appliquées

Novembre 2024

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

DistrictNet : optimisation des districts avec l'apprentissage automatique pour une planification efficace à long terme

présenté par **Cheikh Abdallahi AHMED**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Antoine LEGRAIN, président
Thibaut VIDAL, membre et directeur de recherche
Utsav SADANA, membre

DÉDICACE

À ma famille,
pour votre amour, votre soutien inébranlable,
et vos encouragements constants.

REMERCIEMENTS

Je souhaite exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réalisation de cette thèse.

Tout d'abord, je tiens à remercier sincèrement mon directeur de recherche, Thibaut Vidal, pour son encadrement exceptionnel, ses conseils précieux et son soutien inébranlable tout au long de ce projet. Sa disponibilité, sa patience et son expertise ont été des éléments essentiels à la réussite de cette thèse, et je lui suis infiniment reconnaissant pour tout ce qu'il a apporté.

Je suis également très reconnaissant envers Alexandre Forel pour son travail acharné à mes côtés, ainsi que pour son aide précieuse tant dans le développement du code que dans la rédaction d'articles de recherche, notamment pour NeurIPS et pour ce mémoire.

Mes remerciements vont aussi à Axel Parmentier pour son soutien constant et son aide, et à Youssef Emin, pour ses nombreuses idées qui ont été très utiles pour faire avancer le projet

Je tiens également à exprimer ma gratitude à Scale AI pour le financement généreux qui a permis la réalisation de cette recherche. De plus, je remercie Compute Canada pour les ressources informatiques qui ont été essentielles à l'exécution des expériences nécessaires à ce travail.

Enfin, je voudrais exprimer ma reconnaissance à ma famille, dont le soutien moral et l'encouragement m'ont été d'une grande aide tout au long de ce parcours.

RÉSUMÉ

Le partitionnement des districts, ou *districting*, est un problème combinatoire complexe qui consiste à partitionner une zone géographique composée de petites unités en unités plus grandes appelées districts. Dans le domaine de la logistique, il représente une décision stratégique majeure déterminant les coûts d'exploitation pour plusieurs années. Le résoudre avec des méthodes traditionnelles est extrêmement difficile, même pour de petites zones géographiques, et les heuristiques existantes, bien que rapides, offrent souvent des résultats sous-optimaux. Nous présentons une approche d'apprentissage axée sur la décision pour trouver des solutions de haute qualité aux problèmes de *districting* dans le monde réel en quelques minutes.

Cette approche repose sur l'intégration d'une couche d'optimisation combinatoire le problème de l'arbre de recouvrement minimum avec contraintes de capacité dans une architecture de réseaux de neurones en graphes. Pour entraîner notre modèle de manière orientée vers la décision, nous développons un algorithme qui reconstruit les solutions cibles dans un espace de représentation adapté, utilisé pour l'apprentissage par imitation.

Nous utilisons des villes synthétiques pour l'entraînement et des villes réelles pour les tests. Les expériences montrent que notre approche surpasse les méthodes existantes et peut réduire les coûts jusqu'à 10% dans les villes réelles. De plus, nous démontrons que notre approche généralise efficacement aux instances inédites et reste robuste face à des paramètres d'instance variables.

ABSTRACT

Districting is a complex combinatorial problem that consists in partitioning a geographical area of small units into larger units, called districts. In logistics, it is a major strategic decision impacting operating costs for several years. Solving them using traditional methods is intractable even for small geographical areas and existing heuristics, while quick, often provide sub-optimal results. We present a Decision-Focused Learning approach to find high-quality solutions to real-world *districting* problems in a few minutes. It is based on integrating a combinatorial optimization layer, the capacitated minimum spanning tree problem, into a graph neural network architecture. Our method is trained in a decision-aware fashion, by constructing target solutions embedded in a suitable space and learning from them. Experiments show that our approach outperforms existing methods and can reduce costs by on average 10% compared to benchmarks. Additionally, we show that our approach generalizes well to unseen instances and remains robust under varying instance parameters.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

BU	Basic Unit
ML	Machine Learning
DFL	Decison Focused Learning
PFL	Prediction Focused Learning
NN	Neural Network
GNN	Graph Neural Network
GCNN	Graph Convolutional Neural Network
GAT	Graph Attention Network
MPGNN	Message Passing Graph Neural Network
CNN	Convolutional Neural Network
MLP	Multi-Layer Perceptron
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
MSE	Mean Squared Error
ReLU	Rectified Linear Unit
CO	Combinatorial Optimization
LP	Linear Programming
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
QP	Quadratic Programming
CMST	Capacitated Minimum Spanning Tree
VRP	Vehicle Routing Problem
TSP	Traveling Salesman Problem
BHH	Beardwood-Halton-Hammersley

CHAPITRE 1 INTRODUCTION

Le partitionnement des districts (*districting*) est un problème combinatoire complexe qui consiste à diviser une zone géographique en plusieurs unités basiques (*Basic Unit (BU)*) pour former des zones plus grandes, équilibrées et connectées, appelées districts. Ce processus est crucial dans divers domaines, tels que la politique électorale, où il détermine la répartition des circonscriptions, et la logistique, où il influence directement les coûts de distribution à long terme.

Dans ce mémoire, nous nous concentrons sur une forme plus complexe de *districting* : le *districting* avec optimisation des tournées de véhicules. Dans ce cas, l'objectif est de minimiser les coûts moyens des tournées des véhicules pour tous les districts, chaque district étant desservi par un véhicule partant d'un dépôt commun. Comme le *districting* est une décision stratégique à long terme, les demandes de livraison dans chaque district sont inconnues et modélisées comme un processus de Poisson. En tant que problème combinatoire et stochastique en deux étapes, le *districting* avec optimisation des tournées de véhicules ne peut être résolu de manière optimale que pour de très petites instances.

Les méthodes traditionnelles pour résoudre le *districting*, telles que la programmation linéaire et les algorithmes de recherche exhaustive, se basent généralement sur des modèles pour estimer les coûts moyens des tournées des véhicules pour chaque district, puis optimisent la configuration des districts en fonction de ces estimations. Ce processus est souvent inefficace et ne tient pas compte des interactions complexes entre la prédiction des coûts et les décisions de *districting*. Les heuristiques spécifiques pour l'estimation des coûts, bien que rapides, offrent souvent des résultats sous-optimaux et ne sont pas conçues pour s'adapter à des configurations complexes ou à des variations dans les paramètres des instances.

Pour surmonter ces défis, nous proposons une approche basée sur l'apprentissage automatique. Cette méthode intègre une couche d'optimisation combinatoire dans une architecture de réseaux de neurones en graphes (en anglais *Graph Neural Network (GNN)*). En utilisant le problème de l'arbre de recouvrement minimum avec contraintes de capacité (aussi appelé CMST pour *Capacitated Minimum Spanning Tree*) comme modèle d'approximation, notre approche permet de trouver des solutions de haute qualité en un temps réduit. L'objectif principal de cette recherche est de démontrer que notre approche peut non seulement surmonter les limitations des méthodes traditionnelles et heuristiques, mais aussi offrir des solutions quasi-optimales pour des problèmes de *districting* à grande échelle.

Les expériences montrent que notre méthode peut réduire significativement les coûts de

distribution dans des contextes réels, prouvant ainsi son efficacité et sa pertinence pour des applications pratiques. Ce mémoire présente le cadre de cette recherche, ses objectifs et l'approche envisagée pour résoudre les problèmes de *districting* de manière efficace.

1.1 Définitions et concepts de base

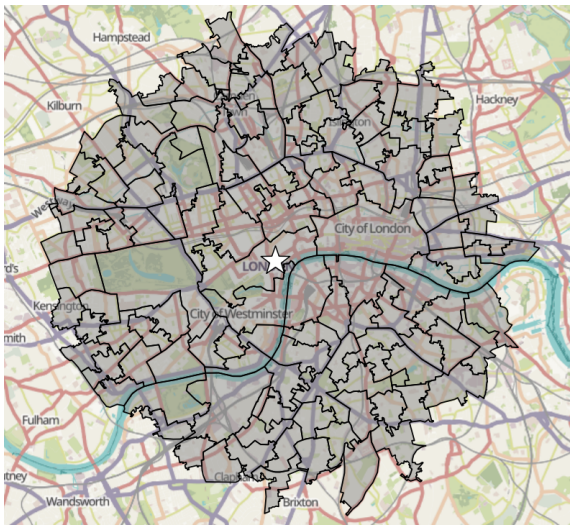
1.1.1 Définition du districting

Le *districting* est un problème combinatoire complexe qui consiste à regrouper de petites zones géographiques, appelées BU, en clusters géographiques plus grands, appelés districts. Ce regroupement doit respecter des critères de planification pertinents pour être considéré comme acceptable. Les exemples typiques de BU incluent les clients, les quartiers et les points de vente. Les exemples typiques de critères de planification incluent l'équilibre, la contiguïté et la compacité [1, 2].

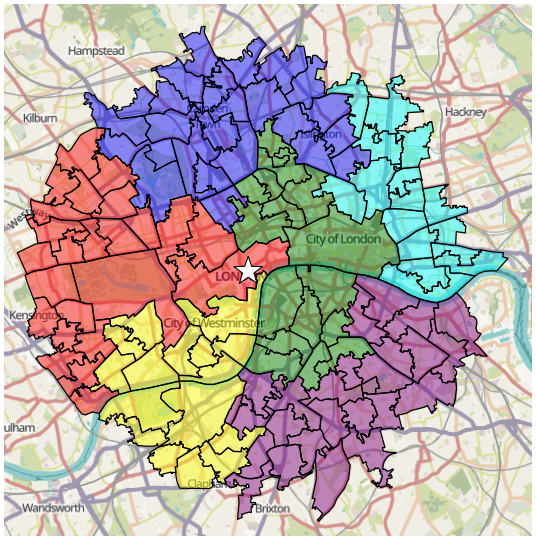
L'équilibre vise à créer des districts de taille équitable selon une mesure de performance. Selon le contexte, cette mesure peut être motivée par des considérations économiques (par exemple, des potentiels de vente égaux, des charges de travail égales ou un nombre équivalent de clients) ou démographiques (par exemple, le même nombre d'habitants ou d'électeurs éligibles). La contiguïté est une autre propriété essentielle, désignant un district comme contigu lorsqu'il forme une seule zone géographique connectée, sans discontinuités. Cette propriété est cruciale pour garantir que les districts soient fonctionnellement efficaces et faciles à gérer. Enfin, la compacité se réfère à la forme géographique des districts, dans le but de minimiser les distances internes et d'éviter des formes géographiques étendues et irrégulières [1].

Comme illustré dans la Figure 1.1, ces critères influencent fortement la configuration des districts. Cette figure compare les limites des districts de Londres avant 1.1(a) et après 1.1(b) la résolution du problème de *districting*. On observe qu'après optimisation, les nouveaux districts 1.1(b) respectent mieux les critères d'équilibre, de contiguïté et de compacité.

Les applications du *districting* sont nombreuses et variées. En politique, le *districting* détermine la répartition des circonscriptions électorales, influençant directement les résultats des élections et la représentation politique des groupes démographiques [3–5]. Dans le domaine de la logistique, il aide à optimiser les routes de distribution pour minimiser les coûts et améliorer l'efficacité [6, 7]. Pour la conception des territoires de vente, il permet de répartir équitablement les clients et les potentiels de vente entre les vendeurs, optimisant ainsi les efforts de vente [8, 9]. En gestion des déchets, le *districting* permet de planifier les itinéraires de collecte de manière efficace, réduisant les coûts opérationnels et les temps de parcours [10]. Dans le domaine des services hivernaux, il aide à planifier les itinéraires de déneigement et



(a). Londres avant la résolution du problème de districting.



(b). Londres après la résolution du problème de districting.

FIGURE 1.1 Comparaison des limites des districts de Londres avant et après la résolution du problème de *districting*.

de salage des routes, assurant une couverture complète et efficace des zones urbaines [11]. Enfin, en répartition scolaire, le *districting* est utilisé pour définir les limites des districts scolaires, équilibrant les populations étudiantes et optimisant l'utilisation des ressources éducatives [12].

1.1.2 Tournées de véhicules dans le contexte du districting

Le problème de tournées de véhicules (en anglais, *Vehicle Routing Problem* ou VRP) est un problème classique en logistique qui consiste à déterminer les itinéraires optimaux pour les véhicules de livraison, afin de minimiser les coûts opérationnels [13]. Dans le contexte du *districting*, on s'intéresse généralement au cas où chaque district est desservi par un véhicule partant d'un dépôt commun. Cela permet de réduire le problème de tournées de véhicules à un ensemble de problèmes de voyageur de commerce (*Traveling Salesman Problem* ou TSP) à résoudre indépendamment pour chaque district.

Le TSP devient particulièrement complexe en raison des caractéristiques géographiques des districts et des demandes de livraison stochastiques. Dans ce cadre, nous nous focalisons sur les coûts opérationnels à long terme, ce qui signifie que nous cherchons à minimiser l'espérance du coût des TSP. Les demandes de livraison, variant de jour en jour, sont modélisées comme un processus de Poisson. L'objectif est de minimiser les coûts des TSP pour l'ensemble des

districts, chaque district étant desservi par un véhicule partant d'un dépôt commun.

1.1.3 Apprentissage profond

L'apprentissage profond, ou *deep learning*, est une sous-discipline de l'apprentissage automatique qui se concentre sur les réseaux de neurones avec plusieurs couches. Contrairement aux méthodes traditionnelles d'apprentissage automatique, les modèles d'apprentissage profond peuvent apprendre des représentations hiérarchiques à partir des données brutes, permettant ainsi de capturer des motifs complexes [14]. Un réseau de neurones profond est composé de couches de neurones interconnectés, chaque neurone appliquant une transformation linéaire suivie d'une fonction d'activation non linéaire. Les paramètres de ces transformations sont appris à partir des données d'entraînement en minimisant une fonction de perte, généralement par une méthode d'optimisation basée sur le gradient, comme la descente de gradient stochastique (SGD).

Dans ce contexte, l'apprentissage supervisé est l'un des cadres les plus couramment utilisés pour entraîner ces modèles d'apprentissage profond. L'apprentissage supervisé est une technique d'apprentissage automatique dans laquelle un modèle est formé pour associer des entrées à des sorties, en utilisant un ensemble de données d'entraînement composé de paires d'entrées x et de sorties y. Les algorithmes d'apprentissage supervisé apprennent à relier certaines entrées à certaines sorties sur la base d'exemples d'entraînement [15]. Ce type d'apprentissage est essentiel dans de nombreux domaines où des prédictions précises sont nécessaires, comme la reconnaissance d'images [16], la détection de fraudes [17] et la prévision des ventes [18]. L'apprentissage supervisé repose sur l'idée que, grâce à un grand nombre d'exemples étiquetés, un modèle peut apprendre des schémas et des relations complexes entre les données d'entrée et de sortie, permettant ainsi des prédictions fiables sur de nouvelles données.

Il existe plusieurs types de réseaux de neurones profonds, chacun adapté à des tâches spécifiques. Par exemple, les réseaux de neurones convolutifs (CNN) sont utilisés pour la vision par ordinateur, les réseaux de neurones récurrents (RNN) pour le traitement du langage naturel, et les GNN pour les données structurées en graphes. Une forme simple de réseau de neurones profond est le perceptron multicouche (MLP), composé de plusieurs couches denses entièrement connectées.

Le perceptron multicouches

Le MLP est une architecture de réseau de neurones profonds composée de plusieurs couches entièrement connectées, également appelées couches denses. Ces réseaux sont les modèles d'apprentissage profond les plus basiques et servent de fondement à de nombreuses applications importantes. L'objectif principal d'un MLP est d'approximer une fonction cible f^*. Par exemple, pour un classificateur, $y = f^*(x)$ associe une entrée x à une catégorie y. Un réseau de neurones MLP définit une fonction de mappage $y = f(x; W)$ et apprend les valeurs des paramètres W qui fournissent la meilleure approximation de cette fonction [15]. Le théorème d'approximation universelle garantit que les réseaux de neurones peuvent approximer n'importe quelle fonction continue sur un ensemble compact à une précision arbitraire [19].

Chaque neurone dans une couche dense applique une transformation linéaire suivie d'une fonction d'activation non linéaire. Pour un vecteur d'entrée $\mathbf{x}$ et une sortie $\mathbf{y}$, la transformation est donnée par :

$$\mathbf{y} = \sigma(W \cdot \mathbf{x} + \mathbf{b})$$

où W est une matrice de poids, $\mathbf{b}$ est un vecteur de biais, et σ est une fonction d'activation telle que ReLU (Rectified Linear Unit). Les paramètres W et $\mathbf{b}$ sont ajustés pendant l'entraînement du réseau pour minimiser une fonction de perte spécifique à la tâche, comme la classification ou la régression.

Réseaux de neurones en graphes

Les GNN représentent une des architectures les plus générales de l'apprentissage profond, permettant de traiter des données structurées en graphes en utilisant les propriétés du groupe de permutations. Pour un graphe spécifié par une matrice d'adjacence A et des caractéristiques de nœuds X, les GNN sont des fonctions $F(X, A)$ équivariantes par permutation, appliquant des fonctions invariantes par permutation sur des voisinages locaux, permettant ainsi de modéliser les interactions complexes au sein des graphes [20]. Ces modèles utilisent des transformations apprenables, notées ψ et ϕ, pour encoder et mettre à jour les caractéristiques des nœuds. Plus précisément, ψ est utilisée pour transformer les caractéristiques des nœuds voisins, tandis que ϕ combine ces caractéristiques transformées avec celles du nœud central. L'agrégation des caractéristiques est effectuée à l'aide de l'opération $\bigoplus$, qui représente une fonction d'agrégation permutation-invariante, comme la somme, la moyenne ou le maximum, pour intégrer les informations des nœuds voisins.

Il existe trois principales couches GNN : réseaux de neurones en graphes convolutionnelle (GCNN) [21–23], réseaux de neurones en graphes attentionnelle (GAT) [24–26] et réseaux de neurones en graphes de passage de messages (MPGNN) [27–29]. Ces couches GNN diffèrent par la manière dont elles agrègent les caractéristiques des nœuds voisins pour mettre à jour les caractéristiques des nœuds actuels.

— GCNN : Les caractéristiques des nœuds du voisinage sont agrégées avec des poids fixes.

$$h_u = \phi\left(x_u, \bigoplus_{v \in \mathcal{N}(u)} c_{uv}\psi(x_v)\right) \tag{1.1}$$

où c_{uv} spécifie l'importance du nœud v pour u.

— GAT : Les interactions sont pondérées par un mécanisme d'attention. Par exemple :

$$h_u = \phi\left(x_u, \bigoplus_{v \in \mathcal{N}(u)} a(x_u, x_v)\psi(x_v)\right) \tag{1.2}$$

où $a(x_u, x_v)$ est un coefficient d'attention.

— MPGNN : Des vecteurs de messages sont calculés à travers les arêtes. Par exemple :

$$h_u = \phi\left(x_u, \bigoplus_{v \in \mathcal{N}(u)} \psi(x_u, x_v)\right) \tag{1.3}$$

où ψ est une fonction de message apprenable.

Les GNN sont utilisés dans diverses applications comme la prédiction de propriétés moléculaires [27], les systèmes de recommandation [30], la détection de fraudes [31] et l'analyse de réseaux sociaux [32], grâce à leur capacité à modéliser des relations complexes dans des données structurées en graphes.

1.1.4 Apprentissage axé sur la décision

L'apprentissage axé sur la décision (en anglais, *Decision Focused Learning (DFL)*) est une approche qui combine l'apprentissage automatique (ML) et l'optimisation combinatoire (CO), pour améliorer la qualité des décisions. Cette approche entraîne des modèles ML dans un système complet (*End-to-End*). Elle peut améliorer la prise de décision combinatoire dans des situations réelles avec de l'incertitude, où estimer les paramètres inconnus dans les modèles décisionnels est un défi important [33].

Contrairement à l'apprentissage axé sur la prédiction (*Prediction Focused Learning (PFL)*),

où le modèle est entraîné pour générer des prédictions précises de paramètres sans se soucier de la qualité des décisions finales, le DFL entraîne le modèle ML à optimiser les critères d'évaluation qui mesurent la qualité des décisions résultantes. Cela nécessite l'intégration des composants de prédiction et d'optimisation dans une architecture unique, permettant ainsi au modèle d'apprendre à prendre des décisions optimales en fonction des prédictions des paramètres, et de quantifier l'erreur encourue après l'optimisation, plutôt que de se concentrer uniquement sur l'exactitude des prédictions des paramètres [33, 34].

Un des principaux défis dans l'apprentissage basé sur les décisions est de permettre la différenciation à travers la couche d'optimisation. Les modèles de deep learning reposent souvent sur l'optimisation par gradients, nécessitant la rétropropagation des gradients à travers toutes les opérations du modèle. Cependant, dans le cadre du DFL, où la fonction de perte dépend directement de la solution d'un problème d'optimisation combinatoire, la présence de variables discrètes dans ces modèles rend la tâche particulièrement complexe, en raison des mappages non continus qui perturbent l'apprentissage par gradients. DFL peut être appliqué à plusieurs applications réelles dans divers domaines, y compris les systèmes énergétiques [35, 36], les réseaux de communication [37], la logistique [38, 39], et la finance [40].

1.2 Éléments de la problématique

1.2.1 Complexité du problème de *districting* et tournées de véhicules

Le *districting* avec optimisation des tournées de véhicules est un problème combinatoire complexe qui devient rapidement très difficile à résoudre pour de grandes instances. Le nombre de districts faisables pour un problème de *districting* croît de manière exponentielle avec le nombre cible des BU dans un district. Cette croissance exponentielle rend la recherche exhaustive de la solution optimale impraticable pour des zones géographiques de taille significative, en grande partie à cause de la nécessité d'évaluer les coûts moyens des TSP pour chaque district.

Lors de la résolution du problème, que ce soit par des heuristiques ou des solveurs exacts, il est nécessaire de résoudre plusieurs TSP afin d'estimer leurs coûts moyens. Or, le TSP est un problème NP-difficile, ce qui complexifie considérablement le processus de *districting* avec optimisation des tournées de véhicules. Même avec des techniques heuristiques avancées telles que l'algorithme LKH [41], la résolution de ces TSP multiples rend le problème global extrêmement complexe et le temps de calcul impraticable pour des instances de taille réelle.

De plus, le TSP, qui cherche à déterminer le tour de coût minimal pour visiter un ensemble de points dans le plan, est un cas particulier du problème de *districting* lorsque la distribution

des demandes de chaque BU est concentrée en un seul point. Ainsi, même l'évaluation du coût d'une solution triviale, où toutes les BU sont contenues dans un seul district, est NP-difficile.

Un autre résultat significatif est que le problème de partitionnement équilibré d'un graphe planaire en sous-graphes connexes est NP-difficile [42]. Cela signifie que même déterminer une partition faisable du graphe sans se préoccuper de l'objectif d'optimisation avec un certain nombre de BU par district (un cas particulier des contraintes de taille minimale et maximale dans notre problème) est NP-difficile.

En conséquence, la résolution optimale du problème de *districting* avec optimisation des tournées de véhicules est extrêmement complexe et nécessite des méthodes efficaces pour estimer rapidement et précisément les coûts moyens des tournées des véhicules.

1.2.2 Limites des méthodes existantes

Les méthodes traditionnelles pour résoudre le *districting* avec optimisation des tournées de véhicules se basent généralement sur des modèles qui estiment les coûts moyens des tournées de véhicules pour chaque district, puis optimisent la configuration des districts en fonction de ces estimations.

Les méthodes couramment utilisées pour estimer les coûts moyens des tournées de véhicules à l'intérieur des districts présentent plusieurs limitations, même lorsqu'elles sont intégrées dans des approches de programmation linéaire ou des heuristiques. Ces méthodes peuvent être regroupées en deux grandes catégories : des formules d'approximation continues pour l'estimation des coûts du TSP et des modèles d'apprentissage avancés pour prédire le coût du TSP à partir des données d'entraînement.

Les heuristiques spécifiques pour l'estimation des coûts, telles que les modèles basés sur des formules empiriques ou des règles simples [43, 44], offrent souvent une estimation rapide des coûts moyens le VRP ou le TSP. Cependant, elles sont généralement moins précises et peuvent produire des résultats sous-optimaux. Ces heuristiques simplifient souvent les caractéristiques géographiques et les demandes de livraison stochastiques, ce qui peut entraîner des surcoûts et des inefficacités opérationnelles [45].

Les modèles de prédiction consistent à utiliser des techniques d'apprentissage supervisé pour prédire les coûts moyens du TSP pour chaque district, puis à optimiser la configuration des districts en fonction de ces prédictions [45, 46]. Cependant, ces méthodes ne prennent pas en compte les interactions complexes entre la prédiction et l'optimisation. Les erreurs de prédiction peuvent entraîner des décisions sous-optimales, et ces approches ne parviennent généralement pas à généraliser efficacement à des instances hors distribution ou à des varia-

tions dans les paramètres des instances. En conséquence, elles manquent de robustesse et de flexibilité dans des scénarios réels variés.

La Figure 1.2 montre la distribution des coûts moyens du TSP pour différentes tailles de districts à Londres, Leeds et Bristol. On peut observer que les coûts moyens du TSP varient considérablement en fonction de la zone géographique et de la taille du district, ce qui rend difficile l'estimation précise des coûts moyens pour des instances de grande taille ou des configurations de districts variées, surtout pour des modèles de prédiction entraînés sur des instances de petite taille.

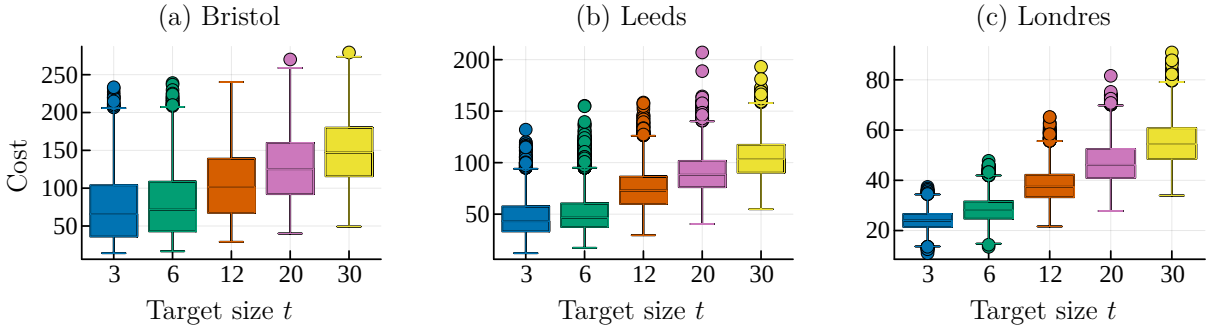


FIGURE 1.2 Distribution du coût du district pour différentes villes et tailles de districts.

Les approches existantes manquent également d'intégration directe du processus décisionnel dans le modèle d'apprentissage. Par conséquent, elles ne peuvent pas ajuster les prédictions en fonction des décisions optimales, limitant ainsi leur performance dans des scénarios complexes. Cette absence d'apprentissage axé sur la décision signifie que les modèles ne peuvent pas apprendre efficacement les relations entre les prédictions de coûts et les configurations optimales des districts.

1.2.3 Opportunités de l'apprentissage axé sur la décision

Face aux limitations des méthodes existantes pour l'estimation des coûts moyens des tournées des véhicules dans les problèmes de *districting*, il est impératif de développer une nouvelle approche capable de surmonter ces défis et de fournir des meilleures solutions. La complexité combinatoire du problème de *districting* et de tournées de véhicules nécessite une méthode innovante qui intègre le DFL pour améliorer la précision et l'efficience des décisions.

Contrairement aux méthodes traditionnelles de prédiction suivie d'optimisation, une nouvelle approche doit intégrer ces deux étapes pour optimiser les configurations des districts de manière plus efficace et précise. En utilisant le DFL, il est possible de former un modèle qui

prend en compte les interactions complexes entre la prédiction des coûts et les décisions de *districting*.

Les méthodes actuelles ont souvent du mal à généraliser à des instances de grande taille ou à des configurations de districts variées. Une nouvelle approche doit être conçue pour s'adapter à une large gamme de tailles d'instances et de paramètres, garantissant ainsi des meilleures performances même dans des scénarios complexes et diversifiés.

1.3 Objectifs de recherche

Les objectifs de cette recherche sont de développer une nouvelle approche basée sur le DFL pour résoudre le problème de *districting* avec optimisation de tournées de véhicules. Le but est de surmonter les limitations des méthodes traditionnelles et d'offrir des meilleures solutions pour des problèmes de *districting* à grande échelle, tout en étant capable de généraliser à des instances hors distribution avec des tailles et configurations variées.

Nous cherchons à développer un modèle capable de résoudre diverses instances de *districting* avec une grande flexibilité. Pour ce faire, notre modèle doit s'adapter à une variété de paramètres et configurations. Les objectifs spécifiques de notre modèle incluent :

— **Adaptabilité aux caractéristiques géographiques et sociales des villes :** Chaque ville possède des particularités géographiques et sociales uniques qui influencent la solution optimale de *districting*. Le modèle doit être capable d'apprendre et de s'adapter à ces variations pour fournir des solutions de haute qualité, quel que soit le contexte urbain.

— **Passage à l'échelle pour des instances de grande taille :** Le modèle doit gérer efficacement des instances de tailles variées, y compris celles avec un nombre de BU supérieur à celui utilisé pour l'entraînement. Cela permet de résoudre des problèmes de plus grande envergure avec un coût d'entraînement réduit, optimisant ainsi les ressources computationnelles.

— **Robustesse aux variations des paramètres du problème :** Le modèle doit être capable de généraliser à différentes configurations du problème, telles que les variations des tailles minimales et maximales des districts. Cette flexibilité garantit que le modèle reste performant même lorsque les contraintes du problème changent.

1.4 Plan du mémoire

Le mémoire est structuré comme suit :

— Chapitre 1 : Introduction : Présentation du sujet, des définitions et des concepts de base, problématique, objectifs de recherche et plan du mémoire.

— Chapitre 2 : Revue de littérature : Analyse des travaux existants sur le *districting* et les approches d'optimisation combinatoire.

— Chapitre 3 : Méthodologie : Description détaillée de l'approche DFL pour résoudre le problème de *districting* avec optimisation de la tournées de véhicules.

— Chapitre 4 : Résultats expérimentaux : Présentation des résultats des expérimentations réalisées, comparaisons avec les méthodes existantes.

— Chapitre 5 : Discussion et conclusion : Discussion des implications des résultats, limitations de l'étude et perspectives pour les recherches futures.

CHAPITRE 2 REVUE DE LITTÉRATURE

Dans ce chapitre, nous introduisons la littérature pertinente pour notre recherche, qui se situe à l'intersection de l'optimisation combinatoire et de l'apprentissage automatique. Nous examinerons d'abord les méthodes traditionnelles utilisées pour estimer les coûts des districts, et leur intégration dans les méthodes heuristiques ou exactes. Ensuite, nous aborderons les approches d'apprentissage pour l'optimisation, suivie de l'apprentissage axé sur la décision. Cette revue mettra en lumière les avancées récentes et les défis actuels dans ces domaines, en soulignant les contributions clés et les lacunes que notre recherche vise à combler.

2.1 Méthodes pour estimer les coûts des districts

Plusieurs algorithmes ont été développés pour résoudre les problèmes de *districting* en logistique. Un élément central de ces approches est le modèle utilisé pour estimer les coûts moyens du TSP dans un district. Ces modèles sont souvent intégrés dans des méthodes exactes ou heuristiques pour résoudre les problèmes de *districting*. Dans cette section, nous examinerons les méthodes traditionnelles utilisées pour estimer les coûts des districts.

Une des meilleures méthodes utilisées pour estimer les coûts au sein d'un district consiste à utiliser une approche de Monte Carlo. Cette méthode génère des scénarios de demande et résout un TSP pour chaque scénario. Cette approche permet d'obtenir une estimation précise des coûts, mais au prix d'un temps de calcul considérable en raison du grand nombre de districts et de scénarios à traiter. Bien qu'efficace, cette méthode peut être coûteuse en temps de calcul, ce qui rend son application impraticable dans les algorithmes de recherche nécessitant l'évaluation de nombreuses solutions de *districting*. Un avantage notable de cette méthode est que l'erreur d'estimation est faible et contrôlée.

La plupart des approches pour le *districting* avec le TSP comme évaluation des coûts s'appuient donc sur des formules d'approximation continue pour évaluer les coûts du TSP. Par exemple, la formule de Beardwood-Halton-Hammersley (BHH) [43], est fréquemment utilisée pour estimer les coûts moyens des TSP à travers n points distribués indépendamment dans une zone compacte de superficie A comme $\alpha\sqrt{nA}$, où α est une constante. Bien que cette méthode soit extrêmement rapide, elle est beaucoup moins précise qu'une approche basée sur des scénarios et peut même orienter la recherche vers des solutions sous-optimales.

La formule BHH a été intégrée dans diverses méthodes de résolution de problèmes de *districting*. Par exemple, des chercheurs ont combiné cette formule avec une méthode du gradient

et un algorithme génétique dans le cadre d'une approche de recherche hybride [47]. D'autres travaux ont utilisé la formule BHH dans une métaheuristique de recherche de voisinage large adaptatif afin d'estimer les coûts du TSP. Cette méthode permet une exploration efficace de vastes portions de l'espace de recherche, en adaptant dynamiquement les mouvements de voisinage selon la qualité des solutions identifiées [48]

Pour améliorer la précision des estimations de coûts, Daganzo [44] a adapté la formule BHH pour les VRP en tenant compte de la variabilité des demandes de livraison. L'approche de Daganzo (BD) estime le coût d'un VRP dans une zone de superficie A comme $\beta\sqrt{AR} + 2\Delta$, où R est le nombre de demandes de livraison dans la zone, Δ est la distance moyenne entre le dépôt et un point de demande, et β est une constante. Chein [49] propose différents modèles et suggère d'estimer les paramètres de la formule en minimisant l'erreur absolue moyenne entre les coûts estimés et les coûts réels. L'un de leurs modèles estime le coût par $\beta_1\Delta + \beta_2\sqrt{A(R-1)}$. Figliozzi [50] a étendu la formule BD pour des demandes non uniformément distribuées. Le coût d'un VRP dans une zone de superficie A est estimé comme $\beta_1\sqrt{AR} + \beta_2\Delta + \beta_3\sqrt{A/R} + \beta_4$, où $\beta = (\beta_1, \beta_2, \beta_3, \beta_4)$ est un vecteur de paramètres

En ce qui concerne les méthodes basées sur l'apprentissage, deux modèles ont été proposés pour estimer les coûts des TSP. Le premier modèle repose sur une approche de régression linéaire, tandis que le second utilise un réseau de neurones simple à trois couches cachées [46]. Des études plus récentes ont proposé des approches d'approximation des coûts des VRP ou TSP à l'aide de modèles de régression, basés sur des caractéristiques liées à la localisation des points de demande [51]. Une méthode similaire a également été développée en utilisant des réseaux de neurones [52]. Plus récemment, un GNN a été entraîné pour prédire les coûts des districts en fonction des caractéristiques des BU du district, telles que les populations, superficies, périmètres, densités de population et distances aux dépôts [45].

2.2 Apprentissage pour l'optimisation

Les dernières années ont vu une augmentation significative de l'utilisation de l'apprentissage automatique pour résoudre des problèmes d'optimisation combinatoire difficiles. Plusieurs algorithmes ont été proposés pour des problèmes combinatoires généraux, comme cela a été décrit dans des travaux récents [53]. L'apprentissage par renforcement profond a été utilisé pour résoudre des problèmes typiques tels que le TSP et les problèmes de sac à dos, démontrant son efficacité [54]. Une autre étude a intégré une étape d'encodage de graphe dans l'apprentissage par renforcement afin de fournir des algorithmes robustes pour divers problèmes d'optimisation combinatoire, y compris le problème de couverture minimale par sommets et le problème de la coupe maximale [55]. Des chercheurs ont proposé une tech-

nique d'amélioration du processus de séparation et évaluation (en anglais *branch and bound*) dans la programmation linéaire en nombres entiers en utilisant des GNN [56]. De plus, des réseaux neuronaux ont été appliqués pour améliorer la sélection des plans sécants dans les programmes quadratiques [57], tandis que des techniques d'apprentissage par renforcement profond ont montré des améliorations dans la sélection des plans sécants dans la programmation en nombres entiers [58]. D'autres chercheurs ont proposé une approche basée sur l'apprentissage supervisé pour décider de l'application de la décomposition de Dantzig-Wolfe dans la programmation en nombres entiers mixtes (MIP) [59]. D'autres ont proposé un modèle pour prédire s'il est préférable de linéariser ou non la partie quadratique des problèmes de programmation quadratique en nombres entiers mixtes (MIQP) [60].

L'avantage principal des méthodes basées sur l'apprentissage est qu'après une procédure d'entraînement potentiellement coûteuse, elles peuvent générer de bonnes solutions à des problèmes combinatoires en un temps court. Cela s'est avéré efficace pour résoudre des problèmes de tournées de véhicules, qui sont des problèmes combinatoires complexes. Par exemple, une approche utilisant des GNN et une recherche en faisceau a montré des améliorations notables en termes de qualité des solutions, de rapidité et d'efficacité pour résoudre le TSP Euclidien [61]. De plus, un modèle d'attention profonde a été proposé pour résoudre divers problèmes complexes de VRP ou TSP, tels que le problème de l'orienteering et le TSP à collecte de prix [62]. Les méthodes d'apprentissage ont également contribué à accélérer la résolution des problèmes stochastiques. Une approche supervisée a été proposée pour prédire rapidement les solutions attendues dans la seconde phase d'un programme stochastique à deux étapes [63]. De même, des réseaux neuronaux ont été utilisés pour approximer les fonctions de valeur attendue dans des programmes stochastiques complexes, entraînant des gains significatifs en termes de vitesse de résolution [64]. Des techniques d'approximation de fonctions de valeur ont également été développées pour des problèmes stochastiques à plusieurs étapes [65]. Pour une revue plus détaillée sur l'apprentissage pour l'optimisation, le lecteur peut se référer à [66, 67].

2.3 Apprentissage axé sur la décision

L'apprentissage axé sur la décision est une approche qui intègre l'apprentissage automatique et l'optimisation combinatoire pour améliorer la qualité des décisions en entraînant des modèles de ML dans un système complet (*End-to-End*). Cette approche montre un potentiel significatif pour révolutionner la prise de décision combinatoire dans des applications réelles opérant en situation d'incertitude, où l'estimation de paramètres inconnus au sein des modèles décisionnels pose un défi majeur. Cette section s'inspire des articles [33] et [34], dont

certaines notations sont par ailleurs utilisées. Elle présente les principes de base du DFL, les défis associés à la différenciation à travers l'optimisation, et les techniques récentes pour surmonter ces défis.

De nombreuses applications réelles, telles que la planification des itinéraires en milieu urbain, le transport maritime ou l'optimisation des horaires de production énergétique, nécessitent des décisions sous incertitude. Dans ces scénarios, les modèles de ML prédisent des quantités incertaines, tandis que les modèles d'optimisation combinatoire cherchent à optimiser des objectifs dans des espaces définis par des contraintes spécifiques.

En général, ces problèmes peuvent être formulés par l'expression suivante :

$$y(\theta) = \arg\min_{y \in \mathcal{Y}} f(y, \theta), \tag{2.1}$$

où $y(\theta)$ représente la solution optimale, $f(y, \theta)$ est la fonction objectif dépendante des paramètres θ, et $\mathcal{Y}$ est la région faisable définie par les contraintes du problème. Comme les paramètres θ sont souvent inconnus, ils doivent être estimés à partir des données observées. Pour mieux comprendre ce processus, la Figure 2.1 illustre le pipeline typique de l'apprentissage et de l'optimisation combinatoire. Ce pipeline montre comment les modèles d'apprentissage automatique et les modèles d'optimisation interagissent pour produire des décisions optimisées dans des contextes incertains.

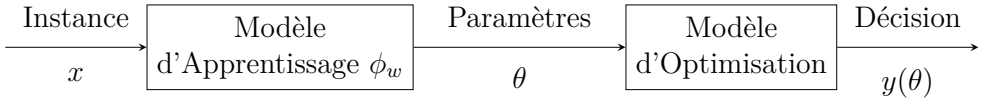


FIGURE 2.1 Pipeline de l'apprentissage et de l'optimisation combinatoire.

Pour aborder ces défis, deux paradigmes principaux peuvent être utilisés : le PFL et le DFL.

En PFL, le modèle de ML est entraîné pour minimiser l'erreur de prédiction des paramètres θ. Cela est souvent réalisé à l'aide d'une fonction de perte telle que l'erreur quadratique moyenne (MSE). Dans ce contexte, les paramètres w du modèle de ML ϕ_w sont ajustés pour minimiser cette fonction de perte. Ce problème est souvent formulé comme suit :

$$\min_w \frac{1}{N} \sum_{i=1}^{N} ||\theta_i - \hat{\theta}_i||^2, \text{ où } \hat{\theta}_i = \phi_w(x_i). \tag{2.2}$$

En revanche, le DFL entraîne le modèle de ML à optimiser directement la qualité des décisions résultantes. La qualité des décisions est mesurée par une fonction de perte de tâche, telle que

le regret :

$$\text{Regret}(y(\hat{\theta}), \theta) = f(y(\hat{\theta}), \theta) - f(y(\theta), \theta). \tag{2.3}$$

Dans ce contexte, les paramètres w du modèle de ML ϕ_w sont ajustés pour minimiser cette fonction de perte. Le DFL peut être formulé comme suit :

$$\min_w \frac{1}{N} \sum_{i=1}^{N} \Big(f(y(\hat{\theta}_i), \theta_i) - f(y(\theta_i), \theta_i) \Big), \text{ où } \hat{\theta}_i = \phi_w(x_i) \text{ et } y(\hat{\theta}_i) = \arg\min_{y \in \mathcal{Y}} f(y, \hat{\theta}_i). \tag{2.4}$$

Ainsi, tandis que le PFL se concentre sur la précision des paramètres estimés, le DFL vise à minimiser l'impact des erreurs de prédiction sur la qualité finale des décisions en intégrant directement la phase de décision dans le processus d'apprentissage, rendant ainsi les décisions finales plus résilientes aux erreurs de prédiction.

Différenciation à travers l'optimisation. L'un des défis principaux du DFL est la différenciation à travers les solutions d'optimisation, ce qui nécessite le calcul des gradients de la fonction de perte $\mathcal{L}$ par rapport aux paramètres du modèle de ML. Cela peut être exprimé par la règle de la chaîne :

$$\frac{d\mathcal{L}(x, y(\hat{\theta}))}{d\omega} = \frac{\partial \mathcal{L}(x, y(\hat{\theta}))}{\partial y(\hat{\theta})} \frac{\partial y(\hat{\theta})}{\partial \hat{\theta}} \frac{\partial \hat{\theta}}{\partial \omega}, \tag{2.5}$$

où ω représente les paramètres du modèle de ML, et $\frac{\partial y(\hat{\theta})}{\partial \hat{\theta}}$ représentant la différenciation à travers l'optimisation.

Un défi majeur réside dans le fait que la plupart des problèmes d'optimisation combinatoire sont non différentiables, ou présentent un gradient nul presque partout. Cela rend difficile le calcul de $\frac{\partial y(\hat{\theta})}{\partial \hat{\theta}}$. Par conséquent, des techniques spécifiques sont nécessaires pour différencier à travers les solutions d'optimisation.

Techniques de différenciation dans le DFL. Pour surmonter ce défi, plusieurs techniques sont proposées pour différencier les solutions d'optimisation. Parmi les approches couramment utilisées figure la différenciation analytique, qui permet de calculer directement les gradients des solutions optimales par rapport aux paramètres. Par exemple, dans le cas d'une fonction convexe$f$, différentiable et sans contrainte, avec des dérivées secondes existantes, certains chercheurs ont appliqué la différenciation implicite des conditions d'optimalité de

premier ordre, aboutissant à l'expression suivante [68] :

$$\frac{dy(\theta)}{d\theta} = -\frac{\partial^2 f(\theta, y(\theta))}{\partial\theta\partial y}\left(\frac{\partial^2 f(\theta, y(\theta))}{\partial y^2}\right)^{-1}. \tag{2.6}$$

Cette méthode peut être étendue aux problèmes d'optimisation sous contraintes en transformant le problème initial en un problème sans contrainte via la fonction de barrière logarithmique. Une autre approche, sans transformation préalable, consiste à différencier directement à travers les conditions doptimalité, comme cela a été fait pour les problèmes d'optimisation quadratique convexe (QP) en différenciant les conditions de Karush-Kuhn-Tucker (KKT) [69].

D'autres techniques incluent l'utilisation de perturbations aléatoires. Par exemple, certains travaux suggèrent de perturber les paramètres θ avec un vecteur d'aléatoire $\epsilon\eta$, où η est échantillonné à partir d'une densité proportionnelle à $\exp(-\nu(\eta))$. Le maximiseur du vecteur perturbé $\theta + \epsilon\eta$ devient alors un échantillon de la distribution conditionnelle [70] :

$$y_\epsilon(\theta) = y(\theta + \epsilon\eta), \tag{2.7}$$

et l'estimation Monte Carlo de $\overline{y_\epsilon}(\theta)$ est donnée par [70] :

$$\overline{y_\epsilon}(\theta) = \frac{1}{M}\sum_{m=1}^{M} y(\theta + \epsilon\eta^{(m)}). \tag{2.8}$$

La dérivée de cette estimation par rapport à θ peut être calculée par une méthode de Monte Carlo, exprimée comme suit [70] :

$$\frac{d\overline{y_\epsilon}(\theta)}{d\theta} = \frac{1}{\epsilon}\frac{1}{M}\sum_{m=1}^{M} y(\theta + \epsilon\eta^{(m)})\nu'(\eta^{(m)})^T. \tag{2.9}$$

Cette approche est applicable à des problèmes d'optimisation combinatoire avec des objectifs linéaires. Une extension de cette technique inclut l'utilisation de perturbations multiplicatives, où la perturbation est définie comme $\theta \odot \exp(\epsilon\eta - 0.5\epsilon^2)$, avec $\odot$ désignant le produit d'Hadamard [71].

Parmi les autres méthodes, la différenciation des fonctions de perte de substitution occupe une place importante. Une approche nommée "Predict, Then Optimize" (SPO) propose une fonction de perte de substitution appelée SPO+, définie comme [72] :

$$L_{SPO+}(y(\hat{\theta})) = 2\hat{\theta}^\top y(\theta) - \theta^\top y(\theta) + \max_{x\in F}\{\theta^\top x - 2\hat{\theta}^\top x\}. \tag{2.10}$$

Étant donné la difficulté de minimiser directement cette fonction pour des réseaux de neurones, un sous-gradient de L_{SPO+} est utilisé pour faciliter l'entraînement, défini par [72] :

$$y(\theta) - y(2\hat{\theta} - \theta). \tag{2.11}$$

Cette méthode s'applique à divers types de problèmes d'optimisation, tels que les LP, QP, ILP et MILP. Pour les ILP, des relaxations LP ont été proposées pour accélérer l'entraînement [73].

Par ailleurs, l'estimation par contraste de bruit (NCE) utilise une fonction de perte de substitution pour éviter le calcul direct du gradient de regret. La perte NCE est définie comme suit [74] :

$$L_{NCE}(\hat{\theta}, \theta) = \sum_{y' \in S} f(y(\theta), \hat{\theta}) - f(y', \hat{\theta}), \tag{2.12}$$

où S représente un ensemble d'exemples négatifs. Cette méthode permet de prédire un $\hat{\theta}$ pour lequel $y(\theta)$ obtient une bonne valeur objective, tandis que les autres solutions y' obtiennent des valeurs objectives moindres. Une approximation par maximum a posteriori (MAP) est également proposée, définie par [74] :

$$L_{MAP}(\hat{\theta}, \theta) = \max_{y' \in S} f(y(\theta), \hat{\theta}) - f(y', \hat{\theta}), \tag{2.13}$$

où y' est l'élément qui minimise la fonction objectif parmi les exemples négatifs.

Des techniques récentes ont été proposées pour combiner apprentissage et optimisation dans des applications réelles sur des graphes. Parmi celles-ci, une approche axée sur la prise de décision, appelée ClusterNet, intègre un proxy différentiable afin de résoudre des problèmes d'optimisation sur des graphes, tels que la détection de communautés ou la localisation de facilités [75]. Cette méthode vise à apprendre des représentations permettant de transformer le problème d'optimisation initial en un problème proxy plus simple et différentiable. Une version modifiée de l'algorithme K-means est utilisée comme couche d'optimisation différentiable. D'autres travaux proposent une approche de clustering différentiable, utilisant une matrice de connectivité partielle des clusters [76]. Bien que similaires à notre méthode, ces recherches se concentrent principalement sur des problèmes de clustering, tandis que notre étude se focalise sur des problématiques de *districting*. La nouveauté de notre approche réside dans la reconstruction d'un moment CMST $\bar{\mu}$ à partir d'une solution de *districting* $\bar{\lambda}$, perçue comme une cible partiellement spécifiée. Les alternatives trouvées dans la littérature visent généralement à compléter la solution partiellement spécifiée en une solution totalement

définie qui minimise la perte de Fenchel-Young [70, 71]. Notre approche présente l'avantage de mener à la perte classique de Fenchel-Young, caractérisée par sa convexité, tandis que d'autres méthodes, telles que celles fondées sur la perte d'infimum, se basent uniquement sur la différence de fonctions convexes [76].

CHAPITRE 3 MÉTHODOLOGIE

Dans ce chapitre, nous détaillons la méthodologie employée pour résoudre le problème de *districting* avec optimisation des tournées de véhicules. Nous décrivons l'architecture du modèle proposé, la formulation mathématique du problème ainsi que les techniques d'optimisation utilisées. Ce chapitre a pour objectif de fournir une compréhension approfondie de notre approche et d'expliquer comment nous avons intégré l'apprentissage automatique pour améliorer les performances du modèle.

3.1 Description du problème

Nous modélisons une zone géographique comme un graphe non orienté $G = (V, E)$, où V est un ensemble de sommets représentant les BU et E est un ensemble d'arêtes représentant les connexions entre elles. Un district $d \subset V$ est un sous-ensemble de BU. Nous disons qu'un district d est connecté si son graphe est connexe. Notons $\mathcal{D}$ l'ensemble des districts connectés et $N = |V|$ le nombre de BU.

3.1.1 Coût d'un district

Pour un district connecté $d \in \mathcal{D}$, le coût d'un district est défini comme l'espérance de la distance minimale parcourue pour satisfaire toutes les demandes de livraison, en tenant compte de la variabilité et de la distribution des demandes à travers le district. Ce coût, noté $C_{TSP}(d)$, correspond au coût attendu d'un TSP appliqué au district d, avec un ensemble de demandes de livraison ξ distribuées aléatoirement dans ce district. Mathématiquement, cela s'exprime par $C_{TSP}(d) = \mathbb{E}_{\xi}[\mathrm{TSP}(d, \xi)]$, où ξ représente l'ensemble des demandes de livraison dans le district d.

Modélisation des demandes de livraison. Les demandes de livraison étant inconnues à l'avance, elles sont modélisées comme des variables aléatoires. Pour chaque BU v, les emplacements des demandes de livraison sont générés aléatoirement selon un processus de Poisson spatial, avec une intensité λ_v proportionnelle à la population de la BU v. Cette modélisation implique que le nombre de demandes de livraison dans chaque BU suit une distribution de Poisson, notée $\xi_i \sim \mathfrak{D}_v$. La valeur de λ_v est définie par $\lambda_v = n_v \times \kappa$, où n_v représente la population de la BU v et κ est une constante de normalisation. Ainsi, les zones avec une population plus élevée génèrent en moyenne un plus grand nombre de demandes par unité

de surface. Enfin, les demandes de livraison sont supposées être réparties uniformément sur la zone géographique couverte par le BU.

Estimation des coûts de tournée de véhicules. Pour calculer le coût d'un district d, nous procédons en plusieurs étapes. Tout d'abord, un sommet supplémentaire v_0, représentant le dépôt, est ajouté et connecté à tous les autres sommets du district d. Nous définissons ensuite $\Pi(\xi)$ comme l'ensemble de tous les itinéraires possibles qui commencent et se terminent au dépôt v_0 et qui couvrent toutes les demandes de livraison représentées par ξ. Chaque itinéraire $\pi \in \Pi(\xi)$ a un coût associé $dist(\pi)$, qui représente la distance totale parcourue. Pour un ensemble donné de demandes de livraison ξ, l'itinéraire optimal est celui qui minimise la distance totale parcourue, tout en répondant à toutes les demandes. Ce problème est formulé comme un TSP, où $\mathrm{TSP}(d, \xi)$ représente la distance minimale parmi tous les itinéraires possibles, c'est-à-dire $\mathrm{TSP}(d, \xi) = \min_{\pi \in \Pi(\xi)} dist(\pi)$.

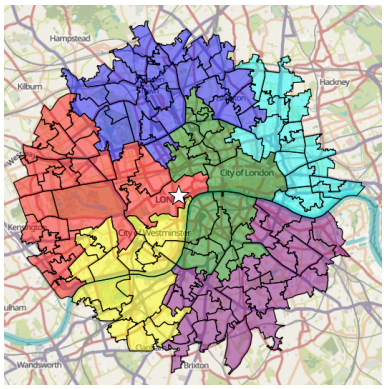
Pour estimer le coût du district d, nous utilisons une méthode d'échantillonnage Monte Carlo. Cela implique la génération d'un ensemble S de $|S|$ scénarios de demandes de livraison ξ aléatoires pour le district d. Chaque scénario représente une configuration possible des demandes de livraison pour toutes les BU du district. Pour chaque scénario généré, nous résolvons un problème de TSP en utilisant un algorithme de type Lin-Kernighan-Helsgaun (LKH) [77]. Cette heuristique est reconnue pour fournir des solutions de haute qualité pour les TSP, proches de l'optimal. Le coût moyen du district est ensuite estimé en prenant la moyenne des coûts calculés pour tous les scénarios, selon la formule suivante :

$$C_{TSP}(d) = \frac{1}{|S|} \sum_{\xi \in S} \min_{\pi \in \Pi(\xi)} dist(\pi), \tag{3.1}$$

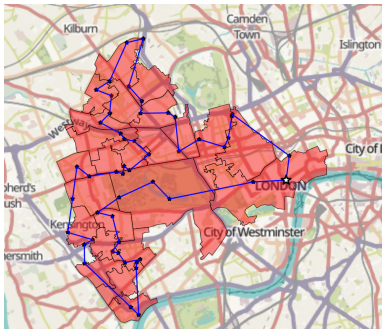
Cette approche permet d'obtenir une estimation précise du coût opérationnel attendu, bien que cela soit computationnellement intensif, en particulier pour les grands districts.

3.1.2 Formulation du problème

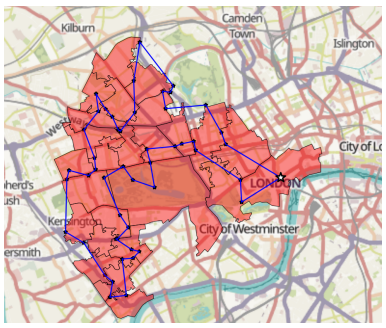
Le problème de *districting* et de tournées de véhicules minimise la somme de tous les coûts des districts. Soit $C_{TSP}(d)$ le coût d'un district d. Un problème de *districting* peut être formulé de manière générale comme suit :



(a). Les nouvelles limites des districts de Londres après la résolution du problème de *districting*.



(b). Scénario de tournées de véhicules pour le district rouge, jour 1. Le chemin montre les étapes de distribution et les zones desservies en premier jour.



(c). Scénario de tournées de véhicules pour le district rouge, jour 2. Une variante des chemins de distribution pour le deuxième jour, couvrant différentes zones.

FIGURE 3.1 Comparaison des nouvelles limites des districts après résolution du problème de *districting* et des scénarios de tournées de véhicules pour le district rouge sur deux jours différents.

$$\min_{\lambda \in \{0,1\}^{|\mathcal{D}|}} \quad \sum_{d \in \mathcal{D}} C_{TSP}(d)\lambda_d, \tag{3.2a}$$

$$\text{s.t.} \quad \sum_{d \in \mathcal{D}} \mathbb{1}(i \in d)\lambda_d = 1, \qquad \forall i \in V, \tag{3.2b}$$

$$\sum_{d \in \mathcal{D}} \lambda_d = k, \tag{3.2c}$$

où λ_d est une variable binaire indiquant si un district est sélectionné. La contrainte (3.2b) stipule que chaque BU est sélectionnée dans exactement un district de la solution. La contrainte (3.2c) spécifie qu'exactement k districts sont sélectionnés.

Des contraintes supplémentaires sur les districts peuvent être facilement ajoutées au problème. Par exemple, le *districting* et la tournées de véhicules visent généralement à obtenir des districts proches d'une taille cible t et incluent des contraintes sur la taille minimale et maximale d'un district. Le problème (3.2) peut être formulé sur un ensemble restreint de districts $\mathcal{D}^r \subseteq \mathcal{D}$ satisfaisant les contraintes de taille minimale et maximale. Formellement, cela assure que $\forall d \in \mathcal{D}^r, \underline{d} \leq |d| \leq \bar{d}$, où $(\underline{d}, \bar{d})$ sont des bornes supérieures et inférieures spécifiées par l'utilisateur sur la taille des districts.

Complexité du problème

Bien que la formulation (3.2) semble simple, elle comporte un nombre exponentiel de variables. Le nombre de districts connectés de taille t possibles dans une zone géographique comprenant N BU est de l'ordre de $\mathcal{O}((e(\delta-1))^{t-1} \cdot \frac{N}{t})$, où δ représente le nombre maximal de voisins pour une BU donnée [78].

Par exemple, pour la ville illustrée à la Figure 3.1, avec $N = 120$ BU, une taille cible de district de $t = 20$, et un maximum de $\delta = 13$ voisins par BU, le nombre de districts connectés possibles est de l'ordre de 10^{29}. Cela signifie que pour résoudre le problème (3.2) de manière optimale, il serait nécessaire de résoudre environ $|S| \times 10^{29}$ Problèmes du Voyageur de Commerce (TSP). Même avec des techniques heuristiques avancées telles que l'algorithme LKH, le temps de calcul requis devient rapidement impraticable pour des instances de taille réelle.

Ces résultats, ainsi que ceux présentés dans la section 1.2.1, montrent clairement que la résolution optimale du problème (3.2) n'est envisageable que pour des problèmes de taille réduite.

3.2 Description du modèle (DistrictNet)

Étant donné la complexité du problème de *districting*, nous proposons d'approcher ce problème d'optimisation combinatoire difficile en le transformant en un problème plus simple et plus traitable. Pour ce faire, nous utilisons une pipeline, comme illustré à la figure 2.1, où un modèle d'apprentissage profond apprend à transformer le problème de *districting* en un problème d'optimisation plus simple. Cette approche nécessite de définir l'architecture du modèle d'apprentissage, de choisir un problème d'approximation adapté, et de concevoir une méthode pour convertir la solution du problème d'approximation en une solution valide pour le problème de *districting* original.

Pour construire une telle pipeline, il est essentiel que le modèle d'apprentissage et le problème de substitution respectent certains critères :

— Architecture du modèle d'apprentissage : L'architecture doit être capable de gérer les variations dans la taille et la structure des données d'entrée tout en restant robuste. Nous utilisons des couches équivariantes qui respectent les symétries du problème, garantissant que les prédictions restent cohérentes malgré les permutations des données d'entrée. Nous avons choisi d'utiliser des GNN, car ils respectent ces propriétés et sont reconnus pour leurs bonnes performances sur les données sous forme de graphes.

— Problème d'approximation : Le problème d'approximation doit être suffisamment simple pour être résolu efficacement, tout en conservant une structure similaire à celle du pro-

blème original. Nous utilisons le problème du CMST comme problème d'approximation, car il partage des caractéristiques structurelles avec le problème de *districting*.

DistrictNet

DistrictNet est composé de trois composants principaux : un GNN, une couche d'optimisation combinatoire basée sur le CMST, et un décodeur. En intégrant ces trois composants, DistrictNet permet de générer des solutions de haute qualité pour des instances complexes et de grande taille.

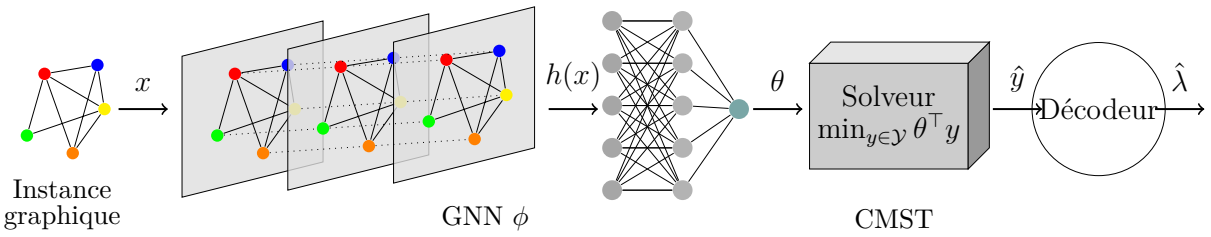


FIGURE 3.2 Architecture de DistrictNet.

Aperçu de l'architecture La figure 3.2 illustre l'architecture de DistrictNet. Le fonctionnement de DistrictNet est divisé en quatre étapes principales :

— Entrée : Le modèle prend en entrée un graphe représentant la zone géographique, où chaque BU est un nœud et les connexions entre les BU sont des arêtes. Chaque nœud est représenté par un vecteur de caractéristiques qui capture les informations géographiques et sociales de la BU, et chaque arête est représentée par un vecteur de caractéristiques qui capture les informations géographiques et sociales de ces deux BU.

— Réseaux de neurones en graphes : Le GNN est utilisé pour apprendre une représentation latente des arêtes, puis de prédire les poids des arêtes.

— Couche d'optimisation combinatoire : La couche d'optimisation combinatoire est utilisée pour résoudre le problème du CMST avec les poids d'arêtes prédits.

— Décodeur : Le décodeur est utilisé pour convertir la solution CMST en une solution de *districting*.

3.2.1 Réseaux de neurones en graphes

Dans DistrictNet, le rôle principal du GNN est de transformer le graphe en un vecteur de poids d'arête. Cette transformation permet de capturer les caractéristiques latentes des BU et de leurs connexions, facilitant ainsi l'optimisation combinatoire ultérieure. Le GNN de

DistrictNet est composé de plusieurs couches de GCNN qui mettent à jour les caractéristiques des arêtes en utilisant les informations de leurs voisins, conformément à l'équation (1.1) présentée précédemment. Dans notre cas, nous avons adopté les fonctions ψ et ϕ, ainsi que l'opération d'agrégation $\bigoplus$, telles que définies dans [79].

- $\psi(x_v) = W_0 x_v$
- $\phi(x_u, X_{\mathcal{N}(u)}) = \sigma\left(W_1 x_u + \frac{1}{|\mathcal{N}(u)|}\sum_{v\in\mathcal{N}(u)} \psi(x_v)\right)$,
- $\bigoplus$: Nous utilisons ici la moyenne des caractéristiques pour agréger les messages des arêtes voisines,

La mise à jour des caractéristiques des arêtes à la l-ième couche est donnée par :

$$h_e^{(l)} = \sigma\left(W_1^{(l)} h_e^{(l-1)} + \frac{1}{|\mathcal{N}(e)|}\sum_{j\in\mathcal{N}(e)} W_0^{(l)} h_j^{(l-1)}\right), \tag{3.3}$$

où :

- $h_e^{(l)}$ est le vecteur de caractéristiques de l'arête e à la l-ième couche,
- σ est une fonction d'activation non linéaire (par exemple, Unité Linéaire Rectifiée ou ReLU),
- $W_0^{(l)}$ et $W_1^{(l)}$ sont des matrices de poids apprenables à la l-ième couche,
- $\mathcal{N}(e)$ est l'ensemble des arêtes voisines de e.
- $X_{\mathcal{N}(e)}$ est l'ensemble des caractéristiques des arêtes voisines de e.
- $\sigma(x) = \max(0, x)$ est la fonction d'activation ReLU.

Les matrices $W_0^{(l)}$ et $W_1^{(l)}$ sont des paramètres appris durant l'entraînement du modèle. Elles permettent de pondérer les caractéristiques des arêtes et de leurs voisins de manière appropriée.

Pour chaque arête e, $\mathcal{N}(e)$ représente l'ensemble des arêtes qui partagent un sommet avec e. Cette définition permet à l'information de se propager à travers le graphe, capturant ainsi la structure locale du graphe.

Transformation finale. Après avoir passé plusieurs couches de convolution de graphe, les représentations latentes des arêtes sont transformées en poids d'arête via trois couches denses entièrement connectées. La transformation est réalisée comme suit :

$$\theta_e = W_3\left(\sigma\left(W_2\left(\sigma\left(W_1 h_e^{(L)}\right)\right)\right)\right), \tag{3.4}$$

où :

— $h_e^{(L)}$ est le vecteur de caractéristiques de l'arête e après la dernière couche de convolution de graphe,
— σ est une fonction d'activation non linéaire (par exemple, ReLU),
— W_1, W_2, et W_3 sont des matrices de poids apprenables des couches denses.

3.2.2 Couche d'optimisation combinatoire

La deuxième composante de DistrictNet est une couche d'optimisation combinatoire, plus précisément le problème du CMST. Cette couche agit comme un modèle d'approximation pour le problème de *districting*. En effet, le problème du CMST a une structure similaire au problème de *districting*, ce qui permet de convertir le problème de *districting* en un problème de CMST. Ce CMST est paramétré par les poids d'arêtes prédits par le GNN et ensuite résolu pour obtenir une solution de *districting*.

Problème de CMST.

Le CMST est un problème d'optimisation combinatoire qui consiste à trouver un arbre couvrant de poids minimum dans un graphe pondéré tout en respectant une contrainte de capacité. La contrainte de capacité garantit que tous les sous-arbres (sous-graphes maximaux connectés à la racine par une seule arête) ont une taille inférieure à une valeur seuil donnée et une taille minimale.

Soit $\mathcal{T}$ l'ensemble des sous-arbres connectés de taille minimale et maximale $(\underline{d}, \bar{d})$. On considère qu'une arête e est dans un sous-arbre s si ses deux extrémités sont à l'intérieur du sous-arbre. Le problème du CMST avec un nombre cible de sous-arbres est alors donné par :

$$\min_{\nu \in \{0,1\}^{|\mathcal{T}|}} \quad \sum_{s \in \mathcal{T}} \nu_s \sum_{e \in s} \theta_e, \tag{3.5a}$$

$$\text{s.t.} \quad \sum_{s \in \mathcal{T}} \mathbb{1}(i \in s) \nu_s = 1, \qquad \forall i \in V, \tag{3.5b}$$

$$\sum_{s \in \mathcal{T}} \nu_s = k. \tag{3.5c}$$

La fonction objectif (3.5a) minimise la somme des poids des arêtes sélectionnées dans les sous-arbres. La contrainte (3.5b) garantit que chaque BU est sélectionnée dans exactement un sous-arbre. La contrainte (3.5c) spécifie qu'exactement k sous-arbres sont sélectionnés. Ce problème est NP-difficile [80], cependant, il peut être résolu efficacement par des méthodes exactes ou des heuristiques. Plus de détails sur la résolution de ce problème seront donnés

dans la section 3.3.

La formulation (3.5) met en évidence le lien entre le CMST et le problème de *districting* dans (3.2). Les deux problèmes partitionnent un graphe en composantes connectées avec des contraintes de cardinalité. Toute solution du CMST peut être convertie en une solution de *districting* en regroupant tous les nœuds d'un sous-arbre dans un district. Comme plusieurs sous-arbres peuvent conduire au même district, il s'agit d'une application surjective de l'espace des sous-arbres $\mathcal{T}$ vers l'espace des districts $\mathcal{D}$. Cela implique également que, pour tout problème de *districting*, il existe toujours un problème de CMST dont les solutions optimales coïncident.

3.2.3 Décodeur

Le décodeur est utilisé pour convertir la solution du CMST en une solution de *districting*. Cette conversion est réalisée en regroupant les nœuds de chaque sous-arbre dans un district.

Conversion de la solution CMST en solution de *districting*. Toute solution du CMST peut être convertie en une solution de *districting* en regroupant tous les nœuds d'un sous-arbre dans un district. Cette conversion est formalisée par une application $\chi : \mathcal{T} \to \mathcal{D}$, où $\chi(s) = \{i \in V | \, \mathbb{1}(i \in s) = 1\}$, associant chaque sous-arbre $s \in \mathcal{T}$ à un district $d \in \mathcal{D}$. Comme plusieurs sous-arbres peuvent conduire au même district, il s'agit d'une application surjective de l'espace des sous-arbres $\mathcal{T}$ vers l'espace des districts $\mathcal{D}$. Cela signifie qu'il existe une relation de correspondance multiple entre les sous-arbres et les districts, ce qui garantit que pour chaque district, il existe au moins un sous-arbre correspondant.

Propriétés

Deux propriétés clés caractérisent la relation entre le problème de *districting* et le problème CMST. La première propriété montre que le CMST préserve l'espace de solution du *districting*, tandis que la seconde démontre qu'il est possible d'obtenir une solution optimale commune pour les deux problèmes sous un vecteur de coûts approprié.

Propriété 1 : Préservation de l'espace de solution. Le problème CMST préserve l'espace de solution du problème de *districting*. Cela signifie que toute solution faisable de *districting* peut être représentée comme une solution faisable du CMST.

Preuve : Soit $\lambda = \{D_1, D_2, \ldots, D_k\}$ une solution faisable de *districting*, où chaque D_i est un sous-ensemble de V tel que $D_i \subseteq V$ pour tout i, $D_i \cap D_j = \emptyset$ pour tout $i \neq j$, et $\bigcup_{i=1}^{k} D_i = V$.

Pour chaque district $D_i \in \lambda$, construisons un arbre couvrant T_i qui connecte tous les nœuds de D_i en utilisant uniquement les arêtes contenues dans D_i. Puisque chaque D_i est un sous-graphe connexe, un tel arbre couvrant T_i existe et relie tous les sommets de D_i. L'ensemble des arbres $\nu = \{T_1, T_2, \ldots, T_k\}$ forme une solution CMST faisable, car chaque T_i est un sous-arbre qui couvre exactement les sommets de D_i tout en respectant les contraintes de capacité. Ainsi, chaque solution de *districting* λ a une solution correspondante ν dans l'espace des solutions CMST. Comme nous avons construit une solution CMST faisable ν pour chaque solution faisable de *districting* λ, nous avons une application surjective de l'espace des solutions de *districting* $\mathcal{D}$ vers l'espace des solutions CMST $\mathcal{T}$. Puisque chaque solution de *districting* faisable peut être représentée comme une solution faisable du CMST, le problème CMST préserve l'espace de solution du problème de *districting*.

Propriété 2 : Optimalité par équivalence de coûts. Pour tout problème de *districting* qui a une solution optimale, il existe au moins un vecteur de coûts pour lequel la solution correspondante du problème CMST est également optimale pour le problème de *districting*. Ainsi, même si plusieurs CMST solutions peuvent correspondre à une solution de *districting*, chacune d'entre elles peut être associée à un vecteur de coûts distinct garantissant que la solution CMST correspondante est optimale.

Preuve :

Soit $\lambda^* = \{D_1^*, D_2^*, \ldots, D_k^*\}$ une solution optimale de *districting*. Pour chaque district $D_i^* \in \lambda^*$, construisons un arbre couvrant minimal T_i^* qui connecte tous les nœuds de D_i^*. L'ensemble $\nu^* = \{T_1^*, T_2^*, \ldots, T_k^*\}$ forme une solution CMST faisable. Définissons un vecteur de coûts $\theta : E \to \mathbb{R}^+$ tel que :

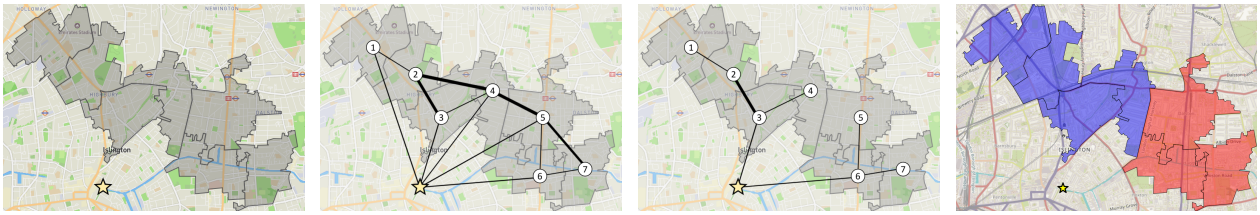
$$\theta(e) = \begin{cases} 0 & \text{si } e \in \nu^* \\ M & \text{si } e \notin \nu^*, \end{cases}$$

où M est un nombre positif. Sous ce vecteur de coûts θ, la solution optimale du problème CMST sera exactement ν^*, car toute solution CMST alternative incluant des arêtes non dans ν^* aurait un coût total supérieur à cause des pénalités M. Ainsi, pour tout λ^* optimal, il existe au moins un vecteur de coûts θ pour lequel la solution optimale du CMST, lorsqu'elle est convertie en solution de *districting*, est également optimale.

3.2.4 Exemple illustratif

Pour mieux comprendre le lien entre les problèmes de *districting* et ceux de CMST, nous présentons un exemple illustratif dans la Figure 3.3.

Tout d'abord, nous considérons une instance simple de *districting* avec $N = 7$ BU, comme illustré dans la sous-figure (a) de la Figure 3.3. Le nœud source est représenté par une étoile jaune. Dans la sous-figure (b), nous montrons l'instance correspondante de CMST obtenue en appliquant le GNN pour prédire les poids des arêtes. La largeur de chaque arête est proportionnelle au poids de l'arête, indiquant les coûts prévus par le modèle. Dans la sous-figure (c), nous montrons la solution optimale de cette instance de CMST pour $t = 3$. La solution comporte deux sous-arbres partant du nœud source, où chaque sous-arbre représente à un district. La sous-figure (d) montre la nouvelle solution de *districting* obtenue en regroupant les nœuds de chaque sous-arbre dans un district.



(a). Instance de districting avec $N = 7$ UBs

(b). Instance de CMST avec la largeur des arêtes proportionnelle à leurs poids

(c). Solution optimale du CMST pour $t = 3$

(d). Solution de districting

FIGURE 3.3 Petit problème de (a) districting converti en (b) un problème de CMST avec des poids d'arête, (c) sa solution correspondante : $d_1 = \{1, 2, 3, 4\}$ et $d_2 = \{5, 6, 7\}$, et (d) nouvelle solution de districting.

3.2.5 Entraînement de DistrictNet

Pour entraîner DistrictNet, nous cherchons à reproduire les solutions d'un ensemble d'entraînement donné, noté $(x_i, \bar{\lambda}_i)_{i=1}^n$. Pour cela, nous utilisons une approche d'apprentissage par imitation. Chaque instance de l'ensemble $x_i = (G_i, f_i)$ correspond à un graphe étiqueté avec $G_i = (V_i, E_i)$ représentant la structure du graphe, et $f_i = \{f_e\}_{e \in E_i}$ l'ensemble des vecteurs caractéristiques des arêtes. À chaque instance x_i est associée une solution de *districting* cible $\bar{\lambda}_i$.

La génération de solutions de *districting* optimales ou presque optimales représente un défi majeur en raison de la complexité et du coût computationnel associés. Ainsi, pour construire cet ensemble d'entraînement, nous utilisons des instances de petite taille, dans l'espoir que le modèle puisse ensuite généraliser efficacement à des instances de plus grande taille.

L'approche d'entraînement par imitation offre plusieurs avantages. D'abord, il est efficace,

car entraîner sur des petites instances permet de réduire les coûts computationnels tout en obtenant des solutions de haute qualité. Ensuite, bien que l'ensemble d'entraînement soit constitué de petites instances, l'objectif est que le modèle apprenne des structures et des motifs qui se généralisent aux grandes instances. Enfin, en se basant sur des solutions optimales, le modèle est formé pour produire des solutions robustes et efficaces dans des situations réelles.

L'entraînement par imitation de DistrictNet consiste à résoudre le problème d'optimisation suivant :

$$\min_{w} \sum_{i=1}^{n} \mathcal{L}(\phi_w(x_i), \bar{y}_i),$$

où $\mathcal{L}$ est une fonction de perte qui quantifie la distance entre une solution CMST correspondant aux poids des arêtes θ et une solution CMST cible $\bar{y}$.

Cependant, notre ensemble d'entraînement ne contient pas de cibles CMST mais uniquement des cibles de *districting*. Pour surmonter ce défi, l'entraînement de DistrictNet est réalisé en trois étapes principales :

— Introduction d'une nouvelle représentation des solutions CMST,
— Conversion des cibles de *districting* en cibles CMST
— Définition d'une fonction de perte appropriée avec des propriétés souhaitables et dérivation de son gradient.

Représentation des Solutions Cibles. Pour faciliter l'entraînement par imitation ainsi que la résolution du problème de CMST, nous introduisons une nouvelle représentation des solutions CMST. Vu que les solutions CMST sont caractérisées par les arêtes, nous pouvons construire un encodage Y de l'ensemble V des solutions ν du problème CMST donné dans $\mathbb{R}^{|E|}$, où chaque solution CMST est représentée par un vecteur de dimension égale au nombre d'arêtes E.

Considérons maintenant une solution ν du CMST. Nous pouvons alors exprimer cette solution sous forme de représentation vectorielle $y(\nu)$ dans l'espace des arêtes comme suit :

$$y(\nu) = \{y_e(\nu)\}_{e \in E}.$$

Dans cette formulation, chaque composante $y_e(\nu)$ est définie par $y_e(\nu) = \sum_{s \in T} \mathbb{1}(e \in s)\nu_s$. Ici, ν_s est une variable binaire indiquant si le sous-arbre s est sélectionné dans la solution ν, et $1(e \in s)$ est une fonction indicatrice qui vaut 1 si l'arête e appartient au sous-arbre s, et

0 sinon.

Avec cette représentation, nous pouvons reformuler le problème de CMST initial. Le problème de CMST peut être exprimé comme la maximisation d'une fonction linéaire sur l'ensemble des représentations Y des solutions :

$$\min_{y \in Y} \theta^\top y, \tag{3.6}$$

où θ représente un vecteur de poids associé aux arêtes. Toutefois, pour être en accord avec les pratiques courantes dans la littérature des méthodes DFL, où les problèmes sont généralement posés en termes de maximisation, nous reformulons ce problème comme une maximisation. Cette transformation ne cause aucune perte de généralité, car il suffit de multiplier les poids des arêtes par -1 pour passer d'un problème de minimisation à un problème de maximisation :

$$\max_{y \in Y} \theta^\top y \tag{3.7}$$

Comme l'ensemble Y est fini, son enveloppe convexe C est un polytope. Par conséquent, le programme linéaire donné par

$$\max_{\mu \in C} \theta^\top \mu, \tag{3.8}$$

est équivalent au problème de maximisation ci-dessus. Le changement de notation, de y à μ, souligne que μ prend des valeurs dans l'enveloppe convexe C.

Conversion des Cibles de *districting* en Cibles CMST. Pour entraîner DistrictNet par imitation, nous devons convertir les cibles de *districting* en cibles CMST. Cette conversion repose sur une surjection de l'espace des solutions CMST vers l'espace des solutions de *districting*. Étant donné une solution de *districting* cible $\bar{\lambda}$, nous désignons par $Y(\bar{\lambda})$ l'ensemble des solutions CMST réalisables qui conduisent à $\bar{\lambda}$.

Pour obtenir une solution CMST correspondant à une solution de *districting* donnée, nous utilisons un algorithme de construction $\mathcal{A} : \lambda \to y$. Cet algorithme associe une solution de *districting* λ à une solution CMST $y \in Y(\lambda)$. Cet algorithme peut être aléatoire. Par exemple, DistrictNet construit des solutions de *districting* en résolvant un problème d'arbre de recouvrement de poids minimal avec des poids d'arêtes aléatoires pour chaque district $d \in \bar{\lambda}$. Cela peut être fait efficacement en appliquant l'algorithme de Kruskal en parallèle pour tous les districts sélectionnés. Finalement, nous définissons notre solution CMST cible $\bar{\mu} \in C$ comme :

$$\bar{\mu} = \mathbb{E}[y|\lambda, \mathcal{A}.] \tag{3.9}$$

Cette espérance est prise sur les réalisations de l'algorithme aléatoire $\mathcal{A}$ et elle peut être approximée par un échantillonnage Monte Carlo. Cela nous permet de convertir notre ensemble d'entraînement $(x_i, \bar{\lambda}_i)_{i=1}^n$ en $(x_i, \bar{\mu}_i)_{i=1}^n$ et d'entraîner DistrictNet par imitation.

Perte de Fenchel-Young et Gradient Stochastique. Maintenant que nous avons défini une représentation des solutions CMST et converti les cibles de *districting* en cibles CMST, nous pouvons établir une fonction de perte adaptée pour entraîner DistrictNet par imitation. Étant donné un ensemble d'entraînement $(x_i, \bar{\mu}_i)_{i=1}^n$, notre objectif est que le modèle apprenne à reproduire les solutions CMST cibles $\bar{\mu}_i$ pour chaque instance x_i. Pour ce faire, nous devons évaluer la non-optimalité de la solution $\bar{\mu}$ par rapport à la solution optimale $\mu^*(\theta)$ obtenue avec les poids θ prédits par le modèle. Cela nous permet de définir la fonction de perte suivante :

$$\mathcal{L}(\theta, \bar{\mu}) = \max_{\mu \in C} \theta^\top \mu - \theta^\top \bar{\mu}. \tag{3.10}$$

Minimiser cette perte directement ne fonctionne pas, car $\theta = 0$ constitue une solution optimale triviale. Cependant, en utilisant une fonction de régularisation lisse et strictement convexe $\Omega(\mu)$, nous pouvons formuler le problème régularisé suivant :

$$\max_{\mu \in C} \theta^\top \mu - \Omega(\mu). \tag{3.11}$$

Cette reformulation permet de définir la fonction de perte régularisée suivante :

$$\mathcal{L}_{\text{FY}}(\theta, \bar{\mu}) = \max_{\mu \in C} \theta^\top \mu - \Omega(\mu) - (\theta^\top \bar{\mu} - \Omega(\bar{\mu})). \tag{3.12}$$

Soit $\Omega^*(\theta)$ le conjugué de Fenchel de Ω. La fonction de perte régularisée dans l'équation ci-dessus est alors donnée par :

$$\mathcal{L}_{\text{FY}}(\theta, \bar{\mu}) = \Omega^*(\theta) + \Omega(\bar{\mu}) - \theta^\top \bar{\mu}. \tag{3.13}$$

Cette expression est connue sous le nom d'inégalité de Fenchel-Young [81]. La théorie de la dualité de Fenchel garantit que cette perte possède des propriétés souhaitables. Notamment, elle est convexe en θ, non-négative et égale à zéro uniquement si $\bar{\mu}$ est la solution optimale

du problème régularisé. Son gradient peut être exprimé comme suit :

$$\nabla_\theta \mathcal{L}(\theta, \bar{\mu}) = \arg\max_{\mu \in C} \theta^\top \mu - \Omega(\mu) - \bar{\mu}. \tag{3.14}$$

Un choix pratique, exploitant le lien entre la perturbation et la régularisation [70,71], consiste à définir $\Omega(\mu)$ comme le conjugué de Fenchel de l'objectif perturbé $\mathbb{E}\left[\max_{\mu \in C}(\theta + \epsilon Z)^\top \mu\right]$, où Z est une variable gaussienne standard. Dans ce cas, le gradient de la perte de Fenchel-Young par rapport à θ peut être calculé comme suit [70] :

$$\nabla_\theta \mathcal{L}_{\mathrm{FY}}(\theta, \bar{\mu}) = \mathbb{E}_Z\left[\mu^*(\theta + \epsilon Z)\right] - \bar{\mu}. \tag{3.15}$$

Ce gradient peut être approximé par une méthode de Monte Carlo de la manière suivante :

$$\frac{1}{M} \sum_{m=1}^{M} \arg\max_{\mu \in C} (\theta + \epsilon Z_m)^\top \mu, \tag{3.16}$$

où $\{Z_m\}_{m=1}^M$ sont des perturbations échantillonnées.

L'entrainement de DistrictNet est résumé dans l'algorithme 1.

Algorithme 1 Entraînement de DistrictNet

Entrée Ensemble de données $\mathcal{D} = \{x_i, y_i\}_{i=1}^{n}$

Initialiser le modèle GNN ϕ_w
Pour $e = 1$ **à** `max_epochs` **Faire**
  Pour tout $b = 1$ **à** `nb_batches` **Faire**
    Obtenir le lot $\mathcal{B}$ d'exemples d'entraînement de $\mathcal{D}$
    Pour tout (x, y) dans $\mathcal{B}$ **Faire**
      Prédire les coûts des arêtes $\theta \leftarrow \phi_w(x)$
      Échantillonner des perturbations $Z^{(1)}, \ldots, Z^{(M)} \sim \mathcal{N}(0, 1)$
      Pour $m = 1$ **à** M **Faire**
        Perturber les coûts des arêtes $\theta^{(m)} \leftarrow \theta + \varepsilon Z^{(m)}$
        Résoudre le CMST perturbé pour obtenir $y^{(m)}$
      Fin Pour
      Moyenne des solutions perturbées $\hat{y} \leftarrow \frac{1}{M} \sum_{m=1}^{M} y^{(m)}$
      Évaluer le gradient de la perte FY $\nabla_\theta \mathcal{L}_{\text{FY}}(y, \phi(x))$
      Faire une étape de gradient sur les paramètres de ϕ
    Fin Pour
  Fin Pour
Fin Pour
Retourner le modèle entraîné ϕ

3.3 Résolution du CMST

La résolution des problèmes de CMST est plus abordable que celle des problèmes de *districting* car il s'agit d'un problème d'optimisation en une seule étape avec un objectif linéaire. Cela reste cependant un problème combinatoire. Plusieurs méthodes ont été proposées pour le résoudre, allant des méthodes exactes coûteuses aux heuristiques rapides.

3.3.1 Méthode de partitionnement de graphe

Cette méthode consiste à énumérer tous les sous-arbres possibles et à appliquer la formulation présentée dans (3.5). Bien que cette approche soit exhaustive, elle devient impraticable pour les grands graphes en raison du nombre exponentiel de sous-arbres possibles.

3.3.2 Méthode de Branch and Cut

Pour accélérer la résolution du problème de CMST sans nécessiter l'énumération exhaustive de tous les sous-arbres possibles, une approche alternative efficace consiste à utiliser une méthode de *Branch and Cut*. Nous nous appuyons sur la formulation initiale proposée par [82], que nous adaptons à notre cas spécifique en ajoutant des contraintes supplémentaires pour garantir que chaque sous-arbre respecte une taille minimale et que le nombre total de sous-arbres est fixé.

Considérons un graphe $G = (V, E)$ orienté, où $V = \{0, \ldots, n\}$ est l'ensemble des nœuds et E est l'ensemble des arêtes. Le nœud n est désigné comme le nœud racine. La formulation du problème est la suivante :

$$\min \sum_{i=0}^{n} \sum_{j=0}^{n-1} \theta_{ij} \cdot y_{ij} \tag{3.17a}$$

$$\text{s.t.} \quad y_{ij} - \sum_{p=1}^{\bar{d}} z_{ijp} = 0 \quad \forall i, j \in \{0, \ldots, n\} \tag{3.17b}$$

$$\sum_{i=0}^{n} y_{ij} = 1 \quad \forall j \in \{1, \ldots, n-1\} \tag{3.17c}$$

$$\sum_{i=0}^{n} \sum_{p=1}^{\bar{d}} p \cdot z_{ijp} - \sum_{i=0}^{n} \sum_{p=1}^{\bar{d}} p \cdot z_{jip} = 1 \quad \forall j \in \{1, \ldots, n-1\} \tag{3.17d}$$

$$\sum_{j=0}^{n-1} y_{nj} = k \tag{3.17e}$$

$$\sum_{p=1}^{\bar{d}} p \cdot z_{nip} \geq \underline{d} \cdot y_{ni} \quad \forall i \in \{1, \ldots, n-1\} \tag{3.17f}$$

$$y_{ij} \in \{0, 1\} \quad \forall i, j \in \{0, \ldots, n\}, \tag{3.17g}$$

$$z_{ijp} \in \{0, 1\} \quad \forall i, j \in \{0, \ldots, n\}, p \in \{1, \ldots, \bar{d}\} \tag{3.17h}$$

Cette formulation est une extension de la formulation CMST proposée dans [82] pour inclure des contraintes supplémentaires de capacité minimale et de nombre de sous-arbres. Les variables de décision y_{ij} indiquent si l'arête ij est sélectionnée dans l'arbre couvrant, tandis que les variables z_{ijp} représentent si un flux de valeur p passe par l'arc ij. Les contraintes (3.17b) lient les variables y et z, et les contraintes (3.17c) assurent que chaque nœud, sauf le nœud racine, est relié par exactement une arête entrante. Les contraintes (3.17d) maintiennent la conservation du flux à travers chaque nœud, tandis que la contrainte (3.17e) garantit que le

nombre de sous-arbres connectés sélectionnés dans la solution est exactement égal à k. Enfin, les contraintes (3.17f) imposent que chaque sous-arbre respecte une capacité minimale de $\underline{d}$.

Dans ce cadre, nous proposons d'intégrer les coupes développées dans les travaux de [83, 84], telles que les coupes de capacité étendues (*extended capacity cuts* ou ECC) et les coupes de Fenchel (*Fenchel cuts*), qui ont été utilisées avec succès pour résoudre des problèmes CMST.

3.3.3 Recherche Locale Itérative (ILS)

Tous les problèmes de *districting* et de CMST sur de grandes instances sont résolus en utilisant l'ILS. Cet algorithme applique de manière itérative deux méthodes : une recherche locale qui guide la solution vers un optimum local, et une perturbation qui explore l'espace des solutions en modifiant aléatoirement la solution.

L'algorithme de recherche locale itérative est basé sur trois étapes principales :

— Initialisation : Une solution initiale de *districting* est générée en utilisant une version modifiée de l'algorithme de Kruskal, qui construit un arbre couvrant en ajoutant des arêtes de poids croissant tout en respectant la contrainte de capacité.

— Recherche locale : À chaque itération, les arêtes de l'arbre sont évaluées et échangées pour réduire le coût total tout en maintenant la connectivité et en respectant les contraintes de capacité.

— Perturbation : Après avoir atteint un minimum local, la solution est légèrement perturbée en modifiant aléatoirement certaines arêtes, puis la recherche locale est réappliquée.

Cette ILS est décrite dans l'algorithme 2. La recherche locale est détaillée dans l'algorithme 3, et la perturbation est détaillée dans l'algorithme 4.

Algorithme 2 Recherche locale itérative

Entrée : Solution initiale S_0
Initialiser : Meilleure solution $S_{\text{best}} \leftarrow S_0$
Tant que critère d'arrêt non atteint **Faire**
  Appliquer la recherche locale à S : $S' \leftarrow LS(S)$
  Si $\texttt{coût}(S') < \texttt{coût}(S_{\text{best}})$ **Alors**
    Enregistrer la meilleure solution : $S_{\text{best}} \leftarrow S'$
  Fin Si
  Appliquer une perturbation à S' : $S \leftarrow P(S')$
Fin Tant que
Retour : S_{best}

Algorithme 3 Recherche locale : LS

Entrée : Solution actuelle S
Tant que une amélioration est trouvée **Faire**
Initialiser : Meilleur mouvement $m_{\text{best}} = \varnothing$
Pour chaque paire de districts $(D_a, D_b) \in S$ dans un ordre aléatoire **Faire**
Identifier les nœuds frontières entre D_a et D_b
Pour chaque nœud frontière i dans D_a **Faire**
Évaluer tous les mouvements possibles : déplacer i vers D_b ou échanger avec un nœud j dans D_b
Comparer les mouvements avec m_{best} et conserver le meilleur
Fin Pour
Si m_{best} améliore la solution **Alors**
Appliquer le meilleur mouvement $S \leftarrow m_{\text{best}}(S)$
Fin Si
Fin Pour
Fin Tant que
Retour : S

Algorithme 4 Perturbation : P

Entrée Solution actuelle S
Assure Solution perturbée
Pour chaque paire de districts (D_i, D_j) dans un ordre aléatoire **Faire**
Identifier les nœuds frontières entre D_i et D_j
Pour chaque nœud frontière nœud_i dans D_i et nœud_j dans D_j **Faire**
Avec une certaine probabilité, appliquer la perturbation : déplacer nœud_i vers D_j ou échanger nœud_i avec nœud_j
Fin Pour
Fin Pour

Solution initiale. Une solution initiale faisable pour le CMST est un ensemble de BU connectés de taille k, où chaque sous-ensemble a un nombre de BU compris entre la taille minimale et maximale. Obtenir une solution faisable satisfaisant ces contraintes est déjà un problème NP-difficile. Nous utilisons donc une heuristique pour générer une solution initiale.

Étant donné un ensemble de poids d'arêtes, nous utilisons d'abord une version modifiée de l'algorithme de Kruskal, décrite dans l'algorithme 5. Cet algorithme commence avec tous les nœuds dans leur propre cluster et trie tous les poids d'arêtes par ordre croissant. Ensuite,

il fusionne de manière gloutonne les clusters aux points extrêmes des arêtes avec le coût le plus bas si la taille du cluster fusionné est inférieure à la taille maximale. Cela fournit une solution initiale qui peut encore avoir trop de clusters. Dans ce cas, nous appliquons l'algorithme de fusion glouton décrit dans l'algorithme 6. Cet algorithme réduit davantage le nombre de clusters en fusionnant continuellement les deux clusters voisins ayant la plus petite taille combinée jusqu'à ce que le nombre de clusters atteigne la cible désirée k

Algorithme 5 Kruskal modifié

Initialiser un cluster pour chaque sommet $u \in G$
Trier les arêtes de G par poids croissant en E_{sorted}
Pour chaque arête (u, v) dans E_{sorted} **Faire**
  Si les sommets u et v sont dans des clusters différents et $|u| + |v| \leq t$ **Alors**
    Fusionner les clusters de u et v
  Fin Si
Fin Pour

Algorithme 6 Fusion glouton

Tant que le nombre de districts est supérieur à la cible k **Faire**
  Trouver les districts voisins D_a et D_b avec la plus petite taille combinée
  Fusionner D_a et D_b
Fin Tant que

Cependant, les solutions initiales obtenues peuvent ne pas toujours respecter les exigences de taille limite. Pour y remédier, nous avons incorporé une fonction de pénalité dans l'algorithme de recherche locale, qui pénalise les clusters ne respectant pas les limites de taille, guidant ainsi la recherche locale vers des solutions faisables. Dans nos expériences, une solution faisable est presque toujours trouvée à la première itération de l'algorithme ILS.

Empiriquement, cette méthode trouve efficacement des solutions initiales faisables dans nos principales expériences. Cependant, elles ne sont pas suffisantes pour les plus grandes instances (lorsque $N \geq 600$ dans nos expériences). Nous introduisons donc un algorithme de réparation supplémentaire donné dans l'algorithme 7. Cet algorithme de réparation ajuste chaque district pour respecter les contraintes de taille minimale et maximale spécifiées en ajoutant ou en supprimant des nœuds des districts voisins tout en maintenant la connectivité globale.

Algorithme 7 Réparation des districts

Entrée Solution actuelle S, taille minimale $\underline{d}$, taille maximale $\bar{d}$, instance du problème, paires de voisins

Trier les districts dans S par taille croissante.
Pour chaque district d dans S **Faire**
Si $|d| < \underline{d}$ **Alors**
Identifier les districts voisins et les nœuds adjacents.
Trier les voisins par taille décroissante.
Tant que $|d| < \underline{d}$ et qu'il reste des nœuds chez les voisins **Faire**
Sélectionner et ajouter un nœud d'un voisin en respectant la connexion et la taille minimale.
Mettre à jour les voisins.
Si aucun nœud approprié n'est disponible **Alors**
break
Fin Si
Fin Tant que
Fin Si
Fin Pour
Trier les districts dans S par taille décroissante.
Pour chaque district d dans S **Faire**
Si $|d| > \bar{d}$ **Alors**
Identifier les nœuds les plus éloignés et les districts voisins qui peuvent recevoir des nœuds.
Tant que $|d| > \bar{d}$ et qu'il y a des voisins pouvant accepter des nœuds **Faire**
Retirer un nœud et l'ajouter à un district voisin plus petit.
Mettre à jour les voisins.
Si aucun nœud approprié n'est disponible **Alors**
break
Fin Si
Fin Tant que
Fin Si
Fin Pour
Vérifier si la solution est conforme aux contraintes de taille.
Si la solution est invalide après plusieurs itérations **Alors**
Arrêter ou recommencer.
Fin Si
Retourner Solution réparée S

CHAPITRE 4 RÉSULTATS EXPÉRIMENTAUX

Dans ce chapitre, nous évaluons la performance de DistrictNet sur des problèmes réels de *districting* et de tournées de véhicules. Nous menons des expériences répétées et comparons notre approche à d'autres approches de référence basées sur l'apprentissage. Nous examinons les aspects suivants de DistrictNet :

— Sa capacité à généraliser à des instances de grande taille hors distribution, à partir d'un entraînement sur quelques petites instances.
— Sa robustesse face aux variations des paramètres d'instance tels que les tailles de district.
— Le rôle du modèle d'approximation CMST pour permettre cette généralisation.

4.1 Approches de référence

Nous évaluons notre méthode par rapport à quatre approches de référence basées sur l'apprentissage des coûts de district $C_{TSP}(d)$: BD, FIG, PREDGNN et TwoStageCMST. BD, FIG et PREDGNN, définies en détail dans la section 2.1, peuvent être combinées avec toute technique de recherche et sont intégrées dans un cadre de recherche locale itérative (ILS) avec une limite de temps de 20 minutes.

BD. Suivant [44], la méthode BD estime le coût d'un district en fonction de la superficie totale du district, du nombre attendu de demandes de livraison et de la distance moyenne entre le dépôt et un point de demande. L'entraînement de BD consiste à ajuster un paramètre β permettant de moduler ces facteurs.

FIG. Suivant [50], la méthode FIG étend l'approche de BD en utilisant un vecteur de paramètres $\beta = (\beta_1, \beta_2, \beta_3, \beta_4)$ pour moduler les estimations de coût. FIG inclut des termes supplémentaires pour améliorer la précision des estimations dans des contextes variés.

PREDGNN. Suivant [45], PREDGNN utilise un GNN pour estimer le coût d'un district. Chaque nœud du graphe représente une BU et est caractérisé par des attributs spécifiques, tels que la population, la densité, la superficie et la distance au dépôt. Le GNN exploite ces attributs pour apprendre une représentation latente, qui est ensuite utilisée pour prédire le coût du district. L'entraînement de PREDGNN vise à minimiser l'erreur absolue moyenne entre les coûts prédits et les coûts réels observés pour chaque district.

Dans les travaux originaux de [45], PREDGNN est conçu pour être entraîné sur un ensemble de districts appartenant à une ville donnée avec des paramètres fixes (comme la taille des districts et le nombre de BU). Le modèle est ensuite utilisé pour prédire les coûts des districts non vus de cette même ville, ce qui permet d'éviter de recalculer leurs coûts de manière explicite. Cela permet de gagner du temps de calcul tout en garantissant une prédiction efficace dans ce cadre spécifique.

Cependant, chaque ville et chaque taille cible de district ont leur propre modèle d'entraînement, ce qui limite la capacité de PREDGNN à généraliser à d'autres villes ou à des tailles de districts différentes. Cette spécialisation, bien qu'efficace dans un contexte donné, rend difficile l'adaptation du modèle à des configurations de villes ou de problèmes qui diffèrent de celles sur lesquelles il a été initialement entraîné.

Bien que cette méthode ait été conçue pour le problème de partitionnement de districts dans le cadre d'une ville fixe, avec des paramètres fixes (comme la taille des districts, le nombre de BU et la localisation du dépôt), elle se généralise bien pour prédire le coût des districts qui n'ont pas été inclus dans les données d'entraînement. En effet, le modèle est entraîné sur un ensemble de districts d'une ville donnée, puis utilisé pour prédire le coût d'autres districts de cette même ville. Un paramètre d'appartenance à un district est ajouté pour chaque BU, permettant ainsi au modèle de mieux capturer les relations entre les BU et le district auquel ils appartiennent, ce qui contribue à la précision des prédictions dans ce cadre spécifique.

Dans notre travail, nous avons cherché à tester la capacité de PREDGNN à généraliser dans des contextes différents. Pour cela, nous avons entraîné PREDGNN sur un ensemble d'instances provenant de plusieurs villes, mais avec une taille de district fixe de 3 et un nombre de BU fixe de 30 par ville. Ensuite, nous avons testé le modèle sur des villes différentes, ayant un nombre de BU et des tailles de districts variés. Contrairement à l'approche originale, où un modèle distinct est formé pour chaque ville et chaque taille de district, nous avons entraîné un modèle unique capable de s'adapter à des contextes variés.

Nous anticipons donc qu'elle ne performe pas aussi bien dans des contextes où la ville et ses caractéristiques varient, car elle a été initialement développée pour des scénarios fixes. Nous avons inclus PREDGNN dans notre comparaison afin de mettre en évidence les différences entre les méthodes conçues pour des paramètres de ville fixes et celles capables de s'adapter aux changements de villes et de paramètres.

TwoStageCMST. Nous présentons la méthode **TwoStageCMST** comme un benchmark additionnel pour résoudre le problème de *districting* en le modélisant comme un CMST. Cette méthode suit une approche d'apprentissage différente de notre méthode principale, en

se concentrant sur le PFL, tandis que notre approche se concentre sur le DFL.

TwoStageCMST se compose de deux étapes : d'abord, prédire les coûts des arêtes du graphe, puis résoudre le problème de CMST avec ces coûts prédits. L'idée principale est de déterminer les poids optimaux des arêtes θ pour que le coût de toute solution de *districting* soit proche du coût de la solution CMST correspondante.

Dans TwoStageCMST, nous travaillons avec des données sous la forme $\{(x_i, \{\lambda_i^{(j)}, c_i^{(j)}\}_{j=1}^m)\}_{i=1}^n$, où x_i représente une instance du problème de *districting*. Pour chaque instance x_i, nous considérons m solutions faisables $\lambda_i^{(j)}$ (qui ne sont pas nécessairement optimales), et $c_i^{(j)}$ est le coût associé à chaque solution de *districting* $\lambda_i^{(j)}$. La première étape de cette méthode consiste à utiliser notre algorithme aléatoire $\mathcal{A}$ pour convertir chaque solution de *districting* $\lambda_i^{(j)}$ en une solution CMST correspondante $y_i^{(j)}$. La deuxième étape consiste à apprendre un modèle ϕ_w qui prédit des poids θ_i tels que les coûts des solutions de *districting* $\lambda_i^{(j)}$ et les solutions CMST $y_i^{(j)}$ soient proches $\theta_i^\top y_i^{(j)} \approx c_i^{(j)}$. Le problème d'apprentissage peut être formulé comme suit :

$$\min_w \frac{1}{n} \sum_{i=1}^{n} \frac{1}{m} \sum_{j=1}^{m} \left\| c_i^{(j)} - \theta_i^\top y_i^{(j)} \right\|^2, \quad \text{où } \theta_i = \phi_w(x_i).$$

Une fois le paramètre optimal w trouvé, le modèle ϕ_w prédit les poids optimaux θ_i pour une nouvelle instance x_i. Le problème CMST est ensuite résolu en utilisant ces poids, ce qui est formulé comme suit :

$$y_{\text{opt}} = \operatorname*{argmin}_{y \in \mathcal{Y}} \theta_i^\top y.$$

La solution y_{opt} est ensuite utilisée pour générer la solution de *districting* optimale λ_{opt} correspond à l'instance x_i.

4.2 Collecte des données

Pour nos tests, nous utilisons des données réelles collectées et générées pour toutes les expériences. Tout d'abord, nous décrivons les instances de test, les sept grandes villes sur lesquelles nous évaluons toutes les méthodes. Ensuite, nous présentons comment nous utilisons des données réelles pour générer un ensemble d'instances d'entraînement de taille arbitraire. Nous discutons également des hypothèses de distribution utilisées pour simuler la demande aléatoire dans une ville.

Instances de test : villes réelles et population. Pour nos instances de test, nous utilisons les villes de Bristol, Leeds, Londres, Manchester, Paris, Lyon et Marseille. Un résumé

de ces instances est présenté dans le tableau 4.1. Le tableau montre les statistiques sur la population, la superficie et la densité des BU composant les sept villes testées. Bien que la superficie et la densité puissent varier d'une ville à l'autre, les statistiques de population sont relativement constantes. Cela n'est pas surprenant, car les BU tendent à être conçues pour avoir des populations similaires. Les données géographiques, y compris les frontières de chaque ville et des BU, peuvent être consultées sur Uber Movement [1]. De plus, les données de recensement sont disponibles sur le site de l'Office for National Statistics (ONS) [2]. Pour les villes françaises, les données géographiques, y compris les frontières, ont été obtenues auprès d'Opendatasoft. Les données de population pour ces régions proviennent de l'Institut National de la Statistique et des Études Économiques (INSEE) [3].

Tableau 4.1 Statistiques des villes utilisées pour les instances de test.

		Bristol	**Leeds**	**Londres**	**Manchester**	**Paris**	**Lyon**	**Marseille**
Population (milliers)	Moyenne	8.28	7.76	9.01	8.21	2.49	2.71	2.29
	Std.	2.13	1.80	1.90	2.12	0.74	1.37	0.77
	Minimum	5.23	5.20	5.43	4.96	0.36	0.28	0.58
	Médiane	7.85	7.40	8.69	7.76	2.33	2.46	2.27
	Maximum	18.16	16.17	24.97	15.87	4.83	6.47	4.88
Surface (km^2)	Moyenne	16.39	6.80	1.62	3.05	0.093	6.77	0.52
	Std.	30.16	10.40	1.88	3.60	0.07	7.62	0.94
	Minimum	0.63	0.35	0.30	0.59	0.018	0.11	0.07
	Médiane	2.80	3.05	1.16	2.13	0.07	4.21	0.29
	Maximum	171.21	94.13	22.30	35.69	0.45	38.96	6.81
Densité	Moyenne	3.12	2.99	8.92	4.01	34.3	1.94	9.59
	Std.	2.62	2.80	5.25	2.32	16.5	2.649	6.98
	Minimum	0.05	0.10	0.36	0.14	2.6	0.0039	85.18
	Médiane	2.89	2.41	7.77	3.79	34.73	0.71	7.5
	Maximum	12.72	25.20	28.27	16.36	129.09	11.45	32.6

Génération d'instances d'entraînement. Pour assembler un ensemble d'entraînement diversifié, nous générons de nouvelles villes en perturbant des villes réelles. Tout d'abord, nous lisons les données géographiques de 27 villes réelles en Angleterre, répertoriées dans le tableau 4.2 (à l'exclusion des villes de l'ensemble de test mentionnées dans la section 4.2). À partir de ces villes initiales, nous générons $n = 100$ sous-graphes connectés aléatoires de taille $N = 30$ BU et échantillonnons la population de chaque BU selon une distribution normale $\mathcal{N}(8000, 2000)$ tronquée entre 5000 et 20000. Ce choix étant justifié par les observations réelles au Royaume-Uni où les populations des BU varient généralement entre ces valeurs,

1. `https://movement.uber.com/`
2. `https://www.ons.gov.uk/`
3. `https://www.insee.fr/fr/accueil`

avec une moyenne proche de 8000, comme le montre le tableau 4.1. Pour chaque instance x_i, nous calculons sa solution optimale pour la taille cible $t = 3$ en énumérant complètement les solutions de *districting* possibles et en évaluant leurs coûts. Pour évaluer ces coûts, nous utilisons la méthode de Monte Carlo décrite précédemment 3.1 avec 100 échantillons. Nous fixons la valeur de κ à $\frac{96}{8000 \times t}$ pour contrôler le nombre de demandes de livraison par BU, où 8000 représente la population moyenne d'une BU au Royaume-Uni. Ce choix de κ est fait pour garantir qu'un district typique génère environ 96 demandes par jour, reflétant ainsi des scénarios réalistes d'itinéraires de livraison couvrant une centaine d'arrêts, ce qui est courant dans les livraisons de colis [45]. Cette procédure génère notre ensemble d'entraînement de $n = 100$ instances et solutions associées.

Les caractéristiques des BU incluent la population, la densité, la superficie, le périmètre, la compacité et la distance au dépôt. Le vecteur de caractéristiques arêtes est construit en moyennant les vecteurs de caractéristiques des deux BU qu'elle connecte et en incluant la distance entre les centres des deux BU connectées. Ces caractéristiques permettent à DistrictNet de capturer efficacement les informations géographiques et sociales pertinentes pour le problème de *districting*.

Pour générer les graphes de villes utilisés dans l'entraînement de DistrictNet, nous échantillonnons des sous-graphes connectés aléatoires à partir des villes d'entraînement. Le processus de génération des instances d'entraînement est donné en pseudo-code dans l'algorithme 8, et l'algorithme d'échantillonnage des sous-graphes est donné dans l'algorithme 9. Enfin, un dépôt central artificiel est placé au centroïde du polygone résultant. Cette procédure nous permet de créer un ensemble d'entraînement de taille arbitraire contenant des instances d'entraînement réalistes (mais de petite taille). Cette configuration assure qu'il n'y a pas de contamination entre les instances d'entraînement et de test.

Tableau 4.2 Villes utilisées pour générer des instances d'entraînement et nombre correspondant de BU

Ville	N	Ville	N	Ville	N	Ville	N
Barnet	41	Coventry	42	Leicester	37	Sefton	38
Birmingham	132	Derby	31	Liverpool	61	Sheffield	70
Bolton	35	Doncaster	39	Newham	37	Southwark	33
Bradford	61	Ealing	39	Nottingham	38	Stoke	34
Brent	34	Greenwich	33	Oldham	33	West Midlands	684
Brighton	33	Kirklees	59	Plymouth	32	Wigan	40
Cardiff	48	Lambeth	35	Rotherham	33		

Algorithme 8 Génération des instances d'entraînement

Entrée Liste des villes d'entraînement C, Nombre d'instances n

Initialiser l'ensemble d'entraînement $\mathcal{D}$
Pour $i = 1$ **à** n **Faire**
    Sélectionner une ville aléatoire c dans C
    Échantillonner un sous-graphe G_i à partir de c en utilisant l'algorithme 9
    Générer les populations des BU de G_i selon une distribution normale $N(8000, 2000)$ tronquée
    Ajouter G_i à $\mathcal{D}$
Fin Pour
Retourner $\mathcal{D}$

Algorithme 9 Échantillonnage des sous-graphes

Entrée Ville c, Taille du sous-graphe N

Initialiser un sous-graphe vide $G = (V, E)$
Sélectionner aléatoirement une BU v de c et l'ajouter à V
Tant que taille de $V < N$ **Faire**
    Sélectionner aléatoirement une BU voisine u de V et l'ajouter à V
    Ajouter l'arête (v, u) à E
Fin Tant que
Retourner G

Détails de l'entraînement

Pour entraîner notre modèle DistrictNet ainsi que les approches de référence, nous avons utilisé différentes configurations de paramètres et d'hyperparamètres adaptés à chaque méthode.

Le modèle **DistrictNet** utilise un batch de 1 et un taux d'apprentissage initial de 10^{-3} avec un facteur de décroissance de 0.9 appliqué toutes les 10 époques, jusqu'à un taux minimal de 10^{-4}. L'entraînement est effectué sur 100 époques. Pour la construction de la solution cible CMST dans l'équation 3.9, nous utilisons une seule observation de notre constructeur aléatoire (Kruskal avec des poids aléatoires). La perturbation Z est fixée à une gaussienne standard avec $\epsilon = 1.0$, et nous utilisons $M = 20$ échantillons pour approximer le gradient attendu dans l'équation 3.16.

Pour **PREDGNN**, nous utilisons un ensemble de 10,000 instances et un batch de 64, un taux d'apprentissage de 10^{-4}, et un entraînement sur 10^4 époques. Une limite de temps de

24 heures est imposée pour l'entraînement, et le processus est arrêté tôt si aucun changement significatif de la perte (défini comme supérieur à 10^{-4}) n'est observé dans les 1000 dernières époques.

Les approches **BD** et **FIG** sont également entraînées sur 10,000 instances d'entraînement. Pour ces méthodes, le seul hyperparamètre est le nombre d'échantillons utilisés dans l'approximation Monte Carlo de la distance moyenne aux demandes. Nous utilisons 100 scénarios dans toutes nos expériences pour cette approximation.

Pour l'approche **TwoStageCMST**, nous avons généré un ensemble de 100 instances de test, et pour chaque instance x_i, nous avons produit $m = 50$ solutions faisables $\lambda_i^{(j)}$. Le modèle est ensuite entraîné pendant 2000 époques, avec un batch de 50 et un taux d'apprentissage initial de 10^{-3} avec un facteur de décroissance de 0.9 appliqué toutes les 10 époques, jusqu'à un taux minimal de 10^{-4}. Pour l'optimisation, nous utilisons l'algorithme d'optimisation Adam.

Ces configurations d'entraînement ont été choisies pour optimiser les performances de chaque modèle en fonction de leurs spécificités et des caractéristiques des données utilisées.

4.3 Résultats

Nous avons évalué la performance de DistrictNet et des benchmarks sur un ensemble diversifié d'instances hors distribution afin de mesurer leur capacité à produire des solutions optimales. En effet, contrairement aux instances d'entraînement, générées à partir de villes perturbées avec des tailles de districts et des structures homogènes (30 BU par instance et une taille de district fixée à 3), les instances de test sont basées sur des villes réelles comme Bristol, Leeds, Londres, Manchester, Paris, Lyon et Marseille, avec des tailles de districts variables. Ces villes, décrites dans la section 4.2, ont été choisies pour représenter une diversité géographique et démographique. Pour chaque ville, nous utilisons des instances de test de taille $N = 120$ BU, et nous faisons varier la taille cible des districts parmi $t \in \{3, 6, 12, 20, 30\}$. Cela nous permet de constituer un total de 35 instances de test, toutes distinctes des données d'entraînement.

Les résultats sont présentés dans le tableau 4.3 sous forme d'une étude d'ablation. Ce tableau montre la différence relative des coûts moyens obtenus par chaque méthode sur les instances de test par rapport à DistrictNet. Les coûts moyens sont calculés sur toutes les instances de test, et pour chaque taille cible de district. Pour évaluer une solution de *districting*, nous utilisons la méthode de Monte Carlo décrite précédemment avec 100 échantillons 3.1. Les résultats montrent que DistrictNet surpasse systématiquement les benchmarks en produisant des solutions de *districting* avec des réductions de coûts significatives, d'environ 10% par

rapport à toutes les autres méthodes. Comme nous l'avions anticipé, PREDGNN, qui utilise un GNN, n'a pas bien performé dans les contextes où les villes et les paramètres des districts varient. En effet, ses résultats sont similaires à ceux des autres méthodes heuristiques, ce qui souligne ses limites en matière de généralisation, lorsque les caractéristiques géographiques et les paramètres du problème changent. Cela met en évidence la capacité de DistrictNet à généraliser à travers diverses structures de ville et pour des instances plus grandes que celles utilisées pour l'entraînement. Les détails des résultats sont présentés dans le tableau 4.4. Ce tableau montre les coûts de *districting* obtenus par les différentes méthodes pour chaque ville testée et chaque taille cible de district. Le coût le plus bas est indiqué en bleu et le deuxième meilleur en orange. Il est démontré que DistrictNet fournit le coût le plus bas dans 31 des 35 instances de test. De plus, il est bien établi que la complexité du problème de *districting* augmente avec la taille cible du district, et dans le tableau 4.4, on observe que DistrictNet permet des économies significatives, réduisant les coûts jusqu'à 21% dans le meilleur des cas.

Tableau 4.3 Étude d'ablation montrant la valeur de l'apprentissage axé sur la décision.

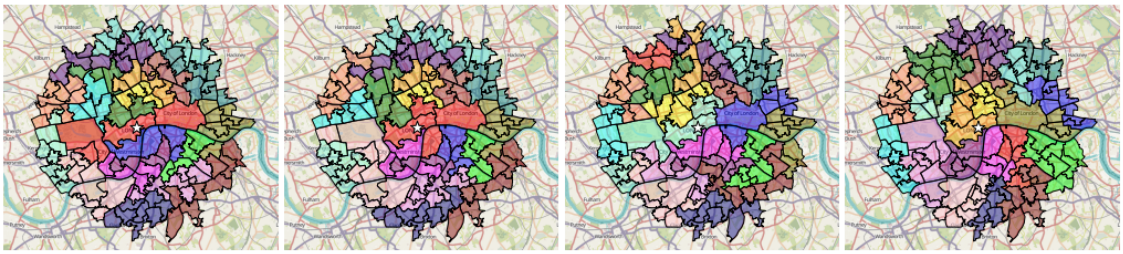
	Coût relatif moyen	p-value
BD	9.92 %	4.9e-09
FIG	10.01 %	8.9e-09
PredGnn	11.91 %	1.5e-10
DistrictNet	0.0 %	-

Nous présentons également des exemples de solutions de *districting* pour les villes de Londres et Manchester dans les figures 4.1 et 4.2 pour $t = 6$ et $t = 20$ BU. Ces solutions correspondent aux coûts présentés dans le tableau 4.4. Les figures montrent que DistrictNet à tendance à produire des districts compacts et homogènes. En revanche, les trois benchmarks ont tendance à générer des districts en forme de "U", ce qui est particulièrement visible pour les grandes tailles de districts, comme $t = 20$. Il est intéressant de noter que BD et FIG produisent des résultats visuellement similaires, notamment pour la ville de Londres. Pour confirmer ces observations, nous avons calculé la compacité des solutions de *districting* en utilisant la mesure de Polsby-Popper [85], définie par

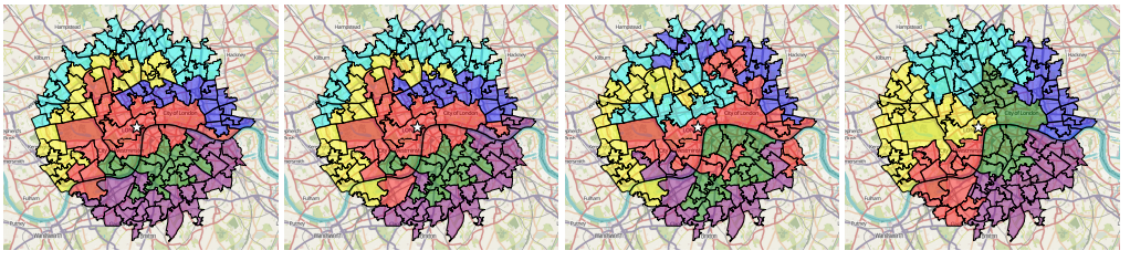
$$\text{Polsby-Popper} = \frac{4\pi A}{P^2},$$

où A est la superficie du district et P est son périmètre. Les résultats sont présentés dans le tableau 4.5, où l'on constate que DistrictNet obtient généralement la meilleure compacité

en moyenne.

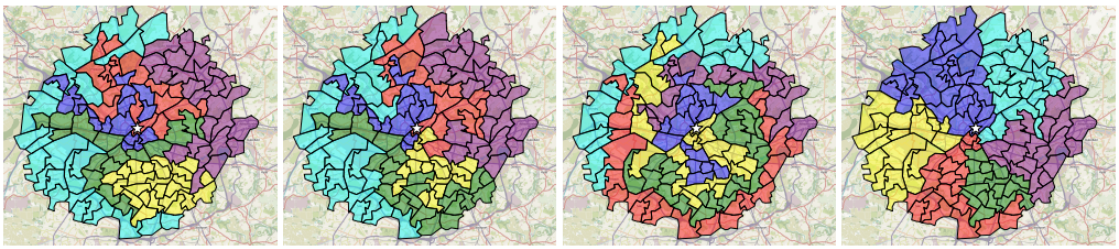


(a) BD, $t = 6$ (b) FIG, $t = 6$ (c) PREDGNN, $t = 6$ (d) DISTRICTNET, $t = 6$



(e) BD, $t = 20$ (f) FIG, $t = 20$ (g) PREDGNN, $t = 20$ (h) DISTRICTNET, $t = 20$

FIGURE 4.1 Exemples de solutions de *districting* pour la ville de Londres avec $t = 6$ et $t = 20$ BU.



(a) BD, $t = 20$ (b) FIG, $t = 20$ (c) PREDGNN, $t = 20$ (d) DISTRICTNET, $t = 20$

FIGURE 4.2 Exemples de solutions de *districting* pour la ville de Manchester avec $t = 6$ BU.

TABLEAU 4.5 Évaluation de la compacité des solutions de découpage en utilisant la formule de Polsby-Popper . Les résultats montrent que DistrictNet obtient généralement la meilleure compacité en moyenne.

Ville	BD	FIG	PredGNN	DistrictNet
Bristol	0.124	0.125	0.124	0.173
Leeds	0.168	0.154	0.187	0.268
Londres	0.086	0.089	0.08	0.11
Lyon	0.183	0.177	0.209	0.258
Manchester	0.167	0.165	0.142	0.272
Marseille	0.143	0.143	0.177	0.185
Paris	0.169	0.182	0.157	0.274
Moyenne	0.149	0.148	0.154	0.22

Résultat 1 : *DISTRICTNET peut généraliser à des instances de grande taille hors distribution, avec des données géographiques et démographiques variées ainsi que des paramètres de problème différents.*

Résultat 2 : *DISTRICTNET fournit des solutions de districting qui sont plus compactes et homogènes que les benchmarks.*

Généralisation et scalabilité à grande échelle. Nous avons mené une expérience supplémentaire pour étudier la généralisation et la scalabilité de **DistrictNet** à des villes de grande taille. Pour cela, nous avons généré des instances de test pour les villes de Londres, Leeds et Bristol. Pour Londres, nous avons généré des instances de tailles variant de 200 à 900. Pour Leeds, nous avons utilisé des instances de taille N variant parmi $\{150, 190, 240, 290\}$, et pour Bristol, des instances de taille N variant parmi $\{110, 160, 210, 260\}$. Pour chaque taille de ville, nous avons fixé la taille cible des districts à $t = 20$. La limite de temps pour l'algorithme ILS a été maintenue à 20 minutes pour toutes les méthodes lorsque $N < 400$ et augmentée à 60 minutes lorsque $N \geq 400$.

Les résultats sont présentés dans la figure 4.3, qui montre le coût de la solution de *districting* obtenue avec les méthodes de référence par rapport à **DistrictNet** pour différentes tailles de ville. Une valeur supérieure à 100 % indique que la méthode de référence performe moins bien que **DistrictNet**. Nous observons que **DistrictNet** fournit de très bonnes solutions, même pour les plus grandes tailles de ville, et surpasse systématiquement les benchmarks, même pour les grandes tailles.

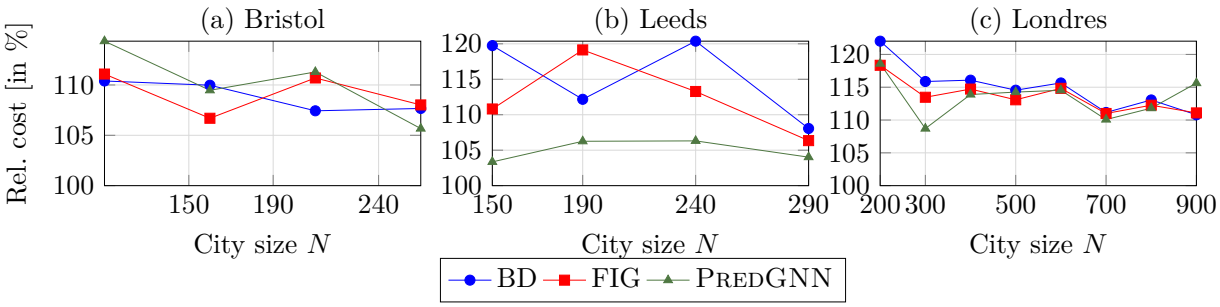


FIGURE 4.3 Coût relatif par rapport à DISTRICTNET pour une taille de district cible $t = 20$ et une taille de ville variable.

Résultat 3 : *DISTRICTNET fournit des solutions de districting de haute qualité meme pour des villes de grande taille.*

Importance des données pour l'apprentissage axé sur la décision. Comme mentionné précédemment, la création de l'ensemble d'entraînement est computationnellement coûteuse. Pour cette raison, nous avons mené une étude pour évaluer l'importance de la taille de l'ensemble d'entraînement pour DISTRICTNET. Nous avons entraîné DISTRICTNET sur des ensembles d'entraînement de tailles variables $n \in \{20, 50, 150, 200\}$ et évalué les performances sur les instances de test qui ont les mêmes que celui dans notre première expérience. À savoir les villes de Bristol, Leeds, Londres, Manchester, Paris, Lyon et Marseille. Les résultats sont présentés dans la figure 4.4, qui montre le coût moyen de *districting* pour toutes les villes et tailles cibles par rapport au coût pour $n = 20$, avec un intervalle de confiance à 95% représenté par une zone ombrée. Une valeur inférieure à 100% signifie que DISTRICTNET s'améliore par rapport à son entraînement avec $n = 20$. La figure montre que DISTRICTNET peut obtenir des coûts faibles même avec un nombre étonnamment faible d'exemples d'entraînement ($n = 50$). Augmenter le nombre d'exemples tend à améliorer les résultats, bien qu'avec un rendement décroissant.

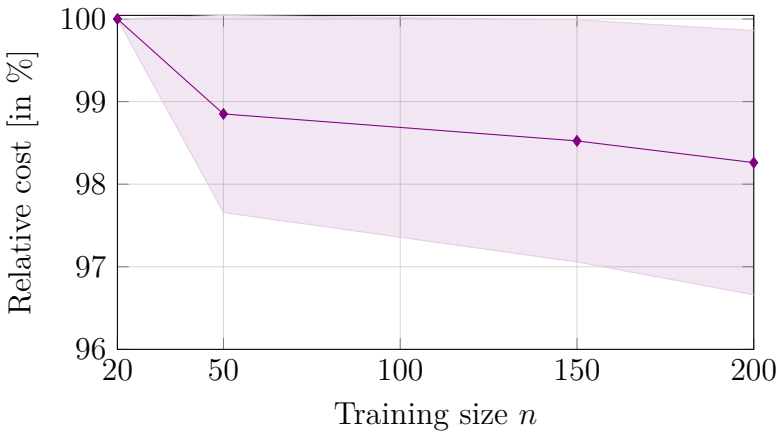


FIGURE 4.4 Coût relatif par rapport à DISTRICTNET pour différentes tailles d'ensemble d'entraînement.

Résultat 4 : *DISTRICTNET peut être entraîné avec un nombre relativement faible d'exemples d'entraînement et bénéficie de l'augmentation de la taille de l'ensemble d'entraînement.*

La valeur de l'utilisation du CMST et de l'apprentissage axé sur la décision. Dans cette section, nous menons une étude d'ablation pour évaluer l'impact de l'utilisation du CMST en tant que modèle d'approximation pour le *districting*, ainsi que l'apport du DFL. Nous comparons les performances de DISTRICTNET, qui intègre le DFL, à celles de TWOSTAGECMST, une méthode alternative basée sur une approche d'apprentissage orientée vers la prédiction. Cette comparaison permet de mettre en évidence les différences entre ces deux stratégies et d'évaluer l'efficacité du CMST dans la structuration du problème de *districting*. Les résultats de cette analyse sont présentés dans le tableau 4.6.

TABLEAU 4.6 Étude d'ablation montrant la valeur de de l'utilisation de CMST pour approximer les coûts moyens des tournées des véhicules.

	Coût relatif moyen	p-value
BD	6.41 %	3.3e-05
FIG	6.58 %	3.3e-05
PREDGNN	8.27 %	5.9e-08
DISTRICTNET	-3.11 %	4.3e-07
TwoStageCMST	0.0 %	1.0e+00

En examinant les résultats de 4.6, nous observons que DistrictNet est meilleur que TwoSta-

geCMST de 3.11%, et TwoStageCMST est supérieur aux autres méthodes d'environ 6%.

Ces résultats soulignent plusieurs points importants :

- L'utilisation du CMST comme modèle d'approximation permet d'améliorer considérablement les performances de *districting*, même sans l'apprentissage axé sur la décision. Cette amélioration de 6% démontre que le CMST capture efficacement la structure générale du problème de *districting*, en fournissant des solutions initiales de haute qualité.
- DistrictNet, en intégrant le DFL avec le CMST, améliore encore plus les performances, indiquant que l'apprentissage des poids d'arêtes spécifiques à chaque instance permet de mieux adapter les solutions aux particularités des données d'entrée. Cette capacité à s'ajuster dynamiquement aux variations des instances rend DistrictNet particulièrement puissant pour les problèmes de grande taille et complexes.
- La comparaison entre TwoStageCMST et les autres benchmarks met en évidence l'importance de la structuration des problèmes d'optimisation. Les méthodes traditionnelles qui ne tiennent pas compte de la structure sous-jacente du problème de *districting* ont des performances inférieures, car elles ne peuvent pas efficacement capturer les contraintes et les relations complexes entre les BU et les districts.

En conclusion, l'utilisation du CMST comme modèle d'approximation, combinée avec des techniques d'apprentissage avancées, offre une voie prometteuse pour résoudre des problèmes de *districting* complexes. DistrictNet, en particulier, montre une robustesse et une capacité de généralisation, en tirant parti des forces combinées de l'apprentissage de bout en bout et de la modélisation structurelle.

Résultat 5 : *L'utilisation du CMST comme modèle d'approximation améliore les performances de districting, et le DFL avec le CMST permet d'obtenir des solutions de districting de haute qualité.*

Généralisation intra-distribution. Enfin, nous réalisons une expérience pour examiner la capacité de généralisation intra-distribution des différentes méthodes. Autrement dit, nous étudions le cadre où les instances d'entraînement et de test sont échantillonnées à partir de la même distribution. Plus précisément, nous avons utilisé des villes synthétiques, générées en perturbant les mêmes villes que celles utilisées pour l'entraînement, et en échantillonnant à nouveau la population de chaque BU selon la même distribution que celle utilisée pour l'ensemble d'entraînement. Nous générons 200 instances en suivant la procédure décrite dans la section précédente, et les divisons en deux ensembles de 100 instances. Le premier ensemble est utilisé pour entraîner toutes les méthodes et le second pour les évaluer hors échantillon.

Toutes les instances ont une taille $N = 30$ et une taille de district cible $t = 3$. Ainsi, nous pouvons les résoudre de manière optimale en un temps raisonnable en énumérant complètement les districts possibles et en utilisant la formulation exacte donnée dans le problème 3.2. Nous résolvons de manière optimale tous les problèmes de *districting* pour BD, FIG et PREDGNN, ainsi que tous les problèmes de CMST pour DistrictNet, à la fois pendant l'entraînement et les phases de test. De plus, puisque nous ne considérons ici que de petites instances, nous pouvons mesurer l'écart d'optimalité de toutes les méthodes : la différence relative entre le coût de la solution optimale réelle et celui des solutions fournies par les méthodes.

Les écarts d'optimalité des quatre méthodes sont présentés dans le tableau 4.7 pour les instances d'entraînement et de test. Les résultats montrent que DistrictNet obtient le plus petit écart moyen par rapport aux solutions optimales comparé aux autres benchmarks. BD et FIG ont des performances similaires, ce qui est attendu puisqu'ils estiment les coûts des districts de manière similaire. PREDGNN présente les performances moyennes les plus faibles, bien que les variations soient moindres, car il a un écart maximum plus faible que BD et FIG.

Tableau 4.7 Les coûts de *districting* moyens, maximaux et minimaux relatifs à la solution optimale pour les instances de test et d'entrainement.

	Train			Test		
	Avg.	Max.	Min.	Avg.	Max.	Min.
BD	4.0 %	11.25 %	0.27 %	4.09 %	11.06 %	0.42 %
FIG	4.24 %	15.98 %	0.27 %	4.09 %	12.16 %	0.42 %
PredGnn	3.37 %	10.05 %	0.05 %	4.64 %	11.89 %	0.22 %
DistrictNet	1.86 %	7.48 %	0.0 %	2.34 %	5.52 %	0.17 %

Résultat 6 : *DistrictNet fournit des solutions de districting plus proches de l'optimalité que les benchmarks, même dans un cadre d'intra-distribution.*

Solveur exact vs ILS. Nous avons mené une expérience pour comparer les performances de DistrictNet avec un solveur exact (utilisant la méthode de branch and cut) et la méthode ILS pour résoudre le problème du CMST et ensuite convertir la solution en une solution de *districting* et l'évaluer en utilisant Monte Carlo. Pour le solveur exact, nous avons utilisé l'algorithme de Branch and Cut présenté dans la section 3.17. Un temps limite de 1 heure a été imposé à l'algorithme de Branch and Cut. Si le solveur n'a pas trouvé la solution optimale dans ce délai, il retourne la meilleure solution trouvée au moment de l'arrêt. La majorité des

instances ont été résolues, 26 instances sur 35. Pour les instances non résolues, nous avons un écart moyen de 0.74% par rapport à la solution optimale.

Nous avons comparé les performances de DistrictNet avec l'algorithme exact et l'ILS sur 35 instances de test. Chaque instance a été résolue deux fois : une première fois en utilisant la méthode exacte (Branch and Cut), et une seconde fois en utilisant l'ILS. Le critère de performance utilisé est le coût relatif moyen par rapport à DistrictNet-ILS. Les résultats expérimentaux sont présentés dans le tableau 4.8, où les coûts moyens obtenus avec chaque méthode sont reportés.

Les résultats montrent que DistrictNet avec la méthode exacte (Branch and Cut) offre des performances légèrement meilleures dans certains cas, mais la méthode ILS fournit des résultats de qualité comparable avec des temps de calcul plus courts, en particulier pour les grandes instances. En moyenne, DistrictNet avec l'ILS a permis de réduire les coûts de manière plus constante et rapide que la méthode exacte, bien que cette dernière ait montré une meilleure précision.

Tableau 4.8 Comparaison de DistrictNet utilisant les méthodes ILS et Branch and Cut.

	Coût Relatif Moyen
DistrictNet-B&C : DistrictNet avec Branch and Cut	-0.7%
DistrictNet-ILS : DistrictNet avec Recherche Locale Itérative	0.0%

Résultat 7 : *DistrictNet avec l'ILS fournit des solutions de districting de haute qualité avec des performances comparables à la méthode exacte. Mais aussi DistrictNet avec la méthode exacte (Branch and Cut) offre des performances légèrement meilleures dans plusieurs cas.*

TABLEAU 4.4 Coûts de *districting* dans différentes villes et tailles cibles de districts pour notre méthode et les methodes de référence. Le meilleur résultat est indiqué en bleu et le deuxième meilleur en orange. La dernière colonne montre la différence relative entre DISTRICTNET et la deuxième meilleure méthode.

City	t	BD	FIG	PREDGNN	DISTRICTNET	(Rel.)
Bristol	3	1938.54	1924.02	1971.96	1881.32	(**−2.2**%)
	6	1280.23	1312.04	1287.93	1206.1	(**−5.8**%)
	12	896.94	918.67	869.9	812.53	(**−6.6**%)
	20	691.24	709.33	679.68	635.02	(**−6.6**%)
	30	578.18	578.18	603.69	540.31	(**−6.5**%)
Leeds	3	1439.35	1448.6	1494.32	1407.03	(**−2.2**%)
	6	939.8	956.43	957.16	903.47	(**−3.9**%)
	12	656.53	661.52	668.3	589.79	(**−10.2**%)
	20	529.24	549.46	484.9	453.64	(**−6.4**%)
	30	457.59	466.8	418.36	391.64	(**−6.4**%)
London	3	743.09	742.35	754.14	738.61	(**−0.5**%)
	6	492.26	484.39	504.03	463.67	(**−4.3**%)
	12	337.59	328.23	350.16	296.68	(**−9.6**%)
	20	270.67	255.28	279.77	222.65	(**−12.8**%)
	30	206.92	221.95	231.34	182.4	(**−11.8**%)
Lyon	3	1465.44	1471.53	1544.37	1450.7	(**−1.0**%)
	6	918.5	911.5	958.86	886.18	(**−2.8**%)
	12	638.78	627.71	644.69	582.59	(**−7.2**%)
	20	556.5	562.6	595.21	470.47	(**−15.5**%)
	30	558.58	555.92	506.03	433.89	(**−14.3**%)
Manchester	3	1179.42	1166.96	1236.67	1171.94	(0.4%)
	6	783.15	771.26	816.07	738.49	(**−4.2**%)
	12	563.44	556.7	570.86	474.86	(**−14.7**%)
	20	438.27	433.95	476.29	364.18	(**−16.1**%)
	30	392.39	378.21	382.66	311.82	(**−17.6**%)
Marseilles	3	466.58	469.99	485.24	477.31	(2.3%)
	6	288.77	289.76	293.16	288.73	(**−0.0**%)
	12	210.0	207.41	206.44	195.49	(**−5.3**%)
	20	180.7	180.43	183.26	160.92	(**−10.8**%)
	30	161.67	161.67	175.65	151.03	(**−6.6**%)
Paris	3	185.43	187.71	188.71	192.04	(3.6%)
	6	119.0	115.25	124.5	116.88	(1.4%)
	12	83.9	85.09	88.97	81.39	(**−3.0**%)
	20	80.1	77.1	80.87	67.29	(**−12.7**%)
	30	75.88	79.37	75.93	59.6	(**−21.5**%)

CHAPITRE 5 CONCLUSION

5.1 Synthèse des travaux

Dans cette étude, nous avons présenté DistrictNet, une nouvelle approche pour résoudre le problème de partitionnement géographique avec tournées de véhicules. DistrictNet combine des techniques d'apprentissage automatique et d'optimisation combinatoire pour générer des solutions efficaces. Notre méthode repose sur l'idée d'approximer le problème de partitionnement géographique par un modèle de CMST en utilisant un réseau de neurones graphes (GNN) pour prédire les coûts des arêtes du graphe.

Les résultats expérimentaux montrent que DistrictNet peut être entraîné sur un petit ensemble d'exemples et appliqué à un large éventail de villes, de tailles d'instances et d'hyperparamètres. Cette capacité de généralisation est essentielle dans les applications réelles où les données peuvent être limitées et variées. De plus, la robustesse de DistrictNet face aux variations des paramètres d'instance, tels que les tailles de districts, permet une flexibilité et une adaptabilité accrues.

Les études de cas sur des villes réelles ont illustré la capacité de DistrictNet à générer des solutions économiquement viables et à réduire les coûts de manière significative par rapport aux benchmarks. Cette performance robuste est obtenue malgré un entraînement sur des instances de petite taille, mettant en évidence la puissance de notre modèle DFL. L'expérience supplémentaire avec TwoStageCMST a également montré que même sans apprentissage de bout en bout, l'utilisation du CMST comme modèle d'approximation améliore les performances, soulignant ainsi l'importance de capturer la structure du problème. De plus, nous avons démontré que DistrictNet peut bénéficier de l'utilisation de Branch and Cut pour améliorer la qualité des solutions.

5.2 Limitations de la solution proposée

Bien que notre approche présente de nombreux avantages, elle comporte également certaines limitations. Tout d'abord, DistrictNet a été appliqué principalement au *districting* et aux tournées de véhicules. Les résultats pourraient varier si appliqués à d'autres types de problèmes de partitionnement géographique. De plus, bien que notre méthode généralise bien aux instances hors distribution, elle repose sur un ensemble de données d'entraînement limité en taille et en diversité. La qualité des solutions générées dépend également de la précision des caractéristiques des BU et des arêtes utilisées pour représenter les graphes. Enfin, l'approche

actuelle peut être améliorée en termes d'efficacité computationnelle pour les très grandes instances.

Une limitation majeure de notre méthode est que la création de l'ensemble d'entraînement pour DistrictNet est très coûteuse en temps, car pour chaque instance, il faut calculer toutes les solutions de *districting* possibles afin de trouver la solution optimale.

5.3 Améliorations futures

Pour les recherches futures, plusieurs directions peuvent être explorées :

- Étendre l'application de DistrictNet à d'autres problèmes de partitionnement géographique, comme la conception de districts de vote ou de districts scolaires, où des contraintes similaires de connectivité et de taille sont présentes ou des problèmes de partitionnement de graphe plus généraux.
- Améliorer la modélisation des caractéristiques des BU et des arêtes pour capturer encore plus finement les spécificités géographiques et sociales des zones étudiées.
- Explorer l'impact de différentes stratégies de perturbation et de recherche locale dans l'algorithme ILS pour améliorer encore la qualité des solutions générées.
- Développer des techniques pour mieux gérer les très grandes instances en optimisant l'efficacité computationnelle, possiblement par la parallélisation ou l'utilisation de ressources de calcul distribué.
- Réduire le temps de création de l'ensemble d'entraînement en utilisant des méthodes de génération de données plus efficaces ou des approches de sélection de sous-ensembles représentatifs.

En conclusion, DistrictNet représente une avancée significative dans la résolution du *districting* avec tournées de véhicules en combinant des GNN avec un modèle CMST. Son architecture, combinant des techniques d'apprentissage automatique et d'optimisation combinatoire, ouvre de nouvelles perspectives pour des applications variées nécessitant des solutions efficaces et robustes. Les travaux futurs permettront de renforcer et d'étendre les capacités de DistrictNet, contribuant ainsi à son adoption dans divers domaines d'application.

RÉFÉRENCES

[1] J. Kalcsics et R. Z. Ríos-Mercado, "Districting problems," *Location Science*, p. 705–743, 2019.

[2] T. Vidal *et al.*, "A concise guide to existing and emerging vehicle routing problem variants," *European Journal of Operational Research*, vol. 286, n°. 2, p. 401–416, 2020.

[3] J. Williams, "Political redistricting : a review," *Papers in Regional Science*, vol. 74, n°. 1, p. 13–40, 1995.

[4] G. R. Webster, "Reflections on current criteria to evaluate redistricting plans," *Political Geography*, vol. 32, p. 3–14, 2013.

[5] F. Ricca *et al.*, "Political districting : from classical models to recent approaches," *Annals of Operations Research*, vol. 204, p. 271–299, 2013.

[6] L. C. Galvão *et al.*, "A multiplicatively-weighted voronoi diagram approach to logistics districting," *Computers & Operations Research*, vol. 33, n°. 1, p. 93–114, 2006.

[7] D. Haugland *et al.*, "Designing delivery districts for the vehicle routing problem with stochastic demands," *European Journal of Operational Research*, vol. 180, n°. 3, p. 997–1010, 2007.

[8] A. A. Zoltners et P. Sinha, "Sales territory design : Thirty years of modeling and implementation," *Marketing Science*, vol. 24, n°. 3, p. 313–331, 2005.

[9] J. F. López-Pérez et R. Z. Ríos-Mercado, "Embotelladoras arca uses operations research to improve territory design plans," *Interfaces*, vol. 43, n°. 3, p. 209–220, 2013.

[10] A. F. Saïd Hanafi et P. Vaca, "Municipal solid waste collection : An effective data structure for solving the sectorization problem with local search methods," *INFOR : Information Systems and Operational Research*, vol. 37, n°. 3, p. 236–254, 1999.

[11] L. Muyldermans *et al.*, "Districting for salt spreading operations," *European Journal of Operational Research*, vol. 139, n°. 3, p. 521–532, 2002.

[12] J. A. Ferland et G. Guénette, "Decision support system for the school districting problem," *Operations Research*, vol. 38, n°. 1, p. 15–21, 1990.

[13] G. B. Dantzig et J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, n°. 1, p. 80–91, 1959.

[14] Y. LeCun *et al.*, "Deep learning," May 2015.

[15] I. Goodfellow *et al.*, *Deep Learning*. MIT Press, 2016.

[16] A. Krizhevsky *et al.*, “Imagenet classification with deep convolutional neural networks,” dans *Advances in Neural Information Processing Systems*, F. Pereira *et al.*, édit., vol. 25. Curran Associates, Inc., 2012.

[17] E. Ngai *et al.*, “The application of data mining techniques in financial fraud detection : A classification framework and an academic review of literature,” *Decision Support Systems*, vol. 50, n°. 3, p. 559–569, 2011, on quantitative methods for detection of financial fraud.

[18] Q. Yu *et al.*, “Application of long short-term memory neural network to sales forecasting in retail—a case study,” dans *Advanced Manufacturing and Automation VII*, K. Wang *et al.*, édit. Singapore : Springer Singapore, 2018, p. 11–17.

[19] B. Csáji, “Approximation with artificial neural networks,” Thèse de doctorat, 06 2001.

[20] M. M. Bronstein *et al.*, “Geometric deep learning : Grids, groups, graphs, geodesics, and gauges,” 2021.

[21] T. N. Kipf et M. Welling, “Semi-supervised classification with graph convolutional networks,” 2017.

[22] T. Kipf et M. Welling, “Variational graph auto-encoders,” *ArXiv*, vol. abs/1611.07308, 2016.

[23] F. Wu *et al.*, “Simplifying graph convolutional networks,” dans *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri et R. Salakhutdinov, édit., vol. 97. PMLR, 09–15 Jun 2019, p. 6861–6871.

[24] P. Velikovi *et al.*, “Graph attention networks,” 2018.

[25] F. Monti *et al.*, “Fake news detection on social media using geometric deep learning,” 2019.

[26] J. Zhang *et al.*, “Gaan : Gated attention networks for learning on large and spatiotemporal graphs,” 2018.

[27] J. Gilmer *et al.*, “Neural message passing for quantum chemistry,” 2017.

[28] P. W. Battaglia *et al.*, “Relational inductive biases, deep learning, and graph networks,” 2018.

[29] ——, “Interaction networks for learning about objects, relations and physics,” 2016.

[30] W. Fan *et al.*, “Graph neural networks for social recommendation,” 2019.

[31] S. Motie et B. Raahemi, “Financial fraud detection using graph neural networks : A systematic review,” *Expert Systems with Applications*, vol. 240, p. 122156, 2024.

[32] W. L. Hamilton *et al.*, “Inductive representation learning on large graphs,” 2018.

[33] J. Mandi *et al.*, “Decision-focused learning : Foundations, state of the art, benchmark and future opportunities,” 2024.

[34] U. Sadana *et al.*, “A survey of contextual optimization methods for decision-making under uncertainty,” *European Journal of Operational Research*, vol. 320, nº. 2, p. 271–289, 2025.

[35] D. Wahdany *et al.*, “More than accuracy : end-to-end wind power forecasting that optimises the energy system,” *Electric Power Systems Research*, vol. 221, p. 109384, 2023.

[36] L. Sang *et al.*, “Electricity price prediction for energy storage system arbitrage : A decision-focused approach,” *IEEE Transactions on Smart Grid*, vol. 13, nº. 4, p. 2822–2832, 2022.

[37] Z. Chai *et al.*, “Port selection for fluid antenna systems,” *IEEE Communications Letters*, vol. 26, nº. 5, p. 1180–1184, 2022.

[38] T. Greif *et al.*, “Combinatorial optimization and machine learning for dynamic inventory routing,” 2024.

[39] L. Baty *et al.*, “Combinatorial optimization-enriched machine learning to solve the dynamic vehicle routing problem with time windows,” *Transportation Science*, vol. 58, nº. 4, p. 708–725, 2024.

[40] A. Butler et R. H. Kwon, “Integrating prediction in mean-variance portfolio optimization,” *Quantitative Finance*, vol. 23, nº. 3, p. 429–452, 2023.

[41] K. Helsgaun, *An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems : Technical report.* Roskilde Universitet, déc. 2017.

[42] M. Dyer et A. Frieze, “On the complexity of partitioning graphs into connected subgraphs,” *Discrete Applied Mathematics*, vol. 10, nº. 2, p. 139–153, 1985.

[43] J. Beardwood *et al.*, “The shortest path through many points,” *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 55, nº. 4, p. 299327, 1959.

[44] C. F. Daganzo, “The distance traveled to visit n points with a maximum of c stops per vehicle : An analytic model and an application,” *Transportation Science*, vol. 18, nº. 4, p. 331–350, 1984.

[45] A. Ferraz *et al.*, “Deep learning for data-driven districting-and-routing,” *arXiv preprint*, 2024.

[46] O. Kwon *et al.*, “Estimating the length of the optimal tsp tour : An empirical study using regression and neural networks,” *Computers & Operations Research*, vol. 22, nº. 10, p. 1039–1046, 1995.

[47] A. G. Novaes *et al.*, "A continuous approach to the design of physical distribution systems," *Computers & Operations Research*, vol. 27, nº. 9, p. 877–893, 2000.

[48] H. Lei *et al.*, "Dynamic design of sales territories," *Computers and Operations Research*, vol. 56, p. 84–92, 2015.

[49] T. Chien, "Operational estimators for the length of a traveling salesman tour," *Computers & Operations Research*, vol. 19, nº. 6, p. 469–478, 1992.

[50] M. A. Figliozzi, "Analysis of the efficiency of urban commercial vehicle tours : Data collection, methodology, and policy implications," *Transportation Research Part B : Methodological*, vol. 41, nº. 9, p. 1014–1032, 2007.

[51] F. Akkerman et M. Mes, "Distance approximation to support customer selection in vehicle routing problems," *Annals of Operations Research*, 2022.

[52] T. Varol *et al.*, "Neural network estimators for optimal tour lengths of traveling salesperson problem instances with arbitrary node distributions," *Transportation Science*, vol. 58, nº. 1, p. 45–66, 2024.

[53] Q. Cappart *et al.*, "Combinatorial optimization and reasoning with graph neural networks," *Journal of Machine Learning Research*, vol. 24, nº. 130, p. 1–61, 2023.

[54] I. Bello *et al.*, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv :1611.09940*, 11 2016.

[55] H. Dai *et al.*, "Learning combinatorial optimization algorithms over graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[56] M. Gasse *et al.*, "Exact combinatorial optimization with graph convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[57] R. Baltean-Lugojan *et al.*, "Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks," 2019.

[58] Y. Tang *et al.*, "Reinforcement learning for integer programming : Learning to cut," 2020.

[59] M. Kruber *et al.*, "Learning when to use a decomposition," dans *Integration of AI and OR Techniques in Constraint Programming*, D. Salvagnin et M. Lombardi, édit. Cham : Springer International Publishing, 2017, p. 202–210.

[60] P. Bonami *et al.*, "Learning a classification of mixed-integer quadratic programming problems," dans *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, W.-J. van Hoeve, édit. Cham : Springer International Publishing, 2018, p. 595–604.

[61] C. K. Joshi *et al.*, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv :1906.01227*, 2019.

[62] W. Kool *et al.*, “Attention, learn to solve routing problems !” dans *International Conference on Learning Representations*, 2019.

[63] E. Larsen *et al.*, “Predicting tactical solutions to operational planning problems under imperfect information,” *INFORMS Journal on Computing*, vol. 34, n°. 1, p. 227242, janv. 2022.

[64] J. Dumouchelle *et al.*, “Neur2SP : Neural two-stage stochastic programming,” *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[65] H. Bae *et al.*, “Deep value function networks for large-scale multistage stochastic programs,” dans *International Conference on Artificial Intelligence and Statistics*. PMLR, 2023, p. 11 267–11 287.

[66] Y. Bengio *et al.*, “Machine learning for combinatorial optimization : A methodological tour dhorizon,” *European Journal of Operational Research*, vol. 290, n°. 2, p. 405–421, 2021.

[67] J. Kotary *et al.*, “End-to-end constrained optimization learning : A survey,” *International Joint Conference on Artificial Intelligence*.

[68] S. Gould *et al.*, “On differentiating parameterized argmin and argmax problems with application to bi-level optimization,” 2016.

[69] B. Amos et J. Z. Kolter, “OptNet : Differentiable optimization as a layer in neural networks,” dans *International Conference on Machine Learning*, vol. 70. PMLR, 2017, p. 136–145.

[70] Q. Berthet *et al.*, “Learning with differentiable perturbed optimizers,” *Advances in Neural Information Processing Systems*, vol. 33, p. 9508–9519, 2020.

[71] G. Dalle *et al.*, “Learning with combinatorial optimization layers : a probabilistic approach,” *arXiv preprint arXiv :2207.13513*, 2022.

[72] A. N. Elmachtoub et P. Grigas, “Smart predict, then optimize,” *Management Science*, vol. 68, n°. 1, p. 9–26, 2022.

[73] J. Mandi et T. Guns, “Interior point solving for LP-based prediction+optimisation,” dans *Advances in Neural Information Processing Systems*, vol. 33, 2020, p. 7272–7282.

[74] M. Mulamba *et al.*, “Contrastive losses and solution caching for predict-and-optimize,” 2021.

[75] B. Wilder *et al.*, “End to end learning and optimization on graphs,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[76] L. Stewart *et al.*, “Differentiable clustering with perturbed spanning forests,” dans *Advances in Neural Information Processing Systems*, 2023.

[77] S. Lin et B. W. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem,” *Operations Research*, vol. 21, p. 498–516, 1973.

[78] C. Komusiewicz et F. Sommer, “Enumerating connected induced subgraphs : Improved delay and expérimental comparison,” *Discrete Applied Mathematics*, vol. 303, p. 262–282, 2021.

[79] C. Morris *et al.*, “Weisfeiler and leman go neural : Higher-order graph neural networks,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, n°. 01, 2019.

[80] C. H. Papadimitriou, “The complexity of the capacitated tree problem,” *Networks*, vol. 8, p. 217–230, 1978.

[81] M. Blondel *et al.*, “Learning with Fenchel-Young losses,” *Journal of Machine Learning Research*, vol. 21, n°. 1, p. 1314–1382, 2020.

[82] L. Gouveia, “A 2n constraint formulation for the capacitated minimal spanning tree problem,” *Operations Research*, vol. 43, n°. 1, p. 130–141, 1995.

[83] E. Uchoa *et al.*, “Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation,” *Math. Program.*, vol. 112, n°. 2, p. 443472, apr 2008.

[84] ——, “Branch-and-cut and hybrid local search for the multi-level capacitated minimum spanning tree problem,” *Networks*, vol. 59, n°. 1, p. 148–160, 2012.

[85] D. D. Polsby et R. D. Popper, “The third criterion : Compactness as a procedural safeguard against partisan gerrymandering,” *Yale Law & Policy Review*, vol. 9, n°. 2, p. 301–353, 1991.