International Journal of Geo-Information

MDPI

Article

A Trade-Off Algorithm for Solving p-Center Problems with a Graph Convolutional Network

Haojian Liang [1], Shaohua Wang [2,3,4,5,*], Huilai Li [6], Huichun Ye [2,4,5] and Yang Zhong [7]

[1] School of Artificial Intelligence, Jilin University, Changchun 130012, China; hjliang20@mails.jlu.edu.cn
[2] International Research Center of Big Data for Sustainable Development Goals, Beijing 100094, China; yehc@aircas.ac.cn
[3] State Key Laboratory of Remote Sensing Science, Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing 100094, China
[4] Key Laboratory of Digital Earth Science, Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing 100094, China
[5] Key Laboratory of Earth Observation of Hainan Province, Aerospace Information Research Institute, Chinese Academy of Sciences, Sanya 572029, China
[6] School of Mathematics, Jilin University, Changchun 130012, China; lihuilai@jlu.edu.cn
[7] Department of Information System and Technology, Claremont Graduate University, Claremont, CA 91711, USA; yang.zhong@cgu.edu
* Correspondence: wangshaohua@aircas.ac.cn; Tel.: +86-010-8217-8178

Abstract: The spatial optimization method between combinatorial optimization problems and GIS has many geographical applications. The p-center problem is a classic NP-hard location modeling problem, which has essential applications in many real-world scenarios, such as urban facility locations (ambulances, fire stations, pipelines maintenance centers, police stations, etc.). This study implements two methods to solve this problem: an exact algorithm and an approximate algorithm. Exact algorithms can get the optimal solution to the problem, but they are inefficient and time-consuming. The approximate algorithm can give the sub-optimal solution of the problem in polynomial time, which has high efficiency, but the accuracy of the solution is closely related to the initialization center point. We propose a new paradigm that combines a graph convolution network and greedy algorithm to solve the p-center problem through direct training and realize that the efficiency is faster than the exact algorithm. The accuracy is superior to the heuristic algorithm. We generate a large amount of p-center problems by the Erdos–Renyi graph, which can generate instances in many real problems. Experiments show that our method can compromise between time and accuracy and affect the solution of p-center problems.

Keywords: p-center problems; graph convolutional network; heuristic algorithm; minimum domain set; deep learning

Citation: Liang, H.; Wang, S.; Li, H.; Ye, H.; Zhong, Y. A Trade-Off Algorithm for Solving p-Center Problems with a Graph Convolutional Network. *ISPRS Int. J. Geo-Inf.* **2022**, *11*, 270. https://doi.org/10.3390/ijgi11050270

Academic Editors: Peng Yue, Danielle Ziebelin and Yaxing Wei

Received: 15 February 2022
Accepted: 15 April 2022
Published: 19 April 2022

1. Introduction

With the excellent performance of deep learning in a growing number of tasks, researchers have begun to use deep learning to solve combinatorial optimization problems (COPs). COPs are of great significance in practical problems, such as logistics transportation [1–4], facility location analytics [5–9], and urban planning [10,11].

There are three main types of classic methods to solve this problem.

- **Exact algorithm.** An exact algorithm refers to the method to find the optimal solution to the problem [12]. When the size of the problem is small, the exact algorithm can obtain the optimal solution in an acceptable time. However, the scale of the problem is often significant in industrial application scenarios. The amount of calculation and storage space required to obtain the optimal solution increases rapidly, prone to "combinatorial explosion". It is not easy to find the optimal solution. The essence of the

exact algorithm is to search the space of the solution. Therefore, with the increase of the scale of the problem, the time complexity is exponential order or even factorial order.

- **Approximate algorithm.** An approximate algorithm refers to using approximate methods for solving optimization problems [13]. For an NP-hard problem, since an exact solution cannot be obtained in polynomial time, it is considered to use an approximate algorithm to obtain an available sub-optimal solution in polynomial time. One of the simplest approximation algorithms is to search for an approximate solution of the original problem with an exact algorithm within a given solution time and then measure whether the approximate solution is feasible.
- **Heuristic algorithm.** A heuristic algorithm is an algorithm based on intuition or empirical, which is widely used in various optimization problems [14,15]. It can give a relatively optimal solution to the problem in an acceptable time. However, there is no theoretical guarantee, and it is impossible to measure the relationship with the optimal solution.

The rise of neural networks provides new ideas for solving COPs. There are two main directions. One direction is reinforcement learning (RL) combined with combinatorial optimization, in which many problems can be modeled as sequential decision-making processes. RL is an effective tool for processing the Markov decision process. Numerous researchers consider using RL to deal with sequential decision-making processes in COPs [16,17]. There are still many challenges, such as the feasibility of the solution, the difficulty of modeling, the difficulty of migrating to large-scale problems, and the trouble of data generation. Another direction is deep learning combined with combinatorial optimization, which combines learning and optimization to improve the performance of solving real problems. Previous literature on using deep learning to solve COPs contains three types: First, pure end2end [18,19] predicts decisions directly from input, but optimization is hard to encode in a neural network. Second, two-stage training predicts and then optimizes [20]. Third, given that the accuracy and consistency of decision-making results cannot be guaranteed at present, decision-focused learning [21] establishes a differentiable optimization objective in the training process. However, it is still a challenge to establish a differentiable optimization system.

We mainly study the p-center (PC) problems to establish a more effective model to deal with practical problems. PC is a classic NP-hard problem and has extremely important guiding significance in the urban facility location, social network analytics, and other issues. PC can be described as selecting p points as the center point in a point dataset and assigning other points to the p-center points so that the maximum distance from all points to their corresponding center points is minimum. Usually, the design of objective functions and constraints for optimization problems is determined according to the practical problems. There are many ways to solve PC problems. In our study, we give an exact algorithm based on minimum domain sets (MDS) [22] and a greedy approximation algorithm [23] to solve the problems. MDS can give the optimal solution to the problem. However, it cannot be solved easily in polynomial time as the problem size is increased. The greedy algorithm can give a sub-solution of the problem in a short time, but the quality of the solution is closely related to the setting of the initial value. We focus more on combining graph learning with optimization problems. A general framework that combines graph learning with optimization is proposed. Using a graph convolutional network (GCN) to give the result of clustering to achieve the efficiency is superior to the exact algorithm and the accuracy is better than the greedy algorithm.

Our contributions are as follows:

- A new approach with a greedy algorithm is proposed to solve p-center problems by directly training GCN.
- Our method achieves that the solution accuracy is superior to the greedy algorithm and the efficiency is better than the exact algorithm.
- The method is transferable and can be combined with various existing approximation or heuristic algorithms.

This study is divided into six sections. The rest of the article is organized as follows. Section 2 introduces the related work about COPs, which combines with deep learning. Section 3 presents the preliminary knowledge for this study. Section 4 is our approach to solving PC and introduces a clustering algorithm to implement the GCN training strategy. Section 5 shows the experiments. We use a different algorithm to solve PC problems on many different scale graphs and achieve the desired results. Section 6 is the conclusions.

2. Related Work

There are many types of research on solving combinatorial optimization problems by training neural networks [24,25]. The pointer network (PN) is one of the representative networks for processing COPs and was first proposed by Vinyals et al. [26]. PN is a sequence-to-sequence learning paradigm, which can mainly solve the problem of the immutable size of the output vocabulary. It can be used to solve the problem of the variable number of nodes in the COPs [27]. The traditional seq2seq cannot solve the problem that the output sequence will change with the different input sequence lengths, especially the problem that the output is heavily dependent on the input. In essence, PN solves the problem of forced constraints on the input and output by simplifying and adjusting the attention mechanism. As a predefined heuristic algorithm is complicated, Zhang et al. used a graph neural network (GNN) to solve the link prediction problem and proposed to learn a heuristic algorithm from a given network instead of a predefined one [28]. By extracting the local sub-graphs around each target link, it can learn a mapping from the sub-graph pattern to the existence of the link to learn a heuristic method suitable for the current network. Highlight a method of learning heuristics from local sub-graphs using GNN. Khalil et al. combined RL and graph embedding to solve COPs [17]. The article mainly used the graph embedding method of struc2vec and proposed a meta-algorithm to solve this problem.

Through meta-algorithms, the same type of COPs based on graph theory can be solved directly, relying on meta-algorithms and not requiring solving one by one. Bello et al. combined RL with PN to solve the COPs [16]. They used the policy gradient method to calculate the parameters of the PN model for TSP and then optimize the solution to the TSP problem. Kool et al. proposed a model based on the transformer model [29]; they built a model architecture that unifies COPs with RL and solved TSP, VRP, OP, and PCTSP.

Tian et al. first proposed learning deep representations for graph clustering [30]. The idea of the article is simplicity. Firstly, they advised applying the autoencoder to the graph structure to get feature extraction. Then, achieve clustering by K-means directly, which came from spectral clustering. Based on it, Yang [18] replaced the Laplace matrix with a modularity matrix. The optimization of the modularity matrix is equivalent to spectral clustering. Xie et al. proposed the deep embedding clustering (DEC) for cluster analysis [31], which defines a parameterized nonlinear mapping from data space X to low-dimensional feature space Z and optimizes the clustering target in low-dimensional space. Guo et al. considered manipulating the feature space to disperse data points by preserving the data structure and the clustering loss as a guide [32]. They put forward the improvement of the deep embedding clustering (IDEC) algorithm. IDEC joint clustering label allocation and learning are suitable for clustering and retaining the characteristics of the data structure by fusing the clustering loss and the loss of the autoencoder.

Wilder et al. thought that the BP in a neural network is in a continuous space, while K-means is used to deal with discrete space problems as an algorithm [33]. They present a differentiable K-means algorithm to effectively deal with modularity and PC problems that combine GNN and K-means clustering. The paper treats CoreData as a PC problem to solve. Although GNN can classify CoreData sets well, it cannot effectively solve p-center. The CoreData is a kind of unweighted graph in which isolated points (not connected with any point or indirectly) will exist in the graph. In their paper, the distance between the isolated and center points is directly set as the current maximum distance, which is problematic and unreliable. We mainly explore and research the PC problem and propose a new method to

solve the PC problems. Meanwhile, we use ER graph to randomly generate a large number of examples of PC problems to prove the feasibility of our method.

3. Preliminary

This section introduces some basic concepts and algorithms about PC problems for our work. We will solve the PC problems with two classic algorithms: an exact algorithm based on minimum domain sets and a fast-approximation method by a greedy algorithm.

3.1. The Definition of the p-Center Problem

The description of PC is as follows [34]: Given an undirected graph G = (V, E) and a positive integer p, the aim is to find a subset C $\in$ V as centers with |C| $<$ p, such that the distance from farthest vertex v to its closest center in S is minimized. We give a strict mathematical definition of PC [35] for having a clear understanding of the algorithm. Figure 1 is a diagram of the p-center problem. The objective of the p-center problem is to minimize the maximal distance for all demand points.

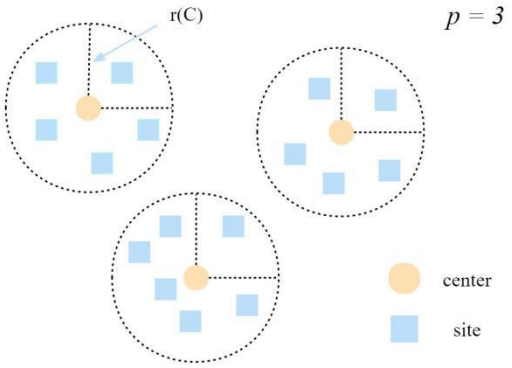


Figure 1. The diagram of the p-center problem.

In metric space X, $d : X \times X \to R$ is the metric function in X, which indicates the similarity or proximity between two elements in X. S is a set in X, the proposal is to find a set $C \subseteq S$ of the most p centers and generate an assignment $\psi : S \to C$. The operator ψ maps each element in S to one of the points in C. According to the above definition, for each $i \in S, d(i, \psi(i))$ is present the distance between i and the cluster center $\psi(i)$. The distance can not only express the true distance between two points, but also the similarity or proximity for any two elements. In many individual issues, the goal is to require $d(i, \psi(i))$ to be as small as possible. The optimization objective of PC is to be minimized $\max_{i \in S} d(i, \psi(i))$.

For each point $i \in S$, we define $U_i = \max_{j \in C_i} d(i, j)$. Obviously, $C_i \subseteq \{j \in S \mid d(i, j) \leq U_i\}$.

PC is an NP-hard problem, under $P \neq NP$, PC cannot be solved in polynomial time [36]. Therefore, we measure the quality of p-center problems' solutions by approximation ratio and relative error.

Definition 1. *If the optimal solution to an optimization problem is c*. c is the best solution obtained by an approximate algorithm. Then the approximate ratio γ of the approximate algorithm is defined as:*

$$\gamma = \max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\} \tag{1}$$

$\gamma \leq r(n)$, where $r(n)$ is a function only related to the scale n of the problem.

Relative error λ is defined as follows:

$$\lambda = \left|\frac{c - c^*}{c^*}\right| \tag{2}$$

$\lambda \leq \eta(n)$, where $\eta(n)$ is the relative error bound which is only related to the scale n of the problem. According to the above definition, it is obviously $\eta(n) \leq r(n) - 1$.

Obviously, the smaller the approximation ratio or relative error is, the performance of the approximation algorithm is better.

3.2. Two Algorithms for Solving p-Center Problems

There are three algorithms for solving PC problems: exact algorithm (EA), approximate algorithm, and a heuristic algorithm. Exact algorithms are usually based on linear programming or mixed-integer programming [37–39], and the optimal solution can be obtained for most problems. We use an exact algorithm based on minimum domain sets (MDS) to solve PC instances in our work. EA can get the optimal solution to the problem, which can be used as a reference to measure the quality of the algorithm. Approximation algorithms include the SH algorithm [36], Gon algorithm [39], and HS algorithm [40]. These algorithms have proved to have the best approximation factor in theory, but they perform poorly on many benchmark datasets. Although the heuristic algorithm has no strict theoretical proof and cannot guarantee rapid convergence, it has performed well on many benchmark datasets [41].

In order to quickly analyze the effectiveness of our method, we choose a greedy approximation algorithm, which is proven to have a 2-approximation ratio in theory [34]. It is a fast heuristic algorithm, which is the easiest to implement. Next, we will mainly show two algorithms for solving PC problems.

Exact Algorithm. Our exact algorithm is based on the minimum dominating set in graph theory. This section will give the mixed linear programming form of MDS and some definitions related to it. We first give the definition of the minimum dominating set [42].

Definition 2. *Given an input graph G = (V, E), a dominating set is a subset $D \in V$ such that for every vertex $v \in V$, an edge $(v, u) \in E$ with either u or v in D exists.*

We show a dominating set in Figure 2 for understanding the definition. Figure 2 shows the domain set of the graph. $\{v0, v2, v4\}$ is one of the domain sets of the graph. According to the definition of a domain set, $\{v1, v3, v5\}$ is also a domain set of the graph.

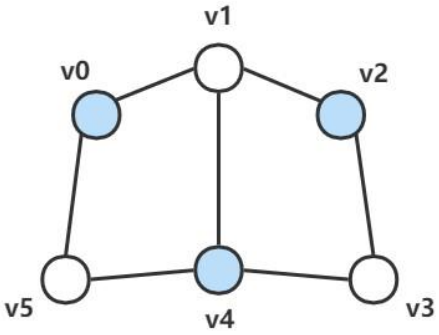


Figure 2. $\{v_0, v_2, v_4\}$ is a domain set of the graph.

Definition 3. *A minimum dominating set is a set of minimum cardinality among all the dominating sets.*

Figure 3 shows a minimum dominating set. Apart from the shown in Figure 3, $\{v2, v5\}$ is also a minimum dominating set of the graph. It is obvious that the MDS is not unique and must be a subset of domain sets.

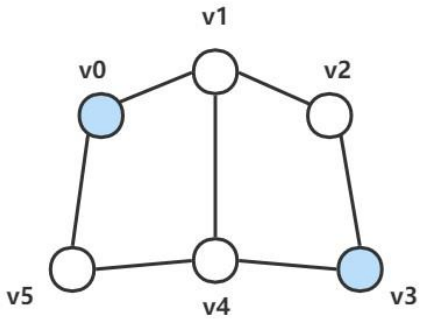


Figure 3. $\{v_0, v_3\}$ is a minimum domain set of the graph.

The minimum dominating set is a sub-problem of the p-center problem, so the solution of the p-center problem can be converted to some MDS problems. However, MDS requires that the graph data is a complete graph. This procedure can get the exact solution but is time-consuming and low efficiency. The solution of MDS is to convert the MDS problem into a mixed linear programming problem [43] and solve it with an open LP solver. The following is the mixed linear programming formulation of MDS:

$$\begin{aligned} &Minimize: && \sum_{v \in G} b_v \\ &s.t. && \forall v \in G, b_v + \sum_{(u,v) \in edges} b_u \geq 1 \\ & && \forall x \in G, b_x \text{ is a binary variable} \end{aligned} \tag{3}$$

Based on the MDS algorithm, the basic exact algorithm for solving PC is shown in Algorithm 1.

Algorithm 1 An Exact Algorithm for the PC problem (EA)

Input:
An undirected graph $G = (V, E)$, a integer p
Output:
Centers set C, distance D

According to G generate a complete graph G_C;
Get an order list of n edge weights of G_C, $w(e_1), w(e_2), \ldots, w(e_n)$, where $w(e_i) \leq w(e_{i+1}), i = 1,2,...,n\text{-}1$;
Let high = n, low = 1;
While high - low > 1 **do**
    mid = ⌈(high + low)/2⌉;
    $r_{max} = w(e_{mid})$;
    for i =1 to n **do**
        if $w(e_i) > r_{max}$ **then**
            Remove e_i;
    Get the bottleneck graph G_r;
    C = minimumDominatingSet(G_r)
    if $|C| \leq p$ **then**
        high = mid;
    else
    low = mid;
Return C, r_{max}.

Lemma 1 [22]. *The time complexity of the exact algorithm is* $O(n^2 logn)$.

Greedy Algorithm. The idea of the greed strategy is to select the current optimal solution at each step. This greed algorithm ensures that it has no more than a 2-approval approximation ratio. The detail of the algorithm is shown in Algorithm 2.

Algorithm 2 Greedy Algorithm for the PC problem (GA)

Input:

An undirected graph $G = (V, E)$, a integer p

Output:

Centers set C, distance D

Get the n vertices of G;
Random generate a starting index from $\|n\|$ and put in C;
While $|C| < p$ **do**
  for i =1 to n **do**
    if $v_i \in C$ **then**
      Continue.
    Calculator the distance $d(v_i, C)$
    $d_{min} = \min_{c \in C} d(v_i, C)$
    if $d_{min} > d_{max}$ **then**
      $d_{max} = d_{min}, c_{best} = v_i$
    Append d_{max} *to deltas*
    Append c_{best} *to* C
Return C, $min(deltas)$.

Lemma 2 [34]. *The time complexity of the greedy algorithm is $O(pn)$, where p is the number of centers, n is the number of nodes.*

3.3. Graph Neural Network

A convolutional neural network makes all the difference in deep learning, but it cannot handle graph structure data. Graph convolutional network (GCN) was first proposed by Kipf and Welling [44] for a semi-supervised classification, which can effectively process graph structure data. GCN is a spectral-based graph convolution model. We usually use G = (V, E) to represent a graph, where V is the set of nodes in the graph, |V| is the number of nodes, and E is the set of edges in the graph.

The convolution of a pixel in the image is to sum the weight of the pixel and adjacent pixels, so the convolution of a node in the graph structure can be shown as the weighted sum of the node and adjacent nodes.

We first consider the simplest convolution operation.

$$X^{(l+1)} = \sigma(AX^{(l)}\Theta^{(l)}) \tag{4}$$

In the above formula, Θ is the parameter of convolution transformation, which needs training and optimization. A represents the adjacency matrix of the graph. $A_{ij} \neq 0$ represent the node i and node j are adjacent. AX is to add the vectors of all neighbor nodes. However, the formula only obtains the information of neighbor nodes and ignores its information. Therefore, A is improved by adding a unit matrix. The formula is as follows:

$$X^{(l+1)} = \sigma(\widetilde{A}X^{(l)}\Theta^{(l)}) \tag{5}$$

$$\widetilde{A} = A + I_N \tag{6}$$

In the process of calculation, Formula (5) will add all neighbor vectors of the node. After multi-layer convolution, the value of the vector is extraordinarily large. Therefore, matrix A needs to be normalized. D denotes the degree matrix of the graph and the degree indicates the number of neighbors of each node. The normalization of A can be achieved by multiplying the inverse of the degree matrix D^{-1}. To ensure that the normalized matrix is still symmetric, the symmetric normalization formula is as follows:

$$A = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{7}$$

After the above derivation, the core formula of GCN is as follows:

$$X^{(l+1)} = \sigma(\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}X^{(l)}\Theta^{(l)}) \tag{8}$$

where $\widetilde{D}$ is the degree of $\widetilde{A}$, $X \in R^{N\times C}$, $\Theta \in R^{C\times F}$. N, C, F represent the number of nodes, the number of channels, and the number of convolution kernels, respectively.

The computational complexity of GCN is O(|E|). It has a linear relationship with the number of edges E. When the graph is sparse, the complexity is much lower than $O(n^2)$. The algorithm only considers the first-order information of the neighborhood. Stacking multiple layers can effectively increase the receptive field. In the experiment, it is not that the more layers of GCN we use, the better the training effect. We only use two layers of GCN to solve our problems.

The superiority of GCN:

- GCN can extract features from graph data. It can perform node classification, graph classification, edge prediction, and graph embedding on the graph data, which traditional CNN cannot process.
- The computational complexity of GCN is low. In our problem, the training speed is fast. Compared with graph attention networks, Graphsage, and other graph neural networks, the solution is faster and the model is more effective.

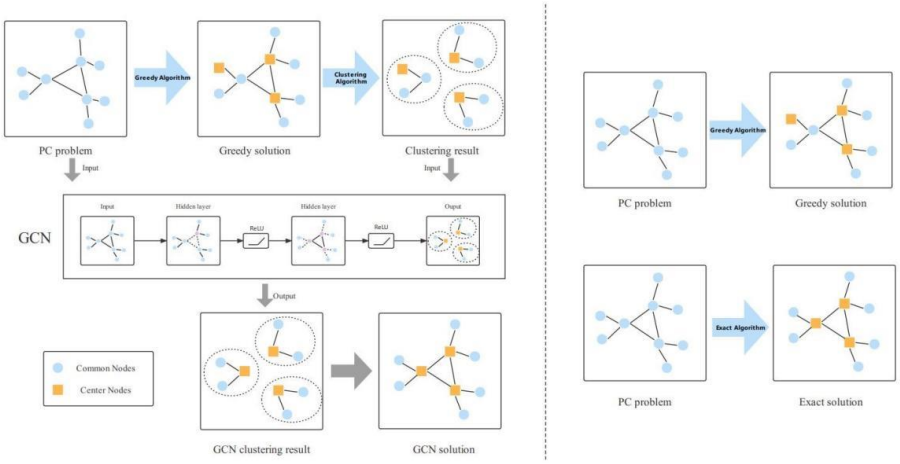
4. Methodology

As deep learning gives a good performance on many tasks, many researchers try to use deep learning to solve combinatorial optimization problems. In previous studies, the main idea must be based on a sufficiently large training set to get a robustness model which can effectively deal with the problems in test sets. Therefore, even if the training time is very long, it can effectively solve many problems as long as the trained model is well enough. Our work presents a new idea that combines the traditional algorithms with GCN to solve PC problems. The main workflow of our approach is shown in Figure 4.

Our method does not need to rely on large datasets. We train the model directly. The idea is straightforward, but it can effectively improve the solution of PC problems and can be extended to analogous COPs.

4.1. Solution Method

For the graph G = (V, E) corresponding to any PC problems, we use two basic algorithms to seek the solution to the problems. The detail of the algorithms is in Section 3. In our approach, we first use a greedy algorithm to work out the center points in the graph. Then a clustering algorithm is proposed to calculate the category of each node which is according to the center points. Next, G, features, and the clustering results are input parameters into GCN to obtain new clusters. Finally, based on the guidance of new clusters, each category's center point and maximum distance are computed, and the solution to the original problem is obtained.



(**a**) (**b**)

Figure 4. (**a**) The process of our approach to solving PC problems. (**b**) The greedy algorithm and exact algorithm.

We most notably train the GCN directly rather than the GCN model in advance. Therefore, this part of the training time needs to be calculated into the solution of the whole problem.

Cluster Algorithm. We need a clustering algorithm to calculate the category of each node which is according to the center point. We only need to cycle all vertices once, calculate the distance from each node to each center point and take the nearest center point as the label of the current vertex. The algorithm is shown in Algorithm 3. It is obvious that the time complexity of our cluster algorithm is $O(pn)$, p is the number of centers, n is the number of nodes.

Algorithm 3 Clusters Algorithm (CA)

Input:
An undirected graph $G = (V, E)$, a list $c = \{c_j\}_{j=1}^{p}$
Output:
$C = \{C_k\}_{k=1}^{p}$

```
for center = 1 to p do
    Clusters = {center: []}
for i =1 to n do
    if v_i ∈ c then
        continue.
    for j =1 to p do
        d_j = d(v_i, c_j)
    k = arg min_j d_j
    Append v_i to C_k
Return C.
```

4.2. Feasibility Analysis of Algorithm

The exact algorithm can give the optimal solution to the problem, but the efficiency is low. The greedy algorithm has high efficiency, but the accuracy of the solution is not enough. We propose a simple and effective method to solve the p-center problem. The solution efficiency is better than the accurate algorithm, the solution accuracy is better than the greedy algorithm, and a balance is achieved between efficiency and accuracy.

Lemma 3. *The time complexity of our approach is* $O(pn) + O(|E|)$*, where p is the number of centers, n is the number of nodes,* $|E|$ *is the number of edges.*

Proof of Lemma 3. Our algorithm is based on a greedy algorithm and a clustering algorithm. In Lemma 2, the time complexity of the greedy algorithm is $O(pn)$. According to Algorithm 3, the time complexity of the clustering algorithm is also $O(pn)$. GCN has $O(|E|)$ computational complexity in one layer. Our model has two layers of GCN (Algorithm 4). The time complexity of our algorithm is $O(pn) + O(|E|)$.□

Algorithm 4 GCN with Greedy Algorithm

Input:

An undirected graph $G = (V, E)$, a integer p, features f

Output:

Centers set c, distance D

```
c1 = GreedyAlgorithm(G, p)
C1 = ClusteringAlgorithm(G, c1)
C = GCN(G, f, C1)
for i=1 to p do
    Calculate the center c_i in C_i
    d_i = d(c_i, C_i)
D = max_i d_i
Return c, D.
```

5. Experiments

The GCN can effectively process graph structure data since the backup of solid theoretical results. However, the training of neural networks is a black-box model. Many phenomena in training have no sufficient theoretical guarantees. Following Lemmas 1 and 2, it is evident that the time complexity of our algorithm is superior to EA. In this section, we used many experiments to verify that the solution accuracy of our approach was better than GA.

5.1. Implementation

Data Generation. We generated many Erdos–Renyi (ER) graphs as instances for the p-center problems. ER graphs can represent a lot of practical problems in reality. Since PC is an NP-hard problem and the solution time increases rapidly with the scale of the problem, we first compared the running time of the two algorithms in Section 3 under different scale problems. Figure 5 is a curve line chart of the running time of the two algorithms varying with the number of nodes. Considering the time cost, we mainly conducted experiments on instances with several nodes ranging from 100 to 200. One hundred instances were generated using the ER random graph for each type of problem and solved by three methods, respectively. The solution time and accuracy of the three algorithms were compared. The final results were expressed by the mean and standard deviation of 100 groups of experiments.

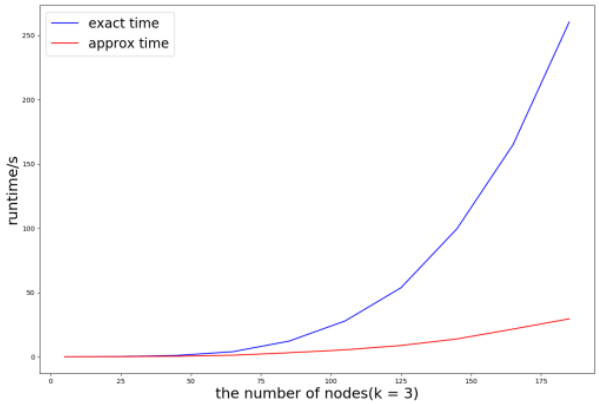


Figure 5. Exact vs greedy algorithm runtime comparisons (prob = 0.4).

Algorithm Procedure. We first used two basic algorithms to solve each problem and then combined GCN and GA to find the problem's solution. The procedure of the experimental is as follows:

(1) Generated a PC instance with ER random graph;
(2) Solved the instance with EA;
(3) Solved the instance with GA;
(4) According to the solution of GA, each node was classified by CA;
(5) Using GCN to get the new clusters;
(6) New clusters obtained the solution to the original problem.

5.2. Results on p-Center Problems

The approximation ratio is positively correlated with the relative error. Both are used to measure the proximity between the approximate solution and the optimal solution. The smaller the approximation ratio is, the closer the approximate solution is to the optimal solution, indicating that the quality of the solution is better. For the cases of $p = 3$ and $p = 5$, we considered three cases of n = 100, 150, and 200, respectively. Table 1 shows the result of $p = 3$, when n = 100, 150, and 200. It can be seen that our results have achieved lower solution time than EA and better solution accuracy than GA for three scale problems. Table 2 shows the result of $p = 5$, when n = 100, 150, and 200. It can be seen that our results have achieved lower solution time than EA and better solution accuracy than GA expected for n = 100, 150, and 200.

Table 1. The results on $p = 3$, when n = 100, 150, and 200.

n = 100, $p = 3$			
Model	**Runtime**	**Appro_ratio**	**Re_error**
GA	1.35 ± 0.09	1.51	0.51
EA	22.64 ± 0.84	1	0
GCN (ours)	10.56 ± 2.32	1.36	0.36
n = 150, $p = 3$			
Model	**Runtime**	**Appro_ratio**	**Re_error**
GA	4.55 ± 0.31	1.45	0.45
EA	114.24 ± 2.81	1	0
GCN (ours)	41.68 ± 1.33	1.25	0.25
n = 200, $p = 3$			
Model	**Runtime**	**Appro_ratio**	**Re_error**
GA	10.81 ± 0.87	1.44	0.44
EA	355.86 ± 7.96	1	0
GCN (ours)	46.69 ± 2.30	1.27	0.27

Table 2. The results on p = 5, when n = 100, 150, and 200.

n = 100, p = 5			
Model	**Runtime**	**Appro_ratio**	**Re_error**
GA	4.66 ± 0.22	1.56	0.56
EA	22.83 ± 0.77	1	0
GCN (ours)	11.26 ± 0.54	1.67	0.67
n = 150, p = 5			
Model	**Runtime**	**Appro_ratio**	**Re_error**
GA	16.08 ± 0.69	1.57	0.57
EA	114.27 ± 2.67	1	0
GCN (ours)	53.32 ± 1.92	1.52	0.51
n = 200, p = 5			
Model	**Runtime**	**Appro_ratio**	**Re_error**
GA	37.86 ± 1.81	1.54	0.54
EA	358.61 ± 7.95	1	0
GCN (ours)	114.47 ± 3.04	1.46	0.46

Result Analysis. Our method combines a heuristic algorithm and GCN, which realizes that the solution efficiency is better than the accurate algorithm on a scale of 100–200, and its accuracy is superior to the greedy algorithm. We expected that a general framework suitable for a specific form of a particular problem could be obtained from training in previous related work. However, learning a general model is a great challenge because of the particularity of graph data. The fundamental reason is that there are no consistent characteristics of different graph structure data of the same problem which cannot be trained well. For example, two ER graphs with 100 nodes were randomly generated and recorded as graphs A and B. Since nodes were randomly generated, there may be a node c, which exists in both A and B and belongs to different categories, so it is not feasible to classify node c with GCN.

Therefore, we proposed a new method—training directly. We do not need to generate a unified model but only need to train their own GCN model for each graph. We used GCN to learn the classification results directly and then computed the center point and the minimum distance according to the classification results. Experiments showed that the training time of the GCN network increases very slowly with the increase of the number of nodes. Only 20–30 s after 10,000 iterations in GCN is relatively small compared with the solution time of the exact algorithm in large-scale problems. It indicates that training directly is feasible and can effectively guide PC problems.

6. Conclusions

In summary, we first implement two basic algorithms: an exact algorithm based on a minimum domain set and a greedy approximate algorithm. Then, we propose a new method to solve p-center problems combined with a graph convolutional network. Our method achieves that the solution accuracy is better than the greedy algorithm and the solution efficiency is superior to the exact algorithm. The time complexity of our algorithm is analyzed theoretically, and a better solution, which is compared with the greedy algorithm, is achieved on a large number of PC instances generated by ER graph. GCN is used as a classifier for specific instances and the output is directly used to solve the origin problems, which is different from previous works. Experiments show that our approach is practical and can better deal with PC problems.

A trade-off approach is proposed to deal with PC problems in our study. Although the experiments are restricted by the solution time, the results also show that our approach is effective. This study proposed our algorithm based on an exact algorithm and a greedy algorithm. We achieved a trade-off between the solution time and the quality of the solution.

Our study can easily combine with other heuristic algorithms, such as simulated annealing, ant colony algorithm, and genetic algorithm. Our idea can extend to spatial optimization problems. However, the specific solution details need to be further explored.

In future work, we will continue to study in following three aspects:

- Combine with other approximate or heuristic algorithms: in our study, we only rely on two basic algorithms to assist the solving, which can be replaced with other algorithms.
- Experiment on realistic datasets: Our study use an ER graph to simulate a real-world dataset. It is necessary and significant to apply the approach to real data.
- Transform to other homologous problems: The paradigm can be transferred to other COPs. We will explore the effectiveness of our approach on p-median problems, location set covering problems, and maximal coverage location problems.

Author Contributions: Conceptualization, Shaohua Wang and Haojian Liang; methodology, Shaohua Wang and Haojian Liang; software, Haojian Liang; validation, Haojian Liang, Huilai Li and Yang Zhong.; formal analysis, Haojian Liang, Huilai Li and Huichun Ye; investigation, Huilai Li and Yang Zhong; resources, Yang Zhong; data curation, Haojian Liang; writing—original draft preparation, Haojian Liang and Shaohua Wang; writing—review and editing, Haojian Liang, Shaohua Wang, Huilai Li, Huichun Ye and Yang Zhong; visualization, Haojian Liang; supervision, Shaohua Wang; project administration, Shaohua Wang; funding acquisition, Huichun Ye and Shaohua Wang. All authors have read and agreed to the published version of the manuscript.

Funding: Supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDA28100500), the Hundred Talents Program Youth Project (Category B) of the Chinese Academy of Sciences (E2Z10501), and the Youth Innovation Promotion Association CAS (2021119).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Zhang, M.; Le, L.; Fang, J.; Ai, X.; Yao, W.; Wen, J. Stochastic unit commitment with air conditioning loads participating in reserve service. *IET Renew. Power Gener.* **2019**, *13*, 2977–2985. [CrossRef]
2. Liu, K.; Gao, S.; Qiu, P.; Liu, X.; Yan, B.; Lu, F. Road2vec: Measuring traffic interactions in urban road system from massive travel routes. *ISPRS Int. J. Geo-Inf.* **2017**, *6*, 321. [CrossRef]
3. Zhang, Y.; Cheng, T.; Ren, Y.; Xie, K. A novel residual graph convolution deep learning model for short-term network-based traffic forecasting. *Int. J. Geogr. Inf. Sci.* **2020**, *34*, 969–995. [CrossRef]
4. Sun, C.-H.; Cheng, C.-Y.; Wang, C.-H.; Hsiao, P.-H. Dynamic floating stations model for emergency medical services with a consideration of traffic data. *ISPRS Int. J. Geo-Inf.* **2020**, *9*, 336. [CrossRef]
5. Gui, Z.; Sun, Y.; Yang, L.; Peng, D.; Li, F.; Wu, H.; Guo, C.; Guo, W.; Gong, J. Lsi-lstm: An attention-aware lstm for real-time driving destination prediction by considering location semantics and location importance of trajectory points. *Neurocomputing* **2021**, *440*, 72–88. [CrossRef]
6. Wang, S.; Gao, S.; Feng, X.; Murray, A.T.; Zeng, Y. A context-based geoprocessing framework for optimizing meetup location of multiple moving objects along road networks. *Int. J. Geogr. Inf. Sci.* **2018**, *32*, 1368–1390. [CrossRef]
7. Zhou, L.; Wang, S.; Xu, Z. A multi-factor spatial optimization approach for emergency medical facilities in beijing. *ISPRS Int. J. Geo-Inf.* **2020**, *9*, 361. [CrossRef]
8. Zhu, Y.; Du, Q.; Tian, F.; Ren, F.; Liang, S.; Chen, Y. Location optimization using a hierarchical location-allocation model for trauma centers in shenzhen, china. *ISPRS Int. J. Geo-Inf.* **2016**, *5*, 190. [CrossRef]
9. Han, B.; Hu, M.; Zheng, J.; Tang, T. Site selection of fire stations in large cities based on actual spatiotemporal demands: A case study of nanjing city. *ISPRS Int. J. Geo-Inf.* **2021**, *10*, 542. [CrossRef]
10. Gan, W.; Ai, X.; Fang, J.; Yan, M.; Yao, W.; Zuo, W.; Wen, J. Security constrained co-planning of transmission expansion and energy storage. *Appl. Energy* **2019**, *239*, 383–394. [CrossRef]
11. Gao, P.; Wang, H.; Cushman, S.A.; Cheng, C.; Song, C.; Ye, S. Sustainable land-use optimization using nsga-ii: Theoretical and experimental comparisons of improved algorithms. *Landsc. Ecol.* **2021**, *36*, 1877–1892. [CrossRef]

12. Church, R.L.; Wang, S. Solving the p-median problem on regular and lattice networks. *Comput. Oper. Res.* **2020**, *123*, 105057. [CrossRef]
13. Feng, X.; Wang, S.; Murray, A.T.; Cao, Y.; Gao, S. Multi-objective trajectory optimization in planning for sequential activities across space and through time. *Environ. Plan. B Urban Anal. City Sci.* **2021**, *48*, 945–963. [CrossRef]
14. Liu, X.; Li, P.; Meng, F.; Zhou, H.; Zhong, H.; Zhou, J.; Mou, L.; Song, S. Simulated annealing for optimization of graphs and sequences. *Neurocomputing* **2021**, *465*, 310–324. [CrossRef]
15. Ding, Q.; Hu, X.; Sun, L.; Wang, Y. An improved ant colony optimization and its application to vehicle routing problem with time windows. *Neurocomputing* **2012**, *98*, 101–107. [CrossRef]
16. Bello, I.; Pham, H.; Le, Q.V.; Norouzi, M.; Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv* **2016**, arXiv:1611.09940.
17. Khalil, E.; Dai, H.; Zhang, Y.; Dilkina, B.; Song, L. Learning combinatorial optimization algorithms over graphs. *Adv. Neural Inf. Processing Syst.* **2017**, *30*, 6351–6361.
18. Donti, P.; Amos, B.; Kolter, J.Z. Task-based end-to-end model learning in stochastic optimization. *Adv. Neural Inf. Processing Syst.* **2017**, *30*, 5490–5500.
19. Wilder, B.; Dilkina, B.; Tambe, M. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; pp. 1658–1665.
20. Yang, L.; Cao, X.; He, D.; Wang, C.; Wang, X.; Zhang, W. Modularity Based Community Detection with Deep Learning. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, New York, NY, USA, 9–15 July 2016; pp. 2252–2258.
21. Amos, B.; Kolter, J.Z. Optnet: Differentiable optimization as a layer in neural networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 6–11 August 2017; pp. 136–145.
22. Robič, B.; Mihelič, J. Solving the k-center problem efficiently with a dominating set algorithm. *J. Comput. Inf. Technol.* **2005**, *13*, 225–234.
23. Rana, R.; Garg, D. The analytical study of k-center problem solving techniques. *Int. J. Inf. Technol. Knowl. Manag.* **2008**, *1*, 527–535.
24. Li, Z.; Chen, Q.; Koltun, V. Combinatorial optimization with graph convolutional networks and guided tree search. *Adv. Neural Inf. Processing Syst.* **2018**, *31*, 539.
25. Drori, I.; Kharkar, A.; Sickinger, W.R.; Kates, B.; Ma, Q.; Ge, S.; Dolev, E.; Dietrich, B.; Williamson, D.P.; Udell, M. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In Proceedings of the 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 14–17 December 2020; pp. 19–24.
26. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer networks. *Adv. Neural Inf. Processing Syst.* **2015**, *28*, 2692–2700.
27. Gu, S.; Hao, T.; Yao, H. A pointer network based deep learning algorithm for unconstrained binary quadratic programming problem. *Neurocomputing* **2020**, *390*, 1–11. [CrossRef]
28. Zhang, M.; Chen, Y. Link prediction based on graph neural networks. *Adv. Neural Inf. Processing Syst.* **2018**, *31*, 5171–5181.
29. Kool, W.; Van Hoof, H.; Welling, M. Attention, learn to solve routing problems! *arXiv* **2018**, arXiv:1803.08475.
30. Tian, F.; Gao, B.; Cui, Q.; Chen, E.; Liu, T.-Y. Learning deep representations for graph clustering. In Proceedings of the AAAI Conference on Artificial Intelligence, Québec, QC, Canada, 27–31 July 2014.
31. Xie, J.; Girshick, R.; Farhadi, A. Unsupervised deep embedding for clustering analysis. In Proceedings of the 33rd International Conference on International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; p. 487.
32. Guo, X.; Gao, L.; Liu, X.; Yin, J. Improved deep embedded clustering with local structure preservation. In Proceedings of the 26th International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017; pp. 1753–1759.
33. Wilder, B.; Ewing, E.; Dilkina, B.; Tambe, M. End to end learning and optimization on graphs. *Adv. Neural Inf. Processing Syst.* **2019**, *32*, 4674–4685.
34. Garcia-Diaz, J.; Menchaca-Mendez, R.; Menchaca-Mendez, R.; Hernández, S.P.; Pérez-Sansalvador, J.C.; Lakouari, N. Approximation algorithms for the vertex k-center problem: Survey and experimental evaluation. *IEEE Access* **2019**, *7*, 109228–109245. [CrossRef]
35. Chakrabarti, D.; Dickerson, J.P.; Esmaeili, S.A.; Srinivasan, A.; Tsepenekas, L. A new notion of individually fair clustering: $\alpha $-equitable $ k $-center. *arXiv* **2021**, arXiv:2106.05423.
36. Plesník, J. A heuristic for the p-center problems in graphs. *Discret. Appl. Math.* **1987**, *17*, 263–268. [CrossRef]
37. Daskin, M.S. A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results. *Commun. Oper. Res. Soc. Jpn.* **2000**, *45*, 428–436.
38. Özsoy, F.A.; Pınar, M.Ç. An exact algorithm for the capacitated vertex p-center problem. *Comput. Oper. Res.* **2006**, *33*, 1420–1436. [CrossRef]
39. Dyer, M.E.; Frieze, A.M. A simple heuristic for the p-centre problem. *Oper. Res. Lett.* **1985**, *3*, 285–288. [CrossRef]
40. Hochbaum, D.S.; Shmoys, D.B. A best possible heuristic for the k-center problem. *Math. Oper. Res.* **1985**, *10*, 180–184. [CrossRef]
41. Mladenović, N.; Labbé, M.; Hansen, P. Solving the p-center problem with tabu search and variable neighborhood search. *Netw. Int. J.* **2003**, *42*, 48–64. [CrossRef]
42. Grandoni, F. A note on the complexity of minimum dominating set. *J. Discret. Algorithms* **2006**, *4*, 209–214. [CrossRef]

43. Fan, N.; Watson, J.-P. Solving the connected dominating set problem and power dominating set problem by integer programming. In *International Conference on Combinatorial Optimization and Applications*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 371–383.
44. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv* **2016**, arXiv:1609.02907.