Optim Lett (2016) 10:1269–1285
DOI 10.1007/s11590-015-0924-1

ORIGINAL PAPER

A primal-dual approximation algorithm for a two depot heterogeneous traveling salesman problem

Jungyun Bae[1] · **Sivakumar Rathinam**[1]

Received: 21 May 2013 / Accepted: 25 June 2015 / Published online: 2 August 2015

Abstract Surveillance applications require a collection of heterogeneous vehicles to visit a set of targets. We consider a fundamental routing problem that arises in these applications involving two vehicles. Specifically, we consider a routing problem where there are two heterogeneous vehicles that start from distinct initial locations and a set of targets. The objective is to find a tour for each vehicle such that each of the targets is visited at least once by a vehicle and the sum of the distances traveled by the vehicles is minimal. We consider an important special case of this routing problem where the travel costs satisfy the triangle inequality and the following monotonicity property: the first vehicle's cost of traveling between any two targets is at most equal to the second vehicle's cost of traveling between the same targets. We present a primal-dual algorithm for this case that provides an approximation ratio of 2.

Keywords Approximation algorithms · Primal-dual method · Traveling salesman problem

1 Introduction

Heterogeneous unmanned vehicles are commonly used in surveillance applications for monitoring and tracking a set of targets. For example, in the Cooperative Operations in Urban Terrain project [2] at the Air Force Research Laboratory, a team of unmanned vehicles are required to monitor a set of targets and send information/videos about the targets to the ground station. A human operator enters the locations of the targets through a human-machine interface, and the central computer associated with the

✉ Sivakumar Rathinam
srathinam@tamu.edu

[1] 3123 TAMU, College Station, TX 77843, USA

interface has a few minutes to determine the motion plans for each of the vehicles. A fundamental subproblem that has to be solved by this computer is the problem of finding a tour for each vehicle so that each target is visited at least once by some vehicle and the sum of the distances traveled by all of the vehicles is minimal. This routing problem is known as the traveling salesman problem (TSP) in the case where there is only one vehicle. In the case where there are multiple vehicles that possibly start from different initial locations or depots, this routing problem is known as the Multiple Depot TSP. Once the routing problem is solved and the tours have been determined, a nominal trajectory can be specified for each vehicle that includes other kinematic constraints of the vehicles, using the results in [6].

A multiple depot TSP is a generalization of the TSP, and thus it is NP-hard. This routing problem is further complicated if the vehicles involved are heterogeneous. In this article, vehicles are heterogeneous because the distance to be traveled between any two targets depends on the type of the vehicle used. In the context of unmanned applications, which generally require solving a multiple depot heterogeneous TSP as a subproblem, we are interested in developing fast algorithms that produce approximate solutions rather than finding optimal solutions, which may be relatively difficult. Therefore, the main focus of this article is on developing approximation algorithms for heterogeneous TSPs. An approximation algorithm for a problem is an algorithm that runs in polynomial time and produces a solution whose cost is at most a given factor away from the optimal cost for every instance of the problem.

The objective of this article is to develop a primal-dual algorithm for a two depot heterogeneous TSP (TDHTSP). In addition to assuming that the costs satisfy the triangle inequality for each vehicle, we assume that the travel costs satisfy the following monotonicity property: the first vehicle's cost of traveling between any two targets is at most equal to the second vehicle's cost of traveling between the same targets. Using these assumptions, we show that our proposed primal-dual algorithm has an approximation ratio of 2. We are motivated to address this case of the TDHTSP for the following reasons:

1. The TDHTSP is one of the simplest cases of the general multiple depot heterogeneous TSP. The objective of this work is to develop a good algorithm that can handle this simple case efficiently.
2. Consider a scenario where each of the vehicles is modeled as a ground robot that can move both forwards and backwards with a constraint on its minimum turning radius [7]. If the approach angle at each target is given, the optimal distance required to travel between any two targets for the vehicles can be computed using the result in [7]. In addition, if the minimum turning radius of the first vehicle is at most equal to the minimum turning radius of the second vehicle, then the optimal distance required for the first vehicle to travel between any two targets will be at most equal to that of the second vehicle.
3. In scenarios where the travel cost is defined as the travel time (D_{ij}/v_k, where D_{ij} is the Euclidean distance between targets i and j and v_k is the speed of the kth vehicle), the travel costs satisfy the monotonicity property if the vehicles are sorted such that $v_1 \geq v_2$.

4. In scenarios where vehicles have fuel constraints, it is possible that the vehicles need to refuel by revisiting the depots. The cost of traveling between any two target locations including the refueling stops will increase as the fuel capacity of a vehicle decreases [8]. Hence, given a collection of vehicles with different fuel capacities, one can order the vehicles based on their decreasing fuel capacities such that travel costs satisfy the monotonicity property.

Without the assumptions about the costs of the two vehicles, the TDHTSP is a generalization of the standard variant of the prize-collecting TSP considered by Goemans and Williamson in [3]. In this variant, each target has a penalty associated with it. The objective of the prize-collecting TSP is to find a tour for the vehicle that starts and ends at the depot such that the sum of the tour's cost plus the penalties of each target not included in the tour is minimal. If π_i and π_j respectively denote the penalties of the two vehicles i and j, the prize-collecting TSP can be cast as a TDHTSP by setting the second vehicle's cost of traveling the edge joining vertices i and j to be equal to $\frac{\pi_i+\pi_j}{2}$. By choosing the penalty variable corresponding to the second depot to be equal to 0, one can deduce that the second vehicle's travel cost is actually equal to the sum of the penalties of the targets that are not present in the first vehicle's tour. Even though there are no penalties explicitly mentioned in the TDHTSP, the tour cost of the second vehicle (which accounts for the targets not visited by the first vehicle) acts as penalties do. For these reasons, the primal-dual algorithm presented in this article is based on the primal-dual algorithm available for the prize-collecting TSP in [3].

Most of the work in the literature related to approximation algorithms for multiple depot TSPs deals with identical vehicles. For example, when the costs satisfy the triangle inequality, there are several approximation algorithms for the multiple depot homogeneous TSP (see, for example, [4–6]). Recently, [10] presented a 3-approximation algorithm for a two depot heterogeneous TSP. This algorithm partitions the targets by solving a linear programming relaxation and then uses Christofides's algorithm [1] to find a sequence of targets for each vehicle. An approximation algorithm has been also developed for the variant of the heterogeneous TSP where each vehicle starts and ends at the same depot, and the maximum completion time of the tours is minimized [11]. A preliminary version of this paper without the main proofs appeared in [12].

The 2-approximation algorithms in the literature for the multiple depot TSP generally follow a two-step procedure. In the first step, a constrained forest problem that is generally a relaxation of the multiple depot TSP is solved optimally. In the second step, an Eulerian graph is found for each vehicle based on the constrained forest. Using the Eulerian graphs, a tour can be found for each vehicle by shortcutting any target already visited by a vehicle. In this article, we follow a similar procedure. We first find a heterogeneous spanning forest using a primal-dual algorithm by solving a relaxation of the TDHTSP. Then we double the edges in the heterogeneous spanning forest to obtain an Eulerian graph for each vehicle. Given these Eulerian graphs, it is always possible to find a tour for each vehicle that visits each of the targets exactly once [9]. *The crux of this procedure depends on finding a good heterogeneous spanning forest*. Using a primal-dual algorithm, we find a heterogeneous spanning forest whose cost is at most equal to the optimal cost of the TDHTSP in polynomial time. Hence, it follows that the approximation ratio for our proposed procedure is 2.

2 Problem statement

Let $D = \{d_1, d_2\}$ represent the two depots (initial locations) corresponding to the first and the second vehicle, respectively. Let T represent the set of targets, and let $V_1 := T \cup \{d_1\}$ and $V_2 := T \cup \{d_2\}$ be the sets of vertices corresponding to the first and the second vehicle, respectively. For $i = 1, 2$, let E_i denote the set of all the edges that join any two distinct vertices in V_i, and let $cost_e^i$ denote the ith vehicle's cost of traversing an edge $e \in E_i$. For every edge e joining two targets, we assume that $cost_e^1 \leq cost_e^2$. We also assume that the costs satisfy the triangle inequality for both the vehicles. A tour for a vehicle begins at its depot, visits a set of targets in some sequence, and finally returns to its depot. The objective of the TDHTSP is to find a tour for each vehicle such that each target is visited exactly once by one of the vehicles and the sum of the costs of the edges traveled by the vehicles is minimal.

3 Problem formulation

Let x_e be an integer variable that represents whether edge $e \in E_1$ is present in the first vehicle's tour. For any edge e joining *two targets*, x_e can only take values in the set $\{0, 1\}$; $x_e = 1$ if e is present in the first vehicle's tour and $x_e = 0$ otherwise. To allow for a tour to visit only one target if required, x_e can be any of the values in the set $\{0, 1, 2\}$ for an edge e joining the depot d_1 and a target $v \in T$. Similarly, let y_e be an integer variable that represents whether edge $e \in E_2$ is present in the second vehicle's tour. Let z_U be a binary variable that determines the partition of targets connected to the first and the second depot; $z_U = 1$ if each target in $U \subseteq T$ is connected to the second depot and each target in $T \setminus U$ is connected to the first depot. At most one subset U of targets is allowed to have $z_U = 1$. Let $\delta_i(S)$, for $i = 1, 2$, denote the subset of the edges of E_i that have one end in S and the other end in $V_i \setminus S$. $\delta_i(S)$ is also called the *cut set* of S corresponding to the ith vehicle.

For any $S \subseteq T$, at least two edges must be chosen from $\delta_1(S)$ for the first vehicle's tour if there is at least one vertex in S that is not connected to the second depot, i.e., $\sum_{e \in \delta_1(S)} x_e \geq 2$ if $\sum_{T \supseteq U \supseteq S} z_U = 0$. This requirement can be written as $\sum_{e \in \delta_1(S)} x_e + 2 \sum_{T \supseteq U \supseteq S} z_U \geq 2$. Similarly, for any $S \subseteq T$, at least two edges must be chosen from $\delta_2(S)$ for the second vehicle's tour if all the vertices in S are required to be visited by the second vehicle. This requirement can be expressed as $\sum_{e \in \delta_2(S)} y_e \geq 2 \sum_{T \supseteq U \supseteq S} z_U$. Now consider the following relaxation of the integer program for the TDHTSP without the degree constraints:

$$\min \sum_{e \in E_1} cost_e^1 \, x_e + \sum_{e \in E_2} cost_e^2 \, y_e \tag{1}$$

$$\sum_{e \in \delta_1(S)} x_e + 2 \sum_{T \supseteq U \supseteq S} z_U \geq 2 \qquad \forall S \subseteq T, \tag{2}$$

$$\sum_{e \in \delta_2(S)} y_e \geq 2 \sum_{T \supseteq U \supseteq S} z_U \quad \forall S \subseteq T, \tag{3}$$

$$\sum_{U \subseteq T} z_U \leq 1, \tag{4}$$

$$x_e, y_e \in \{0, 1\}\ \forall e \text{ joining any two targets}, \tag{5}$$

$$x_e \in \{0, 1, 2\}\ \forall e \text{ joining } d_1 \text{ and a target}, \tag{6}$$

$$y_e \in \{0, 1, 2\}\ \forall e \text{ joining } d_2 \text{ and a target}, \tag{7}$$

$$z_U \in \{0, 1\}\ \forall U \subseteq T. \tag{8}$$

Consider a linear programming (LP) relaxation of the above integer program with the objective stated in (1) subject to the constraints in (2)–(3) and $\forall e \in E_1\ x_e \geq 0$, $\forall e \in E_2\ y_e \geq 0$, $\forall U \subseteq T\ z_U \geq 0$. A dual of this LP relaxation can be formulated as follows:

$$C_{dual} = \max\ 2 \sum_{S \subseteq T} Y_1(S) \tag{9}$$

$$\sum_{S: e \in \delta_1(S)} Y_1(S) \leq cost_e^1 \quad \forall e \in E_1, \tag{10}$$

$$\sum_{S: e \in \delta_2(S)} Y_2(S) \leq cost_e^2 \quad \forall e \in E_2, \tag{11}$$

$$\sum_{S \subseteq U} Y_1(S) \leq \sum_{S \subseteq U} Y_2(S) \quad \forall U \subseteq T, \tag{12}$$

$$Y_1(S), Y_2(S) \geq 0 \quad \forall S \subseteq T. \tag{13}$$

We use this dual problem to find a heterogeneous spanning forest (HSF). A HSF consists of two trees such that the first tree spans a subset of targets and d_1 and the second tree connects the remaining set of targets to d_2.

4 Primal dual algorithm

The primal-dual algorithm is an iterative algorithm that follows the greedy procedure outlined by Goemans and Williamson in [3]. The basic structure of the algorithm involves maintaining a forest of edges corresponding to each vehicle (called F_1 and F_2 for the first and second vehicle, respectively) and an implicit solution to the dual problem. The edges in these forests are candidates for the set of edges that appear in the algorithm's final HSF output. The pseudo code of the primal-dual algorithm is presented in **Algorithm 1**.

Initialization steps Initially, both F_1 and F_2 are empty (Fig. 1). Each connected connected component in F_1 and F_2 contains exactly one vertex. All components except the depots are considered active, and each vertex is unlabeled.[1] All the dual variables of the components in F_1 and F_2 are initially set to zero.

[1] This labeling will be used in the pruning procedure at the end of the algorithm and aids in the proof of the approximation ratio of the algorithm.

Algorithm 1 : *Primal-dual algorithm*

1: Initialization

2: $F_1 \leftarrow \emptyset$; $F_2 \leftarrow \emptyset$; $\mathcal{C}_1 \leftarrow \{\{v\} : v \in V_1\}$; $\mathcal{C}_2 \leftarrow \{\{v\} : v \in V_2\}$
3: **for** $v \in T$ **do**
4: Set v as unlabeled; $p_1(v) \leftarrow 0$; $p_2(v) \leftarrow 0$;
//**Comment:** *Implicitly set* $Y_1(S), Y_2(S) \leftarrow 0,\ \forall S \subseteq T$
5: $w_1(\{v\}) \leftarrow 0$; $w_2(\{v\}) \leftarrow 0$; $Bound(\{v\}) \leftarrow 0$
6: $active_1(\{v\}) \leftarrow 1$;$active_2(\{v\}) \leftarrow 1$
7: **end for**
8: $active_1(\{d_1\}) \leftarrow 0$; $active_2(\{d_2\}) \leftarrow 0$; $w_1(\{d_1\}) \leftarrow 0$; $w_2(\{d_2\}) \leftarrow 0$; $Bound(\{d_1\}) \leftarrow 0$

9: Main loop

10: **while** $\exists C \in \mathcal{C}_1$ such that $active_1(C) = 1$ **do**
11: **for** $i = 1, 2$ **do**
12: Find an edge $e_i = (u, v) \in E_i$ with $u \in C_{ix}, v \in C_{iy}$, where $C_{ix}, C_{iy} \in \mathcal{C}_i, C_{ix} \neq C_{iy}$ that minimizes $\varepsilon_i = \frac{(cost^i_{e_i} - p_i(u) - p_i(v))}{active_i(C_{ix}) + active_i(C_{iy})}$
13: **end for**
14: Let $\mathfrak{C} := \{C : active_1(C) = 1, Children(C) = \emptyset, C \in \mathcal{C}_1\}$. Find $\overline{C} \in \mathfrak{C}$ that minimizes $\varepsilon_3 = Bound(\overline{C}) - w_1(\overline{C})$
15: $\varepsilon_{min} = \min(\varepsilon_1, \varepsilon_2, \varepsilon_3)$
16: //`Comment: If more than one value in` $\{\varepsilon_1,\varepsilon_2,\varepsilon_3\}$ `is equal to` ε_{min}, `then give priority first to` *Case* ε_1, `then to` *Case* ε_2 `and finally to` *Case* ε_3

17: **for** $i = 1, 2$ **do**
18: **for** each active component $C \in \mathcal{C}_i$ **do**
19: $w_i(C) \leftarrow w_i(C) + \varepsilon_{min}$
20: For all $v \in C$, $p_i(v) \leftarrow p_i(v) + \varepsilon_{min}$
//**Comment:** *Implicitly set* $Y_i(C) \leftarrow Y_i(C) + \varepsilon_{min},\ \forall$ *active* $C \in \mathcal{C}_i$
21: **if** $i = 1$ **then**
22: $Bound(C) \leftarrow Bound(C) + \varepsilon_{min}|Children(C)|$
23: **end**
24: **end for**
25: **end for**

26: **if** $\varepsilon_{min} = \varepsilon_i$ for $i = 1$ or 2 **then**
27: $F_i \leftarrow F_i \cup \{e_i\}$
28: $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{C_{ix} \cup C_{iy}\} - C_{ix} - C_{iy}$
29: $w_i(C_{ix} \cup C_{iy}) \leftarrow w_i(C_{ix}) + w_i(C_{iy})$
30: **if** $i = 1$ **then** $Bound(C_{1x} \cup C_{1y}) \leftarrow Bound(C_{1x}) + Bound(C_{1y})$ **end**
31: **if** $d_i \in C_{ix} \cup C_{iy}$ **then**
32: $active_i(C_{ix} \cup C_{iy}) \leftarrow 0$
33: **if** $i = 1$ **then** $active_2(C) \leftarrow 0\ \forall C \in Children(C_{1x} \cup C_{1y})$ **end**
34: **else** $active_i(C_{ix} \cup C_{iy}) \leftarrow 1$
35: **end**
36: **else**
37: $active_1(\overline{C}) \leftarrow 0$
38: Label all the unlabeled vertices of $\overline{C}$ with label $\overline{C}$
39: **end if**
40: **end while**

41: Pruning Step

42: F_1' is obtained from F_1 by the following procedure:

- Find $L := \{u : u \in T, u$ is a target on the path joining any unlabeled target to $d_1\}$. *Note: L includes all the unlabeled targets and other targets that lie on the paths joining the unlabeled targets and d_1.*
- Find $L' := \{v : v \notin L,$ label of $v \supseteq$ label of $u,\ u$ is a labeled target in $L\}$.
- Remove all the edges incident on any vertex not present in $L \cup L'$.

43: F_2' is obtained from F_2 by removing all the edges incident on any vertex $u \in L \cup L'$.

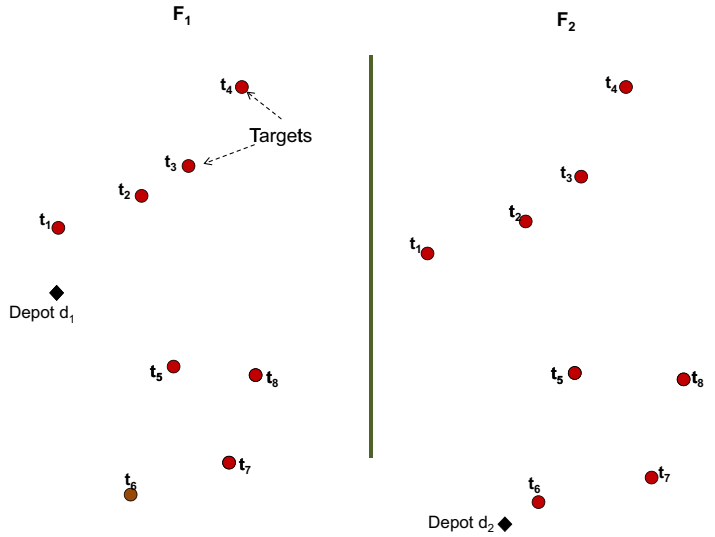

Fig. 1 An example illustrating the basic steps in the primal dual algorithm. There are 8 targets (denoted by t_i for $i = 1, \ldots, 8$) in this example. The forests F_1 and F_2 are initially empty. Each component that contains a target is active. The components that contain the depots are inactive

The initialization steps are presented in lines 2–8 of the pseudo code. In this code, $\mathcal{C}_1$ and $\mathcal{C}_2$ denote the set of connected components in F_1 and F_2, respectively. A dual solution is implicitly generated using variables $p_i(u) := \sum_{S:u\in S} Y_i(S)$ for any target $u \in T$, and $w_i(C) := \sum_{S\subseteq C} Y_i(S)$ for any component $C \in \mathcal{C}_i$ $(i = 1, 2)$. The algorithm also keeps track of the activity of a component $C \in \mathcal{C}_i$ using the variable $active_i(C)$ for $i = 1, 2$. For any component $C \in \mathcal{C}_1$, the variable $Bound(C) := \sum_{S\subseteq C} Y_2(S)$ is used to enforce the feasibility constraint (12) in the dual problem.

Main loop In each iteration, the algorithm *uniformly* increases the dual variable of each active component by a value ε_{min} that is as large as possible without violating any of the constraints (10)–(12) in the dual (lines 11–25 of the pseudo code). As the components in $\mathcal{C}_1$ tend to merge first, we refer to the components in $\mathcal{C}_1$ as parents and the components in $\mathcal{C}_2$ as their children. Specifically, for any component $C \in \mathcal{C}_1$, we define $Children(C) := \{\hat{C}\colon \hat{C} \in \mathcal{C}_2, \hat{C} \subseteq C\}$. The feasibility of the dual solution is ensured in each iteration in the following way (lines 11–15 of the pseudo code):

- For $i = 1, 2$, as long as targets u and v are not connected in F_i, $\sum_{S:e\in\delta_i(S)} Y_i(S) = p_i(u) + p_i(v)$ for the edge e joining u and v. It follows that the dual variable of the components containing u and v can be increased at most by $\frac{cost_e^i - p_i(u) - p_i(v)}{active_i(C_{ix}) + active_i(C_{iy})}$, where $e = (u, v)$, $u \in C_{ix}$, $v \in C_{iy}$, and $C_{ix}, C_{iy} \in \mathcal{C}_i$ (lines 11–13 of the pseudo code). Once the edge $e = (u, v)$ has been added to the forest F_i, the dual cost $\sum_{S:e\in\delta_i(S)} Y_i(S)$ does not increase, and hence the packing constraint corresponding to e in (10)–(11) will continue to hold.
- The algorithm also ensures the dual solution satisfies (12) using the set $Children(C)$ and the variable $Bound(C)$ defined for any component $C \in \mathcal{C}_1$. Given any $C \in \mathcal{C}_1$, $\varepsilon_3 := Bound(C) - w_1(C) = \sum_{S\subseteq C} Y_2(S) - \sum_{S\subseteq C} Y_1(S)$

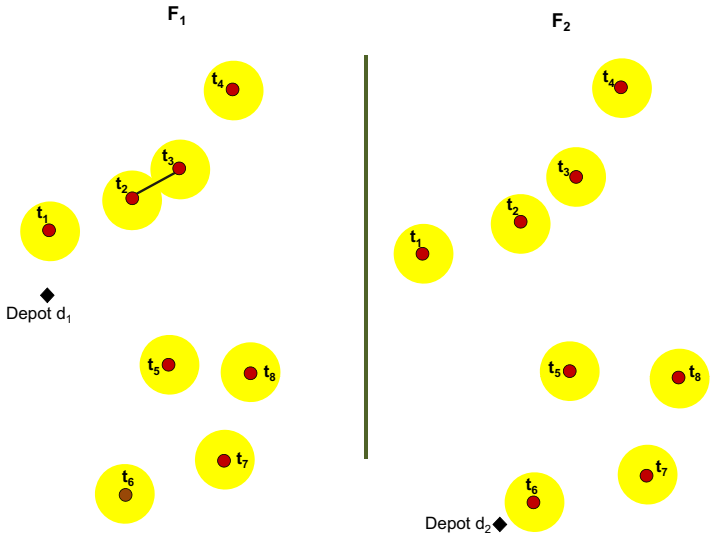

Fig. 2 Snapshot of the forests at the end of the first iteration of the main loop. The radius of the circular region, $p_i(u) := \sum_{S:u\in S} Y_i(S)$, around a target u in the forest F_i is equal to the sum of the dual variables of all components that contain u in F_i. Edge $e := (t_2, t_3)$ is added to F_1 as $p_1(t_2) + p_1(t_3)$ becomes equal to $cost_e^1$. In this case, $\varepsilon_{min} = \varepsilon_1$

indicates the maximum amount by which any dual variable can be increased without violating Eq. (12) (line 14 of the pseudo code).

The increase in the dual variables results in one of the following *outcomes* in each iteration:

- If any of the constraints in (10)–(11) becomes tight[2] for some edge between two distinct components in F_i, at least one of which is active, the algorithm adds the edge to F_i and merges the two components (lines 27–35 of the pseudo code). If the merged component does not contain a depot, it becomes active (Figs. 2, 3); otherwise it is inactive (Fig. 4).
- If a constraint in (12) becomes tight for a component in F_1, then the component is deactivated (Figs. 4, 5, 6, 7) and each unlabelled target in the component is labeled with the name of the component (lines 37–38 of the pseudo code).

The iterative process terminates when all components become inactive (Fig. 7).

Pruning step The final step of the algorithm removes any unnecessary edges that are not required to be in F_1 or F_2, using a labeling procedure from the prize-collecting TSP in [3] (Fig. 8). Specifically, this procedure prunes edges in F_1 while maintaining two properties. First, any unlabeled target in F_1 must be visited by the first vehicle since this target was never in any deactivated component and the algorithm never

[2] A constraint becomes tight if none of the variables present in the constraint can be increased without violating the constraint.

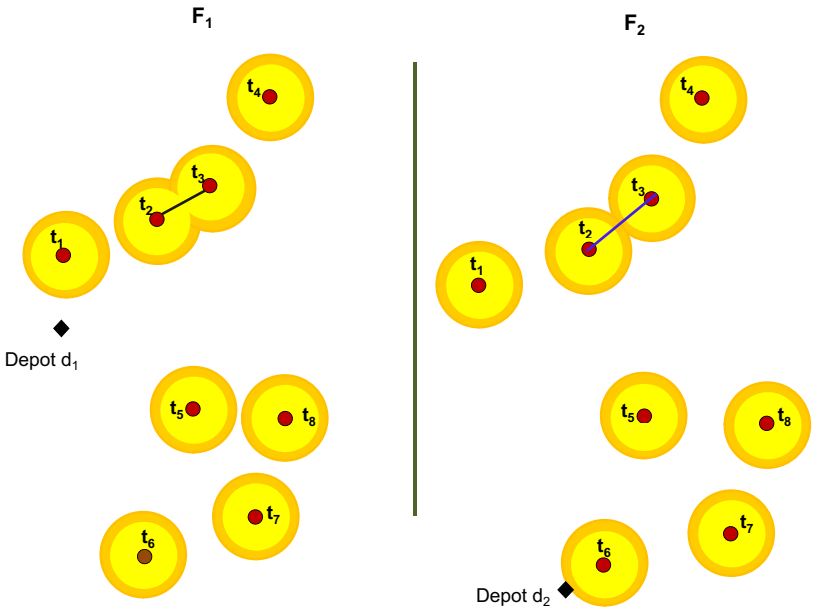

Fig. 3 Snap shot of the forests at the end of the second iteration. Edge $e := (t_2, t_3)$ is added to F_2 as $p_2(t_2) + p_2(t_3)$ becomes equal to $cost_e^2$. In this case, $\varepsilon_{min} = \varepsilon_2$

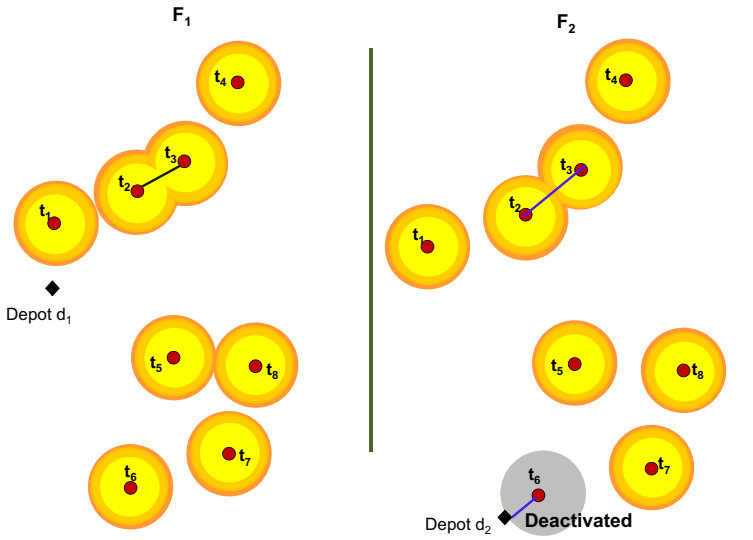

Fig. 4 Snapshot of the forests after three iterations of the main loop. The constraint corresponding to the edge joining target t_6 and depot d_2 becomes tight. Edge (t_6, d_2) is added to F_2 and the merged component is deactivated as it contains d_2. The dual variable $Y_2(\{t_6\})$ does not increase further and will serve as an upper bound on $Y_1(\{t_6\})$. In this case, $\varepsilon_{min} = \varepsilon_2$

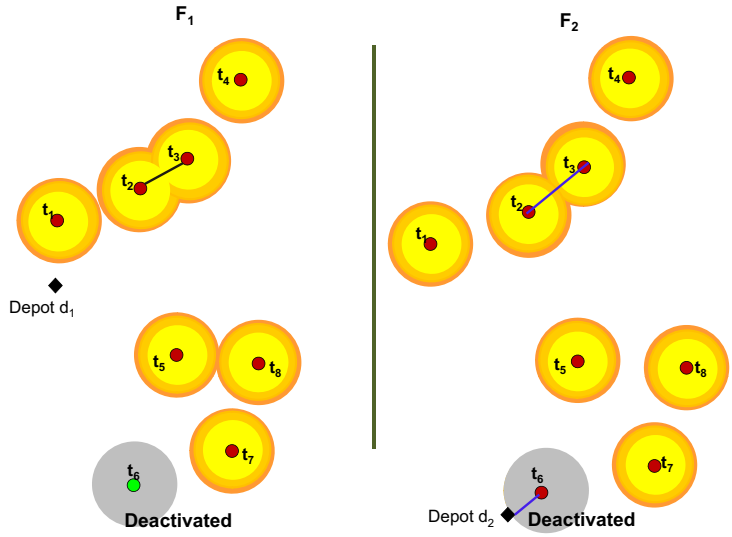

Fig. 5 Snapshot of the forests at the end of the fourth iteration of the main loop. Component $\{t_6\}$ in F_1 is deactivated because $w_1(\{t_6\}) := Y_1(\{t_6\})$ becomes equal to $w_2(\{t_6\}) := Y_2(\{t_6\})$. In this case, $\varepsilon_{min} = \varepsilon_3$

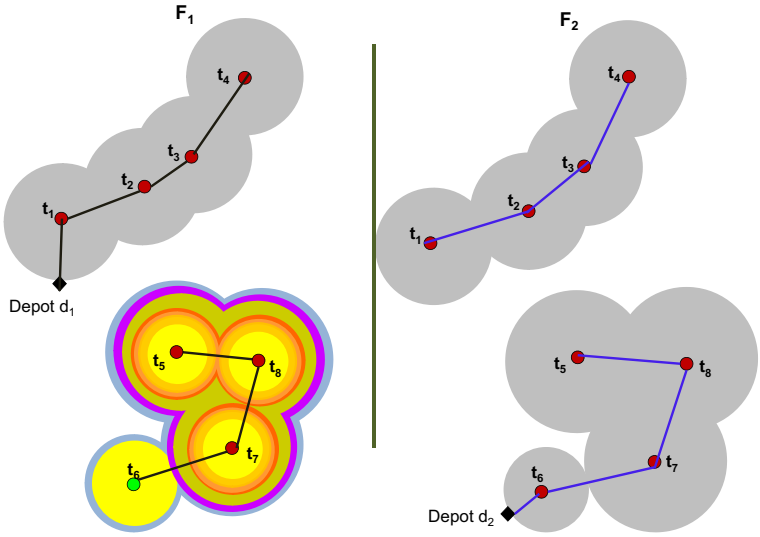

Fig. 6 Snapshot of the forests after few iterations of the main loop. All components are inactive except $\overline{C} := \{t_5, t_6, t_7, t_8\}$ of F_1. Notice that all the targets are connected to one of the two depots and the algorithm can stop if needed. However, the algorithm is not terminated until all components are inactive. The reason for this is that $w_1(\overline{C}) := \sum_{S \subseteq \overline{C}} Y_1(S)$ can be less than $w_2(\overline{C}) := \sum_{S \subseteq \overline{C}} Y_2(S)$, and therefore, it is possible that the targets in $\overline{C}$ can get connected to d_1 at a lower cost

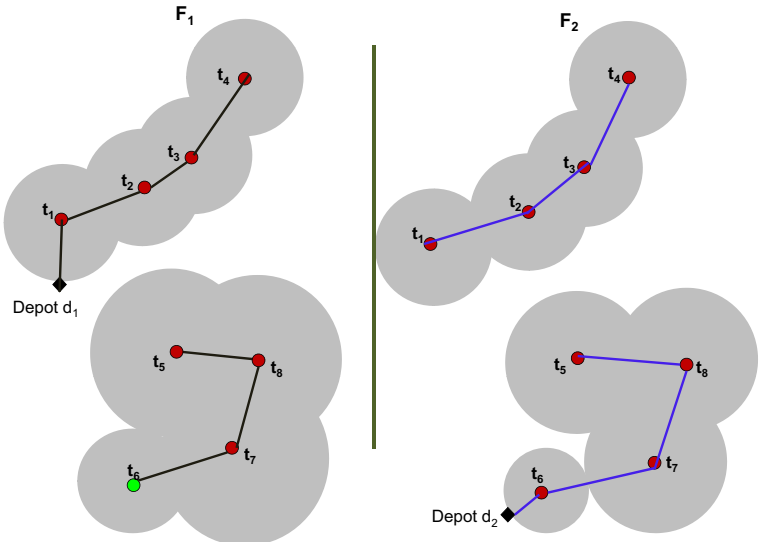

Fig. 7 Snapshot of the forests at the end of the main loop. $\overline{C} := \{t_5, t_6, t_7, t_8\}$ of F_1 is deactivated because $w_1(\overline{C})$ becomes equal to $w_2(\overline{C})$. In this case, $\varepsilon_{min} = \varepsilon_3$. The main loop terminates because all components are now inactive

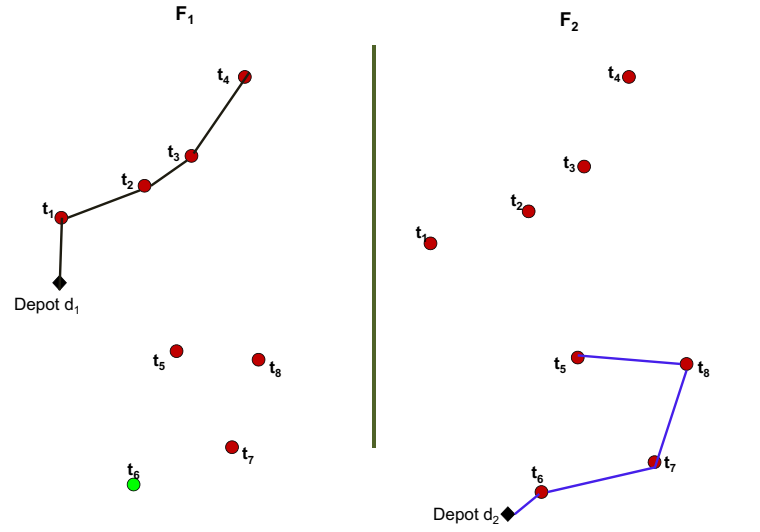

Fig. 8 The final output (HSF) of the primal-dual algorithm after the unnecessary edges are removed in the pruning step

considered this target for a visit by the second vehicle. Second, if any target with label C is visited by the first vehicle, then any other target with a label $C' \supseteq C$ must also be visited by the first vehicle. In addition, the pruning procedure removes as many edges

as possible from F_2 while ensuring that any target not visited by the first vehicle is visited by the second vehicle (lines 42–43 of the pseudo code).

A key feature of our primal-dual procedure is that it ensures that the components in F_1 tend to merge before the components in F_2. Even if the forests contain a feasible HSF at some point, the primal-dual algorithm continues in hope of connecting the targets to d_1 at a cheaper cost (see Fig. 6). It turns out, as we will show, that the way that this is accomplished is useful for obtaining a good approximation ratio.

Remark Any component $\hat{C} \in \mathcal{C}_2$ becomes inactive and seizes to be a child of any component in $\mathcal{C}_1$ once $\hat{C}$ merges with a component in $\mathcal{C}_2$ containing d_2. For this reason, after few iterations, a component in $\mathcal{C}_1$ may still be active but not have any children.

5 Properties of the primal-dual algorithm

Consider any target $u \in T$. At the start of the kth iteration, let $C_1^k(u)$ denote the component of F_1 containing u and let $C_2^k(u)$ denote the component of F_2 containing u. Also, $C_2^k(u)$ is referred as a child of $C_1^k(u)$ if $C_2^k(u) \subseteq C_1^k(u)$.

Lemma 1 *The following statements are true for all k:*

1. $C_2^k(u) \subseteq C_1^k(u)$, *unless* $C_2^k(u)$ *contains the depot* d_2.
2. $active_1(C_1^k(u)) \geq active_2(C_2^k(u))$.

Proof Let us prove this lemma by mathematical induction.

Induction basis ($k = 1$) At the start of the first iteration, $C_1^1(u) = C_2^1(u) = \{u\}$, and the components $C_1^1(u)$ and $C_2^1(u)$ are both active. Therefore, Lemma 1.1 and 1.2 hold for $k = 1$.

Induction hypothesis Let us assume that the lemma is true for the lth iteration for any $1 \leq l \leq k$. As $active_1(C_1^l(u)) \geq active_2(C_2^l(u))$ for any $1 \leq l \leq k$, it follows that $p_1(u) \geq p_2(u)$. Therefore, at the start of the kth iteration,

$$\begin{aligned} \varepsilon_1 &= \frac{cost^1_{(u,v)} - p_1(u) - p_1(v)}{active_1\left(C_1^k(u)\right) + active_1\left(C_1^k(v)\right)} \\ &\leq \frac{cost^2_{(u,v)} - p_2(u) - p_2(v)}{active_2\left(C_2^k(u)\right) + active_2\left(C_2^k(v)\right)} \\ &= \varepsilon_2. \end{aligned} \tag{14}$$

□

Induction step

Proof of Lemma 1.1 During the kth iteration, there are three possible cases for the components $C_1^k(u)$ and $C_2^k(u)$: (1) $C_1^k(u)$ merges with another component in $\mathcal{C}_1$, (2) $C_2^k(u)$ merges with another component in $\mathcal{C}_2$, or (3) $C_1^k(u)$ is deactivated because its corresponding constraint in (12) becomes tight. It is easy to see that $C_2^{k+1}(u) \subseteq C_1^{k+1}(u)$ in the first case. $C_1^k(u)$ can be deactivated, as described in the third case, only when $C_1^k(u)$ does not have any children, i.e., $C_2^k(u)$ already contains d_2. Therefore, Lemma 5 is true vacuously in the third case.

Let us now examine the second case. If $C_2^k(u)$ is active and merges with a component that contains d_2, then Lemma 1.1 is true for $l = k+1$. If $C_2^k(u)$ is active and merges with another active component $C_2^k(v)$ corresponding to target v, we claim that $C_1^k(u) = C_1^k(v)$ since $\varepsilon_1 \leq \varepsilon_2$ by hypothesis, Algorithm 1 will not merge $C_2^k(u)$ and $C_2^k(v)$ unless it merges $C_1^k(u)$ and $C_1^k(v)$. And if $C_1^k(u) = C_1^k(v)$, it follows that the merged component $C_2^{k+1}(u)$ will be a child of $C_1^{k+1}(u)$.

If $C_2^k(u)$ is inactive because $C_2^k(u) \subseteq C_1^k(u)$ and $d_1 \in C_1^k(u)$, we claim that $C_2^k(u)$ will never merge with any other component. For if $C_2^k(u)$ (which is inactive) merges with some other component $C_2^k(v)$ corresponding to target v, then $C_2^k(v)$ must be active and $C_1^k(u) \neq C_1^k(v)$. Again by the hypothesis (Eq. 14), the algorithm will prefer to merge $C_1^k(u)$ and $C_1^k(v)$ before merging their children $C_2^k(u)$ and $C_2^k(v)$. But once $C_1^k(u)$ and $C_1^k(v)$ are merged, the component $C_2^k(v)$ becomes a child of $C_1^k(u) \cup C_1^k(v)$ and as a result will be deactivated (lines 31–33 of the Algorithm 1). Therefore, $C_2^k(u)$ will remain inactive and will never merge with any other component during the kth iteration. Hence, Lemma 1.1 is true in this case. □

Proof of Lemma 1.2 If $C_2^k(u)$ is **inactive**, either $d_2 \in C_2^k(u)$ or $C_2^k(u) \subseteq C_1^k(u), d_1 \in C_1^k(u)$.

- If $C_2^k(u)$ already contains d_2, then $C_2^{k+1}(u)$ must also be inactive. Therefore, $active_1(C_1^{k+1}(u)) \geq active_2(C_2^{k+1}(u)) = 0$.
- If $C_2^k(u)$ is inactive because $C_2^k(u) \subseteq C_1^k(u)$ and $d_1 \in C_1^k(u)$, then we have already shown (in the proof of Lemma 1.1) that $C_2^k(u)$ can never merge with any other component during the kth iteration. Therefore, $active_1(C_1^{k+1}(u)) \geq active_2(C_2^{k+1}(u))$.

If $C_2^k(u)$ is **active**, then by the hypothesis, $C_2^k(u) \subseteq C_1^k(u)$ and $C_1^k(u)$ is active. Since $C_1^k(u)$ has at least one active child in $C_2^k(u)$, $C_1^k(u)$ can become inactive during the kth iteration only when $C_1^k(u)$ merges with another component containing d_1. But this will deactivate all the children of $C_1^k(u)$, including $C_2^k(u)$. Therefore, $active_1(C_1^{k+1}(u)) \geq active_2(C_2^{k+1}(u))$. □

5.1 Feasibility and running time analysis

Let X denote the set of vertices not spanned by F_1'. Based on the label of each vertex in X, X can be partitioned into disjoint deactivated components $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_m$, where each $\overline{C}_i$ denotes the maximal label of its corresponding component.

Lemma 2 *The primal-dual algorithm produces a feasible HSF in $O(|T|^2 \log |T|)$ steps. Also, no vertex spanned by the edges in F_1' is spanned by the edges in F_2', and vice versa.*

Proof Since the sum of the number of components in $\mathcal{C}_1$, the number of active components in $\mathcal{C}_1$, and the number of components in $\mathcal{C}_2$ decreases by at least 1 during each iteration, the primal-dual algorithm must terminate after at most $3|T| + 2$ iterations.

Using the techniques given in [3], this primal-dual algorithm can be implemented in $O(|T|^2 \log |T|)$ steps.

The algorithm terminates when all the components of $\mathcal{C}_1$ become inactive. This is only possible if each of the targets in T is connected to either d_1 or d_2. Note that F_1' is formed from F_1 in such a way that each of the unlabeled vertices remains connected to d_1. The only vertices not spanned by F_1' are some of the labeled vertices. These vertices were labeled because the components in $\mathcal{C}_1$ that span these vertices were deactivated for making their associated constraints in (12) tight. In addition, a component in $\mathcal{C}_1$ can become deactivated due to a constraint in (12) only if it has already lost all its children, i.e., each of these vertices in the component is already connected to d_2. Therefore, by the construction of F_2', each of the labeled vertices not spanned by F_1' must be connected to d_2 and spanned by F_2'. Hence, the algorithm produces a feasible HSF.

Consider any deactivated component $\overline{C}_i \subseteq X$. $\overline{C}_i$ can be deactivated during an iteration only if $\overline{C}_i$ does not have children and $\sum_{S\subseteq\overline{C}_i} Y_1(S) = w_1(\overline{C}_i) = Bound(\overline{C}_i) = \sum_{S\subseteq\overline{C}_i} Y_2(S)$. Note that $\overline{C}_i$ can lose all its children only if all the targets in $\overline{C}_i$ are already connected to d_2 in F_2. Also, *during the iteration* when $\overline{C}_i$ is deactivated, no target $u \in \overline{C}_i$ is connected to any other target $v \in T\backslash\overline{C}_i$ in F_1. As a result, we claim that u does not have an adjacent vertex v in F_2 such that $v \in T\backslash\overline{C}_i$. For by Lemma 1 and Eq. (14), the algorithm would have added the edge (u, v) to F_1 before adding (u, v) to F_2. Thus, since target u is not connected to target $v \in T\backslash\overline{C}_i$ in F_1, u and v cannot be connected in F_2 (*during the considered iteration*). Therefore, during the construction of F_2', all the edges that are incident to any vertex $u \notin X$ could be dropped. Hence, any vertex spanned by the edges in F_1' is not spanned by the edges in F_2', and vice versa. □

5.2 Approximation ratio analysis

The proposed algorithm for the TDHTSP requires doubling the edges in the HSF obtained by the primal-dual algorithm and using the resulting Eulerian graph to find a tour for each vehicle.

Theorem 1 *The proposed algorithm has an approximation ratio of 2.*

Proof We need to show that the sum of the costs of the edges in the HSF found by the primal-dual algorithm is at most equal to the optimal cost of the TDHTSP. To do this, we first simplify the dual cost obtained by the primal-dual algorithm as follows:

$$\begin{aligned} 2\sum_{S\subseteq T} Y_1(S) &= 2 \sum_{S\subseteq T, S\nsubseteq\overline{C}_i, i=1,\ldots,m} Y_1(S) + 2\sum_{i=1}^{m}\sum_{S\subseteq\overline{C}_i} Y_1(S) \\ &= 2 \sum_{S\subseteq T, S\nsubseteq\overline{C}_i, i=1,\ldots,m} Y_1(S) + 2\sum_{i=1}^{m}\sum_{S\subseteq\overline{C}_i} Y_2(S). \end{aligned} \tag{15}$$

Next, we express the cost of the edges in the first tree in terms of the dual variables as follows. An edge e is added to F_1 and consequently appears in F_1' only if the corresponding constraint in (10) is tight, i.e., $cost_e^1 = \sum_{S:e\in\delta_1(S)} Y_1(S)$. Therefore,

$$\sum_{e \in F_1'} cost_e^1 = \sum_{e \in F_1'} \sum_{S: e \in \delta_1(S)} Y_1(S)$$
$$= \sum_{S \subseteq T} Y_1(S) \left| F_1' \bigcap \delta_1(S) \right|.$$

Since $F_1' \bigcap \delta_1(S) = 0$ for any $S \subseteq \overline{C}_i$, we can further simplify the above equation to

$$\sum_{e \in F_1'} cost_e^1 = \sum_{S \subseteq T, S \nsubseteq \overline{C}_i, i=1,\ldots,m} Y_1(S) \left| F_1' \bigcap \delta_1(S) \right|. \tag{16}$$

Similarly, we can express the cost of the edges in the second tree in terms of the dual variables as follows. From Lemma 2, note that F_2' can be decomposed into a set of disjoint sets F_{2i}' where each F_{2i}' consists of edges that form a tree spanning each target from $\overline{C}_i$ and d_2. Since an edge e is added to F_2 and consequently appears in F_{2i}' only if the corresponding constraint in (11) is tight, $cost_e^2 = \sum_{S: e \in \overline{\delta}_{2i}(S), S \subseteq \overline{C}_i} Y_2(S)$, where $\overline{\delta}_{2i}(S)$ consists of all the edges with one endpoint in S and another endpoint in $\overline{C}_i \cup \{d_2\} \backslash S$.

$$\sum_{e \in F_2'} cost_e^2 = \sum_{i=1}^{m} \sum_{e \in F_{2i}'} cost_e^2$$
$$= \sum_{i=1}^{m} \sum_{e \in F_{2i}'} \sum_{S: e \in \overline{\delta}_{2i}(S), S \subseteq \overline{C}_i} Y_2(S)$$
$$= \sum_{i=1}^{m} \sum_{S \subseteq \overline{C}_i} Y_2(S) \left| F_{2i}' \bigcap \overline{\delta}_{2i}(S) \right|. \tag{17}$$

Therefore, from Eqs. (15), (16), and (17), all we need to do to complete the proof is to show that

$$\sum_{S \subseteq T, S \nsubseteq \overline{C}_i, i=1,\ldots,m} Y_1(S) \left| F_1' \bigcap \delta_1(S) \right| + \sum_{i=1}^{m} \sum_{S \subseteq \overline{C}_i} Y_2(S) \left| F_{2i}' \bigcap \overline{\delta}_{2i}(S) \right|$$
$$\leq 2 \sum_{S \subseteq T, S \nsubseteq \overline{C}_i, i=1,\ldots,m} Y_1(S) + 2 \sum_{i=1}^{m} \sum_{S \subseteq \overline{C}_i} Y_2(S). \tag{18}$$

This can be established by proving that during any iteration, the increase in the primal cost (the left-hand side of the inequality) is at most equal to the increase in the dual cost (the right-hand side of the inequality). To see this, choose any iteration of the primal-dual algorithm. Let $\mathcal{N}_a$ be the set of all active components in $\mathcal{C}_1$ that are not subsets of X at the start of this iteration, and let $\mathcal{N}_d$ be the set of all inactive components in $\mathcal{C}_1$ that are not subsets of X at the start of the iteration. Note that one of the inactive components of $\mathcal{N}_d$ must contain the depot d_1. For $i = 1, \ldots, m$, let $\mathcal{M}_{ai}$ denote the

set of all active components in $\mathcal{C}_2$ that are subsets of $\overline{C}_i$. Also, let $\mathcal{M}_d$ denote the inactive component in $\mathcal{C}_2$ that contains d_2.

Now form a graph H_1 with components in $\mathcal{N}_a \cup \mathcal{N}_d$ as its vertices and the edges $e \in F_1' \cap \delta_1(C)$, for $C \in \mathcal{N}_a \cup \mathcal{N}_d$, as its edges. H_1 is a tree that spans all of the vertices in $\mathcal{N}_a \cup \mathcal{N}_d$. Similarly, form a graph H_{2i} with components in $\mathcal{M}_{ai} \cup \{\mathcal{M}_d\}$ as its vertices and the edges $e \in F_{2i}' \cap \delta_2(C)$, for $C \in \mathcal{M}_{ai} \cup \{\mathcal{M}_d\}$, as its edges. H_{2i} is a tree that spans all of the vertices in $\mathcal{M}_{ai} \cup \{\mathcal{M}_d\}$.

Let $deg(v, G)$ represent the degree of vertex v in graph G. During the iteration, the dual variable corresponding to each of the active components is increased by ε_{min}. As a result, the left-hand side of the inequality will increase by $\varepsilon_{min}(\sum_{v\in\mathcal{N}_a} deg(v, H_1) + \sum_{i=1}^{m}\sum_{v\in\mathcal{M}_{ai}} deg(v, H_{2i}))$, while the right-hand side of the inequality will increase by $2\varepsilon_{min}(|\mathcal{N}_a| + \sum_{i=1}^{m}|\mathcal{M}_{ai}|)$. Therefore, the proof will be complete if we can show that

$$\sum_{v\in\mathcal{N}_a} deg(v, H_1) + \sum_{i=1}^{m}\sum_{v\in\mathcal{M}_{ai}} deg\,(v, H_{2i}) \leq 2\left(|\mathcal{N}_a| + \sum_{i=1}^{m}|\mathcal{M}_{ai}|\right). \tag{19}$$

We now claim that any vertex v in H_1 that represents an inactive component in $\mathcal{N}_d$ must have its degree $deg(v, H_1) \geq 2$ unless the inactive component contains the depot d_1. This result follows from the fact that a component that does not contain d_1 can become inactive in $\mathcal{C}_1$ only if the constraint associated with this component in (12) becomes tight. Therefore, all the vertices in this inactive component must be labeled. Also, if vertex v is a leaf (i.e., $deg(v, H_1) = 1$), then pruning all the edges from this inactive component will not disconnect any unlabeled target from d_1. Thus, the pruning step of the algorithm ensures that an inactive component can never be a leaf vertex in H_1 unless it contains d_1. Hence,

$$\sum_{v\in\mathcal{N}_d} deg(v, H_1) \geq 2|\mathcal{N}_d| - 1. \tag{20}$$

We now show the final part of the proof:

$$\sum_{v\in\mathcal{N}_a} deg(v, H_1) + \sum_{i=1}^{m}\sum_{v\in\mathcal{M}_{ai}} deg(v, H_{2i}) \tag{21}$$

$$= \sum_{v\in\mathcal{N}_a\cup\mathcal{N}_d} deg(v, H_1) - \sum_{v\in\mathcal{N}_d} deg(v, H_1) + \sum_{i=1}^{m}\left[\sum_{v\in\mathcal{M}_{ai}\cup\{\mathcal{M}_d\}} deg\,(v, H_{2i}) - deg\,(\mathcal{M}_d, H_{2i})\right] \tag{22}$$

$$\leq \sum_{v\in\mathcal{N}_a\cup\mathcal{N}_d} deg(v, H_1) - \sum_{v\in\mathcal{N}_d} deg(v, H_1) + \sum_{i=1}^{m}\left[\sum_{v\in\mathcal{M}_{ai}\cup\{\mathcal{M}_d\}} deg(v, H_{2i})\right] \tag{23}$$

H_1 is a tree that spans all the vertices in $\mathcal{N}_a \cup \mathcal{N}_d$. Therefore, the sum of the degrees of all the vertices in H_1 is $2(|\mathcal{N}_a| + |\mathcal{N}_d| - 1)$. Similarly, H_{2i} is a tree that spans all the vertices in $\mathcal{M}_{ai} \cup \{\mathcal{M}_d\}$. Therefore, the sum of the degrees of all the vertices in H_{2i} is $2|\mathcal{M}_{ai}|$. Hence

$$\begin{aligned} &\sum_{v \in \mathcal{N}_a} deg(v, H_1) + \sum_{i=1}^{m} \sum_{v \in \mathcal{M}_{ai}} deg(v, H_{2i}) \\ &\quad \leq 2\,(|\mathcal{N}_a| + |\mathcal{N}_d| - 1) \\ &\quad - \sum_{v \in \mathcal{N}_d} deg(v, H_1) + 2 \sum_{i=1}^{m} |\mathcal{M}_{ai}| \end{aligned} \tag{24}$$

$$\leq 2\,(|\mathcal{N}_a| + |\mathcal{N}_d| - 1) - (2|\mathcal{N}_d| - 1) + 2 \sum_{i=1}^{m} |\mathcal{M}_{ai}| \quad \text{(from Eq. 20)} \tag{25}$$

$$< 2|\mathcal{N}_a| + 2 \sum_{i=1}^{m} |\mathcal{M}_{ai}|. \tag{26}$$

This completes the proof. □

References

1. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Carnegie Mellon University, Pittsburgh (1976)
2. Feithans, G.L., Rowe, A.J., Davis, J.E., Holland, M., Berger, L.: Vigilant spirit control station (vscs)the face of counter. In: Proceedings of the AIAA Guidance, Navigation and Control Conference Exhibition. AIAA (2008)
3. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM J. Comput. **24**(2), 296–317 (1995)
4. Malik, W., Rathinam, S., Darbha, S.: An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem. Oper. Res. Lett. **35**(6), 747–753 (2007). doi:10.1016/j.orl.2007.02.001
5. Rathinam, S., Sengupta, R.: 3/2-approximation algorithm for two variants of a 2-depot hamiltonian path problem. Oper. Res. Lett. **38**(1), 63–68 (2010)
6. Rathinam, S., Sengupta, R., Darbha, S.: A resource allocation algorithm for multivehicle systems with nonholonomic constraints. Autom. Sci. Eng. IEEE Trans. **4**(1), 98–104 (2007). doi:10.1109/TASE.2006.872110
7. Reeds, J., Shepp, L.: Optimal paths for a car that goes both forwards and backwards. Pac. J. Math. **145**(2), 367–393 (1990)
8. Sundar, K., Rathinam, S.: Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots. Autom. Sci. Eng. IEEE Trans. **11**, 287 (2013). doi:10.1109/TASE.2013.2279544
9. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2001)
10. YadlapallI, S., Rathinam, S., Darbha, S.: 3-Approximation algorithm for a two depot, heterogeneous traveling salesman problem. Optim. Lett. **6**, 1–12 (2010). doi:10.1007/s11590-010-0256-0
11. Gørtz, I.L., Molinaro, M., Nagarajan, V., Ravi, R.: Capacitated vehicle routing with non-uniform speeds. In: Integer Programming and Combinatoral Optimization (IPCO), LNCS 6655, pp. 235–247. Springer (2011)
12. Bae, J., Rathinam, S.: An approximation algorithm for a heterogeneous traveling salesman problem. ASME Dynamic Systems and Control Conference, pp. 637–644. Arlington, Virginia, 31 Oct–2 Nov 2011