A Primal-Dual Algorithm for a Heterogeneous Traveling Salesman Problem

Jungyun Bae¹, Sivakumar Rathinam²

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 a minimum. We present a primal-dual algorithm for a variant of this routing problem that provides an approximation ratio of 2.

Keywords

Approximation algorithms, Primal-Dual method, Traveling Salesman Problem, Heterogeneous vehicles, Prize collecting TSP

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 [1] at the Air Force Research Laboratory, a team of unmanned vehicles are required to monitor a set of targets and send information/video about the targets to the ground station controlled by a human operator. The human operator may further add new locations of potential targets or task the vehicles to revisit the targets at different angles. Once the human operator enters his/her input through the human-machine interface, the central computer associated with the interface has 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 an objective that depends on the distances traveled by the vehicles is a minimum. A common objective that is used for these applications is the sum of the total distances traveled by all the vehicles. If there is only one vehicle, this routing problem is referred to as the Traveling Salesman Problem (TSP) in the literature. If there are multiple vehicles that (possibly) start from different initial locations or depots, then this routing problem is referred to 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 to include other kinematic constraints of the vehicles using the results in [2], [3].

A multiple depot, TSP is a generalization of the single TSP and is NP-Hard. This routing problem is further complicated if the vehicles involved are heterogeneous. In this article, vehicles are considered to be heterogeneous if the distance to travel between any two targets depend on the type of the vehicle used. In the context of unmanned applications, as a multiple depot heterogeneous TSP is generally a subproblem that needs to be solved, we are interested in developing fast algorithms that produce approximate solutions than find optimal solutions that may be relatively difficult to solve. Therefore, the main focus of this article is to develop 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.

- 1. Graduate Student, Department of Mechanical Engineering, Texas A & M University, College Station, Texas, U.S.A 77843.
- 2. Assistant Professor, Department of Mechanical Engineering, Texas A & M University, College Station, Texas, U.S.A 77843.

The objective of this article is to develop a primal-dual algorithm for a two depot, heterogeneous TSP (2DHTSP). In addition to assuming that the costs satisfy the triangle inequality for each vehicle, we consider a variant of the problem where the cost of traveling between any two targets for the first vehicle is at most equal to the cost of traveling between the same targets for the second vehicle. Using these assumptions, we show that the developed primal-dual algorithm has an approximation ratio of 2. We are motivated to address this variant of the 2DHTSP due to the following reasons:

- 1. The 2DHTSP is one the simplest cases of the general multiple depot, heterogeneous TSP. The objective of this work is to first develop good algorithms that can handle these simpler cases 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[4]. If the approach angle at each target is given and the minimum turning radius of the first vehicle is at most equal to the minimum turning radius of the second vehicle, it follows that the optimal distance required to travel between any two targets for the first vehicle will be at most equal to the optimal distance required for the second vehicle. Therefore, the problem addressed in this article is a useful variant to address.
- 3. The 2DHTSP is a generalization of a 2 depot, homogeneous TSP where there are additional *vehicle-target* constraints which require one of the vehicles to necessarily visit a given subset of targets in addition to visiting any common target available for both the vehicles. This variant of 2 depot, homogeneous TSP arises in applications where the distance to travel between the targets are identical for both the vehicles, but one of the vehicles carry sensors that require the vehicle to visit a subset of targets compulsorily.
- 4. For some missions involving identical vehicles, it is sometimes necessary to minimize the maximum cost incurred by any of the vehicles. This problem is referred to as the min-max, multiple depot, homogeneous TSP in the literature. If there are only two vehicles involved, one can use the variant of the heterogeneous TSP considered in this article to compute bounds for the min-max problem. Specifically, let $TOUR_1$ and $TOUR_2$ denote a feasible pair of tours for the first and the second vehicle respectively. Also, for i = 1, 2, let $cost(TOUR_i)$ denote the cost of traversing the tour for the i^{th} vehicle. Then, the min-max problem can be formulated as $\min_{TOUR_1, TOUR_2} z$ subject to the constraints $cost(TOUR_1) \leq z$, and $cost(TOUR_2) \leq z$. By dualizing the constraints, one obtains a relaxed problem of the form $\max_{\pi_1+\pi_2=1} \min_{TOUR_1, TOUR_2} [\pi_1 cost(TOUR_1) + \pi_2 cost(TOUR_2)]$. Therefore, for a given value of the penalty variable π_1 , the relaxation involves solving the heterogeneous TSP considered in this article.

Without the assumptions on the costs of the two vehicles, the 2DHTSP is a generalization of the standard variant of the prize collecting TSP considered by Goemans and Williamson in [5]. In this variant, each target essentially 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 cost of the tour plus the sum of the penalties of each target not present in the tour is a minimum. For any two vertices i and j, if π_i , π_j denote the penalties of i and j respectively, then one can pose the prize collecting TSP as a 2DHTSP by setting the cost of traveling the edge joining vertices i and j for the second vehicle to be equal to $\frac{\pi_i + \pi_j}{2}$. Essentially, by choosing the penalty variable corresponding to the second depot to be equal to 0, one can deduce that the travel cost for the second vehicle is actually equal to the sum of the penalties of the targets not present in the tour of the first vehicle. Even though there are no penalties explicitly mentioned in the 2DHTSP, the tour cost for the second vehicle which essentially account for targets not visited by the first vehicle act as penalties. Essentially, our algorithm is based on the well known moat growing procedure proposed by Goemans and Williamson in [5]. 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 [5].

Most of the work in the literature related to approximation algorithms for multiple depot, TSPs deal with identical vehicles. For example, when the costs satisfy the triangle inequality, there are several approximation algorithms for the multiple depot, homogeneous TSP in [3],[6],[7],[8]. Recently, a 3-approximation algorithm was presented for a two depot, heterogeneous TSP in [9]. This algorithm partitions the targets by solving a linear programming relaxation and then uses Christofides algorithm [10] to find a sequence of targets for each vehicle.

The 2-approximation algorithms available in the literature for the multiple depot, TSP generally follow a two-step procedure. In the first step, a constrained forest problem which 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. From these Eulerian graphs, a tour can be found for each vehicle by short-cutting any target already visited by a vehicle. In this article, we follow a similar procedure where we first find a heterogeneous spanning forest using a primal-dual algorithm by solving a relaxation of the 2DHTSP. Then, the edges in the heterogeneous spanning forest are doubled to obtain an Eulerian graph for each vehicle. Given these Eulerian graphs, one can [11] always find a tour for each vehicle that visits each of the targets exactly once. 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 2DHTSP in polynomial time. Hence, it follows that the approximation ratio of the proposed procedure is 2.

I. Problem Statement

Let $D = \{d_1, d_2\}$ denote the two depots (initial locations) corresponding to the first and the second vehicle respectively. Let T be the set of targets to be visited by both the vehicles. Let $V_1 := T \bigcup \{d_1\}$ be the set of vertices corresponding to the first vehicle. Similarly, let $V_2 := T \bigcup \{d_2\}$ be the set of vertices corresponding to the second vehicle. For i = 1, 2, let E_i denote the set of all the edges that join any two distinct vertices in V_i . Let the cost of traversing an edge $e \in E_1$ for the first vehicle be denoted by $cost_e^1$. Similarly, let the cost of traversing an edge $e \in E_2$ for the second vehicle be denoted by $cost_e^2$. We will assume that it is always cheaper to travel between any two targets using the first vehicle as compared to using the second vehicle, *i.e.*, for any edge e joining two targets, $cost_e^1 \leq cost_e^2$. We also assume that the costs satisfy the triangle inequality for both the vehicles.

A tour for the first vehicle starts from its depot d_1 , visits a set of targets in a sequence and finally returns to d_1 . A tour for the second vehicle starts from its depot d_2 , visits a set of targets in a sequence and finally returns to d_2 . The objective of the 2DHTSP is to find a tour for each vehicle such that each target is visited exactly once by some vehicle and the sum of the cost of the edges traveled by both the vehicles is a minimum.

II. Problem formulation

Let x_e be an integer variable that represents whether edge $e \in E_1$ is present in the tour corresponding to the first vehicle. For any edge e joining two targets, x_e can take values only in the set $\{0,1\}$; $x_e = 1$ if e is present in the tour of the first vehicle and $x_e = 0$ otherwise. In order for a tour to visit just one target if required, x_e is allowed to choose 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 tour corresponding to the second vehicle. Let z_U be a binary variable that determines the partition of targets connected to the first and the second depot; z_U is equal to 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. There is at most one subset of targets, U, that is allowed to have z_U to be equal to 1. Let $\delta_i(S)$ (for i = 1, 2) denote the subset of all the edges of E_i with one end in S and an other end in $V_i \setminus S$. $\delta_i(S)$ is also referred to as the cut set of S corresponding to the i^{th} vehicle.

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

$$C_{lp} = \min \sum_{e \in E_1} cost_e^1 \ x_e + \sum_{e \in E_2} cost_e^2 \ y_e$$

$$\sum_{e \in \delta_1(S)} x_e + 2 \sum_{T \supseteq U \supseteq S} z_U \ge 2 \qquad \forall S \subseteq T, \tag{1}$$

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

$$\sum_{U \subset T} z_U \le 1,\tag{3}$$

$$x_e, y_e \in \{0, 1\} \ \forall e \text{ joining any two targets},$$
 (4)

$$x_e \in \{0, 1, 2\} \ \forall e \text{ joining } d_1 \text{ and a target},$$
 (5)

$$y_e \in \{0, 1, 2\} \ \forall e \text{ joining } d_2 \text{ and a target},$$
 (6)

$$z_U \in \{0, 1\} \ \forall U \subseteq T. \tag{7}$$

Consider a Linear Programming (LP) relaxation of the above integer program where the constraints (3)-(7) are relaxed as follows:

$$C_{lp} = \min \sum_{e \in E_1} cost_e^1 \ x_e + \sum_{e \in E_2} cost_e^2 \ y_e \tag{8}$$

$$\sum_{e \in \delta_1(S)} x_e + 2 \sum_{T \supseteq U \supseteq S} z_U \ge 2 \qquad \forall S \subseteq T, \tag{9}$$

$$\sum_{e \in \delta_2(S)} y_e \ge 2 \sum_{T \supseteq U \supseteq S} z_U \quad \forall S \subseteq T, \tag{10}$$

$$x_e \ge 0 \ \forall e \in E_1, \quad y_e \ge 0 \ \forall e \in E_2,$$

$$z_U \ge 0 \ \forall U \subseteq T. \tag{11}$$

A dual of the above LP relaxation can be formulated as follows:

$$C_{dual} = \max \ 2 \sum_{S \subseteq T} Y_1(S) \tag{12}$$

$$\sum_{S:e\in\delta_1(S)} Y_1(S) \le cost_e^1 \qquad \forall e \in E_1,$$
(13)

$$\sum_{S:e \in \delta_2(S)} Y_2(S) \le cost_e^2 \qquad \forall e \in E_2, \tag{14}$$

$$\sum_{S \subseteq U} Y_1(S) \le \sum_{S \subseteq U} Y_2(S) \quad \forall U \subseteq T, \tag{15}$$

$$Y_1(S), Y_2(S) \ge 0 \qquad \forall S \subseteq T.$$
 (16)

We use the above dual problem to find a Heterogeneous Spanning Forest (HSF). A HSF is a collection of two trees where the first tree spans a subset of targets and d_1 , and the second tree connects the remaining set of targets to d_2 . In the next section, we discuss the main ideas involved in the primal-dual algorithm that finds a HSF. We later present the details of the algorithm and show that the cost of this HSF is at most equal to the optimal cost of the above dual. This leads to a 2-approximation algorithm for the 2DHTSP.

III. MAIN IDEAS OF THE PRIMAL DUAL ALGORITHM

The primal-dual algorithm follows the greedy procedure outlined by Goemans and Williamson in [5]. The basic structure of the algorithm involves maintaining a forest of edges corresponding to each vehicle, and a solution to the dual problem. The edges in the forests are candidates for the set of edges that finally appear in the output (HSF) of the algorithm. Suppose F_1 and F_2 denote the forest corresponding to the first and the second vehicle respectively. Let the set of connected components in F_1 and F_2 be denoted by C_1 and C_2 respectively. Initially, both C_1 and C_2 consist of components where each vertex is in its own connected component, i.e., $C_1 = \{\{v\} : v \in V_1\}$ and $C_2 = \{\{v\} : v \in V_2\}$. That is, both F_1 and F_2 are empty. All the components are initially active except the components that contain the depots (Refer to the figures 1-8 for an illustration of the algorithm). Also, all the dual variables are set to zero.

The primal-dual algorithm is an iterative algorithm where in each iteration, at most one edge is added between two distinct components of F_1 or F_2 thus merging the two components. The choice of selecting the appropriate edge to be added is based on a dual solution which is also updated during each iteration. Specifically, in each iteration, the algorithm uniformly increases the dual variable of each active component by a value that is as large as possible such that none of the constraints in (13)-(15) are violated. When the dual variables are increased, one of the following outcomes is possible:

• If any of the constraints in (13)-(14) becomes tight for some edge $(u, v) \in E_i$, i = 1, 2 between two distinct components in F_i , then the algorithm adds (u, v) to F_i and merges the two components (Refer to figures 2-4). If the merged component contains a depot, it becomes inactive; otherwise it is active. We can also explain this outcome in the following way: Suppose $p_i(u) := \sum_{S:u \in S} Y_i(S)$ is the total price all the components containing target u are willing to pay to develop a network F_i that can connect u to depot d_i . Then, the edge e := (u, v) is added to F_i when $p_i(u) + p_i(v) = cost_e^i$, i.e., the price paid by the components containing u and the components containing v equals the cost of adding an edge (u, v) to the network. If a component \overline{C} of F_2 merges with the depot d_2 (figure 4), then $\overline{C} \bigcup \{d_2\}$ becomes inactive, and the total price $\sum_{S \subset \overline{C}} Y_2(S)$ serves as an upper bound for $\sum_{S \subset \overline{C}} Y_1(S)$.

• If a constraint in (15) becomes tight for a component \overline{C} , then \overline{C} is deactivated in F_1 (Refer to figures 5,7). This outcome occurs when the total price $(\sum_{S\subseteq \overline{C}} Y_1(S))$ that all the vertices in \overline{C} are willing to pay to get connected to d_1 becomes as costly as the total price $(\sum_{S\subseteq \overline{C}} Y_2(S))$ that the same vertices have already paid to get connected to d_2 .

The iterative process terminates when all the components become inactive. The final step of the algorithm removes any unnecessary edges (refer to figure 8) that are not required to be in F_1 or F_2 using a marking procedure that was previously used for the prize collecting TSP in [5].

There is a key feature to note in our primal-dual procedure. We ensure that the dual variables of all the active components in F_1 and F_2 are increased uniformly by the same amount in each iteration. The components in F_1 tend to merge first as compared with the components in F_2 due to the choice of our dual increase and the fact that it is cheaper to travel between any two targets using the first vehicle as compared with the second vehicle. As the algorithm progresses, for any $U \subseteq T$, it is likely that $\sum_{S \subset U} Y_1(S) < \sum_{S \subset U} Y_2(S)$ as there may be fewer active components of F_1 in U as compared to F_2 . For example, in figure 2, there is exactly one active component of F_1 in $U := \{t_2, t_3\}$ as compared to two active components of F_2 in U. Even if the edges in the forests contain a feasible solution for the HSF, we do not terminate the algorithm if there is at least one active component \overline{C} of F_1 such that $\sum_{S \subset \overline{C}} Y_1(S) < \sum_{S \subset \overline{C}} Y_2(S)$. For example, consider the snap shot of the algorithm in figure 6. Each target in this snap shot is either connected to d_1 or d_2 and hence, one can possibly terminate the algorithm at this step. However, we find that the component $\overline{C} := \{t_5, t_6, t_7, t_8\}$ of F_1 is still active and $\sum_{S \subseteq \overline{C}} Y_1(S) < \sum_{S \subseteq \overline{C}} Y_2(S)$, i.e., the total price that \overline{C} has paid till this iteration to get connected to d_1 is less than the total price that the \overline{C} has already paid to get connected to d_2 . Therefore, the algorithm continues to increase $Y_1(\overline{C})$ to check if all the vertices in \overline{C} can get connected to d_1 at a lower cost. This feature is useful from the point of obtaining a good approximation ratio because the cost of the edges in the HSF has to be bounded in terms of the cost of the dual solution which in turn depends only on $\sum_{S \subset T} Y_1(S)$. Hence, the algorithm terminates only when all the components in F_1 become inactive.

There are other possible ways of increasing the dual variables so that the primal-dual algorithm is simpler. For example, one can increase the dual variable associated with each active component in F_1 by the same amount while growing the dual variable of an active component in F_2 at a slower rate so that the associated constraints in (15) are always tight in each iteration of the algorithm. Specifically, if a dual variable $Y_1(S)$ is increased by ϵ , then the dual variable of each active component of F_2 in S can be increased by $\frac{\epsilon}{k}$ where k is the number of active components of F_2 in S. Even though this type of a dual increase will result in a simpler primal-dual procedure, the active components of F_2 could grow at different rates. If the active components grow at different rates in a forest, as pointed out in the case of the minimum spanning tree problem [5], one can develop instances where the approximation ratio of the algorithm may be greater than 2.

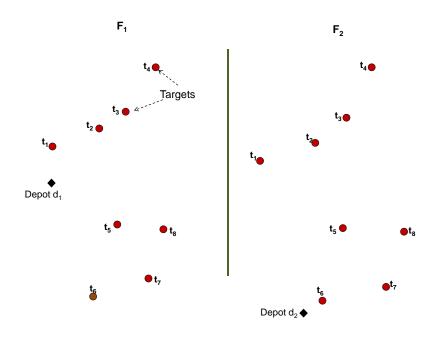


Fig. 1

An example illustrating the basic steps in the primal dual algorithm. There are 8 targets 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.

IV. IMPLEMENTATION DETAILS OF THE PRIMAL-DUAL ALGORITHM

The initialization, the main steps and the final pruning step of the primal-dual algorithm are presented in **Algorithms 1**, **2 and 3**. For any $\forall C \in \mathcal{C}_1$, the internal variable w(C) keeps track of $\sum_{S \subseteq C} Y_1(S)$, i.e, $w(C) = \sum_{S \subseteq C} Y_1(S)$. Similarly, $\forall C \in \mathcal{C}_1$, Bound(C) keeps track of $\sum_{S \subseteq C} Y_2(S)$. Essentially, w(C) and Bound(C) are used to enforce the constraints in (15). Initially, all the dual variables, w(C) and Bound(C) are set to zero. (Refer to the initialization steps in algorithm 1). Also, each vertex in V_1 is initially unmarked.

As the components in C_1 tend to merge first, we refer to the components in C_1 as parents and the components in C_2 as their children. For components $C_1 \in C_1$ and $C_2 \in C_2$, we define C_1 as the parent of C_2 and C_2 as a child of C_1 if $C_2 \subseteq C_1$ and $d_2 \notin C_2$. For any component $C_1 \in C_1$, we use $Children(C_1)$ to denote all the children of C_1 present in C_2 . For any component $C_2 \in C_2$, $d_2 \notin C_2$, we use $Parent(C_2)$ to denote the parent of C_2 present in C_1 . According to the definition, if C_2 contains the depot d_2 , C_2 doesn't have a parent; however, to simplify the presentation, we let $Parent(C_2)$ be an empty set if C_2 contains d_2 . At the start of the algorithm, for any target $v \in T$, $Children(\{v\})$ is assigned to be equal to $\{v\}$ and $Parent(\{v\})$ is assigned to be equal to $\{v\}$ and Parent(v) is assigned to be equal to $\{v\}$ and Parent(v) is assigned to be equal to $\{v\}$ and Parent(v) is assigned to be equal to $\{v\}$ and Parent(v) is assigned to be equal to $\{v\}$. Also, the components that consist of just the depots neither have a parent or a child Parent(v) is an algorithm Parent(v).

In each iteration of the algorithm, the dual variable corresponding to each of the active components in C_1 and C_2 are

increased as much as possible by the same amount until one of the constraints stated in (13-15) becomes tight (Refer to lines 2-5 of the algorithm 2). For any two disjoint components C_{1x} , $C_{1y} \in C_1$, consider the constraint in (13) corresponding to the edge $e = \{u, v\}$ that could potentially connect vertex u in C_{1x} to vertex v in C_{1y} : $\sum_{S:e\in\delta_1(S)} Y_1(S) \leq cost_e^1$. Since e has not yet been added to F_1 , this constraint can be re-written as $\sum_{S:u\in S} Y_1(S) + \sum_{S:v\in S} Y_1(S) \leq cost_e^1$, or as $p_1(u) + p_1(v) \leq cost_e^1$. Therefore, to add an edge (u,v) during the iteration, each of the dual variables of the active components have to be increased by an amount given by $\frac{cost_e^1 - p_1(u) - p_1(v)}{active_1(C_{1x}) + active_1(C_{1y})}$ in order to make the constraint, $p_1(u) + p_1(v) \leq cost_e^1$, tight. Hence, in step 2 of the algorithm 2, we basically find the minimum amount by which each of the dual variables of the active components in C_1 have to be increased so that none of the constraints are violated and at least one of the constraints in (13) just becomes tight. Similarly, in step 3 of the algorithm 2, we find the minimum amount by which each of the dual variables of the active components in C_2 have to be increased so that none of the constraints are violated and at least one of the constraints in (14) just becomes tight. For i = 1, 2, note that $p_i(u)$ is increased during an iteration only if u belongs to a component in C_i that is active; else $p_i(u)$ does not change.

If a constraint in (13) becomes tight for some edge $e \in E_1$, F_1 is augmented with this new edge and the two components (say C_{1x} , C_{1y} in C_1) connected by e are merged to form a single connected component. The children of each of the two components C_{1x} , C_{1y} now together become the children of the resulting component $C_{1x} \cup C_{1y}$. The resulting component becomes inactive if it contains the depot d_1 ; otherwise, it is active. In the case when the resulting component becomes

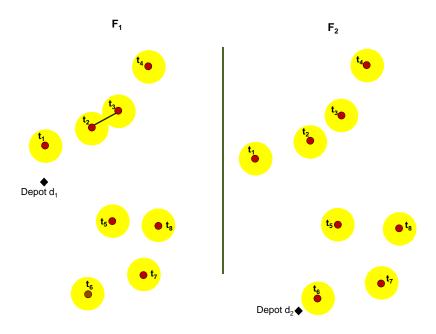


Fig. 2

Snap shot of the forests at the end of the first iteration. 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 the 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 $cost_e^1$.

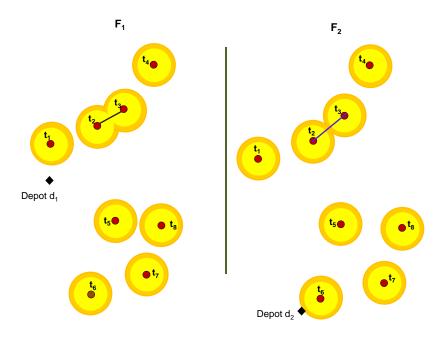


Fig. 3

SNAP SHOT OF THE FORESTS AT THE END OF THE SECOND ITERATION. EDGE (t_2,t_3) is added to F_2 as the sum of the prices paid by the components containing targets t_2 and t_3 becomes equal to the cost of constructing the edge (t_2,t_3) for the second vehicle.

inactive, all the children of the resulting component also become inactive (Refer to lines 17-27 of the algorithm 2).

Similarly, if one of constraints in (14) becomes tight for some edge $e \in E_2$, F_2 is augmented with this new edge and the two components (say C_{2x}, C_{2y} in C_2) connected by e are merged to form a single connected component (Refer to lines 29-39 of the algorithm 2). The resulting component becomes inactive if it contains the depot d_2 ; otherwise, it is active. In the case when the resulting component is active, the parent of either C_{2x} or C_{2y} is assigned as the parent of the resulting component (It turns out that due to our assumptions on the costs, when the algorithm enters this part of the implementation, both C_{2x} and C_{2y} must be active and must be the children of the same parent; we will show this result later in lemma 1). In the case when the resulting component becomes inactive, and say C_{2x} was the active component during the iteration which did not contain the depot, the parent of C_{2x} loses C_{2x} as its child.

Once an active parent \overline{C} loses all its children, $Bound(\overline{C})$ specifies the maximum value that can be attained by $w(\overline{C})$. Suppose an active component $\overline{C} \in \mathcal{C}_1$ does not have any children and the increase in the dual variables results in the constraint $w(\overline{C}) \leq Bound(\overline{C})$ becoming tight. Then, the algorithm deactivates \overline{C} and marks each of the unmarked vertices in the component with \overline{C} (Refer to lines 41-42 of the algorithm 2).

The algorithm terminates when all the components in C_1 become inactive. After termination, the algorithm makes one final pass at all the edges (refer to algorithm 3) and removes any edge that is not required to be in the HSF. Basically,

during the final step of the primal dual algorithm, any unnecessary edges in F_1 and F_2 are pruned further to find a tree for each of the vehicles. Specifically, the tree F'_1 corresponding to the first vehicle is obtained from F_1 by removing as many edges as possible from F_1 so that the following properties hold: 1) All the unmarked vertices of V_1 are connected to the first depot d_1 ; 2) If any vertex with label C is connected to the depot d_1 , then any other vertex with a label $C' \supseteq C$ is also connected to the depot d_1 . The tree F'_2 corresponding to the second vehicle is obtained from F_2 by removing as many edges as possible from F_2 such that any target not spanned by F'_1 is connected to d_2 in F'_2 .

Since the sum of the number of components in C_1 , the number of active components in C_1 and the number of components in C_2 decreases at least by one during each iteration, the primal-dual algorithm must terminate after at most 3|T|+2 iterations. Using the techniques given in [5], this primal-dual algorithm can be implemented in $|T|^2 \log |T|$ steps.

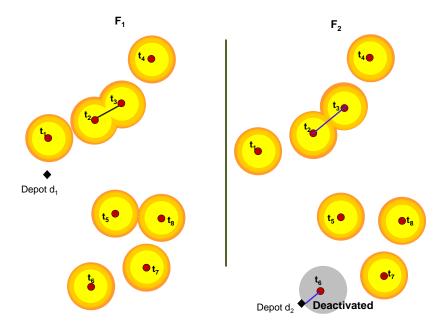


Fig. 4

Snap shot of the forests at the end of the third iteration. 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 t_6 is now connected to d_2 in F_2 . The dual variable $Y_2(\{t_6\})$ does not increase further and will serve as an upper bound on $Y_1(\{t_6\})$.

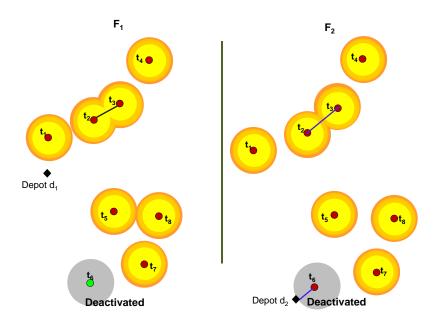


Fig. 5

Snap shot of the forests at the end of the fourth iteration. Component $\{t_6\}$ in F_1 is deactivated because $Y_1(\{t_6\})$ becomes equal to $Y_2(\{t_6\})$.

```
Algorithm 1 Primal-dual algorithm: Initialization
```

```
F_1 \leftarrow \emptyset; \quad F_2 \leftarrow \emptyset; \quad \mathcal{C}_1 \leftarrow \{\{v\} : v \in V_1\}; \quad \mathcal{C}_2 \leftarrow \{\{v\} : v \in V_2\} for v \in V_1 do
Unmark v; p_1(v) \leftarrow 0; w(\{v\}) \leftarrow 0; Bound(\{v\}) \leftarrow 0
If v = d_1, then Children(\{v\}) \leftarrow \emptyset, else Children(\{v\}) \leftarrow \{v\}
If v = d_1, then active_1(\{v\}) = 0, else active_1(\{v\}) = 1
end for
for v \in V_2 do
p_2(v) \leftarrow 0
If v = d_2, then Parent(\{v\}) \leftarrow \emptyset, else Parent(\{v\}) \leftarrow \{v\}
If v = d_2, then active_2(\{v\}) = 0, else active_2(\{v\}) = 1
end for
```

A. Properties of the primal-dual algorithm

Consider any target $u \in T$. At the start of the k^{th} iteration, let $C_1^k(u)$ denote the component in \mathcal{C}_1 containing u, and $C_2^k(u)$ represent the component in \mathcal{C}_2 containing u.

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

- 1. $C_2^k(u)$ is always a child of $C_1^k(u)$, i.e., $C_2^k(u) \subseteq C_1^k(u)$ unless $C_2^k(u)$ contains the depot d_2 .
- 2. $active_1(C_1^k(u)) \ge active_2(C_2^k(u))$.

```
Algorithm 2: Primal-dual algorithm - Main steps
     while \exists C \in \mathcal{C}_1 such that active_1(C) = 1 do
     Find edge e_1 = (i,j) \in E_1 with i \in C_{1x}, j \in C_{1y} where C_{1x}, C_{1y} \in C_1, C_{1x} \neq C_{1y} that minimizes \varepsilon_1 = C_{1x}
          (cost_{e_1}^1 - p_1(i) - p_1(j))
      \overline{active_1(C_{1x}) + active_1(C_{1y})}
     Find edge e_2 = (i,j) \in E_2 with i \in C_{2x}, j \in C_{2y} where C_{2x}, C_{2y} \in C_2, C_{2x} \neq C_{2y} that minimizes \varepsilon_2 =
          (cost_{e_2}^2 - p_2(i) - p_2(j))
      \overline{active_2(\tilde{C}_{2x}) + active_2(C_{2y})}
     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(\overline{C})
     \varepsilon_{min} = \min(\varepsilon_1, \varepsilon_2, \varepsilon_3)
     for each active component C \in \mathcal{C}_1 do
6:
     w(C) \leftarrow w(C) + \varepsilon_{min}
7:
     For all v \in C, p_1(v) \leftarrow p_1(v) + \varepsilon_{min}
8:
     Bound(C) \leftarrow Bound(C) + \varepsilon_{min}|Children(C)|
9:
10: end for
     for each active component C \in \mathcal{C}_2 do
     For all v \in C, p_2(v) \leftarrow p_2(v) + \varepsilon_{min}
12:
13: end for
14: switch \varepsilon_{min}
     //Comment: If more than one value in \{\varepsilon_1, \varepsilon_2, \varepsilon_3\} is equal to \varepsilon_{min}, then give priority first to Case \ \varepsilon_1,
      then to Case \ \varepsilon_2 and finally to Case \ \varepsilon_3
16:
     Case \varepsilon_1:
17:
          F_1 \leftarrow F_1 \bigcup \{e_1\}
          C_1 \leftarrow C_1 \bigcup \{C_{1x} \bigcup C_{1y}\} - C_{1x} - C_{1y}
18:
19:
          w(C_{1x} \bigcup C_{1y}) \leftarrow w(C_{1x}) + w(C_{1y})
          Children(C_{1x} \bigcup C_{1y}) \leftarrow Children(C_{1x}) \bigcup Children(C_{1y})
20:
          For all C \in Children(C_{1x} \bigcup C_{1y}), Parent(C) \leftarrow C_{1x} \bigcup C_{1y}
21:
          Bound(C_{1x} \bigcup C_{1y}) \leftarrow Bound(C_{1x}) + Bound(C_{1y})
22:
          if d_1 \in C_{1x} \bigcup C_{1y}, then
23:
          active_1(C_{1x} \bigcup C_{1y}) = 0
24:
          active_2(C) = 0 for all C \in Children(C_{1x} \bigcup C_{1y})
25:
          else active_1(C_{1x} \bigcup C_{1y}) = 1
26:
          end
27:
      Case \varepsilon_2:
28:
          F_2 \leftarrow F_2 \bigcup \{e_2\}
29:
          C_2 \leftarrow C_2 \bigcup \{C_{2x} \bigcup C_{2y}\} - C_{2x} - C_{2y}
30:
          if d_2 \in C_{2x} \bigcup C_{2y} then
31:
            active_2(C_{2x} \bigcup C_{2y}) \leftarrow 0
32:
            Parent(C_{2x} \bigcup C_{2y}) \leftarrow \emptyset
33:
            Let C \in \{C_{2x}, C_{2y}\} such that d_2 \notin C; Children(Parent(C)) \leftarrow Children(Parent(C)) - C
34:
          else active_2(C_{2x} \bigcup C_{2y}) \leftarrow 1
35:
            C_{temp} \leftarrow Parent(C_{2x})
36:
            Parent(C_{2x} \bigcup C_{2y}) \leftarrow C_{temp}
37:
            Children(C_{temp}) \leftarrow Children(C_{temp}) \bigcup \{C_{2x} \bigcup C_{2y}\} - C_{2x} - C_{2y}
38:
           end if
39:
      Case \varepsilon_3:
40:
          active_1(\overline{C}) \leftarrow 0
41:
          Mark all the unlabeled vertices of \overline{C} with label \overline{C}
42:
43: end switch
44: end while
```

Algorithm 3: Primal-dual algorithm - Pruning step

- 1: F'_1 is obtained from F_1 by removing as many edges as possible from F_1 so that the following properties hold: 1) All the unmarked vertices of V_1 are connected to the first depot d_1 ; 2) If any vertex with label C is connected to the depot d_1 , then any other vertex with a label $C' \supseteq C$ is also connected to the depot d_1 .
- 2: F'_2 is obtained from F_2 by removing as many edges as possible from F_2 such that any target not spanned by F'_1 is spanned by F'_2 .

Proof: Let us prove this lemma by induction. At the start of the first iteration, $C_1^1(u) = C_2^1(u) = \{u\}$ and the components $C_1^1(u)$, $C_2^1(u)$ are both active. Therefore, lemma 1.1 and lemma 1.2 are correct for k = 1. Now, let us assume that the statements in the lemma are true for the l^{th} iteration for any $l = 1, \dots, k$. As $active_1(C_1^l(u)) \geq active_2(C_2^l(u))$ for any $l = 1, \dots, k$, it follows that $p_1(u) \geq p_2(u)$ at the start of the k^{th} iteration.

Proof of lemma 1.1: During the k^{th} 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 C_1 , or, 2) $C_2^k(u)$ merges with another component in C_2 , or, 3) $C_1^k(u)$ gets deactivated because its corresponding constraint in (15) becomes tight. It is easy to note that $C_2^{k+1}(u)$ will remain a child of $C_1^{k+1}(u)$ in the first case. $C_1^k(u)$ can get deactivated as 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 1.1 is true by default 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 the depot d_2 , then lemma 1.1 is true for l = k + 1 by default. If $C_2^k(u)$ is active and merges with another active component $C_2^k(v)$ corresponding to target v, we claim that both $C_2^k(u)$ and $C_2^k(v)$ must have the same parent. If this is not true, note that

$$\varepsilon_1 = \frac{cost_{(u,v)}^1 - p_1(u) - p_1(v)}{active_1(C_1^k(u)) + active_1(C_1^k(v))} \le \frac{cost_{(u,v)}^2 - p_2(u) - p_2(v)}{active_2(C_2^k(u)) + active_2(C_2^k(v))} = \varepsilon_2. \tag{17}$$

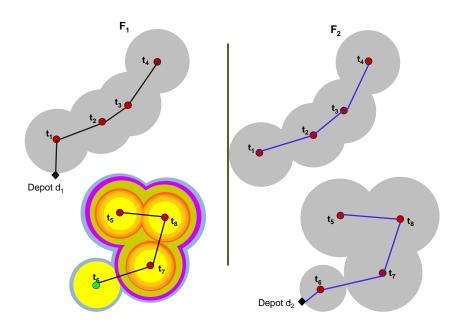


Fig. 6

Snap shot of the forests after few iterations of the algorithm. All the 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. So, the algorithm can possibly stop if needed. However, it turns out that the total price paid by the components in \overline{C} to get connected to the first depot is less than the total price the components in \overline{C} have already paid for F_2 . Therefore, $Y_1(\overline{C})$ is increased further in the next iteration to check if \overline{C} can get connected to d_1 at a lower cost.

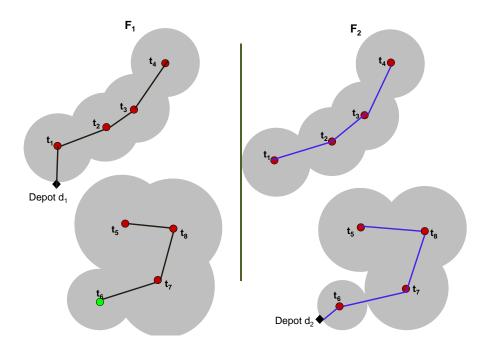


Fig. 7

Snap shot of the forests at the end of the main loop of the algorithm. $\overline{C}:=\{t_5,t_6,t_7,t_8\}$ of F_1 is deactivated because $\sum_{S\subseteq \overline{C}} Y_1(S)$ becomes equal to $\sum_{S\subseteq \overline{C}} Y_2(S)$. The main part of the algorithm terminates because all the components are now inactive.

Therefore, the algorithm 2 will not merge $C_2^k(u)$ and $C_2^k(v)$ unless it merges the parents of $C_2^k(u)$ and $C_2^k(v)$. If $C_2^k(u)$ and $C_2^k(v)$ have the same parent, it then 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 its parent contains d_1 , we claim that $C_2^k(u)$ will never merge with any other component. If this claim is not true and say $C_2^k(u)$ (which is inactive) merges with some other component $C_2^k(v)$ corresponding to target v, then $C_1^k(u) \neq C_1^k(v)$ and $C_2^k(v)$ must be active. Again from equation (17), the algorithm will prefer to merge $C_1^k(u)$ and $C_1^k(v)$ before merging their children, i.e., $C_2^k(u)$ 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. Therefore, $C_2^k(u)$ will remain inactive and will never merge with any other component during the k^{th} iteration. Hence, lemma 1.1 is true by default.

Proof of lemma 1.2:

If $C_2^k(u)$ is inactive, either $C_2^k(u)$ must contain the depot d_2 or its parent $C_1^k(u)$ must contain the depot d_1 .

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

If $C_2^k(u)$ is active, then $active_1(C_1^k(u)) \geq active_2(C_2^k(u))$ implies that $C_1^k(u)$ is also active. From lemma 1.1 it follows that $C_1^k(u)$ is a parent of $C_2^k(u)$. Since the component, $C_1^k(u)$, has at least one active child in $C_2^k(u)$, $C_1^k(u)$ can never become inactive due to its associated constraint in (15) during the k^{th} iteration. The only way $C_1^k(u)$ can lead to an inactive $C_1^{k+1}(u)$ is if $C_1^k(u)$ merges with another component containing d_1 during the iteration in which case all the children of $C_1^k(u)$ including $C_2^k(u)$ also get deactivated. Therefore, $active_1(C_1^{k+1}(u)) \geq active_2(C_2^{k+1}(u))$.

Let \mathfrak{X} denote the set of vertices not spanned by F'_1 . Based on the label of each vertex in \mathfrak{X} , \mathfrak{X} can be partitioned into disjoint, deactivated components $\overline{C}_1, \overline{C}_2, \cdots, \overline{C}_m$ where each \overline{C}_i denotes the maximal label of its respective component. The following lemma shows that the primal-dual algorithm produces a feasible solution in which each target is connected to exactly one depot.

Lemma 2: The algorithm produces a feasible, heterogeneous spanning forest, i.e., the trees specified by the collection of edges in F'_1 and F'_2 connect each of the targets to one of the depots. Any vertex spanned by the edges in F'_1 is not spanned by the edges in F'_2 and vice versa.

Proof: The algorithm terminates when all the sets of C_1 become inactive. This is only possible if each of the targets in T is either connected to d_1 or d_2 . Note that F'_1 is formed from F_1 such that each of the unmarked vertices remain connected to d_1 . The only vertices not spanned by F'_1 are some of the marked vertices. These vertices were marked because the components in C_1 that span these vertices were deactivated for making their associated constraints in (15) tight. In addition, a component in C_1 can become deactivated due to a constraint in (15) only if it has already lost all

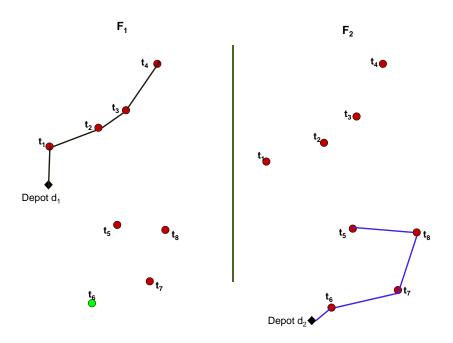


Fig. 8

THE FINAL OUTPUT (HSF) OF THE PRIMAL-DUAL ALGORITHM AFTER THE UNNECESSARY EDGES ARE REMOVED IN THE PRUNING STEP.

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 marked vertices not spanned by F'_1 must be connected to d_2 and spanned by F'_2 . Hence, the algorithm produces a feasible, heterogeneous spanning forest.

Consider any deactivated component $\overline{C}_i \subseteq \mathfrak{X}$. \overline{C}_i can get deactivated during an iteration only if \overline{C}_i does not have children and $\sum_{S \subseteq \overline{C}_i} Y_1(S) = w(\overline{C}_i) = Bound(\overline{C}_i) = \sum_{S \subseteq \overline{C}_i} Y_2(S)$. Note that \overline{C}_i could have lost 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 gets deactivated, no target $u \in \overline{C}_i$ is connected to any other target $v \in T \setminus \overline{C}_i$ in F_1 . As a result, from lemma 1, we claim that u does not have an adjacent vertex v in F_2 such that $v \in T \setminus \overline{C}_i$. If this claim is not true, then from lemma 1 and equation (17), it follows that the algorithm would have added edge (u,v) to F_1 before adding (u,v) to F_2 . Since target u is not connected to target $v \in T \setminus \overline{C}_i$ in F_1 , u and v cannot be connected in F_2 . Therefore, during the construction of F'_2 , all the edges that are incident on any vertex $u \notin \mathfrak{X}$ can be dropped. Hence, any vertex spanned by the edges in F'_1 is not spanned by the edges in F'_2 and vice versa.

The main result of this article is in the following subsection.

B. Proof of the Approximation Ratio

Theorem IV.1: The primal-dual algorithm produces a tree with edges denoted by F'_1 for the first vehicle and a tree with edges denoted by F'_2 for the second vehicle such that the cost of the edges in these trees is bounded by the cost for the dual problem, *i.e.*,

$$\sum_{e \in F_1'} cost_e^1 + \sum_{e \in F_2'} cost_e^2 \le 2 \sum_{S \subseteq T} Y_1(S).$$

Since $2\sum_{S\subseteq T} Y_1(S)$ is a lower bound to the optimal cost of the 2DHTSP, it follows that the cost of the HSF found by the primal dual algorithm is at most equal to the optimal cost of the 2DHTSP. This provides a 2-approximation algorithm for the 2DHTSP.

Proof: In order to prove the above theorem, we first simplify the dual cost obtained by the algorithm as follows:

$$2\sum_{S\subseteq T} Y_1(S) = 2\sum_{S\subseteq T, S\nsubseteq \overline{C}_i, i=1,...,m} Y_1(S) + 2\sum_{i=1}^m \sum_{S\subseteq \overline{C}_i} Y_1(S)$$

$$= 2\sum_{S\subseteq T, S\nsubseteq \overline{C}_i, i=1,...,m} Y_1(S) + 2\sum_{i=1}^m \sum_{S\subseteq \overline{C}_i} Y_2(S).$$
(18)

Now, we express the cost of the edges in the first tree in terms of the dual variables as follows. Note that edge e is added to F_1 and consequently appears in F'_1 only if the corresponding constraint in (13) is tight, i.e., $cost_e^1 = \sum_{S:e \in \delta_1(S)} Y_1(S)$. Therefore,

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

Since $F'_1 \cap \delta_1(S) = 0$ for any $S \subseteq \overline{C}_i$, we can further simplify the above equation to

$$\sum_{e \in F_1'} cost_e^1 = \sum_{S \subseteq T, S \nsubseteq \overline{C}_i, i=1,\dots,m} Y_1(S) |F_1' \bigcap \delta_1(S)|. \tag{19}$$

Similarly, we can also 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 the depot d_2 . An edge e is added to F_2 and consequently appears in F'_{2i} only if the corresponding constraint in (14) 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 end point in $\overline{C}_i \bigcup \{d_2\} \setminus S$.

$$\sum_{e \in F_2'} cost_e^2 = \sum_{i=1}^m \sum_{e \in F_{2i}'} cost_e^2$$

$$= \sum_{i=1}^m \sum_{e \in F_{2i}'} \sum_{S: e \in \overline{\delta}_{2i}(S), S \subseteq \overline{C}_i} Y_2(S)$$

$$= \sum_{i=1}^m \sum_{S \subset \overline{C}_i} Y_2(S) |F_{2i}' \cap \overline{\delta}_{2i}(S)|. \tag{20}$$

Therefore, from equations (18), (19), (20), the proof for the theorem reduces to showing the following result:

$$\sum_{S \subseteq T, S \nsubseteq \overline{C}_i, i=1,..,m} Y_1(S)|F_1' \bigcap \delta_1(S)| + \sum_{i=1}^m \sum_{S \subseteq \overline{C}_i} Y_2(S) |F_{2i}' \bigcap \overline{\delta}_{2i}(S)|$$
(21)

$$\leq 2 \sum_{S \subseteq T, S \nsubseteq \overline{C}_i, i=1,\dots,m} Y_1(S) + 2 \sum_{i=1}^m \sum_{S \subseteq \overline{C}_i} Y_2(S). \tag{22}$$

The above result can be shown by proving that during any iteration, the increase in the primal cost (the left-hand side of the above inequality) is at most equal to the increase in the dual cost (the right-hand side of the above inequality). To see this, let us choose any iteration of the primal-dual algorithm. At the start of this iteration, let N_a be the set of all the active components in C_1 such that each active component in this set is not a subset of \mathfrak{X} and N_d be the set of all the inactive components in C_1 such that each inactive component in this set is not a subset of \mathfrak{X} . Note that one of inactive components of N_d must consist of the depot d_1 . For $i = 1, \dots, m$, let M_{ai} denote the set of all the active components in C_2 such that each active component in this set is a subset of \overline{C}_i . Also, let M_d denote the inactive component in C_2 that consists of the depot d_2 .

Now, form a graph H_1 with components in $N_a \bigcup N_d$ as its vertices and edges $e \in F_1' \cap \delta_1(C)$ for $C \in N_a \bigcup N_d$ as edges of H_1 . H_1 is a tree that spans all the vertices in $N_a \bigcup N_d$. Similarly, form a graph H_{2i} with components in $M_{ai} \bigcup M_d$ as its vertices and edges $e \in F_{2i}' \cap \delta_2(C)$ for $C \in M_{ai} \bigcup \{M_d\}$ as edges of H_{2i} . H_{2i} is a tree that spans all the vertices in $M_{ai} \bigcup \{M_d\}$.

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 the result, the left hand side of the inequality will increase by $\varepsilon_{min}(\sum_{v\in N_a}deg(v,H_1)+\sum_{i=1}^m\sum_{v\in M_{ai}}deg(v,H_{2i}))$ whereas the right hand side of the inequality will increase by $2\varepsilon_{min}(N_a+\sum_{i=1}^mM_{ai})$. Therefore, basically, the proof is complete if we can show that

$$\sum_{v \in N_a} deg(v, H_1) + \sum_{i=1}^m \sum_{v \in M_{ai}} deg(v, H_{2i}) \le 2(|N_a| + \sum_{i=1}^m |M_{ai}|).$$
(23)

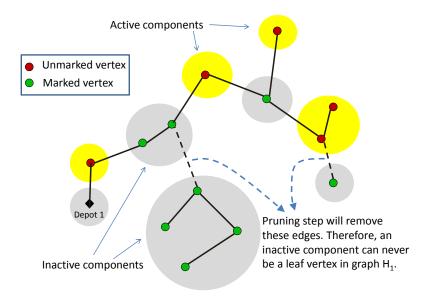


Fig. 9

An example which illustrates that the graph H_1 cannot have an inactive component as its leaf vertex unless it contains d_1 .

The circles indicate all the active and the inactive components corresponding to the first vehicle at the start of an iteration.

We now claim that any vertex v in H_1 that represents an inactive component in 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, which does not contain d_1 , can become inactive in C_1 only if the constraint associated with this component in (15) becomes tight. Therefore, all the vertices in this inactive component must be marked. Also, if vertex v is a leaf $(deg(v, H_1) = 1)$ then pruning all the edges from this inactive component will not disconnect any unmarked target from d_1 . Hence, the pruning step of the algorithm will ensure that an inactive component can never be a leaf vertex in H_1 unless it contains d_1 . Refer to figure 9 for an illustration of this claim. Hence, $\sum_{v \in N_d} deg(v, H_1) \geq 2|N_d| - 1$. We now show the final part of the proof:

$$\sum_{v \in N_{c}} deg(v, H_1) + \sum_{i=1}^{m} \sum_{v \in M_{c,i}} deg(v, H_{2i})$$
(24)

$$= \sum_{v \in N_a \bigcup N_d} deg(v, H_1) - \sum_{v \in N_d} deg(v, H_1)$$
 (25)

$$+\sum_{i=1}^{m} \left[\sum_{v \in M_{ai} \cup \{M_d\}} deg(v, H_{2i}) - deg(M_d, H_{2i}) \right]$$
 (26)

$$\leq \sum_{v \in N_a \bigcup N_d} deg(v, H_1) - \sum_{v \in N_d} deg(v, H_1) \tag{27}$$

$$+\sum_{i=1}^{m} \left[\sum_{v \in M_{ai} \bigcup \{M_d\}} deg(v, H_{2i}) \right]$$
 (28)

(29)

 H_1 is a tree that spans all the vertices in $N_a \bigcup N_d$. Therefore, the sum of the degree of all the vertices in H_1 is $2(|N_a| + |N_d| - 1)$. Similarly, H_{2i} is a tree that spans all the vertices in $M_{ai} \bigcup \{M_d\}$. Therefore, the sum of the degree of all the vertices in H_{2i} is $2|M_{ai}|$. Hence, continuing with the proof,

$$\sum_{v \in N_a} deg(v, H_1) + \sum_{i=1}^m \sum_{v \in M_{ai}} deg(v, H_{2i})$$
(30)

$$\leq 2(|N_a| + |N_d| - 1) - (2|N_d| - 1) + 2\sum_{i=1}^{m} |M_{ai}|$$
(31)

$$<2|N_a| + 2\sum_{i=1}^m |M_{ai}|. (32)$$

Hence proved.

References

- [1] J. E. Davis M. Holland G. L. Feithans, A. J. Rowe and L. Berger, "Vigilant spirit control station (vscs)the face of counter," in *Proc. AIAA Guidance, Navigation and Control Conf. Exhibition.* 2008, AIAA.
- [2] Jae-Ha Lee, Otfried Cheong, Woo-Cheol Kwon, Sung Yong Shin, and Kyung-Yong Chwa, "Approximation of curvature-constrained shortest paths through a sequence of points," in *Proceedings of the 8th Annual European Symposium on Algorithms*, London, UK, 2000, ESA '00, pp. 314–325, Springer-Verlag.
- [3] S. Rathinam, R. Sengupta, and S. Darbha, "A resource allocation algorithm for multivehicle systems with nonholonomic constraints," Automation Science and Engineering, IEEE Transactions on, vol. 4, no. 1, pp. 98-104, 2007.
- [4] J.A. Reeds and L.A. Shepp, "Optimal paths for a car that goes both forwards and backwards," Pacific Journal of Mathematics, vol. 145, no. 2, pp. 367393, 1990.
- [5] Michel X. Goemans and David P. Williamson, "A general approximation technique for constrained forest problems," SIAM J. Comput., vol. 24, no. 2, pp. 296–317, 1995.
- [6] W. Malik, S. Rathinam, and S. Darbha, "An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem," *Operations Research Letters*, vol. 35, no. 6, pp. 747 753, 2007.
- [7] S. Rathinam and R. Sengupta, "3/2-approximation algorithm for two variants of a 2-depot hamiltonian path problem," *Operations Research Letters*, vol. 38, no. 1, pp. 63 68, 2010.
- [8] Zhou Xu and Brian Rodrigues, "A 3/2-approximation algorithm for multiple depot multiple traveling salesman problem," in *Algorithm Theory SWAT 2010*, Haim Kaplan, Ed., vol. 6139 of *Lecture Notes in Computer Science*, pp. 127–138. Springer Berlin / Heidelberg.
- [9] Sai Yadlapalli, Sivakumar Rathinam, and Swaroop Darbha, "3-approximation algorithm for a two depot, heterogeneous traveling salesman problem," Optimization Letters, pp. 1–12, 2010, 10.1007/s11590-010-0256-0.

- [10] Nicos Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Tech. Rep., Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1976.
- $[11]\,$ Vijay V. Vazirani, Approximation~Algorithms,~Springer-Verlag,~Berlin,~2001.