mathematics

MDPI

Article

A Graph-Induced Neighborhood Search Heuristic for the Capacitated Multicommodity Network Design Problem

Houshan Zhang

School of Science, Beijing University of Posts and Telecommunications, Beijing 100876, China; houshan_zhang@bupt.edu.cn

Abstract: In this work, an efficient graph-induced neighborhood search heuristic is proposed to address the capacitated multicommodity network design problem. This problem, which commonly arises in transportation and telecommunication, is well known for its inherent complexity and is often classified as $\mathcal{NP}$-hard. Our approach commences with an arbitrary feasible solution and iteratively improves it by solving a series of small-scale auxiliary mixed-integer programming problems. These small-scale problems are closely tied to the cycles inherent in the network topology, enabling us to reroute the flow more effectively. Furthermore, we have developed a novel resource-efficient facility assignment technique that departs from standard variable neighborhood search algorithms. By solving a series of small knapsack problems, this technique not only enhances the quality of solutions further but also can serve as a primary heuristic to generate initial feasible solutions. Furthermore, we theoretically guarantee that our algorithm will always produce an integer-feasible solution within polynomial time. The experimental results highlight the superior performance of our method compared to other existing approaches. Our heuristic algorithm efficiently discovers high-quality feasible solutions, substantially reducing the computation time and number of nodes in the branch-and-bound tree.

Keywords: capacitated multicommodity network design; large neighborhood search; mixed-integer programming

MSC: 90C59

check for updates

Academic Editor: Andrea Scozzari

Received: 7 January 2025
Revised: 8 February 2025
Accepted: 10 February 2025
Published: 11 February 2025

Citation: Zhang, H. A Graph-Induced Neighborhood Search Heuristic for the Capacitated Multicommodity Network Design Problem. *Mathematics* **2025**, *13*, 588. https://doi.org/10.3390/math13040588

Copyright: © 2025 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The *capacitated multicommodity network design* (CMND) problem is a fundamental optimization challenge that plays a critical role in various network applications, particularly within supply chain management [1], logistics planning [2,3], transportation [4], and telecommunications [5–7]. In these scenarios, networks are constructed by linking nodes, and limited resources are allocated to deliver products across these nodes [6–11]. The transmission capacities on each link can be the sum of integer multiples of differently sized capacity facilities. The term "facility" refers to the capacity base unit, which can be installed on integer multiples. This problem focuses on optimizing traffic routing to minimize overall cost, which includes the costs of traffic flow and capacity installation. One of the most prominent applications of CMND is in telecommunication networks, where it plays a vital role in ensuring efficient data transmission and bandwidth allocation. For instance, telecommunication providers must design robust and scalable fiber-optic networks to meet the rapidly growing demand for high-speed Internet and data services. By effectively addressing the CMND problem, these providers can reduce network expansion costs

while maintaining service quality and minimizing latency. These applications underscore the critical contributions of CMND to advancing modern communication systems and addressing the needs of a digitally connected world.

1.1. Problem Classification

For each CMND problem instance, the type of facility used determines whether the problem is categorized as a single-facility or multi-facility version. In the single-facility version, only one type of facility is installed on each link in the network [12,13]. Practical examples include routing vehicles with uniform capacity within a logistics network or deploying a communication network utilizing only one cable type. Conversely, the multi-facility problem allows for the installation of diverse facility types with varying capacities. Each facility is associated with a fixed-charge cost, and its unit capacity cost differs, typically decreasing with larger capacity sizes [10]. The optimization challenge involves determining the optimal quantity and the combination of facilities to install across the network. These facilities must meet the demand for each commodity. Compared with the single-facility CMND problem, the multi-facility version is more challenging.

In addition, in network design problem formulations, capacity constraints are tailored to fit three distinct application scenarios: *directed* [8,13,14], *bidirected* [15,16], and *undirected* [6,7]. In the directed link model, the flow on each arc is limited by its installed capacity. In the bidirected link model, identical capacities are installed on both the arc and its corresponding reverse arc. Conversely, in the undirected link model, the total flow on an arc and its reverse counterpart is at most the capacity of the undirected edge that they collectively represent. We will refer to the *mixed-integer programming* (MIP) formulations based on these respective link models as the directed, bidirected, and undirected models [9].

1.2. Research Objectives and Contributions

Most existing heuristic methods are constrained by the specific facility and link types, limiting their adaptability to different CMND formulations. However, due to the complexity of real-world problems, highly customized approaches are often too restrictive. Additionally, in the field of network design, models with similar names often differ significantly, which makes it difficult to find a broadly applicable method. Therefore, it is essential to design an effective heuristic method for general CMND problems, regardless of link and facility types. In this paper, we introduce a novel *graph-induced neighborhood search* (GINS) heuristic method for solving the general capacitated multicommodity network design problem. To the best of our knowledge, this is the first heuristic method broadly applicable to all three link types of CMND models with any number of facilities.

Compared to other available methods, the GINS algorithm incorporates two key innovations in a theoretical context:

1. **Graph-Induced Custom Heuristic:** Unlike other methods that use mathematical programming techniques, which often overlook or fail to utilize the underlying topology of the graph, GINS adopts a graph-induced custom heuristic to strategically identify and extract this underlying topology. Specifically, the network is divided into cycles centered around a specific *central link*. These cycles are used to construct a subnetwork, with facilities on links outside this subnetwork fixed. Within this subnetwork, an auxiliary problem is iteratively constructed and solved, continuously enhancing solution quality until no further improvement can be achieved. This process significantly improves computational efficiency.
2. **Resource-Efficient Facility Assignment Technique:** Traditional heuristic methods often result in resource wastage during flow rerouting. To overcome this limitation, GINS incorporates a *resource-efficient facility assignment* (REFA) technique to leverage

the distinctive characteristics of the CMND problem. By fixing all traffic flows, the original problem is transformed into multiple small-scale independent knapsack problems, which can be solved efficiently using a dynamic programming approach. This transformation enhances solution quality and enables the rapid generation of high-quality initial integer-feasible solutions.

To demonstrate the effectiveness and efficiency of our proposed heuristic, we have implemented GINS in two ways:

1. **Integration with CPLEX:** GINS was embedded as a callback routine within the widely used MIP solver CPLEX. Using this implementation, we demonstrate that the proposed GINS heuristic provides a more effective acceleration of the branch-and-bound algorithm compared to the built-in heuristics of CPLEX. This improvement is reflected in the reduced computational time and the number of explored nodes within the branch-and-bound tree, underscoring its capability to enhance the overall efficiency of hybrid solution frameworks.
2. **Standalone Implementation:** GINS was implemented independently without relying on the branch-and-bound framework of the MIP solver CPLEX. This approach provides strong empirical evidence that GINS consistently outperforms other existing heuristics in solving the CMND problem. Specifically, it achieves higher-quality solutions while requiring less computational time. These results validate the practical advantages and superior performance of the GINS algorithm compared to traditional heuristic methods.

In summary, our method demonstrates superiority over existing approaches in two aspects. Theoretically, the GINS method fully exploits network topology to enhance computational efficiency and adopts a resource-efficient facility assignment strategy, addressing the limitations of existing heuristics. Practically, its integration with CPLEX effectively optimizes hybrid frameworks. At the same time, its standalone implementation showcases exceptional performance in solving CMND problems, establishing its robustness and efficiency in both theoretical and practical contexts.

1.3. Organization

The rest of this article is organized as follows. The state of the art and relevant studies are listed in Section 2. In Section 3, we describe the mathematical models of the CMND problem. Then, our heuristic and the related detailed information on implementation are presented in Section 4. Next, Section 5 presents the computational results. Finally, we summarize our conclusions in Section 6.

2. Literature Review

Many studies in solving the CMND problem concentrate on developing exact approaches to find optimal solutions, including deriving tight lower bounds or conducting polyhedra analysis. Magnanti et al. [17] first introduced the residual capacity inequality, and Atamtürk and Rajan [18] gave the polynomial time separation procedure for residual capacity inequalities. Brockmüller et al. [19] introduced the c-strong inequality, which was further studied by Atamtürk and Rajan [6] and van Hoesel et al. [20]. The review of cut-based inequalities for network design can be found in Raack et al. [21]. In this literature, some cutting planes only apply to a specific link model. To address this problem, Atamtürk and Günlük [9] studied the connection of valid inequalities between the three types of model links. We refer to [10] for more details about exact approaches.

The CMND problem has been extensively studied using heuristic methods. Various heuristics, such as evolutionary algorithms and large neighborhood search methods, have been developed to obtain high-quality solutions within reasonable computation times.

Before delving into customized heuristics for CMND problems, it is helpful to introduce some general heuristics that are useful for solving general MIP problems. Many algorithms explicitly tailored for CMND are adaptations of these general techniques. For instance, the *local branching* heuristic [22] is effective for binary integer programming problems. It identifies and fixes certain binary variables (which correspond to the activation or deactivation of facilities in the CMND problem), dividing the original problem into a subproblem and a main problem. High-quality solutions can often be obtained by solving the subproblem; otherwise, a cutting plane is added to the main problem, repeating this way until a solution is obtained. Another notable method is the *relaxation-induced neighborhood search* (RINS) method [23], which applies to general integer programming problems. This heuristic fixes certain variables based on linear relaxation and integer-feasible solutions. It then solves an auxiliary MIP subproblem and adjusts the remaining variables to find a better solution.

However, the algorithms mentioned above are general heuristics and only sometimes effective for CMND problems. In order to solve CMND more effectively, a customized heuristic algorithm is necessary. A well-researched area is the single-facility CMND problem, particularly the fixed-charge version, where only one facility can be installed on each link. For this problem, the method of improving solution quality locally by adjusting flow along a cycle was proposed early. It can be combined with tabu search [24,25] or an evolutionary algorithm [26]. In reference [12], Hewitt et al. proposed solving MIPs to search neighborhoods of a solution. Recently, the combination of slope scaling with MIP neighborhood search [13,14] or greedy algorithms [27] has been proposed as an effective approach for the fixed-charge CMND problem. Recent advancements in heuristic algorithms have further contributed to solving network optimization problems like CMND. Liu et al. [28] developed a two-stage hybrid algorithm combining genetic algorithms and depth-first search, effectively handling large-scale problems with complex constraints. Ahmed et al. [29] proposed an enhanced ant colony optimization algorithm that integrates subpath-based strategies and K-nearest neighbor methods, achieving strong performance in capacitated vehicle routing problems. Li et al. [30] introduced an improved particle swarm optimization algorithm with variable neighborhood search, enhancing local search capabilities for both continuous and discrete optimization problems.

Although the above methods have proven effective for solving the single-facility CMND problem or other network optimization problems, they cannot be directly applied to tackle the multi-facility version. The multi-facility CMND problem has a broader range of applications and poses a more significant challenge in terms of its solvability. However, the literature on heuristic approaches specifically designed to tackle the multi-facility CMND problem is scarce. In the context of the multi-facility CMND problem, Agarwal [6] proposed a neighborhood search technique for optimizing a specific class of undirected network design problems, focusing on local improvement strategies. This method iteratively evaluates subsets of links, termed subnetworks, to reroute traffic flow within the subnetwork, leaving the flow outside unaffected during each iteration. Additionally, in [7], Salman et al. introduced a novel branch-and-bound algorithm that can serve as a practical heuristic approach, providing upper bounds for the same undirected model. In reference [31], Rahmaniani and Ghaderi [31] proposed an evolutionary approach for the directed CMND problem with extra consideration for facility location. For a specific class of undirected CMND problems, Khelifi [32] introduced a hybrid approach combining greedy constructive heuristics with iterative local search. Chen et al. [33] reviewed uncertainty analysis and optimization modeling in supply chain management, highlighting robust and stochastic optimization for addressing uncertainties like demand fluctuations and supply disruptions. Recent highly remarkable advancements in heuristic and metaheuristic algorithms, particularly hybrid approaches that combine ant optimization with the whale optimization

algorithm [34,35] and Harris hawks optimization for multi-objective problems [36], have shown significant potential for solving complex optimization challenges. Although these methods have primarily been applied in areas such as task scheduling and workflow optimization, their adaptability suggests that they could also be effective in addressing CMND problems. However, their application in this specific context remains an open area for further investigation. For a more comprehensive review of heuristic methods for the CMND problem, we refer to [11,37]. Overall, we summarize the difference between our study and these articles in Table 1 regarding facility and model types. The symbols 'D', 'U', and 'B' in Table 1 represent directed, undirected, and bidirected link models, respectively.

Table 1. The related literature on heuristic algorithms for solving the CMND problem.

Reference	Research Focus	Facility Types		Link Types		
		Single	Multiple	D	U	B
Fischetti et al., 2003 [22]	Local branch	✓		✓	✓	✓
Danna et al., 2005 [23]	Neighborhood search	✓	✓	✓	✓	✓
Ghamlouche et al., 2003 [24]	Cycle-based tabu search	✓		✓		
Ghamlouche et al., 2004 [25]	Cycle-based tabu search, path relink	✓		✓		
Hewitt et al., 2010 [12]	Neighborhood search	✓		✓		
Paraskevopoulos et al., 2016 [26]	Cycle-based evolutionary algorithm	✓		✓		
Gendron et al., 2018 [14]	Slope scaling, local branch	✓		✓		
Katayama, 2019 [27]	Slope scaling, greedy algorithm	✓		✓		
Katayama, 2020 [13]	Slope scaling, local branch	✓		✓		
Agarwal, 2002 [6]	Local adjustment	✓	✓		✓	
Salman et al., 2008 [7]	Branch-and-bound	✓	✓		✓	
Rahmaniani et al., 2013 [31]	Facility location	✓	✓	✓		
Khelifi et al., 2015 [32]	Neighborhood search	✓	✓		✓	
This article	**Graph-induced neighborhood search**	✓	✓	✓	✓	✓

3. Mathematical Formulations

Before presenting the mathematical formulation of the CMND problem, Table 2 lists the key notations used throughout this paper for clarity and quick reference. The CMND problem can be defined on both directed and undirected networks, and can be formulated as three different MIPs, each MIP formulation corresponding to a unique link model [9,10]. The directed network design instance is given by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. For each node $v \in \mathcal{V}$, let $\delta^+(v)$ and $\delta^-(v)$ represent the arcs in $\mathcal{A}$ that have v as their source and target node, respectively. Let $\mathcal{Q}$ denote the commodities set, indexed by q. For each commodity $q \in \mathcal{Q}$, we associate a vector d^q to denote the demand of each node, such that $\sum_{v \in \mathcal{V}} d_v^q = 0$. The facility types set is $\mathcal{K}$, where a unit of facility $k \in \mathcal{K}$ provides capacity u_k. For each arc $a \in \mathcal{A}$, let h_a denote the unit flow

cost, c_a^k denote the fixed-charge cost of the installed capacity of facility k, and u_a indicate the pre-installed capacity. The directed CMND problem is formulated as follows [10]:

$$\min_{x,y} \sum_{a\in\mathcal{A}}\left(\sum_{q\in\mathcal{Q}} h_a^q x_a^q + \sum_{k\in\mathcal{K}} c_a^k y_a^k\right) \tag{1a}$$

$$s.t. \sum_{a\in\delta^+(v)} x_a^q - \sum_{a\in\delta^-(v)} x_a^q = d_v^q, \qquad \forall\, v\in\mathcal{V},\ q\in\mathcal{Q}, \tag{1b}$$

$$\sum_{q\in\mathcal{Q}} x_a^q \le u_a + \sum_{k\in\mathcal{K}} u_k y_a^k, \qquad \forall\, a\in\mathcal{A}, \tag{1c}$$

$$x_a^q \in \mathbb{R}^+, \qquad \forall\, a\in\mathcal{A},\ q\in\mathcal{Q}, \tag{1d}$$

$$y_a^k \in \mathbb{Z}^+, \qquad \forall\, a\in\mathcal{A},\ k\in\mathcal{K}. \tag{1e}$$

Table 2. Summary of notations and their descriptions for the mathematical models of capacitated multicommodity network design problem.

Notation	Description
$\mathcal{V}$	Set of nodes in the network.
$\mathcal{A}$	Set of arcs in the directed network.
$\mathcal{E}$	Set of edges in the undirected network.
$\delta^+(v)$	Set of arcs in $\mathcal{A}$ leaving node $v\in\mathcal{V}$.
$\delta^-(v)$	Set of arcs in $\mathcal{A}$ entering node $v\in\mathcal{V}$.
$\mathcal{Q}$	Set of commodities, indexed by q.
d^q	Demand vector for commodity $q\in\mathcal{Q}$.
d_v^q	Demand at node $v\in\mathcal{V}$ for commodity $q\in\mathcal{Q}$.
$\mathcal{K}$	Set of facility types, indexed by k.
u_k	Capacity provided by one unit of facility $k\in\mathcal{K}$.
h_a	Unit flow cost on arc $a\in\mathcal{A}$.
c_a^k	Fixed-charge cost of installing facility $k\in\mathcal{K}$ on arc $a\in\mathcal{A}$.
u_a	Pre-installed capacity on arc $a\in\mathcal{A}$.
c_e^k	Fixed-charge cost of installing facility $k\in\mathcal{K}$ on edge $e\in\mathcal{E}$.
u_e	Pre-installed capacity on edge $e\in\mathcal{E}$.
e^+, e^-	Anti-parallel directed arcs for edge $e\in\mathcal{E}$.

In the above MIP formulation, the flow variable x_a^q is the commodity $q\in\mathcal{Q}$ to be transferred along arc a and y_a^k is the number of facilities k installed on the arc. The objective function (1a) is to minimize the total cost, which is the sum of the flow costs of commodities plus the sum of the installation costs of facilities. Constraint (1b) consists of flow conservation equations, which indicate that the sum of the path flows of commodity q is equal to the demand. Constraint (1c) ensures that the capacity on arc a is sufficient to accommodate the flow on it. Constraints (1d) and (1e) define all the variables in the model.

The bidirected and undirected CMND problems are defined on an undirected graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of arcs. The facilities are installed on the edges of the undirected network. Let c_e^k denote the fixed-charge cost of facility k installed on edge $e\in\mathcal{E}$, and u_e indicate the pre-installed capacity. For each edge $e\in\mathcal{E}$, we introduce a pair of anti-parallel (directed) arcs e^+ and e^- to handle the flow on edges. In the bidirected and undirected cases, $\mathcal{A}$ is the set of all such arcs, i.e., $\mathcal{A}:=\{e^+=(i,j), e^-=(j,i)\ :\ e=\{i,j\}\in\mathcal{E}\}$.

The undirected CMND problem is formulated as follows [21]:

$$\min_{x,y} \sum_{a\in\mathcal{A}}\sum_{q\in\mathcal{Q}} h_a^q x_a^q + \sum_{e\in\mathcal{E}}\sum_{k\in\mathcal{K}} c_e^k y_e^k \tag{2a}$$

$$s.t. \sum_{a\in\delta^+(v)} x_a^q - \sum_{a\in\delta^-(v)} x_a^q = d_v^q, \qquad \forall\, v\in\mathcal{V},\ q\in\mathcal{Q}, \tag{2b}$$

$$\sum_{q\in\mathcal{Q}} x_{e^+}^q + \sum_{q\in\mathcal{Q}} x_{e^-}^q \le u_e + \sum_{k\in\mathcal{K}} u_k y_e^k, \qquad \forall\, e\in\mathcal{E}, \tag{2c}$$

$$x_a^q \in \mathbb{R}^+, \qquad \forall\, a\in\mathcal{A},\ q\in\mathcal{Q}, \tag{2d}$$

$$y_e^k \in \mathbb{Z}^+, \qquad \forall\, e\in\mathcal{E},\ k\in\mathcal{K}. \tag{2e}$$

In the bidirected link model, the flow in the two arcs is bounded by the corresponding edge capacity. Therefore, the bidirected model is formulated by (2a), (2b), (2d), (2e), and the following constraint [21]:

$$\max\left\{\sum_{q\in\mathcal{Q}} x_{e^+}^q, \sum_{q\in\mathcal{Q}} x_{e^-}^q\right\} \le u_e + \sum_{k\in\mathcal{K}} u_k y_e^k,\ \forall\, e\in\mathcal{E}. \tag{3}$$

We provide the following descriptions and examples to clarify the distinctions between directed, undirected, and bidirected link models. In a directed model, each link has a specific direction, represented by an arc (i,j). This signifies that flows can only move from node i to node j. In contrast, an undirected model allows for bidirectional flow along an edge $\{i,j\}$, which is directionless, representing the possibility of flow between nodes i and j. The bidirected model includes two directed arcs for every pair of nodes, such as (i,j) and (j,i). In this model, flow, capacity, and cost can be independently assigned to each direction. The bidirectional network design problem can be challenging to understand, so we illustrate it with an application in rail transit, particularly in metro systems. A typical example is the design of *double-track railways*, which consist of two parallel tracks. One track is designated for trains traveling in one direction while the other is for trains traveling in the opposite direction. Building only a single track would prevent vehicles from returning to their starting point, except in loop lines. Therefore, double tracks are generally constructed simultaneously to accommodate bidirectional travel. This configuration ensures that train flows in opposite directions remain independent and do not interfere with each other, thereby enhancing operational efficiency and safety.

To conclude this section, it is noteworthy that the bidirected CMND problem can be transformed into the directed version. In the transformed directed link model, the capacities on the forward and backward arcs between a pair of nodes are the same, i.e., constraints $y_{e^+}^k = y_{e^-}^k$, $\forall e\in\mathcal{E}$ should be added in model (1). Furthermore, in specific scenarios, the undirected CMND problem can be transformed into the bidirected version, where the commodity between a pair of nodes is the same. In other words, the amount of goods transported from node i to node j must equal the amount transported from node j to node i. See refs. [9,21] for more details on this relationship between different link models.

4. Graph-Induced Neighborhood Search Heuristic Method

For mixed-integer programming problems, primal heuristics are essential in finding feasible solutions. The high-quality solutions can significantly accelerate the problem-solving process and reduce the number of nodes in the branch-and-bound tree. Extensive research has been conducted on primal heuristics [13,14,22,23,32,38]. One particularly efficient heuristic framework is the large neighborhood search (LNS). In this section, a graph-induced neighborhood search algorithm based on the LNS framework is presented for solving the CMND problem. The LNS heuristic framework for MIP problems

consists of two main phases [38]: (i) the formulation of an auxiliary problem and (ii) the subsequent resolution.

The first phase involves formulating a simplified auxiliary mixed-integer optimization problem, denoted as P_{aux}. The auxiliary problem in LNS is designed to reduce the solution space, which can be constructed using various methodologies. One commonly used strategy involves fixing a subset of integer variables at values derived from some reference solutions [23,39]. Another strategy involves introducing additional constraints [22] or modifying the objective function [40] in the original problem formulation. It is important to note that these methods are often combined in practice and customized for specific problems. Reference [39] comprehensively analyzes LNS collective application.

The resolution phase involves solving the auxiliary problem P_{aux} using various optimization algorithms. This step is crucial as the efficiency of the entire LNS algorithm depends on the speed of obtaining a solution for the auxiliary problem. Typically, the P_{aux} resolution is accomplished by directly invoking the MIP solver. Due to its relatively small size, the optimization of this auxiliary problem is comparatively simple. Furthermore, if a distinctive structure is identified within the auxiliary problem, alternative methods, such as dynamic programming, can be employed to expedite the solution process.

In summary, we present the implementation of the LNS in Algorithm 1. Two critical phases, the P_{aux} construction and solving, are bolded. It is worth noting that some commonly used technical details of the LNS algorithm have been omitted for simplicity, such as incorporating the best solution of the original problem as a cutoff value in the auxiliary problem and defining the working limit for it [39].

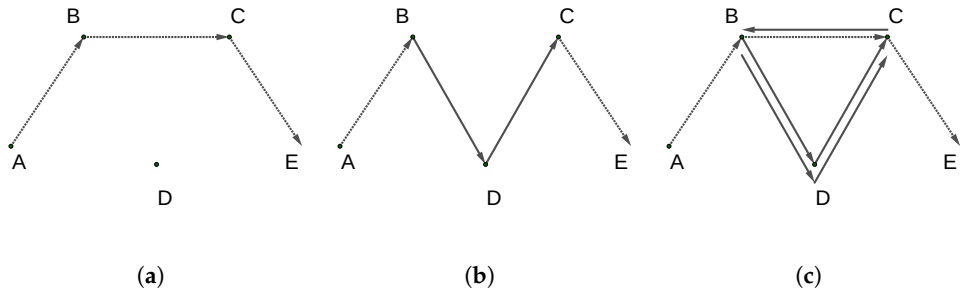
Algorithm 1 Large neighborhood search routine LNS$(\mathcal{N}, \bar{x}, \bar{y})$

Input: Network design problem P, reference solution $(\bar{x}, \bar{y})$;
Output: An improved integer-feasible (x, y), if success;
1: $P_{aux} \leftarrow P$;
2: **Phase 1: Construct (integer) variable subset $\mathcal{N}$;**
3: Fix all variables from the set $\mathcal{N}$ in P_{aux} to $(\bar{x}, \bar{y})$;
4: **Phase 2: Solve P_{aux} using MIP solvers or customized algorithms;**
5: **if** P_{aux} has a feasible solution (x, y) **then**
6: **return** (x, y);
7: **end if**

However, the LNS algorithm framework can just serve as a general paradigm rather than a specific algorithm. To effectively employ LNS in tackling the unique challenges of CMND problems, a custom and problem-specific design is necessary. In response to this need, we propose the GINS heuristic algorithm, which is a tailored implementation of the LNS paradigm. GINS incorporates innovative strategies specifically designed for the CMND problem, including (1) strategic selection of the central link, (2) division of the graph based on directed cycles, and (3) a resource-efficient facility assignment technique. These features distinguish GINS from other LNS-based approaches while fully leveraging the strengths of the LNS framework.

4.1. The Framework of GINS Algorithm

Our GINS algorithm relies on a fundamental principle that, when modifying the flows of a commodity while maintaining feasibility, the adjusted flows must form at least one directed cycle. This means that a set of arcs on a cycle must simultaneously modify its flow, and the corresponding facilities may also need to be adjusted. Figure 1 illustrates how the network flows change between two different feasible solutions.



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

Figure 1. An illustrative example of flow changes and cycle formation in a single-commodity network. (**a**) The original network flow for one commodity. (**b**) The adjusted network flow after adjustment. (**c**) Cycle formed by all links with changed flow.

In Figure 1a, the transportation of the (A, E) commodity demand is facilitated through the path (A, B, C, E). However, through the process of optimization, this demand may be rerouted through the path (A, B, D, C, E), as depicted in Figure 1b. This is equivalent to adding a directed cycle (B, D, C, B) to the previous path; see Figure 1c. In fact, improving a feasible solution of the network design problem to an optimal solution can be achieved by simultaneously changing the flows on several directed cycles. Hence, we determine P_{aux} based on a promising subnetwork consisting of some cycles within the topology structure of the underlying graph. To find a promising subnetwork, we took a two-step approach similar to that in the literature [6].

Firstly, a *central link* is identified within the entire network. For directed model (1), we choose the arc a^* such that the residual of its corresponding constraint (1c) is largest, i.e.,

$$a^* = \arg\max_{a \in \mathcal{A}} \left\{ \sum_{k \in \mathcal{K}} u_k y_a^k - \sum_{q \in \mathcal{Q}} x_a^q + u_a \right\}. \tag{4}$$

The underlying reasoning is that, when the residual is more significant, there is a higher probability of experiencing a decrease in facility costs as the flow over this arc decreases. As a result, we anticipate a substantial reduction in facility costs on the central link after the change in traffic flow. A similar trick can be found in the reference [38]. For undirected network problems, the central link e^* is the edge with the largest residual value of its corresponding constraint (2c). Specifically, e^* is determined by the following formula:

$$e^* = \arg\max_{e \in \mathcal{E}} \left\{ \sum_{k \in \mathcal{K}} u_k y_e^k - \sum_{q \in \mathcal{Q}} x_{e^+}^q - \sum_{q \in \mathcal{Q}} x_{e^-}^q + u_e \right\}. \tag{5}$$

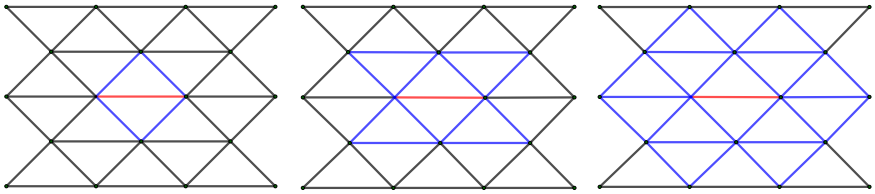
Similarly, for bidirected network problems, the central link e^* is chosen as the edge with the largest residual value of the corresponding constraint (3). The selection is based on the following formula:

$$e^* = \arg\max_{e \in \mathcal{E}} \left\{ \sum_{k \in \mathcal{K}} u_k y_e^k - \max \left\{ \sum_{q \in \mathcal{Q}} x_{e^+}^q, \sum_{q \in \mathcal{Q}} x_{e^-}^q \right\} + u_e \right\}. \tag{6}$$

Secondly, we construct a subnetwork around the central link. To be more specific, we select cycles that meet two criteria: (i) they must include the central link, and (ii) their length is constrained to a predefined value, denoted as "CycLen". The subnetwork is formed by these cycles.

Figure 2 explains the methodology of building a subnetwork through the central link. The red and blue lines in Figure 2 represent the central and other links in the subnetwork.

When CycLen is chosen as 3, 4, and 5, the subnetwork scale gradually increases. It should be noted that we only limit each cycle added to the subnetwork and, eventually, there may be cycles in the subnetwork that exceed the limit in length. With the increase in the size of subnetwork, the scale of the corresponding auxiliary problem also gradually increases. To limit the scale of the auxiliary problem, we introduce the parameter "CycNum" to determine the maximum number of links added in the subnetwork. We add links to the subnetwork one cycle at a time, and when the number of added links reaches the predefined threshold CycNum, we stop adding more links.



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

Figure 2. Illustration of subnetwork construction via a central link with different CycLen parameter settings. The GINS algorithm fixes facilities on all black links and iteratively adjusts facilities within the subgraph centered at the central link, aiming to optimize the network design progressively. (**a**) CycLen $= 3$. (**b**) CycLen $= 4$. (**c**) CycLen $= 5$.

Once we identify the subnetwork, the facilities on the remaining network links are restricted to their reference solution values. Meanwhile, the facilities located inside the subnetwork and the entire traffic flow will be released to their bounds. The objective function of P_{aux} is to minimize the combined cost of facilities within the subnetwork and the cost of the entire traffic flow, thereby facilitating a global adjustment in the traffic flow.

The procedure of using the GINS algorithm to solve the directed model (1) is summarized in Algorithm 2. The algorithm can be started from any initial feasible solution, which can be found by the technique introduced in Section 4.2. The algorithm selects a central link and solves the related auxiliary problem, leading to an improved solution; if successful, the primary step is repeated until no enhanced solution is found.

Algorithm 2 Graph-induced large neighborhood search

Input: Integer feasible solution $(\bar{x}, \bar{y})$, parameters CycLen, CycNum;
Output: An improved integer feasible (x, y), if success;
Initialize $Stop \leftarrow False$, $(x, y) \leftarrow$ **null**, $\mathcal{N} \leftarrow \varnothing$;
while $Stop = False$ **do**
  $Stop \leftarrow True$;
  Select the central link $a^* \in \mathcal{A}$;
  Clear the links set in subnetwork $\mathcal{S} \leftarrow \varnothing$;
  for Cyclic links set $\mathcal{C} \subseteq \mathcal{A}$ ($|\mathcal{C}| \leq CycLen$, and $a^* \in \mathcal{C}$) **do**
    if The number of links in $\mathcal{S}$ less than CycNum **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{C}$;
    else
      break;
    end if
  end for
  Update $\mathcal{N} \leftarrow \left\{ y_a^k \ : \ a \in \mathcal{A} \setminus \mathcal{S}, k \in \mathcal{K} \right\}$;
  Call large neighborhood search subroutine $(\bar{x}, \bar{y}) \leftarrow$LNS$(\mathcal{N}, \bar{x}, \bar{y})$;
  if $(\bar{x}, \bar{y}) \neq$ **null then**
    $Stop \leftarrow False$;
    Update $(x, y) \leftarrow (\bar{x}, \bar{y})$;
  end if
end while
return (x, y);

In each iteration of Algorithm 1, the number of facilities outside the variable fixing neighborhood $\mathcal{N}$ remains constant while the flow outside $\mathcal{N}$ is modified. However, this may result in excessive capacity provided by the facilities and excessive installation costs after each adjustment. To address this issue, we propose a technique to improve the solution quality further.

4.2. Resource-Efficient Facility Assignment Technique

In this subsection, we will introduce an efficient *resource-efficient facility assignment* (REFA) technique to improve the quality of the solutions found by the GINS algorithm further. In addition, the REFA technique can also serve as a primary heuristic to find high-quality initial feasible solutions for the capacitated multicommodity network design problems. For a general integer programming problem, the main difficulty lies in the discreteness of the integer variables, so the traditional LNS heuristics in MIP solvers fix the integer variables in the auxiliary problem to make it easy to solve. However, in the case of CMND problem (1), (2), and (3), this is the opposite.

Taking the directed model (1) as an example, when we fix all traffic flow (continuous) variables to the partial reference solution $\bar{x}$, i.e., choose $\mathcal{N} = \left\{ x_a^q : a \in \mathcal{A}, q \in \mathcal{Q} \right\}$ in Algorithm 1, the auxiliary problem P_{aux} can be expressed as follows:

$$\min_{y} \sum_{a \in \mathcal{A}} \left(\sum_{k \in \mathcal{K}} c_a^k y_a^k \right) \tag{7a}$$

$$s.t. \sum_{q \in \mathcal{Q}} \bar{x}_a^q \leq u_a + \sum_{k \in \mathcal{K}} u_k y_a^k, \qquad \forall\, a \in \mathcal{A}, \tag{7b}$$

$$y_a^k \in \mathbb{Z}^+, \qquad \forall\, a \in \mathcal{A},\ k \in \mathcal{K}. \tag{7c}$$

Next, we present a detailed discussion of the computational complexity of the proposed GINS heuristic method. Firstly, we note that the left side of constraint (7b) is a known value, and the variable y_a^k is only included in the constraint corresponding to arc a. This characteristic of GINS allows us to decompose the MIP problem (7) into $|\mathcal{A}|$ independent knapsack problems.

$$\min_{y} \sum_{k \in \mathcal{K}} c_a^k y_a^k \tag{8a}$$

$$s.t. \sum_{q \in \mathcal{Q}} \bar{x}_a^q \leq u_a + \sum_{k \in \mathcal{K}} u_k y_a^k, \tag{8b}$$

$$y_a^k \in \mathbb{Z}^+, \qquad \forall\, k \in \mathcal{K}. \tag{8c}$$

Secondly, these knapsack problems can be efficiently solved using a dynamic programming algorithm. Once the dynamic programming tables are constructed, the time complexity for resolving (7) is $\mathcal{O}(|\mathcal{A}|)$. Consequently, our GINS heuristic method operates with a computational complexity of $\mathcal{O}(|\mathcal{A}|)$ and guarantees the generation of an integer-feasible solution. This unique feature distinguishes it from other LNS-based heuristic approaches. Traditional LNS heuristics typically focus on reducing the problem size and finding feasible solutions by fixing a subset of integer variables y_a^k. However, the traditional heuristic approach can often result in infeasible auxiliary problems, which waste computational time and fail to produce meaningful results. In contrast, as demonstrated in our analysis, the GINS heuristic consistently generates an integer-feasible solution, although it does not provide a theoretical guarantee regarding the quality of the solution. The procedure for applying the REFA technique is outlined in Algorithm 3.

Algorithm 3 Resource-efficient facility assignment technique

Input: Network design problem P, reference traffic flow values $(\bar{x})$;
Output: An integer feasible (x, y);
1: $P_{aux} \leftarrow P$;
2: Construct continuous variable subset $\mathcal{N} = \left\{ x_a^q \ : \ a \in \mathcal{A}, q \in \mathcal{Q} \right\}$;
3: Fix all variables x_a^q from the set $\mathcal{N}$ in P_{aux} to $(\bar{x})$;
4: Decompose the problem P_{aux} into $|\mathcal{A}|$ independent knapsack problems;
5: Solve $|\mathcal{A}|$ knapsack problems using dynamic programming algorithm;
6: Obtain the solution y_a^k for each knapsack problem corresponding to each arc a;
7: Combine these solutions to construct an integer feasible solution (x, y) for P_{aux};
8: **return** (x, y);

4.3. Implementation Details

As long as we obtain a feasible network traffic flow $\bar{x}$ as the partial reference solution, we can construct a corresponding integer solution $(\bar{x}, \bar{y})$ as the output of Algorithm 1. The construction of this feasible traffic flow $\bar{x}$ can be achieved through various methods, such as utilizing the shortest path algorithm [6,32], solving the linear relaxation solution at each node of the branch-and-bound tree, or employing the GINS algorithm introduced in Section 4.2. To be more specific, in our algorithm implementation, the REFA technique is used in two scenarios.

(1) Whenever our GINS routine obtains a **better integer-feasible solution**, the REFA technique utilizes it to check whether we can improve the solution quality further. The integer-feasible solution can either be returned by CPLEX or obtained by GINS itself.
(2) Whenever our GINS routine obtains a **linear relaxation solution**, the REFA technique can obtain a corresponding integer-feasible solution (has the same flow values as the linear relaxation solution). The way of obtaining linear relaxation solutions differs depending on how GINS is implemented.
    (a) If the GINS is embedded as a callback routine within CPLEX, a linear relaxation solution could be returned in each node in the branch-and-bound tree.
    (b) If the GINS is implemented independently, we call CPLEX once and only once to solve the relaxed CMND problem and obtain the initial linear relaxation solution.

Following the implementation of this technique, the issue of excessive facility installations on each link is effectively resolved. To conclude this section, we draw Figure 3 to demonstrate the flowchart of our proposed algorithm clearly. Precisely, steps (1), (2a), and (2b) in the figure correspond to the three reference solution cases discussed earlier, highlighting their logical connections.

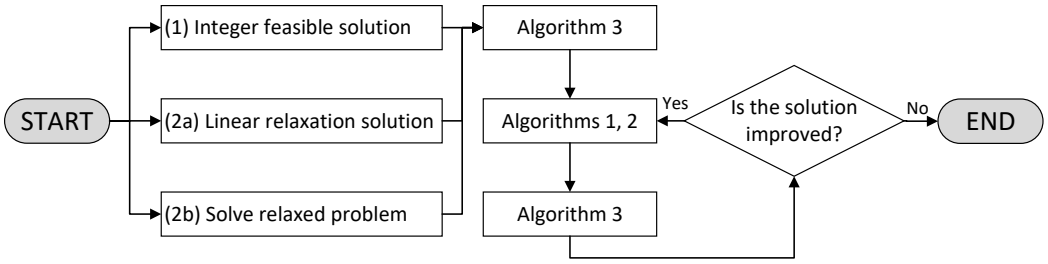


Figure 3. Flowchart of the GINS algorithm integrated with the REFA technique.

5. Computational Results

In this section, we present the performance of our heuristic and compare the results with the widely used commercial MIP solver, CPLEX. Additionally, we benchmark our

method against other available heuristics in solving network design problems, demonstrating its effectiveness and efficiency. While we acknowledge the usefulness of modern metaheuristic approaches, such as hybrid genetic algorithms and ant colony optimization algorithms, this study focuses on comparing our approach with widely adopted baselines and heuristics traditionally used in this domain. Specifically, our numerical experiments are divided into two parts to comprehensively evaluate the performance of our method within the scope of these established comparisons.

- Firstly, we implemented GINS in C and embedded it as a callback routine within the IBM ILOG CPLEX Optimizer 20.1.0 library. This approach is motivated by the fact that GINS is tailored to address general network design problems and aims to handle more complicated situations in practice. Meanwhile, CPLEX is designed to solve general MIP problems using a range of built-in heuristics, which can be further enhanced by GINS. Combining these two tools allows us to effectively test the ability of GINS to solve real-world problems and evaluate its versatility.
- Secondly, we implemented GINS independently without embedding it in CPLEX as a callback routine. For comparison, we also implemented two other heuristics from the literature [14,27] and evaluated the quality of solutions obtained by different algorithms. We applied the same time limits and used CPLEX (with identical parameters) to solve both the linear relaxation problems and the auxiliary MIP problems required by GINS and other heuristics.

Following [8], to avoid changing the problem structure, the presolving features were turned off in our experiments. All computations were executed in a single-threaded environment to eliminate the effect of multithreads. Our experiments were performed on a cluster of Intel(R) Xeon(R) Gold 6140 CPU @ 2.30 GHz computers with 192 GB RAM, running Linux (in 64-bit mode). The parameters CycLen and CycNum of GINS were set to 4 and 30, respectively.

5.1. Testsets

We conducted our computational study on a large-scale network testset encompassing all three link models: the directed model (1), the undirected model (2), and the bidirected model (3). This testset was created using widely accepted random generation methodologies from the literature, as referenced in previous studies [7,8,41]. These methodologies are commonly used to evaluate algorithmic performance and are recognized as standard benchmarks for testing optimization algorithms in capacitated multicommodity multi-facility network design problems. The generation process is as follows:

1. Generate graphs with 50 nodes, where the coordinates of the nodes are uniformly distributed on a 100×100 region in the Euclidean plane.
2. Randomly assign a degree to each point, with a probability of 0.2 for the degree to be either 4 or 7 and a probability of 0.3 for it to be 5 or 6.
3. For each node, sequentially add edges to the nearest node until the degree is met, ensuring that the length of each link does not exceed 50.
4. After all links have been added, verify that the graph is connected; if not, the graph must be regenerated.
5. Each case has 100 sets of demands, each randomly generating a single-source–single-sink pair, and the demand value is uniformly selected from the range from 10 to 190.

The key characteristics and features of the instances are summarized in Table 3. Furthermore, we use all 27 combinations of different sizes and costs studied in [8]. For the same facility, we assume that its costs on all links are the same. The unit flow cost across each link equals the length rounded down to the nearest integer. We randomly generate

three graphs for each capacity and cost combination in Table 4. Thus, in total, we have 81 instances for each link model.

Table 3. Summary of testset characteristics.

Parameter	Value / Range	Notes
Number of nodes	50	Uniformly distributed.
Number of edges	$\left(\sum_{v \in \mathcal{V}} v_{\text{degree}}\right)/2$	Based on node degree.
Length of edges	0–50	Based on Euclidean distance.
Demand value range	10–190	Uniform distribution.
Number of demand pairs	100	Single-source–single-sink pair.

Table 4. All capacity and cost combinations of available facilities.

Number of facilities		**1**	
Instance ID	1-1-1	1-1-2	1-1-3
Capacity	(130)	(130)	(130)
Cost	(10,000)	(18,000)	(25,000)
Number of facilities		**1**	
Instance ID	1-2-1	1-2-2	1-2-3
Capacity	(170)	(170)	(170)
Cost	(10,000)	(18,000)	(25,000)
Number of facilities		**1**	
Instance ID	1-3-1	1-3-2	1-3-3
Capacity	(200)	(200)	(200)
Cost	(10,000)	(18,000)	(25,000)
Number of facilities		**2**	
Instance ID	2-1-1	2-1-2	2-1-3
Capacity	(130, 50)	(130, 50)	(130, 50)
Cost	(10,000, 5000)	(18,000, 9000)	(25,000, 13,000)
Number of facilities		**2**	
Instance ID	2-2-1	2-2-2	2-2-3
Capacity	(170, 70)	(170, 70)	(170, 70)
Cost	(10,000, 5000)	(18,000, 9000)	(25,000, 13,000)
Number of facilities		**2**	
Instance ID	2-3-1	2-3-2	2-3-3
Capacity	(200, 80)	(200, 80)	(200, 80)
Cost	(10,000, 5000)	(18,000, 9000)	(25,000, 13,000)
Number of facilities		**3**	
Instance ID	3-1-1	3-1-2	3-1-3
Capacity	(130, 50, 20)	(130, 50, 20)	(130, 50, 20)
Cost	(10,000, 5000, 2500)	(18,000, 9000, 5000)	(25,000, 13,000, 9000)
Number of facilities		**3**	
Instance ID	3-2-1	3-2-2	3-2-3
Capacity	(170, 70, 30)	(170, 70, 30)	(170, 70, 30)
Cost	(10,000, 5000, 2500)	(18,000, 9000, 5000)	(25,000, 13,000, 9000)
Number of facilities		**3**	
Instance ID	3-3-1	3-3-2	3-3-3
Capacity	(200, 80, 30)	(200, 80, 30)	(200, 80, 30)
Cost	(10,000, 5000, 2500)	(18,000, 9000, 5000)	(25,000, 13,000, 9000)

5.2. Numerical Experiments with GINS Embedded in CPLEX

In this subsection, we present our comprehensive numerical results to prove the effectiveness of the GINS heuristic. We set the time limit to 7200 s. Since none of the instances could be solved within the time limit and the optimal solutions are not available in the literature, we used the best solution found by the CPLEX solver within 5 h with default settings, denoted as v^*, to validate the performance of the approximate solutions obtained by different methods. Specifically, we compared the objective value obtained by each algorithm with v^* and calculated the percentage gap between these values. We set a relative gap tolerance of 1% for all experiments in this paper. The relative gap for each instance was computed as $(\hat{v} - v^*)/v^*$, where $\hat{v}$ represents the best solution value found by each method. This approach allows us to evaluate the quality of solutions provided by each algorithm relative to the optimal solution and provides insights into the effectiveness and efficiency of the algorithms. In the following, we use "CPX" and "GINS + CPX" to denote the default setting of CPLEX and the one with the GINS implemented through callback routines.

- **CPX:** CPLEX with all the parameters set to their default values.
- **GINS + CPX:** CPLEX with all the parameters set to their default values, while the GINS heuristic is embedded as a callback routine.

We provide a summary of the computational results in a similar table to that used in [8]. Tables 5–7 display the overall numerical results for three link models, respectively. Column 'Bracket(s)' marks some subsets of instances depending on the solving time of instances. The 'All' bracket contains all instances of the testset. The '$\geq n$' brackets contain all instances whose solution time is between n seconds and the time limit in at least one setting. Column 'Total' lists the number of models in each bracket of the testset. The 'Solved' column indicates the number of instances that are solved under the GINS + CPX or CPX settings. Column 'Nodes' shows the average of the number of branch-and-bound nodes. Similarly, column 'Time(s)' lists the average solving times.

Tables 5–7 indicate that our GINS algorithm has a significant positive effect on all these three link models. In particular, using the GINS algorithm, we can solve 7, 3, and 17 more instances than those using the default setting on directed, undirected, and bidirected models, respectively. As shown in Tables 5–7, the default CPLEX setting requires nearly two times more computational time compared to the GINS + CPX setting. Furthermore, the number of nodes is significantly reduced. For the more challenging instances, the GINS algorithm demonstrates a significant advantage over the default CPLEX solver. As the network size increases and the problem becomes more complex, the time required by CPLEX grows much faster. In contrast, the GINS algorithm exhibits a more moderate increase in computational time. These computation results clearly show that our GINS heuristic can effectively reduce the scale of the branch-and-bound trees. Additionally, our approach significantly improves the performance of the MIP solver in solving the CMND problems on all three link types.

Table 5. Performance comparison on the directed models.

		GINS + CPX			**CPX**		
Bracket(s)	**Total**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**
All	81	66	3161	1247.30	59	8327	2164.75
≥900	56	45	4038	1580.16	38	13,140	3409.46
≥1800	49	38	4329	1655.00	31	16,995	4273.49
≥3600	45	34	4325	1670.57	27	17,980	4583.15

Table 6. Performance comparison on the undirected models.

		GINS + CPX			CPX		
Bracket(s)	**Total**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**
All	81	79	1138	768.47	76	5715	1331.86
≥900	59	58	1374	903.78	55	8494	1847.78
≥1800	35	34	1637	1043.89	31	14,226	2816.84
≥3600	15	14	1795	1049.65	11	22,869	4392.93

Table 7. Performance comparison on the bidirected models.

		GINS + CPX			CPX		
Bracket(s)	**Total**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**
All	81	74	804	1648.43	57	4428	3119.29
≥900	77	73	736	1569.01	56	4371	3071.30
≥1800	59	55	839	1757.24	38	6408	4491.96
≥3600	43	39	889	1781.54	22	9132	6064.32

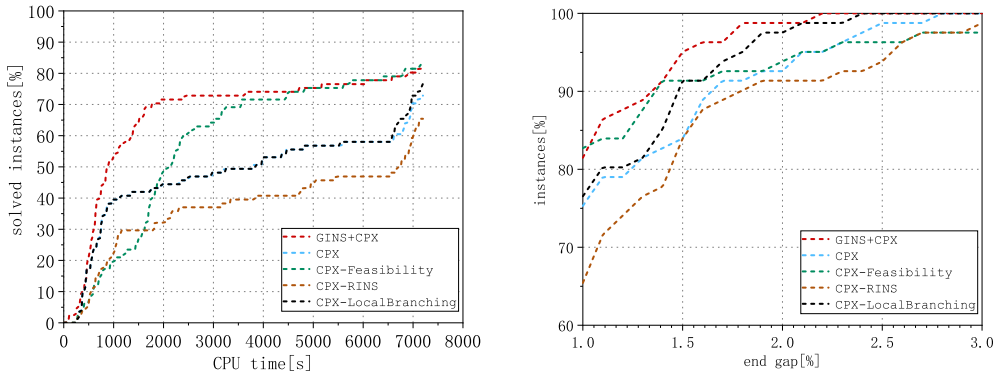
To demonstrate the superiority of our GINS method over the built-in heuristics in CPLEX, we focused on some classical neighborhood search heuristics that involve solving auxiliary MIP problems, similar to how GINS operates. We conducted additional experiments using three alternative parameter settings for CPLEX, which were proposed by Guastaroba et al. [42]. This approach allowed us to evaluate the performance of our GINS algorithm thoroughly. These settings were applied to ensure a fair and comprehensive evaluation of the GINS algorithm. These configurations are denoted as, "CPX-Feasibility", "CPX-RINS", and "CPX-LocalBranching".

- **CPX-Feasibility:** This configuration uses CPLEX with all parameters set to their default values, except for the parameter 'CPXPARAM_Emphasis_MIP', which is set to "feasibility". In this setting, CPLEX is programmed to frequently generate more feasible solutions when optimizing problems, which may result in sacrificing some speed in the pursuit of proving optimality.
- **CPX-RINS:** This setup employs CPLEX with all default parameters, with the exception of the parameter 'CPXPARAM_MIP_Strategy_RINSHeur', which is set to 20. In this configuration, CPLEX executes the RINS heuristic [23] every 20 nodes. This approach can be more aggressive than the default CPX setting.
- **CPX-LocalBranching:** This option utilizes CPLEX with all parameters at their default values, except for the parameter 'CPXPARAM_MIP_Strategy_LBHeur', which has been activated. This setting enables the local branching heuristic [22], which is disabled by default. Previous studies have demonstrated the effectiveness of customized local branching methods in addressing fixed-charge network design problems [13,14]. Thus, this configuration serves as a particularly suitable benchmark for evaluating the effectiveness of GINS.

The three settings were selected because of their frequent utilization of built-in heuristics in CPLEX. Additionally, the last two methods, which involve fixing integer variables and solving auxiliary problems, are similar to our GINS heuristic method. They are all based on the LNS Algorithm 1. The three CPLEX settings emphasize quickly finding the feasible solution early in the optimization process. Tables 8–10 show the results for these three settings. Comparing these settings with GINS + CPX, we can see that the number of instances solved by GINS has increased overall across the three link types. In addition, the number of nodes explored and the running time are significantly reduced. In particular, as presented in Table 10, our GINS algorithm performs significantly in solving the bidirected CMND problem. It significantly increases the number of solved instances and reduces

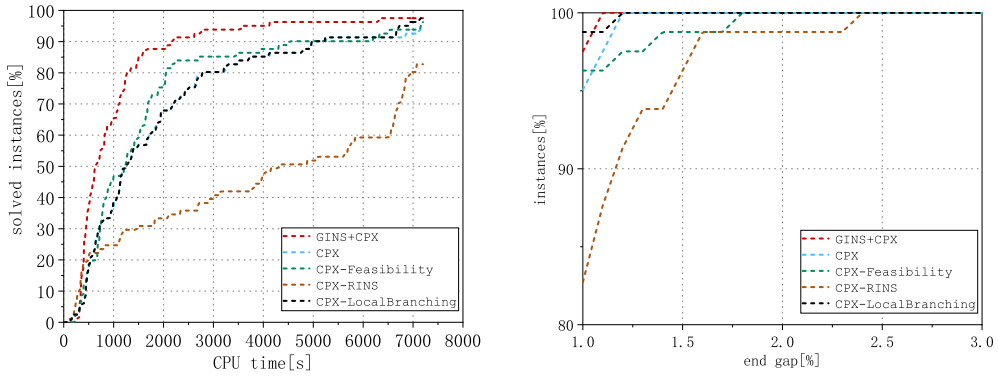
the required solving time. Therefore, GINS demonstrates a superior ability to solve this specific problem type. Overall, according to the results shown in Tables 8–10, we can conclude that the GINS method holds a notable advantage over RINS [23], local branching [22], and other heuristics included in CPLEX in terms of finding feasible solutions for the CMND problems.

Additionally, Figures 4a, 5a, and 6a display the performance profiles in terms of running times visually. Meanwhile, Figures 4b, 5b, and 6b showcase the end gaps for the remaining unsolved instances. In the performance profile graph, each point with coordinates (a, b) in a line represents that, for b% of the instances, reaching their gap tolerance needs less than or equal to a seconds. Similarly, in the end gap graph, each point with coordinates (a, b) represents that, for b% of the instances, the end gap is less than or equal to a%. The three performance profile graphs clearly demonstrate that our GINS method efficiently discovers high-quality solutions. By incorporating GINS callback routines into CPLEX, we can expedite the solving process for all types of CMND instances. Furthermore, the three end gap graphs illustrate that the end gaps of remaining unsolved instances under the GINS + CPX setting are substantially smaller than the four other CPLEX settings. Regardless of the chosen end gap as the optimal tolerance threshold, the GINS algorithm consistently enables the successful solution of more instances across all three link types. In general, the GINS algorithm significantly improves the efficiency and effectiveness of the MIP solver in solving the capacitated multicommodity network design problem.



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

Figure 4. Illustration of the comparison results between GINS + CPX and four CPLEX settings on the directed link models: (**a**) performance profile graph of solved instances; (**b**) end gap graph of unsolved instances.



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

Figure 5. Illustration of the comparison results between GINS + CPX and four CPLEX settings on the undirected link models: (**a**) performance profile graph of solved instances; (**b**) end gap graph of unsolved instances.

Table 8. Comparison of GINS + CPX and three CPLEX settings on the directed models.

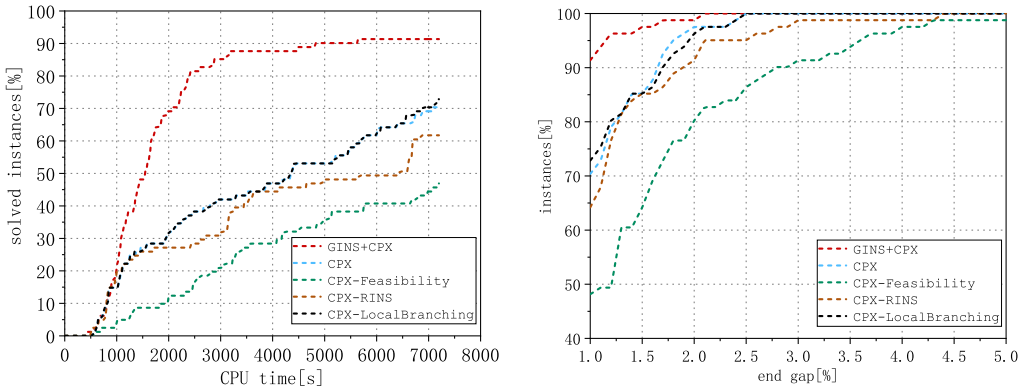
		GINS + CPX			CPX-Feasibility			CPX-RINS			CPX-LocalBranching		
Bracket(s)	**Total**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**
All	81	66	3161	1247.30	67	12,538	2165.34	53	10,398	3045.64	62	8319	2158.22
≥900	78	63	3262	1295.09	64	13,297	2283.37	50	11,163	3297.28	59	8845	2298.28
≥1800	76	61	3342	1335.88	62	13,698	2324.35	48	11,731	3430.02	57	9243	2400.18
≥3600	65	50	4007	1532.64	51	14,728	2524.82	37	14,748	4321.60	46	11,546	2989.08

Table 9. Comparison of GINS + CPX and three CPLEX settings on the undirected models.

		GINS + CPX			CPX-Feasibility			CPX-RINS			CPX-LocalBranching		
Bracket(s)	**Total**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**
All	81	79	1138	768.47	78	5343	1203.05	67	10,646	2432.17	79	5711	1329.03
≥900	75	73	1271	824.45	72	5608	1282.62	61	12,754	2907.61	73	6310	1448.33
≥1800	67	65	1301	843.73	64	5871	1349.31	53	16,681	3701.07	65	7030	1601.06
≥3600	53	51	1418	897.04	50	6384	1470.53	39	24,053	5035.71	51	8192	1766.05

Table 10. Comparison of GINS + CPX and three CPLEX settings on the bidirected models.

		GINS + CPX			CPX-Feasibility			CPX-RINS			CPX-LocalBranching		
Bracket(s)	**Total**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**	**Solved**	**Nodes**	**Time(s)**
All	81	74	804	1648.43	38	8589	4728.73	50	4218	3432.71	59	4422	3115.16
≥900	79	74	759	1588.03	38	8532	4678.67	50	4153	3368.93	59	4349	3049.78
≥1800	76	71	759	1618.58	35	9112	5016.77	47	4373	3578.10	56	4488	3202.18
≥3600	68	63	784	1648.72	27	10,292	5523.80	39	4914	4029.38	48	5002	3529.51



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

Figure 6. Illustration of the comparison results between GINS + CPX and four CPLEX settings on the bidirected link models: (**a**) performance profile graph of solved instances; (**b**) end gap graph of unsolved instances.

5.3. Numerical Experiments with Independent GINS Algorithm

In this subsection, we present numerical results to demonstrate the efficiency of the independent GINS algorithm, which operates outside the CPLEX framework. We compare its performance with two established heuristics that have proven to be highly effective in solving fixed-charge capacitated multicommodity network design problems.

The first comparison heuristic is the *slope scaling* (SS) algorithm [14]. This method iteratively adjusts the objective coefficients by solving linear programming subproblems. In network design problems, fixed installation costs create a nonlinear cost structure and complicate matters. SS heuristic addresses this by "scaling" these nonlinear functions to linear, simplifying the solving process. We provide a brief overview of the SS heuristic as follows:

1. **Initialize:** Set the initial cost weights for each facility based on the original cost function.
2. **Solve subproblem:** Solve a linear relaxation subproblem using the current weights.
3. **Adjust weights:** Update the cost weights based on the linear relaxation solution.
4. **Obtain solution:** Round up all integer variables (corresponding to the number of facilities) from the relaxation solution to obtain a feasible integer solution.

This process is repeated until the time limit is reached. For more details about SS, we refer to literature [14].

The second comparison heuristic is the *slope-scaling-based neighborhood search* (SSBNS) [27] algorithm. In contrast to the SS algorithm, after applying slope scaling, this method excludes links with consistently zero flow, i.e., links where the flow remains zero throughout the iteration process. The SSBNS heuristic assumes that these links will likely have no flow passing through them. Subsequently, the SSBNS heuristic directly utilizes the MIP solver to optimize the MIP subproblem and may obtain the approximate solution. For more details about SSBNS, we refer to algorithms 3 and 4 of reference [27].

The effectiveness of these heuristics is closely related to the running time. The longer the runtime, the more iterations that can be performed, increasing the likelihood of finding higher-quality feasible solutions. To ensure a fair comparison, we set time limits ranging from 50 to 500 s for all heuristics. This allowed us to evaluate the quality of solutions found within a short time and compare the evolution of the best solution over time. We also used the best solution found by the CPLEX solver within 5 h with default settings to validate the performance of the approximate solutions obtained by different methods. The gap

calculation method described in the previous subsection was used. In the following, we use "GINS" to denote the independent GINS algorithm.

Table 11 displays the computational results for different heuristics. Columns 'Time(s)' and 'Models' represent the time limit and corresponding CMND link models. The symbols (D), (U), and (B) represent the directed, undirected, and bidirected link models, respectively. The 'All' bracket contains all instances of the three link types. Column 'Opt' denotes the best values obtained by the CPLEX solver within 5 h. Columns 'Obj' and 'Gap(%)' list the objective values and corresponding gaps of different heuristics, respectively. It is important to mention that all values are calculated as the geometric means of the values across all corresponding instances. To avoid issues with gaps being zero, we employ a shifted geometric mean with a shift of 1 (the shifted geometric mean of values $x_1, x_2, \cdots, x_n$, where shift s is defined as $\prod_{k=1}^{n}(x_k + s)^{1/n} - s$; see [43]).

Table 11. Performance comparison between GINS, SS, and SSBNS.

Time(s)	Models	Opt	GINS		SS		SSBNS	
			Obj	Gap(%)	Obj	Gap(%)	Obj	Gap(%)
50	(D)	3,981,031.78	4,169,974.61	4.04	4,459,898.20	10.59	4,369,777.06	8.13
	(U)	3,809,821.78	3,982,813.09	3.98	4,260,059.52	10.43	4,066,677.27	5.66
	(B)	2,422,765.95	2,695,152.09	9.49	2,898,820.48	14.26	3,666,230.61	19.16
	All	3,324,583.61	3,550,603.75	5.41	3,804,702.00	11.63	4,023,829.68	9.70
100	(D)	3,981,031.78	4,139,207.79	3.42	4,458,709.24	10.57	4,359,208.60	7.69
	(U)	3,809,821.78	3,958,735.85	3.41	4,258,690.71	10.40	4,047,683.97	5.24
	(B)	2,422,765.95	2,664,450.47	8.31	2,892,170.61	14.11	3,601,956.43	18.05
	All	3,324,583.61	3,521,225.22	4.66	3,801,045.39	11.58	3,990,716.73	9.11
200	(D)	3,981,031.78	4,092,975.84	2.43	4,458,709.24	10.57	4,239,642.27	4.46
	(U)	3,809,821.78	3,921,401.42	2.52	4,258,690.71	10.40	3,955,405.85	3.12
	(B)	2,422,765.95	2,592,543.35	5.66	2,888,647.43	14.03	2,911,643.88	15.53
	All	3,324,583.61	3,465,257.45	3.32	3,799,501.31	11.56	3,654,989.85	6.19
300	(D)	3,981,031.78	4,072,006.66	1.92	4,458,697.57	10.57	4,165,593.32	2.90
	(U)	3,809,821.78	3,893,264.15	1.86	4,258,690.71	10.40	3,907,931.08	2.22
	(B)	2,422,765.95	2,545,416.21	3.98	2,888,647.43	14.03	2,827,967.96	12.25
	All	3,324,583.61	3,429,996.85	2.46	3,799,498.00	11.56	3,583,984.19	4.50
500	(D)	3,981,031.78	4,035,163.89	1.12	4,458,697.57	10.57	4,079,544.11	1.67
	(U)	3,809,821.78	3,860,755.61	1.19	4,258,690.71	10.40	3,877,667.35	1.59
	(B)	2,422,765.95	2,488,180.50	2.08	2,888,647.43	14.03	2,655,552.61	6.73
	All	3,324,583.61	3,384,322.91	1.43	3,799,498.00	11.56	3,476,260.92	2.77

The results in Table 11 indicate that, in comparison to the SS and SSBNS heuristics, our GINS method consistently delivers the highest-quality feasible solutions with the smallest end gaps, regardless of the chosen time limits. Notably, GINS not only produces effective solutions within a short time but also exhibits potential for further improvement over time. In contrast, the SS heuristic lacks such potential. When the time limit exceeds 200 s, the SS heuristic is unable to find a better solution. On the other hand, the SSBNS heuristic, being an LNS heuristic, takes more time to solve the auxiliary MIP subproblem, making it relatively more time-consuming than SS (which just solves linear programming subproblems). The SSBNS heuristic can identify high-quality feasible solutions over a longer time, but its performance deteriorates rapidly within a shorter time. In summary, the results presented in Table 11 support the conclusion that the GINS algorithm performs superior to the other available heuristics across all three link models.

The four subplots in Figure 7 collectively demonstrate the superior performance of the proposed GINS algorithm compared to the baseline algorithms, SS and SSBNS, across all tested link models. Specifically, the figure evaluates the average end gaps over time,

comprehensively analyzing these temporal behaviors and effectiveness. Due to the inherent characteristics of the SS algorithm, its performance does not depend on time. While it can achieve a reasonable level of effectiveness initially, especially in the bidirected model, both the GINS and SSBNS methods can outpace it as time progresses. Across all link models, GINS consistently achieves a lower end gap than SS and SSBNS, indicating its faster convergence and higher accuracy. This trend is evident in the early stages of the time axis, where GINS shows a more rapid reduction in the end gap. In the later stages, it stabilizes at a significantly lower value than the other two algorithms. The consistent performance across directed, undirected, and bidirected link models highlights the robustness and adaptability of the proposed algorithm.

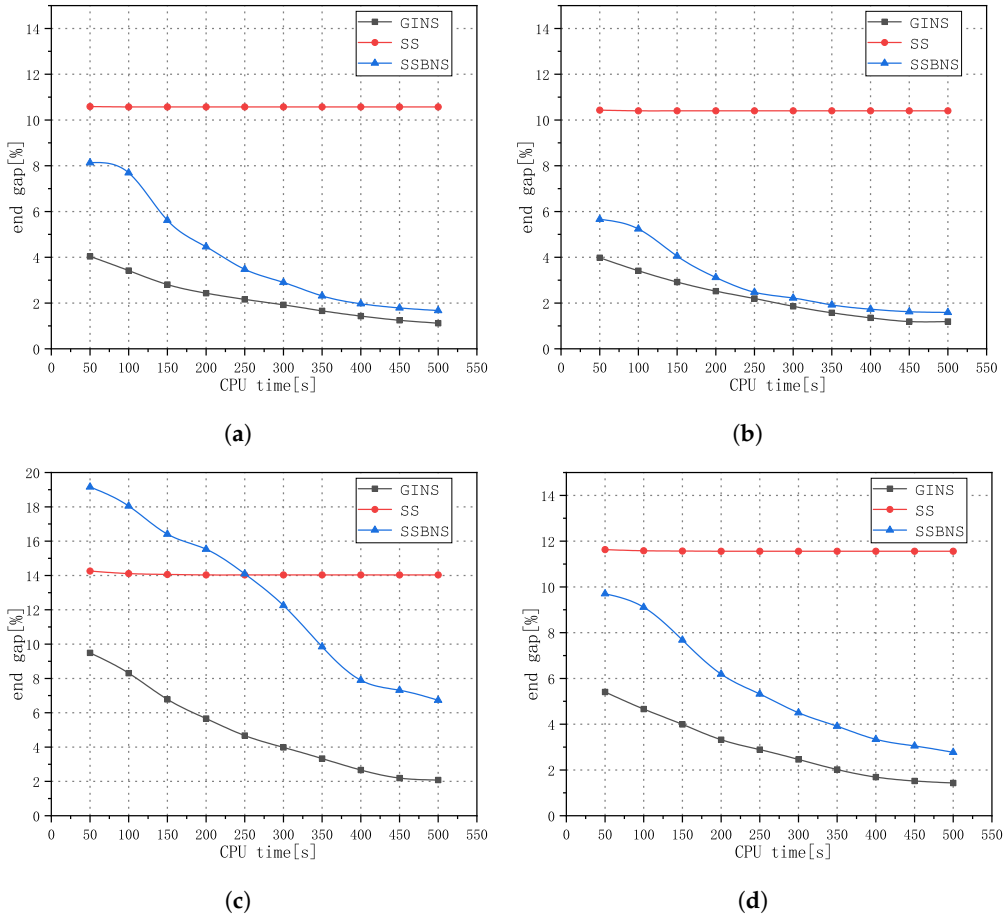


Figure 7. Comparison of average end gaps between GINS, SS, and SSBNS over time. (**a**) Directed link model. (**b**) Undirected link model. (**c**) Bidirected link model. (**d**) All link models.

6. Conclusions

In this paper, we present an efficient graph-induced neighborhood search heuristic algorithm for addressing the large-scale capacitated multicommodity network design problem. By leveraging the topological structure of the underlying graph, we construct a series of auxiliary problems and iteratively refine the solution quality. Extensive computational experiments on three distinct link models demonstrate that our proposed algorithm is superior to other existing heuristics. Notably, our algorithm achieves remarkable reductions in both the size of branch-and-bound trees and the overall computational time. Overall, our research introduces a novel heuristic technique that efficiently solves all three link models of the CMND problem, offering a valuable contribution to network design optimization.

Funding: This research received no external funding.

Data Availability Statement: All the codes and corresponding experimental results that support the findings of this paper are available in the GitHub repository https://github.com/houshan-zhang/graph-induced-heuristic (accessed on 9 February 2025).

Acknowledgments: The computations were performed on the high-performance computers of the State Key Laboratory of Scientific and Engineering Computing, Chinese Academy of Sciences.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Govindan, K.; Fattahi, M.; Keyvanshokooh, E. Supply chain network design under uncertainty: A comprehensive review and future research directions. *Eur. J. Oper. Res.* **2017**, *263*, 108–141. [CrossRef]
2. Cordeau, J.F.; Pasin, F.; Solomon, M.M. An integrated model for logistics network design. *Ann. Oper. Res.* **2006**, *144*, 59–82. [CrossRef]
3. Cordeau, J.F.; Laporte, G.; Pasin, F. An iterated local search heuristic for the logistics network design problem with single assignment. *Int. J. Prod. Econ.* **2008**, *113*, 626–640. [CrossRef]
4. Crainic, T.G.; Hewitt, M. Service Network Design. In *Network Design with Applications to Transportation and Logistics*; Crainic, T.G., Gendreau, M., Gendron, B., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 347–382. [CrossRef]
5. Soriano, P.; Wynants, C.; Séguin, R.; Labbé, M.; Gendreau, M.; Fortz, B. Design and Dimensioning of Survivable SDH/Sonet Networks. In *Telecommunications Network Planning*; Sansò, B., Soriano, P., Eds.; Springer US: Boston, MA, USA, 1999; pp. 147–167. [CrossRef]
6. Agarwal, Y.K. Design of Capacitated Multicommodity Networks with Multiple Facilities. *Oper. Res.* **2002**, *50*, 333–344. [CrossRef]
7. Salman, F.S.; Ravi, R.; Hooker, J.N. Solving the Capacitated Local Access Network Design Problem. *INFORMS J. Comput.* **2008**, *20*, 243–254. [CrossRef]
8. Chen, L.; Chen, W.K.; Yang, M.M.; Dai, Y.H. An exact separation algorithm for unsplittable flow capacitated network design arc-set polyhedron. *J. Glob. Optim.* **2021**, *81*, 659–689. [CrossRef]
9. Atamtürk, A.; Günlük, O. A note on capacity models for network design. *Oper. Res. Lett.* **2018**, *46*, 414–417. [CrossRef]
10. Atamtürk, A.; Günlük, O. Multicommodity Multifacility Network Design. In *Network Design with Applications to Transportation and Logistics*; Crainic, T.G., Gendreau, M., Gendron, B., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 141–166. [CrossRef]
11. Mejri, I.; Layeb, S.B.; Zeghal, F. A survey on network design problems: Main variants and resolution approaches. *Eur. J. Ind. Eng.* **2023**, *17*, 253–309. [CrossRef]
12. Hewitt, M.; Nemhauser, G.L.; Savelsbergh, M.W.P. Combining Exact and Heuristic Approaches for the Capacitated Fixed-Charge Network Flow Problem. *INFORMS J. Comput.* **2010**, *22*, 314–325. [CrossRef]
13. Katayama, N. MIP Neighborhood Search Heuristics for a Capacitated Fixed-Charge Network Design Problem. *Asia-Pac. J. Oper. Res.* **2020**, *37*, 2050009. [CrossRef]
14. Gendron, B.; Hanafi, S.; Todosijević, R. Matheuristics based on iterative linear programming and slope scaling for multicommodity capacitated fixed charge network design. *Eur. J. Oper. Res.* **2018**, *268*, 70–81. [CrossRef]
15. Bienstock, D.; Günlük, O. Capacitated Network Design Polyhedral Structure and Computation. *INFORMS J. Comput.* **1996**, *8*, 243–259. [CrossRef]
16. Günlük, O. A branch-and-cut algorithm for capacitated network design problems. *Math. Program.* **1999**, *86*, 17–39. [CrossRef]
17. Magnanti, T.L.; Mirchandani, P.; Vachani, R. The convex hull of two core capacitated network design problems. *Math. Program.* **1993**, *60*, 233–250. [CrossRef]
18. Atamtürk, A.; Rajan, D. On splittable and unsplittable flow capacitated network design arc–set polyhedra. *Math. Program.* **2002**, *92*, 315–333. [CrossRef]
19. Brockmüller, B.; Günlük, O.; Wolsey, L. Designing Private Line Networks—Polyhedral Analysis and Computation. LIDAM Discussion Papers CORE 1996047, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE). 1996. Available online: https://ideas.repec.org/p/cor/louvco/1996047.html (accessed on 9 February 2025).
20. van Hoesel, S.P.; Koster, A.M.; van de Leensel, R.L.; Savelsbergh, M.W. Polyhedral results for the edge capacity polytope. *Math. Program.* **2002**, *92*, 335–358. [CrossRef]
21. Raack, C.; Koster, A.M.; Orlowski, S.; Wessäly, R. On cut-based inequalities for capacitated network design polyhedra. *Networks* **2011**, *57*, 141–156. [CrossRef]
22. Fischetti, M.; Lodi, A. Local branching. *Math. Program.* **2003**, *98*, 23–47. [CrossRef]

23. Danna, E.; Rothberg, E.; Pape, C.L. Exploring relaxation induced neighborhoods to improve MIP solutions. *Math. Program.* **2005**, *102*, 71–90. [CrossRef]
24. Ghamlouche, I.; Crainic, T.G.; Gendreau, M. Cycle-Based Neighbourhoods for Fixed-Charge Capacitated Multicommodity Network Design. *Oper. Res.* **2003**, *51*, 655–667. [CrossRef]
25. Ghamlouche, I.; Crainic, T.G.; Gendreau, M. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Ann. Oper. Res.* **2004**, *131*, 109–133. [CrossRef]
26. Paraskevopoulos, D.C.; Bektaş, T.; Crainic, T.G.; Potts, C.N. A cycle-based evolutionary algorithm for the fixed-charge capacitated multi-commodity network design problem. *Eur. J. Oper. Res.* **2016**, *253*, 265–279. [CrossRef]
27. Katayama, N. A combined fast greedy heuristic for the capacitated multicommodity network design problem. *J. Oper. Res. Soc.* **2019**, *70*, 1983–1996. [CrossRef]
28. Liu, S.; Yang, F.; Liu, T.; Li, M. An Effective Two-Stage Algorithm for the Bid Generation Problem in the Transportation Service Market. *Mathematics* **2024**, *12*, 1007. [CrossRef]
29. Ahmed, Z.H.; Hameed, A.S.; Mutar, M.L.; Haron, H. An Enhanced Ant Colony System Algorithm Based on Subpaths for Solving the Capacitated Vehicle Routing Problem. *Symmetry* **2023**, *15*, 2020. [CrossRef]
30. Li, H.; Zhan, J.; Zhao, Z.; Wang, H. An Improved Particle Swarm Optimization Algorithm Based on Variable Neighborhood Search. *Mathematics* **2024**, *12*, 2708. [CrossRef]
31. Rahmaniani, R.; Ghaderi, A. A combined facility location and network design problem with multi-type of capacitated links. *Appl. Math. Model.* **2013**, *37*, 6400–6414. [CrossRef]
32. Khelifi, M.; Babes, M.; Ghanemi, S.; Saidi, M.; Boudjit, S. Hybrid Heuristic for Capacitated Network Design Problem. *J. High Speed Netw.* **2015**, *21*, 313–330. [CrossRef]
33. Chen, L.; Dong, T.; Peng, J.; Ralescu, D. Uncertainty Analysis and Optimization Modeling with Application to Supply Chain Management: A Systematic Review. *Mathematics* **2023**, *11*, 2530. [CrossRef]
34. Salehnia, T.; Montazerolghaem, A.; Mirjalili, S.; Khayyambashi, M.R.; Abualigah, L. Chapter 8—SDN-based optimal task scheduling method in Fog-IoT network using combination of AO and WOA. In *Handbook of Whale Optimization Algorithm*; Academic Press: Cambridge, MA, USA, 2024; pp. 109–128. [CrossRef]
35. Salehnia, T.; MiarNaeimi, F.; Izadi, S.; Ahmadi, M.; Montazerolghaem, A.; Mirjalili, S.; Abualigah, L. Chapter 45—A MTIS method using a combined of whale and moth-flame optimization algorithms. In *Handbook of Whale Optimization Algorithm*; Academic Press: Cambridge, MA, USA, 2024; pp. 625–651. [CrossRef]
36. Montazerolghaem, A.; Khosravi, M.; Rezaee, F.; Khayyambashi, M.R. An optimal workflow scheduling method in cloud-fog computing using three-objective Harris-Hawks algorithm. In Proceedings of the 2022 12th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 17–18 November 2022; pp. 300–306. [CrossRef]
37. Salimifard, K.; Bigharaz, S. The multicommodity network flow problem: State of the art classification, applications, and solution methods. *Oper. Res.* **2022**, *22*, 1–47. [CrossRef]
38. Maher, S.J.; Fischer, T.; Gally, T.; Gamrath, G.; Gleixner, A.; Gottwald, R.L.; Hendel, G.; Koch, T.; Lübbecke, M.; Miltenberger, M.; et al. *The SCIP Optimization Suite 4.0*; Technical Report 17-12; ZIB: Berlin, Germany, 2017. Available online: https://opus4.kobv.de/opus4-zib/files/6217/scipoptsuite-401.pdf (accessed on 9 February 2025).
39. Hendel, G. Adaptive large neighborhood search for mixed integer programming. *Math. Program. Comput.* **2022**, *14*, 185–221. [CrossRef]
40. Fischetti, M.; Monaci, M. Proximity search for 0-1 mixed-integer convex programming. *J. Heuristics* **2014**, *20*, 709–731. [CrossRef]
41. Luo, H.; Kianfar, K. Valid inequalities and facets for multi-module survivable network design problem. *Networks* **2023**, *81*, 155–176. [CrossRef]
42. Guastaroba, G.; Speranza, M. A heuristic for BILP problems: The Single Source Capacitated Facility Location Problem. *Eur. J. Oper. Res.* **2014**, *238*, 438–450. [CrossRef]
43. Achterberg, T. Constraint Integer Programming. Ph.D. Thesis, Technische Universität Berlin, Berlin, Germany, 2007. Available online: https://opus4.kobv.de/opus4-zib/files/1112/Achterberg_Constraint_Integer_Programming.pdf (accessed on 9 February 2025).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.