Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 6648044, 8 pages
https://doi.org/10.1155/2021/6648044

Research Article

A Polynomial-Time Exact Algorithm for k-Depot Capacitated Vehicle Routing Problem on a Tree

Liang Xu,[1] **Tong Zhang**,[2] **Rantao Hu**,[1] **and Jialing Guo**[3]

[1]*School of Business Administration, Southwestern University of Finance and Economics, Chengdu 611130, China*
[2]*Institute of Chinese Financial Studies, Southwestern University of Finance and Economics, Chengdu 611130, China*
[3]*School of Finance, Zhongnan University of Economics and Law, Wuhan 430073, China*

Correspondence should be addressed to Liang Xu; arecxuliang1112@gmail.com

Received 12 December 2020; Revised 1 February 2021; Accepted 18 February 2021; Published 8 March 2021

Academic Editor: Lu Zhen

This paper proposes a polynomial-time exact algorithm for the k-depot capacitated vehicle routing problem on a tree for fixed k (k-depot CVRPT for short), which involves dispatching a fixed number of k capacitated vehicles in depots on a tree-shaped graph to serve customers with the objective of minimizing total distance traveled. The polynomial-time exact algorithm improves the 2-approximation algorithm when k is a constant.

1. Introduction

This article investigates routing k capacitated vehicles in fixed depots to serve customers. According to [1], this problem is equivalent to dispatching k vehicles located at distinct depots (referred to as k-depot CVRP). The k-depot CVRP can be described as follows: Given a complete, undirected graph $G = (V, E)$ with vertex set V and edge set E, there is a given depot set D with $|D| = k$, where a vehicle of capacity Q is located at each depot in D, and a nonempty set of customer locations $J = V \setminus D$. Each edge $e \in E$ is associated with a nonnegative edge weight $w(e)$. The k-depot CVRP involves determining a set of k tours to cover all the customers in J such that the total customer demand served by each vehicle does not exceed Q. The objective is to minimize the total edge weight of all tours.

If demand of a customer cannot be split, the k-depot CVRP has a feasible solution if and only if its corresponding Bin Packing Problem [2], with the number of bins equal to that of depots and bin capacity equal to vehicle capacity, has a feasible solution. Then, polynomial-time exact algorithm is unavailable even for the tree-shaped graph. According to Li and Simchi-Levi [3], if demand of a customer can be split (demand of a customer can be served by more than one vehicle), the problem reduces to a k-depot CVRP with each customer having unit demand and with vehicle capacity Q of each vehicle equal to the maximum number of customers that can be served by a vehicle. This paper considers the splittable demand case and develops polynomial-time exact algorithm under the assumption that each customer has a unit demand.

We use $\mathcal{I} = (G, D, J, w, Q)$ to denote an instance of the k-depot CVRP. A feasible solution to $\mathcal{I}$ can be defined as a k-cycle collection $\mathcal{C}$ which satisfies the following: (i) Each customer is served by exactly one cycle in $\mathcal{C}$. (ii) Number of customers served by each cycle in $\mathcal{C}$ does not exceed Q. (iii) Each cycle in $\mathcal{C}$ contains a distinct depot in D. It is noted that if there exists a k-tree collection $\mathcal{T}$ which also satisfies (i), (ii), and (iii), we call it a k-tree cover of G. Duplicating each edge in a k-tree cover $\mathcal{T}$ constitutes a k-cycle collection $\mathcal{C}$.

For the min-max k-traveling salesmen problem on a tree (min-max k-TSPT), which involves determining a set of k tours for k salesmen who have infinite capacity to serve all customers located on a tree-shaped network T, with the objective of minimizing the greatest weight among all tours, Xu et al. [4] developed a pseudo-polynomial-time exact algorithm through dynamic programming. Supposing that the

underlying tree T is a fully binary tree [4], the basic idea underlying the dynamic programming for the min-max k-TSPT is to join subtrees in T^l (the subtree of T rooted at vertex l) and T^r (a subtree of T rooted at vertex r) to form subtrees in T^v, in which l and r are the left child and right child of v, respectively. However, in the k-depot CVRPT, each vehicle has a capacity of Q. Although the dynamic programming in [4] has been extended to solve a k-TSPT under a min-sum objective in the same article, it cannot be directly extended to the capacitated case. Meanwhile, it is noted that the best approximation algorithm for the k-depot CVRPT was provided in the thesis of Hu [5] and achieved an approximation ratio of 2. The CVRPT with number of depot part of the instance has been proved to be NP-hard by [5], and the 2-approximation algorithm was proposed for this NP-hard case.

Concerning the k-depot TSP, a special case of the k-depot CVRP with infinite capacity, and the first constant approximation algorithm was obtained by [6], which achieves an approximation ratio of 2. For the same problem, Xu et al. [7] provided an extended Christofides heuristic, which is proved to achieve a tight approximation ratio of $2 - (1/k)$. To answer the question of whether or not there exists a 3/2-approximation algorithm for the k-depot TSP, Xu and Rodrigues [8] designed an edge exchange algorithm which is proved to achieve an approximation ratio of 3/2.

For the multiple-depot CVRP with unlimited vehicles, Li and Simchi-Levi [3] first provided a tour partitioning approximation algorithm for the problem and analyzed the worst-case performance ratios of the approximation algorithms. Moreover, Cardon et al. [9] also considered multiple-depot CVRP with unlimited vehicle number and extended the Polynomial Time Approximation Scheme (PTAS) for CVRP in [10] to the multiple-depot case and obtained a PTAS with its time complexity corresponding to both the number of vehicles and the vehicle capacity. For the k-depot CVRP on a general graph, which has a fixed number of k vehicles in depots, there is no constant ratio approximation algorithm as far as we know.

We propose the first, as far as we are aware, polynomial-time exact algorithm for the CVRPT with the number of depot k a constant. The proposed algorithm in this article is based on a novel network-flow-based dynamic programming rather than the existing subtree-joining-based dynamic programming in Xu et al.'s work [4]. The network-flow-based dynamic programming provides a more general framework to solve both capacitated and uncapacitated vehicle routing problems on trees. The contribution and significance of this article can be summarized as follows:

(1) This article designs a novel polynomial-time exact algorithm based on a network-flow-based dynamic programming for the k-depot CVRPT

(2) The polynomial-time exact algorithm for this problem improves the existing 2-approximation algorithm when k is a constant

2. Dynamic Programming for the k-Depot CVRPT

To simplify the presentation, we first define a "standard" instance of the k-depot CVRPT as follows.

Definition 1. An instance of the k-depot CVRPT is standard if, and only if, the underlying tree is a full binary tree, which means that each vertex in the tree other than the leaves has exactly two children.

According to [11], given any instance $\mathcal{I}$ of the k-depot CVRPT defined on an arbitrary tree, we can transform it to a standard instance $\mathcal{I}'$ of the k-depot CVRPT defined on a full binary tree as follows:

(1) Let $T' = T$ and $w' = w$.

(2) For each v in T' that has only one child, insert a new vertex v' into T' to be the second child of v by adding an edge (v, v') and setting $w'(v, v') = 0$.

(3) Consider each v in T' that has more than two children. For each child u of v, other than the first child, move u to be a child of a new vertex v' by replacing the edge (v, u) with an edge (v', u) and setting $w'(v', u) = w'(v, u)$. Insert v' into T' to be the second child of v by adding an edge (v, v') and setting $w'(v, v') = 0$.

Similar to the arguments in [11], it can be verified that any feasible solution of $\mathcal{I}$ can be transformed to a feasible solution of $\mathcal{I}'$ with the same objective value, and vice versa. Accordingly, if there exists a polynomial-time exact algorithm for $\mathcal{I}'$, then the same algorithm can be applied to obtain a polynomial-time exact algorithm for $\mathcal{I}$, and vice versa.

Based on the definition of standard instance, we assume without loss of generality that the instances in this article are standard and introduce the dynamic programming for the k-depot CVRPT for those standard instances. An outline of dynamic programming is presented in Section 2.1. The dynamic programming has two main phases, (1) flow-enumerating phase and (2) flow-splitting phase, which are introduced in Sections 2.2 and 2.3, respectively. Finally, the correctness and time complexity for the dynamic programming are proved in Section 2.4.

Furthermore, to facilitate the presentation, we can assume without loss of generality that vertices in $V = \{v_1, v_2, \ldots, v_n\}$ are labeled in a nonincreasing order on their depths in T. We assume without loss of generality that v_n is the root of T, and the tree satisfies that all the customers and depots are at the leaves [5]. Figure 1 shows an example of the mentioned tree. In the tree T, there are in total three customers, v_1, v_4, and v_6, and two depots, v_5 and v_2. Furthermore, the edge weights are shown near the edges. For each $v \in V$, let T^v denote the subtree of T that contains v and all descendants of v. In this problem instance, the vehicle capacity $Q = 2$.

2.1. Algorithm. Denote any edge e as (u, v) and u is the parent of v. Given any feasible solution to the k-depot CVRPT, let the flow $f(u, v)$ be the total customer number within T^v served by the depots outside T^v minus the total number of customers outside T^v served by depots within T^v (also called the net customer number). Thus, for a leaf vertex v which is a customer, the flow $f(u, v)$ equals 1; for a leaf vertex v which is a depot, the flow $f(u, v)$ equals $-s$ and s equals the total number of customers served by the depot. For any nonleaf vertex v, the flow $f(u, v)$ satisfies the flow conservation property [5]; that is, $f(u, v) = f(v, v_1) + f(v, v_2)$, where v_1 and v_2 denote the two children of v. Therefore, given any feasible solution $\mathscr{C}$ to the k-depot CVRPT, there exists a unique flow $f(\mathscr{C})$ corresponding to $\mathscr{C}$, in which the flow $f(e)$ equals the total net customer number on edge e. In the flow-enumerating phase of the dynamic programming, we try to figure out the flow $f(\mathscr{C}^*)$ corresponding to the optimal solution $\mathscr{C}^*$.

Let (v_i, d_i) be the unique edge connected to depot d_i for $i = 1, 2, \ldots, k$. Since the flow $f^*(e)$ on each edge e, connected to a customer vertex, equals 1 in any optimal solution $\mathscr{C}^*$, the flow $f(\mathscr{C}^*)$ can be found by an enumeration of flows on $(v_1, d_1), (v_2, d_2), \ldots, (v_k, d_k)$. In an optimal solution, since the total net customer number $f^*(v_i, d_i)$ on (v_i, d_i) belongs to a bounded set $\{0, 1, \ldots, Q\}$, the flows $[f^*(v_1, d_1), f^*(v_2, d_2), \ldots, f^*(v_k, d_k)]$ also belong to a bounded set $\{0, 1, \ldots, Q\}^k$. Based on the flows on all edges connected to the leaves, the flow $f(\mathscr{C}^*)$ can be constructed according to the flow conservation property.

Then, after we have obtained the flow $f(\mathscr{C}^*)$, in the flow-splitting phase, we further figure out the number of customers served in an optimal solution, via a specific edge e, by the depot d_i for $i = 1, 2, \ldots, k$. For each edge $e = (u, v)$, we associate a state $[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)]$ with e, where $f_i(u, v)$ denotes the net customer number served by d_i in T^v for $i = 1, 2, \ldots, k$. Noticing that, given any T^v, the depot d_i is either outside T^v or inside T^v, we can conclude that $f_i(u, v) \geq 0$ when d_i is outside T^v and $f_i(u, v) \leq 0$ when d_i is inside T^v. Furthermore, according to the definition of $f(u, v)$, we have $f(u, v) = \sum_{i=1}^{k} f_i(u, v)$. For any state $[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)]$, we define its state value $g[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)]$ as the minimum total edge length traversed by all vehicles with the ith vehicle serving $f_i(u, v)$ net customers within T^v for $i = 1, 2, \ldots, k$ (including the length traversed on (u, v)). Then, one can notice that, for any leaf vertex v, if v is a customer, it can be served by exactly one depot, and thus the state $[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)]$ satisfies $\sum_{i=1}^{k} f_i(u, v) = 1$; if v is a depot, the state $[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)]$ satisfies $f_i(u, v) = -s_i$ for a certain i, where s_i denotes the number of customers served by d_i. Therefore, the states for any edge (u, v) connected to a leaf vertex v can be computed by a boundary procedure, and the states are further added to a state collection $\mathscr{H}^{(v)}$. Based on the state collections for all the leaves, the dynamic programming constructs a state collection $\mathscr{H}^{(v)}$ for any nonleaf vertex v in an increasing order, iteratively. In such a manner, when we construct the state collection for a nonleaf vertex v, the state collections $\mathscr{H}^{(v_1)}$ and $\mathscr{H}^{(v_2)}$ have been constructed for the two children of v, respectively. Then, the state collection $\mathscr{H}^{(v)}$ can be constructed based on $\mathscr{H}^{(v_1)}$ and $\mathscr{H}^{(v_2)}$, by a state recursion procedure.

After all the state collections $\mathscr{H}^{(v)}$ have been constructed for $v = v_1, v_2, \ldots, v_n$, the optimal solution to the k-depot CVRPT can be returned as the "feasible" state with minimum state value in $\mathscr{H}^{(v_n)}$.

The details of the dynamic programming can be found in Algorithm 1.

Note that the feasible solution returned by Algorithm 1 is a k-tree cover $\mathscr{T}$. Therefore, the optimal objective value to $\mathscr{I}$ is exactly twice the value of the minimum length of k-tree cover of T.

2.2. Flow-Enumerating Phase. As mentioned before, we present in this section a flow-enumerating procedure, denoted as Flow(T, D), which finds the flow $f(\mathscr{C}^*)$.

Procedure 1. Flow-enumerating procedure, Flow(T, D).

Input: *A tree T and a depot set D.*

Output: *A collection $\mathbb{F}$ of flows f.*

(1) *Let* $\mathbb{F} = \varnothing$.
(2) *for each* $(s_1, s_2, \ldots, s_k) \in \{0, 1, \ldots, Q\}^k$ *such that* $\sum_{i=1}^{k} s_i = |J|$ *do*
(3) *(Construction of flows on edges connected to customers)*
(4) *for each customer v in T do*
(5) *Let $f(u, v) = 1$, where u is the parent of v.*
(6) *end for*
(7) *(Construction of flows on edges connected to depots)*
(8) *for $i = 1, \ldots, k$ do*
(9) *Let $f(u, d_i) = -s_i$, where u is the parent of d_i.*
(10) *end for*
(11) *(Construction of flows on edges connected to nonleaf vertices)*
(12) *Construct the flow $f(s_1, s_2, \ldots, s_k) = \{f(e) | e \in E(T)\}$, by the flow conservation property, i.e., the flow on $f(u, v)$ is calculated by $f(v, v_1) + f(v, v_2)$, where v_1 and v_2 are two children of v.*
(13) *Add the flow $f(s_1, s_2, \ldots, s_k)$ into $\mathbb{F}$.*
(14) *end for*
(15) *Return $\mathbb{F}$.*

In the following lemma, it is stated that the flow $f(\mathscr{C}^*)$ corresponding to $\mathscr{C}^*$ is in $\mathbb{F}$.

Lemma 1. *The flow $f(\mathscr{C}^*)$ corresponding to $\mathscr{C}^*$ is in $\mathbb{F}$, and the flow collection $\mathbb{F}$ returned by Procedure 1 has a cardinality in $O(Q^k)$.*

Proof. Consider the flow $f(\mathscr{C}^*)$. First, for any edge e connected to a customer vertex, the flow on e is equal to one. Second, for any edge e connected to a depot d_i, the flow on e

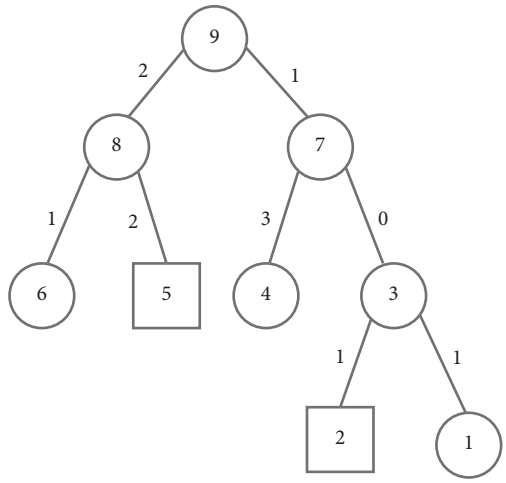

Figure 1: Example to illustrate the tree in the problem instance.

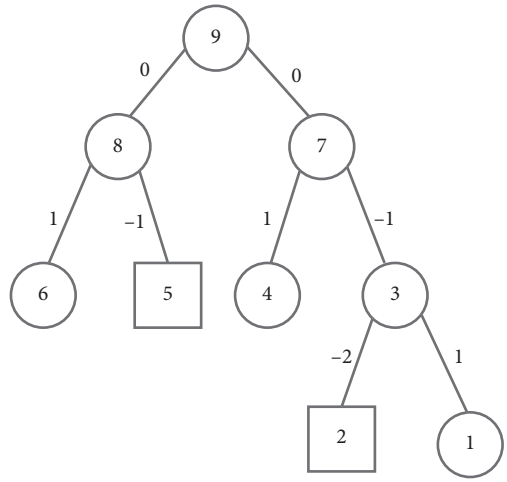

Figure 2: The flow constructed from $\{s_1, s_2\} = \{2, 1\}$ by the flow-enumerating procedure.

is equal to s_i^*, where s_i^* denotes the total number of customers served by d_i in $\mathscr{C}^*$. Thus, since $s_i^* \le Q$ for $i = 1, 2, \ldots, k$, according to Step 8-Step 10 of Procedure 1, there exists a flow $f(s_1^*, s_2^*, \ldots, s_k^*)$ in $\mathbb{F}$ such that flow $f(v_i, d_i)$ satisfies $f(v_i, d_i) = s_i^*$ for $i = 1, 2, \ldots, k$, where v_i denotes the parent of depot d_i for $i = 1, 2, \ldots, k$. Meanwhile, due to Step 12, according to the flow conservation property, it can be verified that $f(u, v)$ equals the net customer number in T^v in the optimal solution $\mathscr{C}^*$. Meanwhile, consider the k edges connected to D. Procedure 1 constructs at most Q^k flow combinations on the k edges. Therefore, the cardinality of $\mathbb{F}$ can be seen to be $O(Q^k)$. □

Example 1. Consider the tree shown in Figure 1. Let $D = \{d_1, d_2\} = \{v_2, v_5\}$, $J = \{v_1, v_4, v_6\}$, and $Q = 2$. We apply the flow-enumerating procedure to construct the flow collection $\mathbb{F}$.

The flow-enumerating procedure is shown for the case in which $\{s_1, s_2\} = \{2, 1\}$, where $f(v_3, v_2) = -2$ and $f(v_8, v_5) = -1$. First, for the vertex v_1, let $f(v_3, v_1) = 1$. Similarly, the flow-enumerating procedure sets $f(v_3, v_2) = 1$, $f(v_7, v_4) = 1$, and $f(v_8, v_6) = 1$. Second, for $d_1 = v_2$, let $f(v_3, v_2) = -2$. Similarly, the flow-enumerating procedure sets $f(v_8, v_5) = -1$. Third, for v_3, noticing that v_2 and v_1 are two children of v_3, the flow $f(v_7, v_3)$ is constructed by the flow conservation property; that is, $f(v_7, v_3) = f(v_3, v_2) + f(v_3, v_1) = -1$. For v_7, noticing that v_4 and v_3 are two children of v_7, the flow $f(v_9, v_7)$ is constructed by the flow conservation property; that is, $f(v_9, v_7) = f(v_7, v_4) + f(v_7, v_3) = 0$. Similarly, we can obtain that $f(v_9, v_8) = 0$. Therefore, the flow $f(2, 1)$ in $\mathbb{F}$ is shown in Figure 2, in which the flows are indicated near the edges.

For the case $\{s_1, s_2\} = \{1, 2\}$, the flow constructed by the flow-enumerating procedure is shown in Figure 3.

Since the capacity $Q = 2$ and the customer number $|J| = 3$, it can be seen that any feasible solution to $\mathscr{I}$ satisfies that $\{s_1, s_2\} = \{2, 1\}$ or $\{s_1, s_2\} = \{1, 2\}$. In summary, the flow collection $\mathbb{F}$ contains the flows $f(2, 1)$ and $f(1, 2)$, as indicated in Figures 2 and 3.

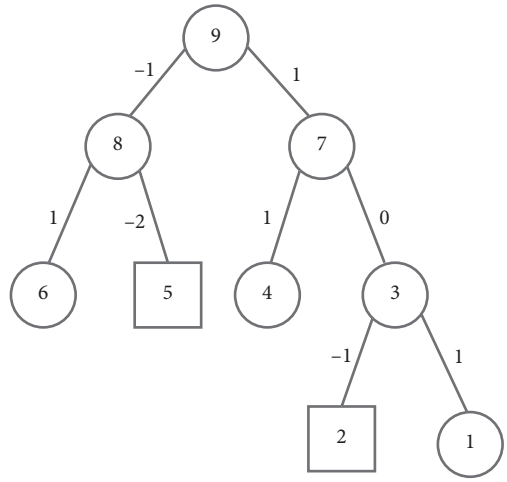

Figure 3: The flow constructed from $\{s_1, s_2\} = \{1, 2\}$ by the flow-enumerating procedure.

2.3. Flow-Splitting Phase. In this section, we present the procedures developed to construct $\mathscr{H}^{(v)}$ for a leaf vertex and a nonleaf vertex, respectively.

Procedure 2. Boundary procedure, Bound$(v, \mathbb{F})$.

Input: *A leaf vertex v and a given flow collection $\mathbb{F}$.*

Output: *A state collection $\mathscr{H}^{(v)}$.*

(1) *Let* $\mathscr{H}^{(v)} = \varnothing$.

(2) **for** *any* $f(s_1, s_2, \ldots, s_k) \in \mathbb{F}$ **do**

(3) **if** *v is a depot and $v = d_i$* **then**

(4) *Add* $[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)]$ *to* $\mathscr{H}^{(v)}$ *with* $f_i(u, v) = s_i$ *and* $f_j(u, v) = 0$ *for any* $j \ne i$. *Let the state value* $g[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)] = w(u, v)$ *if* $s_i > 0$; *and let the state value to be zero if* $s_i = 0$.

(5) **else**

(6) *Add* $[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)]$ *to* $\mathscr{H}^{(v)}$ *with* $f_i(u,v) \in \{0,1\}$ *for* $i = 1,2,\ldots,k$ *and* $\sum_{i=1}^{k} f_i(u,v) = 1$. *Let the state value* $g[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)] = w(u,v)$.

(7) **end if**

(8) **end for**

(9) *Return* $\mathscr{H}^{(v)}$.

Example 2. Let us continue Example 1. We construct the state collection $\mathscr{H}^{(v)}$ for all the leave vertices, v_1, v_2, v_4, v_5, v_6, respectively. Note that the flow collection $\mathbb{F}$ contains two flows $f(2,1)$ and $f(1,2)$. For $v = v_1$, since v_1 is a customer, for $f(2,1)$, the boundary procedure adds two states $\{[0,1],[1,0]\}$ into $\mathscr{H}^{(v_1)}$. Meanwhile, for $f(1,2)$, the boundary procedure adds the same states $\{[0,1],[1,0]\}$ into $\mathscr{H}^{(v_1)}$. Therefore, the state collection $\mathscr{H}^{(v_1)}$ contains two states $\{[0,1],[1,0]\}$ with the state values $g[0,1] = 1$ and $g[1,0] = 1$. For $v = v_2$, since v_2 is a depot where the first vehicle is located and $v_2 = d_1$, for $f(1,2)$, the boundary procedure adds $[-1,0]$ into $\mathscr{H}^{(v_2)}$; for $f(2,1)$, the boundary procedure adds $[-2,0]$ into $\mathscr{H}^{(v_2)}$. Therefore, the state collection $\mathscr{H}^{(v_2)}$ contains two states $\{[-1,0],[-2,0]\}$ with the state values $g[-1,0] = 1$ and $g[-2,0] = 1$. Similarly, one can obtain all the state collections $\mathscr{H}^{(v)}$ for $v = v_1, v_2, v_4, v_5, v_6$ in Table 1.

Next, we present the state recursion procedure. For any nonleaf vertex $v \in V$, it constructs $\mathscr{H}^{(v)}$ based on $\mathscr{H}^{(v_1)}$ and $\mathscr{H}^{(v_2)}$, where v_1 and v_2 are the two children of v.

Procedure 3. State recursion procedure, Recursion $(v, \mathscr{H}^{(v_1)}, \mathscr{H}^{(v_2)})$.

Input: *A non-leaf vertex* $v \in V$, $\mathscr{H}^{(v_1)}$ *and* $\mathscr{H}^{(v_2)}$, *where* v_1 *and* v_2 *are two children for* v.

Output: *A state collection* $\mathscr{H}^{(v)}$.

(1) *Let* $\mathscr{H}^{(v)} = \varnothing$.

(2) **for** *any* $(S_1, S_2) \in \mathscr{H}^{(v_1)} \times \mathscr{H}^{(v_2)}$, *in which* $S_1 = [f_1(v,v_1), f_2(v,v_1), \ldots, f_k(v,v_1)]$ *and* $S_2 = [f_1(v,v_2), f_2(v,v_2), \ldots, f_k(v,v_2)]$ **do**

(3) **for** $j = 1,2,\ldots,k$ **do**

(4) $f_j(u,v) = f_j(v,v_1) + f_j(v,v_2)$.

(5) **end for**

(6) *Determine the state value for* $[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)]$ *as* $\sum_{j=1}^{k} \left\lceil |f_j(u,v)|/Q \right\rceil w(u,v) + \sum_{i=1}^{2} g[f_1(v,v_i), f_2(v,v_i), \ldots, f_k(v,v_i)]$.

(7) *Add* $[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)]$ *to* $\mathscr{H}^{(v)}$ *if it satisfies* $|f_j(u,v)| \le Q$ *for* $j = 1,2,\ldots,k$, *and there exists no state* $[f_1'(u,v), f_2'(u,v), \ldots, f_k'(u,v)]$ *in* $\mathscr{H}^{(v)}$ *with* $f_j(u,v) = f_j'(u,v)$ *for* $j = 1,2,\ldots,k$ *and* $[f_1'(u,v), f_2'(u,v), \ldots, f_k'(u,v)]$ *has a smaller state value.*

(8) **end for**

(9) *Return* $\mathscr{H}^{(v)}$.

Example 3. Let us apply the state recursion procedure to construct $\mathscr{H}^{(v_3)}$, $\mathscr{H}^{(v_7)}$, $\mathscr{H}^{(v_8)}$, and $\mathscr{H}^{(v_9)}$, respectively. First, we construct $\mathscr{H}^{(v_3)}$ based on $\mathscr{H}^{(v_1)}$ and $\mathscr{H}^{(v_2)}$. In Table 2, all combinations of $(S_1, S_2) \in \mathscr{H}^{(v_1)} \times \mathscr{H}^{(v_2)}$ are shown in the first column, the states constructed based on (S_1, S_2) are shown in the second column, and their state values are indicated in the third column.

Furthermore, based on $\mathscr{H}^{(v_3)}$ and $\mathscr{H}^{(v_4)}$, the state collection $\mathscr{H}^{(v_7)}$ can be constructed as depicted in Table 3 (those states with "*" are kept in $\mathscr{H}^{(v_7)}$).

Similarly, we can construct $\mathscr{H}^{(v_8)}$ as in Table 4, in which $(S_1, S_2) \in \mathscr{H}^{(v_6)} \times \mathscr{H}^{(v_5)}$ are shown in the first column.

Finally, we can construct $\mathscr{H}^{(v_9)}$ based on $\mathscr{H}^{(v_7)}$ and $\mathscr{H}^{(v_8)}$. Since $v_n = v_9$, we add one more vertex v_{10} into the graph with $w(v_{10}, v_9) = 0$ and $f(v_{10}, v_9) = 0$. Next, we construct the state collection $\mathscr{H}^{(v_9)}$ as depicted in Table 5.

According to Algorithm 1, only those flows $[f_1(v_{10}, v_9), f_2(v_{10}, v_9)]$ with $f_1(v_{10}, v_9) = 0$ and $f_2(v_{10}, v_9) = 0$ are added to $\mathscr{H}^{(v_9)}$. Then, the two states with "*" are kept. Since the minimum state value in the state collection $\mathscr{H}^{(v_9)}$ is 8, Algorithm 1 returns an optimal objective value as 8. Since the minimum total edge length for serving customers in J equals 8, by the fact that each edge is traversed exactly twice in any tour in a tree, the optimal solution to the instance can be seen to be 16.

2.4. Correctness and Time Complexity for Dynamic Programming. We first establish Lemma 2 to prove the feasibility of all the states in $\mathscr{H}^{(v_n)}$.

Lemma 2. *For each* $v = v_1, v_2, \ldots, v_n$, *and for each state* $[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)] \in \mathscr{H}^{(v)}$ *obtained in Algorithm 1, it satisfies that (i) each customer in* T^v *is served by exactly one depot and (ii) each depot serves no more than* Q *customers.*

Proof. First, according to Step 6 of the boundary procedure, it can be verified that each customer is served by exactly one depot. Second, according to Step 7 of the state recursion procedure, it can be verified that each depot serves no more than Q customers. Thus, (i) and (ii) can be verified for $v = v_1, v_2, \ldots, v_{n-1}$. For the case $v = v_n$, since $f_i(v_{n+1}, v_n) = 0$ $(\sum_j f_i(v_n, v_j) = 0)$ for $i = 1,2,\ldots,k$, one can see that the total customer number assigned to d_i equals s_i, where s_i denotes the flow on (u, d_i) in the flow $f(s_1, s_2, \ldots, s_k)$, which implies that each state in $\mathscr{H}^{(v_n)}$ corresponds to a feasible solution to the k-depot CVRPT.

We then establish Lemma 3 to prove the optimality of Algorithm 1, which, together with Lemma 2, will be used to prove the correctness of Algorithm 1 in Theorem 1. □

Lemma 3. *Consider* $f_1(\mathscr{C}^*), f_2(\mathscr{C}^*), \ldots, f_k(\mathscr{C}^*)$, *in which* $f_i(\mathscr{C}^*)$ *denotes the flow corresponding to* d_i *in* $f(\mathscr{C}^*)$. *For each* $v = v_1, v_2, \ldots v_n$, *there exists a state* $[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)] \in \mathscr{H}^{(v)}$, *obtained by Algorithm 1, such that* $f_i(u,v) = f_i^*(u,v)$ *for* $i = 1,2,\ldots,k$ *and the state value* $g[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)]$ *is not greater than* $g[f_1^*(u,v), f_2^*(u,v), \ldots, f_k^*(u,v)]$, *where*

Input: *Any instance* $\mathcal{I} = (T, D, J, w, Q)$ *to the k-depot CVRPT on a tree* T.
Output: *A minimum length over all k-tree cover of* T.
(1) (Flow-enumerating Phase)
(2) Apply the flow-enumerating procedure Flow(T, D) to return a flow collection $\mathbb{F}$.
(3) (Flow-splitting Phase)
(4) **for** $v = v_1, v_2, \ldots, v_{n-1}$ **do**
(5) Set $\mathcal{H}^{(v)}$ to be empty.
(6) **if** v is a leaf **then**
(7) Construct the state collection $\mathcal{H}^{(v)}$ by the boundary procedure Bound$(v, \mathbb{F})$.
(8) **else**
(9) Construct the state collection $\mathcal{H}^{(v)}$ by the state recursion procedure Recursion$(v, \mathcal{H}^{(v_1)}, \mathcal{H}^{(v_2)})$, where v_1 and $v_2$are two children of v.
(10) **end if**
(11) **end for**
(12) (Optimal solution return phase)
(13) When $v = v_n$, add a new vertex v_{n+1} to the graph with $w(v_{n+1}, v_n) = 0$ and $f(v_{n+1}, v_n) = 0$. Construct $\mathcal{H}^{(v_n)}$ by the state recursion procedure Recursion$(v, \mathcal{H}^{(v_1)}, \mathcal{H}^{(v_2)})$, where v_1 and $v_2$denote the two children of v_n. Among all states $S = [f_1(v_{n+1}, v_n), f_2(v_{n+1}, v_n), \ldots, f_k(v_{n+1}, v_n)]$ in $\mathcal{H}^{(v_n)}$, with $f_i(v_{n+1}, v_n) = 0$ for $i = 1, 2, \ldots, k$, return the objective value as the minimum state value.

Algorithm 1: The dynamic programming algorithm for the k-depot CVRPT.

Table 1: Construction of $\mathcal{H}^{(v)}$ for all the leaves.

State collection	State	State value
$\mathcal{H}^{(v_6)}$	$[0, 1]$	1
	$[1, 0]$	1
$\mathcal{H}^{(v_5)}$	$[0, -2]$	2
	$[0, -1]$	2
$\mathcal{H}^{(v_4)}$	$[0, 1]$	3
	$[1, 0]$	3
$\mathcal{H}^{(v_2)}$	$[-1, 0]$	1
	$[-2, 0]$	1
$\mathcal{H}^{(v_1)}$	$[0, 1]$	1
	$[1, 0]$	1

Table 2: Construction of $\mathcal{H}^{(v_3)}$.

(S_1, S_2)	State	State value
$[0, 1], [-2, 0]$,	$[-2, 1]$	2
$[1, 0], [-2, 0]$	$[-1, 0]$	2
$[1, 0], [-1, 0]$	$[0, 0]$	2
$[0, 1], [-1, 0]$	$[-1, 1]$	2

Table 3: Construction of $\mathcal{H}^{(v_7)}$.

(S_1, S_2)	State	State value
$[0, 1], [-1, 0]$,	$[-1, 1]^*$	7
$[1, 0], [-1, 0]$	$[0, 0]^*$	5
$[1, 0], [-2, 1]$	$[-1, 1]$	7
$[0, 1], [-2, 1]$	$[-2, 2]^*$	7
$[0, 1], [-1, 1]$,	$[-1, 2]^*$	7
$[1, 0], [-1, 1]$	$[0, 1]^*$	6
$[1, 0], [0, 0]$	$[1, 0]^*$	6
$[0, 1], [0, 0]$	$[0, 1]$	6

Table 4: Construction of $\mathcal{H}^{(v_8)}$.

(S_1, S_2)	State	State value
$[0, 1], [0, -2]$	$[0, -1]^*$	5
$[1, 0], [0, -2]$	$[1, -2]^*$	7
$[0, 1], [0, -1]$	$[0, 0]^*$	3
$[1, 0], [0, -1]$	$[1, -1]^*$	7

$f_i^*(u, v)$ denotes the flow in $f_i(\mathcal{C}^*)$ on the edge (u, v), and $g[f_1^*(u, v), f_2^*(u, v), \ldots, f_k^*(u, v)]$ denotes the shortest length traversed by all the k vehicles with the ith vehicle serving $f_i^*(u, v)$ net customers in T^v for $i = 1, 2, \ldots, k$.

Proof. We prove Lemma 3 by induction. For each leaf v which is a customer, we know that $f_i^*(u, v) \in \{0, 1\}$ for $i = 1, 2, \ldots, k$ and $\sum_i f_i^*(u, v) = 1$, which implies that $[f_1^*(u, v), f_2^*(u, v), \ldots, f_k^*(u, v)] \in \mathcal{H}^{(v)}$. For each leaf v which is a depot, we can conclude that $[f_1^*(u, v), f_2^*(u, v), \ldots, f_k^*(u, v)] \in \mathcal{H}^{(v)}$ according to Lemma 1. Furthermore, since $f_i(u, v) = f_1^*(u, v)$ for $i = 1, 2, \ldots, k$, according to the definition of $g[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)]$ and $g[f_1^*(u, v), f_2^*(u, v), \ldots, f_k^*(u, v)]$, we have $g[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)] \leq g[f_1^*(u, v), f_2^*(u, v), \ldots, f_k^*(u, v)]$, which implies that the lemma holds for any leaf vertex v

Next, consider any $v \in V$ that is not a leaf. Let v_1 and v_2 denote the two children of v. Assume that Lemma 3 holds for each child of v, denoted as v_1 and v_2. We are going to show as follows that Lemma 3 holds for v.

First, for any v_i for $i = 1, 2$, there exists a state $[f_1(v, v_i), f_2(v, v_i), \ldots, f_k(v, v_i)] \in \mathcal{H}^{(v_i)}$ such that, for $j = 1, 2, \ldots, k$, $f_j(v, v_i) = f_j^*(v, v_i)$ and $g[f_1(v, v_i), f_2(v, v_i), \ldots, f_k(v, v_i)] \leq g[f_1^*(v, v_i), f_2^*(v, v_i), \ldots, f_k^*(v, v_i)]$. Since $\sum_{i=1}^{2} f_j(v, v_i) = \sum_{i=1}^{2} f_j^*(v, v_i)$ for $j = 1, 2, \ldots, k$, we can conclude that there exists a state $[f_1(u, v), f_2(u, v), \ldots, f_k(u, v)]$ in $\mathcal{H}^{(v)}$ such that $f_i(u, v) = f_i^*(u, v)$ for $i = 1, 2, \ldots, k$. Then, it remains to prove that

Table 5: Construction of $\mathscr{H}^{(v_9)}$.

(S_1, S_2)	State	State value
$[0,0]$, $[0,-1]$	$[0,-1]$	–
$[0,0]$, $[1,-2]$	$[1,-2]$	–
$[0,0]$, $[1,-1]$	$[1,-1]$	–
$[0,0]$, $[0,0]$	$[0,0]^*$	8
$[-1,1]$, $[0,-1]$	$[-1,0]$	–
$[-1,1]$, $[1,-2]$	$[0,-1]$	–
$[-1,1]$, $[1,-1]$	$[0,0]$	14
$[-1,1]$, $[0,0]$	$[-1,1]$	–
$[-2,2]$, $[0,-1]$	$[-2,1]$	–
$[-2,2]$, $[1,-2]$	$[-1,0]$	–
$[-2,2]$, $[1,-1]$	$[-1,1]$	–
$[-2,2]$, $[0,0]$	$[-2,2]$	–
$[1,0]$, $[0,-1]$	$[1,-1]$	–
$[1,0]$, $[1,-2]$	$[2,-2]$	–
$[1,0]$, $[1,-1]$	$[2,-1]$	–
$[1,0]$, $[0,0]$	$[1,0]$	–
$[-1,2]$, $[0,-1]$	$[-1,1]$	–
$[-1,2]$, $[1,-2]$	$[0,0]^*$	14
$[-1,2]$, $[1,-1]$	$[0,1]$	–
$[-1,2]$, $[0,0]$	$[-1,2]$	–
$[0,1]$, $[0,-1]$	$[0,0]^*$	11
$[0,1]$, $[1,-2]$	$[1,-1]$	–
$[0,1]$, $[1,-1]$	$[1,0]$	–
$[0,1]$, $[0,0]$	$[0,1]$	–

$g[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)] \leq g[f_1^*(u,v), f_2^*(u,v), \ldots, f_k^*(u,v)]$.

According to the calculation of $g[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)]$ in Step 6 of Procedure 3, it satisfies

$$\begin{aligned} & g[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)] \\ &= \sum_{j=1}^{k} \left\lceil \left| f_j(u,v) \right| / Q \right\rceil w(u,v) + \sum_{i=1}^{2} g[f_1(v,v_i), f_2(v,v_i), \ldots, f_k(v,v_i)] \\ &\leq \sum_{j=1}^{k} \left\lceil \left| f_j(u,v) \right| / Q \right\rceil w(u,v) + \sum_{i=1}^{2} g[f_1^*(v,v_i), f_2^*(v,v_i), \ldots, f_k^*(v,v_i)] \\ &\leq \sum_{j=1}^{k} c^*(u,v) w(u,v) + \sum_{i=1}^{2} g[f_1^*(v,v_i), f_2^*(v,v_i), \ldots, f_k^*(v,v_i)] \\ &= g[f_1^*(u,v), f_2^*(u,v), \ldots, f_k^*(u,v)], \end{aligned} \tag{1}$$

in which the first inequality holds by the induction, and the second inequality holds by the fact that $\lceil |f_j(u,v)|/Q \rceil$ is the lower bound for the vehicle in d_j to serve $|f_j(u,v)|$ net customers for all vehicles within T^v and $c^*(u,v)$ denotes the number of times for the vehicle in d_j traversing on (u,v) in the optimal solution $\mathscr{C}^*$ (each edge is traversed at most once by one vehicle in an optimal solution).

Based on Lemma 3, the optimality for Algorithm 1 is proved. Notice that, for each leaf vertex v, at most Q^k states $[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)]$ are kept in $\mathscr{H}^{(v)}$ because the cardinality of $\mathbb{F}$ is at most Q^k. Meanwhile, for any nonleaf vertex v, since each $f_i(u,v)$ has at most Q possibilities for $i = 1, 2, \ldots, k$ and thus at most Q^k states $[f_1(u,v), f_2(u,v), \ldots, f_k(u,v)]$ are kept in $\mathscr{H}^{(v)}$, the maximum number of states kept in Algorithm 1 is $O(Q^k)$. Then, the following theorem holds. □

Theorem 1. *Algorithm 1 returns an optimal solution to any given instance of the k-depot CVRPT in $O(nQ^k)$ time.*

Since $Q \leq n$, one can see that Algorithm 1 is a polynomial-time exact algorithm for the k-depot CVRPT.

3. Conclusion

In this article, we provide the polynomial-time exact algorithm for the k-depot CVRPT. We can see that, with $Q = n$, the polynomial-time exact algorithm for the k-depot CVRPT

can be applied to solve the k-depot TSPT. It is worth noting that although the dynamic programming devised in this paper can solve the k-depot CVRPT and k-depot TSPT with a min-sum objective, it cannot be applied directly to the min-max k-depot TSPT in Xu et al.'s work [4].

Furthermore, the exact algorithm may set a basis for developing approximation algorithms for k-depot CVRP on general graphs. The approximation algorithms can be designed as in the following two steps: (1) Find a capacitated constrained spanning forest, in which each tree in the forest contains enough vehicles to serve all the customers within the tree. (2) Apply the exact algorithm for the k-depot CVRPT to find an optimal solution for the subinstance restricted on each tree, respectively. Finally, the optimal solution on the capacitated constrained spanning forest is returned as the approximated solution to the k-depot CVRP. However, we can see that developing approximation algorithms for the k-depot CVRP has the following two difficulties: (1) Polynomial-time exact algorithm for finding a capacitated constrained spanning forest is unknown in the literature. (2) How to bound the total edge weight for the optimal solution to the k-depot CVRPT remains an issue.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant nos. 71971177 and U1811462.

References

[1] A. Lim and W. Zhu, "A fast and effective insertion algorithm for multi-depot vehicle routing problem with fixed distribution of vehicles and a new simulated annealing approach," in *Advances in Applied Artificial Intelligence*, M. Ali and R. Dapoigny, Eds., pp. 282–291, Springer, Berlin, Germany, 2006.

[2] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, JSTOR, San Francisco, CA, USA, 1979.

[3] C.-L. Li and D. Simchi-Levi, "Worst-case analysis of heuristics for multidepot capacitated vehicle routing problems," *ORSA Journal on Computing*, vol. 2, no. 1, pp. 64–73, 1990.

[4] L. Xu, Z. Xu, and D. Xu, "Exact and approximation algorithms for the min-max k-traveling salesmen problem on a tree," *European Journal of Operational Research*, vol. 227, no. 2, pp. 284–292, 2013.

[5] Y. Hu, *Approximation algorithms for the capacitated vehicle routing problem*, PhD Thesis, Simon Fraser University, Canada, 2009.

[6] S. Rathinam, R. Sengupta, and S. Darbha, "A resource allocation algorithm for multi-vehicle systems with non holonomic constraints," *IEEE Transactions on Automation Sciences and Engineering*, vol. 4, no. 1, pp. 98–104, 2006.

[7] Z. Xu, L. Xu, and B. Rodrigues, "An analysis of the extended Christofides heuristic for the -depot TSP," *Operations Research Letters*, vol. 39, no. 3, pp. 218–223, 2011.

[8] X. Zhou and B. Rodrigues, "A 3/2-approximation algorithm for the multiple TSP with a fixed number of depots," *INFORMS Journal on Computing*, vol. 27, no. 4, pp. 636–645, 2015.

[9] S. Cardon, S. Dommers, C. Eksin, A. Sitters, and L. Stougie, *A PTAS for the Multiple Depot Vehicle Routing problem*, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2009.

[10] M. Haimovich and A. H. G. Rinnooy Kan, "Bounds and heuristics for capacitated routing problems," *Mathematics of Operations Research*, vol. 10, no. 4, pp. 527–542, 1985.

[11] A. Tamir, "An O(pn2) algorithm for the p-median and related problems on tree graphs," *Operations Research Letters*, vol. 19, no. 2, pp. 59–64, 1996.