Distributed Vehicle Routing Approximation

Akhil Krishnan
Department of Computing and Software
McMaster University, Canada
Email: krisha4@mcmaster.ca

Mikhail Markov
Department of Computing and Software
McMaster University, Canada
Email: markoma@mcmaster.ca

Borzoo Bonakdarpour
Department of Computing and Software
McMaster University, Canada
Email: borzoo@mcmaster.ca

***Abstract*—The classic vehicle routing problem (VRP) is generally concerned with the optimal design of routes by a fleet of vehicles to service a set of customers by minimizing the overall cost, usually the travel distance for the whole set of routes. Although the problem has been extensively studied in the context of operations research and optimization, there is little research on solving the VRP, where distributed vehicles need to compute their respective routes in a decentralized fashion. Our first contribution is a synchronous distributed approximation algorithm that solves the VRP. Using the duality theorem of linear programming, we show that the approximation ratio of our algorithm is $O(n \cdot (\rho)^{1/n} \cdot log(n+m))$, where ρ is the maximum cost of travel or service in the input VRP instance, n is the size of the graph, and m is the number of vehicles. We report results of simulations and discuss implementation of our algorithm on a real fleet of unmanned aerial systems (UASs) that carry out a set of tasks.**

***Keywords*-Vehicle routing problem; Distributed optimization; Unmanned aerial systems (UAS).**

I. Introduction

The *vehicle routing problem* (VRP) [1] is one of the most studied combinatorial optimization problems and is concerned with the optimal design of routes to be used by a fleet of vehicles to service a set of customers. VRP generalizes the well-known multiple traveling salesmen problem (mTSP) and has numerous applications in the industry such as in transportation. The literature of VRP mainly focuses on developing optimization algorithms for different variations of the problem. Theses attempts tackle the problem in a centralized and offline fashion, meaning that given a set of constraints, the algorithm computes a set of routes. Subsequently, the computed routes can be given to a set of vehicles that will carry out the tasks of traveling to and servicing customers.

With the growing popularity of autonomous vehicles (e.g., driverless cars and unmanned aerial systems) in the modern world, more and more classic problems in computer science are being revisited in the context of decentralized multi-agent systems. The VRP is certainly one such problem that can play a crucial role in design and implementation of a fleet of autonomous vehicles that are required to execute a set of tasks in a cost-effective fashion. To our knowledge, the body of work on distributed VRP is limited to the work in [2], where the authors propose partitioning policies for adaptive vehicle routing in a dynamic environment, where travel and service costs can change.

With this motivation, in our paper, we introduce a distributed approximation algorithm, where a set of independent vehicles attempt to compute their respective routes to service a group of nodes in an undirected graph. Each node of the graph has a constant non-negative *service cost* and each edge in the graph is associated with some constant non-negative *travel cost*. These costs may vary with respect to different vehicles. The variation of VRP we are studying is the *open multi-depot vehicle routing problem* (OMDVRP), where a vehicle starts at a depot and is not required to return to any of the depots after servicing the last node in its route. We assume that there is no priority and/or precedence in servicing the nodes. There are also some side constraints that need to be satisfied:

- All nodes including the depots, are to be serviced by exactly one vehicle;
- A vehicle has to reach a node before satisfying its demand unless the vehicle is already located in the node (i.e., the depot), and
- A vehicle can visit a node only once in its entire route (i.e., each route is a simple path). However, multiple vehicles can travel to the same node, if it results in reducing the costs.

In our computation model, the set of distributed vehicles execute the same local algorithm and communicate using *synchronous* message passing. We assume that the communication network is a complete graph; i.e., each vehicle can broadcast messages to all other vehicles regardless of their location. Each vehicle has a unique *id* and knows the structure of the entire graph and cost functions of all other vehicles.

Our distributed approximation algorithm is inspired by the technique proposed in [3] to solve the facility location problem and in [4] to solve the minimum dominating set problem. In particular, our approach involves the following steps:

- We first transform VRP into an integer linear program (ILP). Then, we relax the integer program to obtain a linear program (LP), where the solution may involve fractional values.
- In our distributed approximation algorithm, each vehicle, through communication with other vehicles, computes a set of fractional values that potentially represent a route and a set of nodes to be serviced by the vehicle. Using the LP duality theorem [5], we show that the approximation ratio of this step is $O(n \cdot (\rho)^{1/n})$, where n is the size of the graph and ρ is the maximum cost of travel or service in the given instance of VRP.

- Next, we obtain integer values from the fractional solution either by (1) a simple rounding technique, which preserves the feasibility of the fractional solution as well as the approximation ratio, or (2) a randomized rounding technique, which results in approximation ratio of $O(n \cdot (\rho)^{1/n} \cdot \log(n+m))$, where m is the size of the set of vehicles. While the former technique results in an on-the-fly algorithm, where the vehicles move along their routes while they communicate and make local decisions, the latter stipulates a semi-offline technique, where the vehicles first compute their full routes and then start traveling and servicing.

To analyze the performance of our algorithm, we report results of simulations on different size of graphs and different number of vehicles. We compare the performance of our algorithm with the optimal solution obtained by an ILP solver as well as a simple centralized greedy algorithm. Furthermore, our algorithm is fully implemented on a set of real unmanned aerial systems (UASs) that attempt to carry out a joint mission to inspect an outdoor area. The video clips of the real experiments are available at http://www.cas.mcmaster.ca/borzoo/uas.

Organization: In Section II, we formally state the VRP. Section III presents transformation of VRP into ILP and its LP relaxation. In Section IV, we discuss the overall idea of our algorithm and its key components. The detailed description of the distributed solution for the LP relaxation is presented in Section V, while integral rounding techniques to obtain the ILP solution are introduced in Section VI. We analyze the results of simulations in Section VII. Our implementation considerations on UASs are presented in Section VIII. Related work is discussed in Section IX. Finally, we make concluding remarks and discuss future work in Section X. All proofs appear in the appendix.

II. Formal Problem Description

We now formally describe the problem. Let A be a finite set of m vehicles and $G = \langle V, E \rangle$ be an undirected finite graph. Each vehicle $a \in A$ is associated with a node v_0^a called the *depot* for that vehicle. The depot also needs to be serviced. We consider two cost functions:

- The first is the cost of *servicing* a demand by a vehicle at a node of G:

$$C_s : A \times V \to \mathbb{Z}_{\geq 0}$$

- The second is the cost of *traveling* from one node to another by a vehicle:

$$C_t : A \times E \to \mathbb{Z}_{\geq 0}$$

A *path* of G is a sequence $p = v_0 \cdots v_n$, where for all $i \in [1, n-1]$, we have $(v_i, v_{i+1}) \in E$. For each vehicle $a \in A$, we require that $v_0 = v_0^a$ (i.e., its route starts from its associated depot). Let V_a^p be the set of all nodes serviced by a vehicle $a \in A$ in a path p, where $V_a^p \subseteq V$. The total cost of a path $p = v_0 \cdots v_n$ chosen by a vehicle $a \in A$ (denoted $C(a, p)$) is

$$C(a,p) = \sum_{v \in V_a^p} C_s(a, v) + \sum_{i=0}^{n-1} C_t(a, (v_i, v_{i+1}))$$

Our version of the VRP intends to find a set P of paths and a plan function $F : P \to A$, such that:

- The paths cover all vertices of G for servicing. That is,

$$\bigcup_{p \in P} V_{F(p)}^p = V$$

- And, F respects the following optimization objective

$$\min \sum_{p \in P} C(F(p), p)$$

among all such possible functions.

III. Transforming VRP into ILP and Primal-Dual LP Relaxation

A. Transformation to ILP

An *integer linear program* (ILP) is of the form:

$$\begin{cases} \text{Minimize} & c.\mathbf{x} \\ \text{Subject to} & B.\mathbf{x} \geq \mathbf{b} \end{cases}$$

where B (a rational $k \times l$ matrix), c (a rational l-vector), and $\mathbf{b}$ (a rational k-vector) are given, and $\mathbf{x}$ is an l-vector of integers to be determined. In other words, we try to find the minimum of a linear function over a feasible set defined by a finite number of linear constraints. A problem with linear equalities (or $\leq$ linear inequalities) can always be put in the above form, implying that this formulation is indeed general. Thus, every integer linear program involves a set of constants, variables, constraints, and an optimization objective.

Recall that we let A be a finite set of vehicles, $G = \langle V, E \rangle$ be a finite graph, and C_s and C_t be two cost functions as defined in Section II. We now transform VRP to ILP by identifying the constants, variables, constraints, and optimization objective.

Constants.

- For each node $v \in V$ and vehicle $a \in A$, constant $C_s(a, v) \geq 0$ is the cost of servicing v by a.
- $C_t(a, (u, v)) \geq 0$ is the cost of travel from node u to node v by vehicle a.

Variables. The set of variables are the following:

- For each edge $(u, v) \in E$ and vehicle $a \in A$, we introduce a binary variable $x_{(u,v)}^a$, where

$$x_{(u,v)}^a = \begin{cases} 1 & \text{if vehicle } a \text{ travels from node } u \text{ to } v \\ 0 & \text{otherwise} \end{cases}$$

- For each node $v \in V$ and vehicle $a \in A$, we introduce a binary variable y_v^a, where

$$y_v^a = \begin{cases} 1 & \text{if vehicle } a \text{ services node } v \\ 0 & \text{otherwise} \end{cases}$$

- Finally, for each edge $(u, v) \in E$ and vehicle $a \in A$, we include two integer variables s_u^a, and s_v^a. These auxilliary variables are used for subtour elimination constraints (explained below).

Constraints. The set of constrains are as follows:

- All variables are binary. That is, for all $a \in A$ and $(v, u) \in E$, we include:

$$x_{(u,v)}^a, y_v^a \in \{0, 1\} \tag{1}$$

- Each node is serviced exactly once. That is, for each $v \in V$, we include the following:

$$\sum_{a \in A} y_v^a = 1 \tag{2}$$

- If a vehicle a services a node v, then it must travel to that city. Thus, for each vehicle $a \in A$ and each node $v \in V - \{v_0^a\}$, we include the following constraint:

$$\sum_{u \in V} x_{(u,v)}^a - y_v^a \geq 0 \tag{3}$$

- A path may enter a node at most once. That is, for each vehicle $a \in A$ and each node $v \in V$, we include the following constraint:

$$\sum_{u \in V} x_{(u,v)}^a \leq 1 \tag{4}$$

- A vehicle may not travel to a node more than once. That is, for each $a \in A$, we include the following:

$$\sum_{v \in V} x_{(u,v)}^a \leq 1 \tag{5}$$

- A vehicle chooses a continuous path. That is, for each $a \in A$ and $v \in V - \{v_0^a\}$, we include the following:

$$\sum_{u \in V} x_{(u,v)}^a - \sum_{u \in V} x_{(v,u)}^a \geq 0 \tag{6}$$

- The constraints stated above ensure that all cities are visited at least once. But there could be multiple cycles (subtours) in the path that a vehicle is assigned. To avoid this multiple cycles within a path we use subtour elimination constraint. This constraint forces a vehicle to take a path without any cycles. This constraint is known as *subtour elimination*: for each vehicle $a \in A$ and edge $(u, v) \in E$, where $u, v \neq v_0^a$:

$$s_u^a - s_v^a + (n - m) \cdot x_{(u,v)}^a \leq (n - m - 1) \tag{7}$$

where $n = |V|$ and $m = |A|$.

Optimization objective. Our objective is to minimize the cost among all vehicles:

$$\min \left(\sum_{a \in A} \sum_{v \in V} y_v^a \cdot C_s(a, v) + \sum_{a \in A} \sum_{(u,v) \in E} x_{(u,v)}^a \cdot C_t(a, (u, v)) \right) \tag{8}$$

B. LP Relaxation

We now relax our ILP constraints to form a *linear program* (LP), which allows fractional values for variables $x_{(u,v)}^a$ and y_v^a. The set of constraints that change ILP are the following:

- We change Constraint 1 to:

$$x_{(u,v)}^a, y_v^a \in \mathbb{R}_{\geq 0} \tag{9}$$

- To ensure that the LP solution is feasible for ILP, we modify Constraint 2 to:

$$\sum_{a \in A} y_v^a = \frac{1}{m} \tag{10}$$

where $m = |A|$.

- We modify Constraint 4 to:

$$\sum_{u \in V} x_{(u,v)}^a \leq \frac{1}{m} \tag{11}$$

- We modify Constraint 5 to:

$$\sum_{v \in V} x_{(u,v)}^a \leq \frac{1}{m} \tag{12}$$

- We modify Constraint 7 to:

$$s_u^a - s_v^a + (n - m) \cdot x_{(u,v)}^a \leq \frac{1}{m} \cdot (n - m - 1) \tag{13}$$

We will later explain why s variables are not required to be relaxed.

C. The Dual Linear Program

The motivation behind using a *dual* LP [6] is that it provides a lower bound on the value of the optimal *primal* LP solution for minimization problems, and an upper bound for maximization problems. The idea of LP duality is to find the optimal multipliers for the constraints, so as to obtain the tightest bound possible. We can express this problem of finding the best multipliers as another LP; this is called the dual LP. The variables in the dual LP represent constraints of the original primal LP. Each constraint in the dual LP refers to one variable of the primal LP and states that the weighted sum of the coefficients corresponding to that variable should be no more than the coefficient of the variable in the objective function. We use the duality theorem to compute the approximation ratio of our distributed algorithm in Section IV.

Given the primal LP defined in Section III-B, the associated dual LP is the following:

Variables. The set of variables are the following:

- Variable α_v represents Constraint 10.
- Variable β_v^a represents Constraint 3.

- Variable γ_v^a represents Constraint 11.
- Variable τ_u^a represents Constraint 12.
- Variable θ_v^a represents Constraint 6.
- Variable $\lambda_{u,v}^a$ represents Constraint 13.

Constraints. The set of variables are the following[1]:

- For each $v \in V$ and $a \in A$, we have

$$y_v^a : \alpha_v - \beta_v^a \leq C_s(a, v) \tag{14}$$

- For each $a \in A$ and $(u, v) \in E$

$$x_{(u,v)}^a : \beta_v^a - \gamma_v^a - \tau_u^a - \lambda_{(u,v)}^a + \theta_v^a \leq C_t(a, (u, v)) \tag{15}$$

-

$$\alpha_v \text{ is free}, \beta_v^a, \gamma_v^a, \tau_u^a, \lambda_{(u,v)}^a, \theta_v^a \geq 0 \tag{16}$$

Optimization objective. The objective of the dual program is maximize the following:

$$\max \frac{1}{m}(\sum_{v \in V} \alpha_v - \sum_{a \in A} \sum_{v \in V} \gamma_v^a - \sum_{a \in A} \sum_{v \in V} \tau_v^a - (n - m - 1) \sum_{a \in A} \sum_{(u,v)} \lambda_{(u,v)}^a)$$

In the next section, we present an algorithm that solves the primal LP relaxation and we provide bounds using the dual program.

IV. Overall Idea and Key Components of the Distributed VRP Algorithm

In this section, we present the intuition behind our algorithm (Subsections IV-A and IV-B) and key conditions based on which a vehicle locally decides whether to *service* or *travel* to a node (Subsections IV-C and IV-D).

A. Model of Distributed Computation

We consider the following characteristics:

- The set of distributed vehicles execute the same local algorithm and can communicate using *synchronous* message passing; i.e., in each round the vehicles send and receive messages synchronously and subsequently engage in internal computation;
- The communication network is a complete graph; i.e., each vehicle can broadcast messages to all other vehicles regardless of their location in G, and
- Each vehicle has a unique *id* and knows the structure of the entire graph G and cost functions C_s and C_t of all vehicles.

[1]For reasons of space, we leave it to the reader to do the math and obtain the dual program.

B. Overall Idea of the Algorithm

Initially, each vehicle is placed in its depot to be serviced. As mentioned earlier, our distributed algorithm is synchronous. In each round, each vehicle:

1) performs some computation and decides either to service its current location or to travel to another location;
2) broadcasts a message containing the details of its decision to all other vehicles, and
3) waits to receive similar messages from all other vehicles.

Thus, in each round, a vehicle can perform two operations when it enters a node:

- A vehicle services the node, if it can service the node with less cost than any rival vehicle or in fewer communication rounds compared to all other rival vehicles from their current location.
- Otherwise, the vehicle travels to a node, where it can reach and service a node with the least cost possible. A vehicle travels to another node, if it finds a rival vehicle can service the current node cheaper than itself or in the case when the node is already serviced.

We explain the conditions based on which a vehicle decides to service a node or travel to another node in Subsections IV-C and Subsection IV-D, respectively.

C. Conditions for Servicing

Step 1 – Computing the vehicle's neighborhood

Let $loc(a)$ denote the current location of a vehicle $a \in A$. The *neighborhood* for a vehicle a from $loc(a)$ (if not serviced) is the set of all simple paths, in which all the nodes are not yet serviced by any vehicle and not traveled a. If $loc(a)$ has already been serviced, then the neighborhood is empty.

Definition 1: Given a graph $G = (V, E)$ and vehicle $a \in A$, the *neighborhood* of a is the following set of simple paths:

$$N^a = \left\{ p \mid p \text{ starts at } loc(a) \ \wedge \ V_p = V_p^{\neg s} \right\}$$

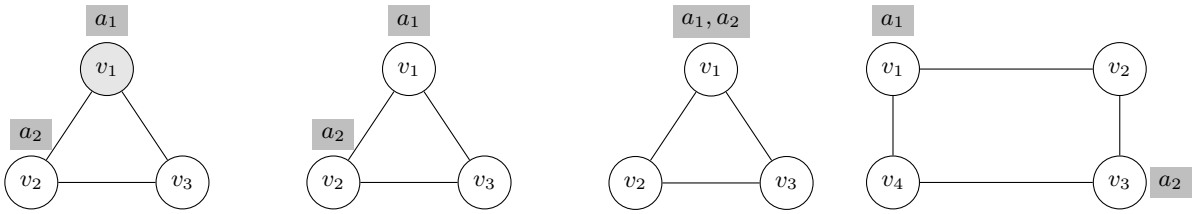
where V^p denotes the set of all nodes in path p and $V_p^{\neg s}$ is the set of nodes of p that are not serviced. ■

In our algorithm, we compute a *dynamic* neighborhood for each vehicle a denoted by N_k^a. Let $a \in A$ be a vehicle and $Rival_a = A - \{a\}$ be the set of all *rival* vehicles of a. A dynamic neighborhood for a is the neighborhood where the length of each path is restricted to the minimum distance from $loc(a)$ and its closest rival vehicle, and, the value of k.

Definition 2: Given a graph $G = (V, E)$ and vehicle $a \in A$, the *dynamic neighborhood* of a is the following:

$$N_k^a = \big\{ p \mid \ p \in N^a \ \wedge \ (|p| < \min\big\{k, \{d(loc(a), a') \mid a' \in Rival_a \wedge loc(a) \neq loc(a')\}\big\}) \vee \exists a' \in Rival_a : loc(a') = loc(a) \rightarrow p = loc(a)) \big\}$$

where $k \leq |A|$ is some natural number, $d(u, v)$ is the distance of node u from v for any $(u, v) \in E$ and $|p|$ is the length of



(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Fig. 1: Computing dynamic neighborhood

path p. ■

Notice that in Definition 2, if there exists $a' \in Rival_a$, such that a and a' are in the same node, then $N_k^a = \{loc(a)\}$. To clarify Definition 2 consider the following examples:

- *Example 1.* In Figure 1a, vehicles a_1 and a_2 are placed at nodes v_1 and v_2, respectively. Node v_1 has been serviced (i.e, shaded). Costs of servicing and traveling are unimportant. In this case, for $k = 2$, we have $N_2^{a_1} = \emptyset$ and $N_2^{a_2} = \{v_2\}$.
- *Example 2.* In Figure 1b, vehicles a_1 and a_2 are placed at nodes v_1 and v_2, respectively. For $k = 2$, we have a $d(loc(a_1), loc(a_2)) = 1$, $N_2^{a_1} = \{v_1\}$ and, hence, $N_2^{a_2} = \{v_2\}$.
- *Example 3.* In Figure 1c, vehicles a_1 and a_2 both placed at node v_1. If $k = 2$, we have $N_2^{a_1} = \{v_1\}$ and $N_2^{a_2} = \{v_1\}$.
- *Example 4.* In Figure 1d, vehicles a_1 and a_2 are placed at nodes v_1 and v_3, respectively. If $k = 2$, since $d(loc(a_1), loc(a_2)) = 2$, we have $N_2^{a_1} = \{v_1, v_1v_4, v_1v_2\}$ and $N_2^{a_2} = \{v_3, v_3v_2, v_3v_4\}$, respectively.

Step 2 – Choosing the best path from N_k^a

Next, a vehicle should decide which path it will choose from the set of paths in N_k^a. This decision is made based on the cost of traveling and servicing along the paths in N_k^a. To this end, we identify the path whose normalized cost of travel and service is minimum:

$$C(N_k^a) = \min \left\{ \frac{\sum_{v \in p} C_s(a, v) + \sum_{(u,v) \in p} C_t(a, (u, v))}{2|p| + 1} \;\middle|\; p \in N_k^a \right\} \quad (17)$$

We denote the path that yields $C(N_k^a)$ by $\pi(N_k^a)$. For example, in Figure 2, let vehicles a_1 and a_2 have identical service and travel costs (each edges is labeled by its travel cost and service costs are shown above nodes). We take $k = 3$ and vehicle a_1 has its depot at v_1 and a_2 has its depot at v_3. Then, for vehicle a_1, we have $N_3^{a_1} = \{v_1, v_1v_2, v_1v_4\}$ and

$$C(N_3^{a_1}) = \min \left\{ 20, \frac{20 + 5 + 10}{3}, \frac{20 + 10 + 10}{3} \right\} = 11.67$$

Hence, $\pi(N_3^{a_1}) = v_1v_2$. For vehicle a_2, we have $N_3^{a_2} = \{v_3, v_3v_2, v_3v_4\}$ and

$$C(N_3^{a_2}) = \min \left\{ 30, \frac{30 + 20 + 10}{3}, \frac{30 + 10 + 10}{3} \right\} = 16.67$$

Hence, $\pi(N_3^{a_2}) = v_3v_4$.

Algorithm 1 Function CheckIfPathFree for vehicle a.

Input: Path $p = \langle v_0 v_1 \dots v_n \rangle$ and rival vehicle a'
Output: Path p'

```
p' ← p;
if (|p| = 0) ∧ (loc(a) = loc(a')) ∧ (C_s(a, loc(a)) > C_s(a', loc(a)) ∨
        (C_s(a, loc(a)) = C_s(a', loc(a)) ∧ id(a) > id(a')) then
    p' ← ∅;
else
    for (i ← 0; i ≤ n; i++) do
        if (2.d(loc(a), v_i) + 1) < d(loc(a'), v_i) then
            p' ← p'.⟨v_i⟩;
        else
            return p';
        end if
    end for
end if
return p';
```

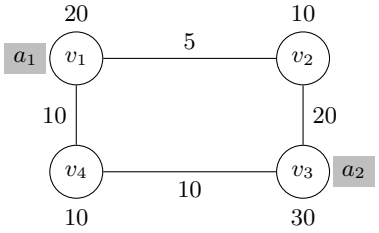


Fig. 2: Choosing the best path from N_k^a

Step 3 – Verify if a rival vehicle may enter $\pi(N_k^a)$

Each vehicle needs to check if it takes fewer number of communication rounds to service all the nodes in $\pi(N_k^a)$ than another vehicles from its current location. This check is done using the function CheckIfPathFree (see Algorithm 1). The function CheckIfPathFree takes as input a path chosen by the current vehicle and a rival vehicle. It returns as output a path p'. If CheckIfPathFree returns a non-empty path p', it ensures that no rival vehicle can enter p' in at most $2|p'| + 1$ communication rounds (we will describe how the vehicles communicate later). Note that in the main algorithm described in the next section, CheckIfPathFree is called for every rival vehicle.

Function CheckIfPathFree is also used to avoid conflicts among the vehicles. That is, if two or more vehicles have traveled to a node where both vehicles have the same cost of servicing, then CheckIfPathFree allows only the vehicle that has the least vehicle id to service the node. There are three possible outcomes of this function:

- If the CheckIfPathFree returns an $\emptyset$, then the vehicle travels to another node from its current location.
- If it returns a path p' where, $|p'| = |\pi(N_k^a)|$ then the vehicle is clear to service all nodes in $\pi(N_k^a)$.
- If it returns a path p' where, $|p'| < |\pi(N_k^a)|$ then the vehicle makes p' its current path and calculates $C(N_k^a)$ based on the new path p' and is then clear to service all nodes in p'. In this case the subsequent function calls to CheckIfPathFree is made with the updated path.

For example, consider Figure 3 where vehicles a_1, a_2, and a_3 with depots v_1, v_2, and v_5, respectively, have identical service and travel costs. We take $k = 2$. After two rounds, vehicle a_1 has serviced vertex v_1 and traveled from vertex v_1 to v_8. Similarly, vehicle a_2, has finished servicing vertex v_2 and traveled to vertex v_3. Vehicle a_3, has finished servicing vertex v_5 and traveled to vertex v_6. The path chosen by vehicle a_1 from its current location is v_8, the path chosen by a_2 from its current location is v_3v_4, and the path chosen by a_3 is v_6v_7 based on their $C(N_k^a)$ values. Let us imagine in this state, the three vehicles invoke function CheckIfPathFree:

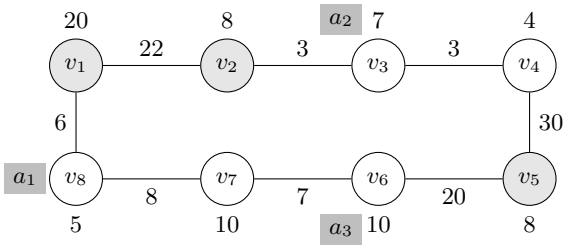


Fig. 3: Example for CheckIfPathFree

- **Vehicle a_1:** Vehicle a_1 calls function CheckIfPathFree with parameters $\pi(N_2^{a_1}) = v_8$ and a_2, where $loc(a_2) = v_3$. Since $loc(a_1) \neq loc(a_2)$, the function returns v_8. Then, it calls the function with the same path but for a_3, where $loc(a_3) = v_6$. The function again returns v_8. Hence, vehicle a_1 can service its node v_8.
- **Vehicle a_2:** Vehicle a_2 calls function CheckIfPathFree with parameters $\pi(N_2^{a_2}) = v_3v_4$ and a_1, where $loc(a_1) = v_8$. Vehicle a_2 can service node v_3 in fewer rounds than a_1. Since, it takes $2|p'| + 1 = 3$ communication rounds for a_2 to service node v_4 and a_1 has traveled from v_1 to v_8 and is not allowed to travel back to v_1, the legitimate distance of the location of a_2 to a_1 is 4. Hence, $d(loc(a_1), v_3) > (2|p'|+1)$ and $d(loc(a_1), v_4) > 2|p'|+1$. This means the function returns v_3v_4. Next, vehicle a_2 calls the function with parameters v_3v_4 and a_3, where $loc(a_3) = v_6$. Notice that since a_3 has traveled from v_5 to v_6 and is not allowed to travel back to v_5, then the legitimate distance of the location of a_3 to a_2 is 5. Hence, $d(loc(a_3), v_3) > 2|p'| + 1$ and also $d(loc(a_3), v_4) > 2|p'| + 1$. Hence, the function returns v_3v_4 and vehicle a_2 can service the this chosen path.
- **Vehicle a_3:** Vehicle a_3 calls function CheckIfPathFree with parameters $\pi(N_2^{a_3}) = v_6v_7$ and a_1, where $loc(a_1) = v_8$. Vehicle a_3 can service node v_6 in fewer rounds than its rival vehicles a_1 and a_2. Vehicle a_3 takes $2|p'| + 1 = 3$ communication rounds to service node v_7. We have $d(loc(a_1), v_6) \leq 2|p'| + 1$. Hence the function returns v_6. Hence, vehicle a_3 updates its path from v_6v_7 to its current location only i.e., v_6 and calls CheckIfPathFree with the updated path. This time, since $loc(a_3) \neq loc(a_1)$ and $loc(a_3) \neq loc(a_2)$, the function returns v_6. Hence, vehicle a_3 can service the v_6.

In summary, for a vehicle a to *service* a path the following two conditions must hold at all times:

- $N_k^a \neq \emptyset$, and
- CheckIfPathFree returns a non-empty path comparing all rival vehicles.

D. Conditions for Traveling

If none of the conditions given above are satisfied, then a vehicle travels from its current node to an another node. The logic behind choosing the next node from a set of possible nodes is as follows. A vehicle evaluates all available nodes that it can reach from its current location in one communication round excluding the nodes it has traveled before in its path. For this set of nodes, a vehicle sums up the cost of traveling to each node and servicing it. All these costs are added into a set and the node which yields the least value possible is chosen as the next node to travel from the vehicle's current position. If the set of available nodes includes nodes that has already been serviced, the vehicle temporarily assign a large value as cost of servicing that node. This large value can simple be the following:

$$\rho = \max\left\{C_s(a, v), C_t(a, (u, v)) \mid (a \in A) \ \wedge \ (u, v \in V)\right\} \tag{18}$$

A vehicle a finds the next best node for it to travel and service as follows:

$$nextbest(a) = \min\{C_s(a, v) + C_t(a, (loc(a), v)) \mid (loc(a), v) \in E\} \tag{19}$$

where v has not been traveled to or from by a.

For example, consider Figure 4, where vehicle a_2 has serviced v_3. Here, the available nodes to vehicle a_3 are v_2, v_3, and v_4. Since $\rho = 35$, we temporarily set $C_s(a_3, v_3) = 35$. Thus, we have

$$nextbest(a_3) = \min\{5 + 10, 20 + 35, 10 + 10\} = 15$$

So vehicle a_3 chooses to go to node v_2. If it happens that the node whose service cost has been temporarily set to ρ is returned from the $nextbest(a)$ value, that node is chosen as the next node to travel.

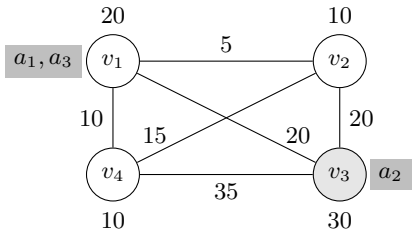


Fig. 4: Travel Based on $nextbest(a)$ value

V. Algorithm for Distributed VRP with Fractional Values

Using the concepts introduced in Section IV, in this section we design an algorithm, where a set of distributed vehicles solve the relaxed LP version of VRP as formalized in Section III-B. We will use the LP relaxation and dual LP constraints identified in Section III to identify the approximation ratio of the algorithm.

All vehicles execute the same local algorithm (see Algorithm 2). The algorithm consists of two nested loops: an outer s-loop and an inner t-loop. The outer s-loop iterates as many times as the number of nodes in the graph. When the value of s becomes zero, it signals the termination of the algorithm. The inner t-loop is initialized by the value of $2|p'| + 1$, which gives value 1 if $N_k^a = \emptyset$ or $p' = \emptyset$ (i.e., when function CheckIfPathFree indicates that no node can be serviced). Similar to the outer s-loop, the inner t-loop is also executed until t becomes zero. That is, until all the nodes in the chosen path is serviced by the current vehicle or the vehicle changes its location from one node to another node without servicing the current node thus computing a new neighborhood. If a node is serviced in a t-iteration, the value of t is decremented by 1 and if a vehicle travels to an another node, then the value is made 0 for the vehicle to compute its new neighborhood.

Initially, all primal and dual variables are set to zero (Line 1). Hence, the initial primal solution is infeasible, and the dual solution is feasible, yet far from optimal. During the course of the algorithm, both the primal and dual variables are gradually increased, thereby decreasing the primal infeasibility, and increasing dual optimality. Each vehicle a starts from its depot v_0^a. From its current location, the vehicle calculates N_k^a, $C(N_k^a)$, and $\pi(N_k^a)$ as explained in Section IV-C, where the value of k is given as an input. In lines 11 – 13, a vehicle checks if its chosen path is free from other rival vehicles. In the inner for-loop line 17 checks if a vehicle can service its chosen path. We now explain the process of servicing and traveling in two sections:

- **Service Section (lines 18 – 30).** If the condition in line 17 is satisfied, then a vehicle enters the servicing part of the algorithm. Line 19 checks if the current vehicle satisfies $y_v^a = 0$. If true, the vehicle services the node and assigns $y_v^a \leftarrow 1/m$. This denotes that node v has been serviced by vehicle a. This information will be broadcast in the current communication round (t-loop). Since the vehicle has not traveled to any node in the current communication round, the dual variables γ_v and τ_u do not increase. The dual variable β_v^a represents ILP Constraint 3. Since a vehicle would have reached the current location (if not its depot) by taking an edge from an adjacent node, we have that β_v^a would have already been increased by the value ρ. A node can be reached or visited using multiple edges. Since we do not know the cost of the incoming edge, we bound this value by ρ. Going by the 2nd LP constraint, we subtract the value of servicing the current node i.e., $C_s(a, v)$. Hence $\Delta\beta_v^a$ is increased by $\rho - C_s(a, v)$. If a vehicle services a node from its depot (i.e., $flag = false$), then Constraint 3 is not true and, hence, the initial value of β_v^a is retained.
  Next, if Line 19 returns false, it means that vehicle a has already serviced its current location and executes Lines 22 - 29. In these lines, the vehicle moves from its current location to the next node in its chosen path. In this process the value of $x^a_{(loc(a),v)}$ is increased by $1/m$ to denote an edge has been taken. The corresponding γ_v and $\tau_{loc(a)}$ are increased by the value of $\Delta x^a_{(loc(a),v)}$ and $-\Delta x^a_{(loc(a),v)}$, respectively. These values are incremented in an equal and opposite way to denote an outgoing edge from $loc(a)$ to an another node v can also be seen as an incoming edge for v from $loc(a)$. Hence, these values are equal and opposite. Line 26 is never executed, as N_k^a would not contain already visited edges and also $nextbest(a)$ would not allow the vehicle to travel back to a node that is already visited. Since the maximum cost of $C_t(a, (u, v))$ can be bounded by ρ, the value β_v^a is increased by ρ in Line 27. Note that $C(N_k^a)$ was normalized by a factor of $2|p'| + 1$. Hence, $C(N_k^a)$ if repeatedly sent over $2|p'| + 1$ communication rounds would yield the actual costs of the vehicle servicing and traveling across all nodes in its path. Hence, the temporary assignment in line 28 is made to ensure $C(N_k^a)$ is send out in each communication round of the service section of the code.
- **Travel Section (lines 31 – 38).** If a vehicle is not allowed or cannot service its current location, it enters the travel section. First, the value of $nextbest(a)$ is calculated as shown in Section IV-D. The primal and dual values are incremented in the same way like the travel case in service section. The only difference between the two, is that here in this section the value of t is assigned zero after one iteration of the t-loop.

After the service and travel sections end, all the values calculated from the current communication round are assigned to the respective primal and dual variables. Then, a broadcast message containing details of the vehicle *id*, current location, cost incurred in the current communication round is sent out and received from all vehicles (lines 42 – 44). The cost incurred in the current communication round (service or travel costs) is multiplied with the corresponding $x^a_{(loc(a),v)}$ or $y^a_{loc(a)}$ value increased in the current communication round and this information is passed in the broadcast message. The messages

Algorithm 2 Algorithm for vehicle a

Input: Graph $G = (V, E)$, cost functions C_s and C_t of all vehicles, and value of k.
Output: Fractional Value for x^a_{uv}, y^a_v

1: $x^a_{(u,v)}, y^a_v, \Delta x^a_{(u,v)}, \Delta y^a_v, \alpha_v,$
$\beta^a_v, \Delta\beta^a_v, \lambda^a_{(u,v)}, \gamma^a_v, \Delta\gamma^a_v, \tau^a_u, \Delta\tau^a_u, \theta^a_v \leftarrow 0$;
2: $loc(a) \leftarrow v^a_0$; $flag \leftarrow true$; $Rival_a \leftarrow A - \{a\}$;
3: **for** $s = |V|$ **to** 0 **do**
4: $p' \leftarrow \emptyset$, $count \leftarrow 0$;
5: Calculate N^a_k;
6: **if** $(N^a_k = \emptyset)$ **then**
7: $|p'| \leftarrow \emptyset$;
8: **else**
9: Calculate $C(N^a_k)$ and $\pi(N^a_k)$;
10: $p' \leftarrow \pi(N^a_k)$;
11: **for each** $a' \in Rival_a$ **do**
12: $p' \leftarrow$ CheckIfPathFree$(p', loc(a'))$;
13: **end for**
14: **end if**
15: **for** $t = 2|p'| + 1$ **to** 0 **do**
16: $isService \leftarrow false$;
17: **if** $(p' \neq \emptyset) \wedge (N^a_k \neq \emptyset)$ **then**
18: $isService \leftarrow true$;
19: **if** $(y^a_{loc(a)} = 0)$ **then**
20: $\Delta y^a_{loc(a)} \leftarrow 1/m$;
21: **if** $(flag \neq true)$ **then** $\Delta\beta^a_{loc(a)} \leftarrow \beta^a_{loc(a)} - C_s(a, loc(a))$;
22: **else**
23: Let v be the node adjacent to $loc(a)$ in p' st. $x^a_{(loc(a),v)} = 0$;
24: $isService \leftarrow false$;
25: $\Delta x^a_{(loc(a),v)} \leftarrow 1/m$;
26: **if** $(x^a_{(v,loc(a))} > 0)$ **then** $\lambda^a_{(loc(a),v)}$++;
27: $\Delta\gamma^a_v \leftarrow \Delta x^a_{(loc(a),v)}$; $\Delta\tau^a_{loc(a)} \leftarrow -\Delta x^a_{(loc(a),v)}$; $\Delta\beta^a_v \leftarrow \rho$;
28: Temporarily assign $C_t(a, (u, v)) \leftarrow C(N^a_k)$;
29: **end if**
30: $t \leftarrow t - 1$;
31: **else**
32: Let v yield $nextbest(a)$;
33: $\Delta x^a_{(loc(a),v)} \leftarrow 1/m$;
34: $\Delta\beta^a_v \leftarrow \rho$;
35: **if** $(x^a_{(v,loc(a))} > 0)$ **then** $\lambda^a_{(loc(a),v)}$++;
36: $\Delta\gamma^a_v \leftarrow \Delta x^a_{(loc(a),v)}$; $\Delta\tau^a_{loc(a)} \leftarrow -\Delta x^a_{(loc(a),v)}$;
37: $t \leftarrow 0$;
38: **end if**
39: $\gamma^a_{loc(a)} \leftarrow \gamma^a_{loc(a)} + \Delta\gamma^a_{loc(a)}$; $\tau^a_{loc(a)} \leftarrow \tau^a_{loc(a)} + \Delta\tau^a_{loc(a)}$;
40: $\beta^a_{loc(a)} \leftarrow \beta^a_{loc(a)} + \Delta\beta^a_{loc(a)}$; $y^a_{loc(a)} \leftarrow y^a_{loc(a)} + \Delta y^a_{loc(a)}$;
41: $x^a_{(loc(a),v)} \leftarrow x^a_{(loc(a),v)} + \Delta x^a_{(loc(a),v)}$;
42: **if** $(isService)$ **then** Send$(id(a), loc(a), \Delta y^a_{loc(a)}.C(N^a_k))$;
43: **else** Send$(id(a), (loc(a), v), \Delta x^a_{(loc(a),v)}.C_t(a, (loc(a), v))$;
44: $\{(id, v, Y)\}_{A' \subseteq A}$, $\{(id, (u, v), X)\}_{A'' \subseteq A} \leftarrow$ Receive();
45: **for each** $\bar{a} \in A'$ **do**
46: $loc(\bar{a}) \leftarrow v$;
47: **if** $(a \neq \bar{a})$ **then** $y^{\bar{a}}_v \leftarrow 1/m$;
48: $\Delta\alpha_v \leftarrow Y$;
49: $\alpha_v \leftarrow \alpha_v + \Delta\alpha_v$;
50: $count \leftarrow count + 1$;
51: **end for**
52: **for each** $a'' \in A''$ **do**
53: $loc(a'') \leftarrow v$;
54: **if** $(a \neq a'')$ **then** $x^{a''}_{(u,v)} \leftarrow 1/m$;
55: $\alpha_v \leftarrow X$;
56: $\alpha_v \leftarrow \alpha_v + \Delta\alpha_v$;
57: **end for**
58: **if** $(t = 0 \wedge s - count = 0)$ **then**
59: $\theta^a_{loc(a)} \leftarrow 1/m$;
60: **end if**
61: $flag \leftarrow false$;
62: **end for**
63: $s \leftarrow s - count$;
64: **end for**

received are distinguished by two sets, one that contain service messages (by vehicles in A') and one that contains travel messages (by vehicles in A'').

In Lines 45 – 51, from A', the location of all vehicles that sent a service and values of corresponding rival vehicle's y^a_v are updated and the count of these messages are taken and the cost sent by each vehicle is added up to the corresponding α_v value in Line 49. The number of service messages received are stored in the variable $count$ and the value of s is decremented by the value of $count$ once the inner for-loop terminates in Line 63. That way, all vehicles have an updated count of the number of nodes yet to be serviced at all times during the algorithm.

Likewise, for the vehicles in A'' that traveled (Lines 52 – 57), the location of all vehicles that sent a travel message is updated. The corresponding rival vehicles $x^a_{(u,v)}$ are updated and also cost sent by each vehicle is added up to the corresponding α_v value in Line 56. The dual variable α_v holds the weighted sum of all service and travel that happened corresponding to a node v. The sum of α_v for all nodes gives the total cost.

At the end of the last iteration of the outer loop, the primal variables y^a_v and $x^a_{(u,v)}$ form a feasible solution to the LP. The value of $\theta^a_{loc(a)}$ is given the value $1/m$ in the last iteration of the s-loop (Line 59). This is to show that there is only an incoming edge and there are no outgoing edges from the last node in the path. In all the other iterations of the s-loop, the value of incoming edges could increase in one iteration, but the corresponding outgoing edge is also increased in the next iteration of the s-loop, if a vehicle performs travel or the value is increased in two communication rounds if a vehicle performs a service.

At the end of the algorithm, each vehicle has some $x^a_{(u,v)}$ fractional values and some y^a_v values and collectively all the nodes have been serviced by exactly one vehicle. While these values constitute a feasible solution to LP, they have fractional values and can therefore not be used as a solution to the original VRP. Hence, we will utilize two approaches to round these fractional values to integer values in $\{0, 1\}$, increasing the approximation ratio from the fractional solution only by a logarithmic factor.

Lemma 1: Algorithm 2 forms a feasible solution for the LP constraints.

Theorem 1: The approximation ratio of Algorithm 2 is

$$O(n \cdot (\rho)^{1/n}).$$

VI. Rounding Techniques to Obtain an Integral Solution

A. Randomized Rounding

In order to generate integer solutions for our problem, we round the fractional solutions obtained from Algorithm 2. In the following, let $\hat{x}^a_{(u,v)}$ and $\hat{y}^a_v$ be the fractional solutions obtained from Algorithm 2 and the variables $x^a_{(u,v)}$ and y^a_v denote the corresponding rounded integer values. The idea is to round the fractional values $\hat{y}^a_v$ in such a way that with

Algorithm 3 Randomized Rounding: For each vehicle a

Input: Fractional solutions $\hat{x}^a_{(u,v)}, \hat{y}^a_v$ from Algorithm 2 of all vehicles.
Output: Integral Solutions $x^a_{(u,v)}, y^a_v$ to ILP.

1: $visited^a \leftarrow \{\}$;
2: $total^a \leftarrow \{v \in V \mid \hat{y}^a_v > 0\} \cup \{u, v \in V \mid (u,v) \in E \wedge \hat{x}^a_{(u,v)} > 0\}$;
3: **for each** $v \in total^a$ **do**
4: $a'' \leftarrow -1$; $cost \leftarrow 0$;
5: **if** $v \notin visited^a$ **then**
6: Let a' be a vehicle st. $(\hat{y}^{a'}_v > 0) \wedge (\exists u \in V.\hat{x}^{a'}_{(u,v)} > 0 \wedge x^{a'}_{vu} \neq 1)$
7: $x^{a'}_{(u,v)} \leftarrow 1$;
8: $p^{a'}_v \leftarrow \min\{1, \hat{y}^{a'}_v \cdot \ln(n+m)\}$;
9: $y^{a'}_v \leftarrow \begin{cases} 1 & \text{with probability } p^{a'}_v \\ 0 & \text{with probability } 1 - p^{a'}_v \end{cases}$
10: $C_v \leftarrow \sum_{a \in A} \sum_{u \in V} \hat{x}^a_{(u,v)} \cdot C_t(a,(u,v))$;
11: $count \leftarrow |\{(u,v) \mid \exists a \in A.\exists u \in V.\hat{x}^a_{(u,v)} > 0\}|$;
12: $A_v \leftarrow \{a \in A \mid \exists u \in V.\hat{x}^a_{(u,v)} > 0 \ \wedge \ C_t(a,(u,v)) \leq \ln(n+m) \cdot C_v\}$;
13: **if** $(count \neq |A_v|)$ **then**
14: $A_v \leftarrow \emptyset$;
15: **end if**
16: **if** $(a' \in A_v) \wedge (y^{a'}_v = 1)$ **then**
17: $a'' \leftarrow a'$;
18: **else**
19: $y^{a'}_v \leftarrow 0$;
20: $cost \leftarrow \min\{C_s(a,v) + C_t(a,(u,v)) \mid \exists a \in A.\exists u \in V.\hat{x}^a_{(u,v)} > 0\}$;
21: Let $\tilde{a} \in A$ yield the value $cost$.
22: **if** $(\exists u \in V.\hat{x}^{\tilde{a}}_{(u,v)} > 0 \wedge x^{\tilde{a}}_{vu} \neq 1)$ **then**
23: $x^{\tilde{a}}_{(u,v)} \leftarrow 1$;
24: **end if**
25: $y^{\tilde{a}}_v \leftarrow 1$;
26: $a'' \leftarrow \tilde{a}$;
27: **end if**
28: $visited^a \leftarrow visited^a \cup \{v\}$;
29: **else**
30: Let a'' be the vehicle, where $y^{a''}_v = 1$.
31: **end if**
32: Send(v, a'');
33: $RV \leftarrow$ Receive();
34: **for each** $(v', a') \in RV$ **do**
35: **if** $(\exists u \in V.\hat{x}^{a'}_{(u,v)} > 0 \wedge x^{a'}_{v'u} \neq 1)$ **then**
36: $x^{a'}_{(u,v')} \leftarrow 1$;
37: **end if**
38: **if** $(v' \notin visited^a) \wedge (v' \in total^a)$ **then**
39: $y^{a'}_{v'} \leftarrow 1$;
40: $visited^a \leftarrow visited^a \cup \{v'\}$;
41: **else if** $(v' \in visited^a) \wedge \exists a''.y^{a''}_{v'} = 1 \wedge (id(a') < id(a''))$ **then**
42: $y^{a''}_{v'} \leftarrow 0$;
43: $y^{\tilde{a}}_{v'} \leftarrow 1$;
44: **else if** $(v' \notin total^a)$ **then**
45: $y^{a'}_{v'} \leftarrow 1$;
46: **end if**
47: **end for**
48: **end for**

probability p, vehicle a that has increased its $\hat{y}^a_v$ fractional value, services node v and with probability $(1-p)$, the vehicle with the least cost to travel and service among the vehicles that have traveled to this node services the node.

Each vehicle $a \in A$, runs Algorithm 3 on $total^a$ which is the set of all nodes that vehicle a has traveled to or serviced during its execution of Algorithm 2. This ensures all vehicles run the algorithm for equal number of rounds; i.e., the size of $total^a$. Each vehicle calculates the rounded values for a node (even if it has not serviced it) and inform other vehicles about the calculated values. These values are updated by other respective vehicles accordingly.

The main steps of the algorithm are as follows. For every node v in $total^a$:

- If $v \notin visited^a$ (Line 5), the following steps are executed.
  1) In Line 6, vehicle a finds a vehicle a' which has traveled to and serviced node v. It, then, makes the corresponding vehicle's incoming edge 1. In Line 8, the fractional value $\hat{y}^{a'}_v$ is multiplied with $\ln(n+m)$ and the value returned is stored as $p^{a'}_v$. In Line 9, with probability of $p^{a'}_v$, the corresponding $y^{a'}_v$ gets the value 1 else it gets the value 0.
  2) In Lines 10, we compute the cost of all vehicles that traveled to v. A vehicle in Line 11 keeps track of the count of vehicles that has entered the node v. Further in Line 12, let A_v be the set of all vehicles whose travel cost is within a factor of $\ln(n+m)$ of C_v.
  3) In Line 14, if there are vehicles that traveled to v, but their cost is not within the $\ln(n+m)$ factor, then the content of A_v is retained, else it is made empty. Then, in Line 16 we check, if A_v contains vehicle a' for which $y^{a'}_v$ has been made 1 in Line 9 of the algorithm. If this condition is satisfied the values increased in Lines 6 – 9 are retained.
  4) If either of the above conditions is not satisfied, the value of $y^{a'}_v$ is made 0 in Line 19. In Line 20 the vehicle with $\hat{x}^a_{(u,v)} > 0$ that returns the minimum cost for traveling to node v and servicing it, is chosen and the corresponding $x^a_{(u,v)}$ and y^a_v variables are made 1 in Lines 23 – 25. After each node in $total^a$ is processed, the node is added to the visited nodes list $visited^a$.
- If $v \in visited^a$ (Line 30) and the node has been already added to $visited^a$, the vehicle id which has increased its corresponding y^a_v is retrieved.
- Finally, a message is sent to all vehicles (Line 32) informing them about the vehicle which has increased its $y^a_v = 1$ for a node in consideration. When a message is received (Line 33), its corresponding $x^a_{(u,v)}$ are made 1 in Line 36. Then the vehicle checks if the node is present in the $total^a$ and not been visited, it adds the node to $visited^a$ and makes the corresponding vehicle's $y^a_v = 1$. If the node is already present in $visited^a$, a vehicle updates the information if there any conflicts in the messages received for a node v, the message that has the least vehicle id is updated. If the received node is not present in $total^a$, its corresponding y^a_v values is increased by 1.

Theorem 2: The approximation ratio of Algorithm 3 is

$$O(n \cdot (\rho)^{1/n} \cdot log(n+m)).$$

B. On-the-fly Algorithm

We can modify Algorithm 2 to solve the VRP in an on-the-fly distributed fashion. We basically need to take out the processing of dual LP variables α_v, βa^a_v, $\lambda^a_{(u,v)}$, γ^a_v, τ^a_u, θ^a_v. Also, in Lines 25, 33, and 54, we increase $\Delta x^a_{(u,v)}$ by value 1 and in lines 20, and 47, we increase Δy^a_v by 1.

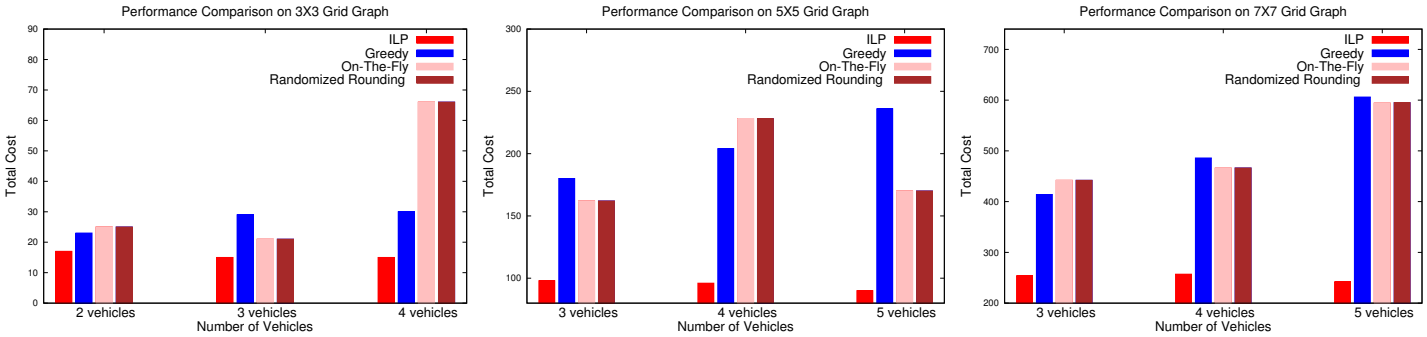


Fig. 5: Simulation results.

VII. Simulation Results

In this section, we present our simulations results comparing Algorithm 2, Algorithm 3, the optimal solution[2], and a simple centralized greedy algorithm. We study grid-shape graphs with different sizes. To ensure that the grids have Hamiltonian paths, we convert them to complete graphs by adding virtual edges between nodes that do not share a physical edge, where the travel cost is equal to the weight of the shortest path between the incident nodes. The greedy algorithm gives each vehicle equal number of nodes to service. If there are not equal nodes, the last vehicle may be given lesser nodes to cover. For the number of nodes assigned, each vehicle chooses the best nodes it can travel and service starting from its depot.

In our simulations, we have taken the travel costs of all vehicles in the ratio 1:2:3 where the first vehicle is given a cost of 2 to travel between physical edges of the grid. For the Case of 3*3 grid graph, the cost assigned to service a node is 1 for all vehicles and for the cases of 5*5 and 7*7 grids, we assign the cost of service as 2 and 4, respectively for all vehicles. The results are shown in the graphs in Figure 5. All our simulations are run with 95 % confidence interval.

We acknowledge that the ILP solution outperforms both Algorithms 2 and 3. Besides the fact that ILP gives us the optimal solution, the solution does not require all vehicles to be equally involved in all steps of servicing and it can require that the vehicle with the highest travel cost not to be used in the solution for better overall costs. But since we have designed a synchronous algorithm, we require that all vehicles perform a travel or service in each communication round until the termination occurs. Hence, the total cost is comparatively higher.

There are cases where both our solution and the greedy solution outperform each other. Both our algorithms allow two or more vehicles to travel to a same node in the same communication round after which only one vehicle services the node. So this unwanted travel is avoided by the greedy algorithm which leads to its better performance in some cases. However, there are cases, where our algorithms perform better than the greedy algorithm. This is because in the greedy algorithm there are chances that all the cheap nodes are taken by the vehicles with cheaper cost and the remaining few nodes might be left for the vehicle with a high travel cost. From the simulation results, we can see that our algorithm would perform particularly well if all vehicles have similar costs.

[2]We utilized the ILP solver tool GAMS (http://www.gams.com) to obtain the optimal solution for each instance of the input VRP.

VIII. Implementation on Unmanned Aerial Vehicles

We have implemented our on-the-fly algorithm on real unmanned aerial systems (UASs). We designed and assembled quadracopters on top of RCTimer Spider with on board Raspberry Pi computer and long-range Wi-Fi module Alfa. Each UAS is controlled by a Pixhawk flight controller which can execute commands from the onboard computer. Onboard computers of each UAS communicate between each other through a Wi-Fi access point. Each UAS finds its position in environment using a GPS/compass module.

We have implemented the algorithm in python using MAVlink protocol for Pixhawk. We use a synchronous broadcast model for communication between UASs using TCP/IP. Each UAS acts as a client and runs the distributed on-the-fly algorithm. A TCP/IP server is implemented on a central computer. This server assigns parameters to each UAS (e.g., the speed, target altitude etc) and sends details of a graph (i.e., GPS coordinates of nodes). Once the UASs receive these parameters and pass all the required safety pre-arm checks, they start executing the algorithm. Each UAS takes off and reaches its target altitude and goes to the exact GPS position of its depot. If the current node has not been serviced, a UAS decides to service the node, shown by a temporary change of its target altitude. Then, each UAS communicates with other rival UASs, informing them about the nodes serviced in the current communication round, and waits for messages from all its rivals. When all the UASs receive messages from each other, the algorithm continues until all the nodes in the graph are serviced.

We have conducted several experiments in a 40×40 outdoor fields. The video clips of the experiments are available at http://www.cas.mcmaster.ca/borzoo/uas.

IX. Related Work

The closest variation of OMDVRP is the problem of *open VRP* wherein all vehicles start from the same depot and are

not required to return back to their depots. In [7], the authors use a Cluster First, Route Second (CFRS) approach to solve the open VRP problem based on a minimum spanning tree with penalties procedure. The work in [8] tackles open VRP using tabu search algorithm (TSA). The approach in [9] uses the *bone route* algorithm (BR) to solve the open VRP. The BR algorithm is a genetic solution procedure in the sense that it produces a new solution out of components of routes of previous good solutions. One can use ant colony optimization (ACO) [10] or particle swarm optimization (PSO) techniques [11] to solve open VRP.

Different methods to solve OMDVRP have been proposed in the literature. In [12], the authors solve the OMDVRP to tackle the distribution of fresh meat from a major Greek industry to customers located in Athens. This paper uses a stochastic search meta-heuristic algorithm called List Based Threshold Accepting (LBTA) which iteratively searches the solution space guided by a deterministic control parameter called threshold which reveals promising regions for better configurations. In [13], the authors solve the OMDVRP problem using UASs. They design a path planning algorithm for UASs using mixed integer linear programming (MILP). The MILP solution gives the UASs points where they need to go and due to the unforeseen disturbance in the environments, the UASs may be forced to apply the path planning again to complete their graph.

X. Conclusion and Future Work

In this paper, we proposed a synchronous distributed approximation algorithm that solves the multi-depot vehicle routing problem (VRP). Our algorithm (1) computes a set of fractional values that solve the linear program relaxation of the input VRP instance, and (2) generates integer values using either a simple or randomized rounding scheme. We show that the approximation ratio of the solution obtained by randomized rounding is $O(n \cdot (\rho)^{1/n} \cdot log(n+m))$, where ρ is the maximum cost of travel or service in the input VRP instance, n is the size of the graph, and m is the number of vehicles. We reported simulation results that compare the performance of our algorithm with a simple centralized greedy algorithm as well as the optimal solution obtained from an integer linear program solver. We also discussed the implementation of our algorithm on a group of real unmanned aerial vehicles (UASs) that intend to inspect a large outdoor area.

For future work, we are considering several directions. First, we would like to reduce the level of synchronization among vehicles to improve the performance of our algorithm. We are currently working on 3D graphs, which is crucial in the context of UAS planning. Another important aspect is to bring energy constraints as well as physical environment parameters (e.g., weather conditions) into the formulation of the VRP.

XI. Acknowledgments

This work was partially sponsored by Canada NSERC Discovery Grant 418396-2012 and NSERC Strategic Grants 430575-2012 and 463324-2014.

Engage grant

References

[1] P. Toth and D. Vigo, "The vehicle routing problem," *SIAM Monographs on Discrete Mathematics and Applications*, 2002.

[2] M. Pavone, E. Frazzoli, and F. Bullo, "Adaptive and distributed algorithms for vehicle routing in a stochastic and dynamic environment," *IEEE Transactions on Automation and Control*, vol. 56, no. 6, pp. 1259–1274, 2011.

[3] T. Moscibroda and R. Wattenhofer, "Facility location: distributed approximation," in *Proceedings of the 24th ACM symposium on Principles of distributed computing (PODC)*, 2005, pp. 108–117.

[4] F. Kuhn and R. Wattenhofer, "Constant-time distributed dominating set approximation," *Distributed Computing*, vol. 17, no. 4, pp. 303–310, 2005.

[5] K. Jain and V. V. Vazirani, "Primal-dual approximation algorithms for metric facility location and k-median problems," in *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE, 1999, pp. 2–13.

[6] A. Bachem and W. Kern, "Linear programming duality," in *Linear Programming Duality*. Springer, 1992, pp. 89–111.

[7] D. Sariklis and S. Powell, "A heuristic method for the open vehicle routing problem," *Journal of the Operational Research Society*, vol. 51, no. 5, pp. 564–573, 2000.

[8] J. Brandão, "A tabu search algorithm for the open vehicle routing problem," *European Journal of Operational Research*, vol. 157, no. 3, pp. 552–564, 2004.

[9] C. D. Tarantilis, D. Diakoulaki, and C. T. Kiranoudis, "Combination of geographical information system and efficient routing algorithms for real life distribution operations," *European Journal of Operational Research*, vol. 152, no. 2, pp. 437–453, 2004.

[10] X. Li, P. Tian, and S. C. Leung, "An ant colony optimization metaheuristic hybridized with tabu search for open vehicle routing problems," *Journal of the Operational Research Society*, vol. 60, no. 7, pp. 1012–1025, 2009.

[11] S. MirHassani and N. Abolghasemi, "A particle swarm optimization algorithm for open vehicle routing problem," *Expert Systems with Applications*, vol. 38, no. 9, pp. 11 547–11 551, 2011.

[12] C. Tarantilis and C. Kiranoudis, "Distribution of fresh meat," *Journal of Food Engineering*, vol. 51, no. 1, pp. 85–91, 2002.

[13] D. Habib, H. Jamal, and S. A. Khan, "Employing multiple unmanned aerial vehicles for co-operative path planning," *International Journal of Advanced Robotic Systems*, vol. 10, 2013.

APPENDIX

Lemma 1: Algorithm 2 forms a feasible solution for the LP.

Proof:

- **Constraint 10.** The function call to CheckIfPathFree in line 12, ensures that a node is serviced by only one vehicle in any iteration of the t-loop. In line 20, the algorithm increases the y_v^a value by $1/m$. Hence,

$$\sum_{a' \in A} y_v^{a'} = \frac{1}{m}$$

If a node is serviced, a message is sent across to all vehicles and received by all vehicles in the same communication round. When a message is received from other rival vehicles, the current vehicle updates the set of serviced nodes and, hence, these already serviced nodes does not feature in the N_k^a in the following communication rounds. So, it cannot be the case that a vehicle services the same node again in any subsequent iterations of the t-loop. Hence, a node is serviced exactly once and we have

$$\sum_{a' \in A} y_v^{a'} = \frac{1}{m}.$$

- **Constraint 3.** There are 3 cases here.
  - A vehicle travels from one node to another without *servicing*: For *travel*, Algorithm 2 increases only the $\Delta x_{(u,v)}^a$ value in line 25 or line 33 and Δy_v^a is initialized to zero in line 1. Hence, we have

$$\sum_{u \in V} x_{(u,v)}^a - y_v^a \geq 0.$$

  - A vehicle services a node other than its depot: Algorithm 2 allows a vehicle to either *service* a node or *travel* from one node to another node in any communication round. This LP constraint gives a relation between $x_{(u,v)}^a, y_v^a$. We need to prove that a vehicle travels to a node before servicing it. Hence, we need to consider two consecutive communication rounds to prove the validity of this constraint. If a vehicle chooses a path of length zero ($|p'| = 0$) and services it in the current communication round, from the definition of N_k^a, we see that N_k^a becomes $\emptyset$ in the next communication round which forces the vehicle to *travel* to another node. With the same logic, we can claim that in the previous communication round before the servicing took place, the vehicle would have traveled from some other node to the current location. Also when a vehicle chooses a path of length more than zero ($|p'| > 0$), our algorithm in every two rounds of communication ensures that the vehicles travels to the next node in the path before servicing it, in the next communication round. This is ensured by the condition check in line 19, which will return false if the current location is serviced and hence forcing the vehicle to take the else loop where it travels to the next node in the path chosen. Therefore, we have that a corresponding $x_{(u,v)}^a$ is increased by a value of $1/m$ before increasing the value of y_v^a by $1/m$ in the next communication round. So this guarantees the fact that if a node is serviced in the current iteration of the t-loop, the corresponding $x_{(u,v)}^a$ value would have been increased from the previous iteration. Hence, we have that Δy_v^a can never be greater than $\Delta x_{(u,v)}^a$. Hence, we have

$$\sum_{u \in V} x_{(u,v)}^a - y_v^a \geq 0.$$

  - The constraint trivially holds for the depot because we actually exclude the depot from the constraint (i.e., $V - \{v_0^a\}$).
- **Constraints 11, 12, 13.** A vehicle always chooses a simple path and hence this path satisfies the constraints 11, 12, and 13. Constraints 11 and 12 are satisfied from the definition of a simple path, which ensures that a vehicle takes a maximum of only one incoming edge and one outgoing edge to travel to and from a node. By the definition of N_k^a and $nextbest(a)$, our algorithm does not allow a node to be visited twice. From line 36, we see that in each iteration of the t-loop, γ_v^a and τ_u^a are increased by at most $\Delta x_{(u,v)}^a$; i.e., by $1/m$. Hence, Constraints 11 and 12 are satisfied. And also the definition of a simple path ensures there is no sub-tour, which satisfies Constraint 13. Once an edge is traversed by a vehicle, the same edge does not feature in the N_k^a in the next iteration of the t-loop and also from the definition of $nextbest(a)$ when a vehicle travels from one node to another node, the vehicle chooses an edge which has not been visited by the vehicle before. Hence, there is no subtour.
- **Constraint 6.** Constraint 3 ensures that a vehicle enters a node before servicing it. So to service a node, the vehicle must take an incoming edge to reach the node. Also Constraints 11, 12 ensure that a vehicle takes a maximum of only one incoming edge and one outgoing edge to travel to and from a node. Also Algorithm 2 guarantees that a vehicle does not exit a node without traveling to it. Hence, its always the case that the sum of incoming edges to a node is greater than or equal to the number of outgoing edges from the node. Hence, this constraint is also satisfied.

■

Lemma 2: At the end of each iteration of the t-loop, the value of the primal and dual objectives are equal. That is,

$$\sum_{a \in A} \sum_{v \in V} y_v^a \cdot C_s(a, v) + \sum_{a \in A} \sum_{(u,v) \in E} x_{(u,v)}^a \cdot C_t(a, (u, v)) =$$

$$\frac{1}{m}\Bigg(\sum_{v\in V}\alpha_v - \sum_{a\in A}\sum_{v\in V}\gamma_v^a - \sum_{a\in A}\sum_{u\in V}\tau_u^a - (n-m-1)\sum_{a\in A}\sum_{(u,v)}\lambda_{(u,v)}^a\Bigg)$$

Proof: After each iteration of the inner t-for loop, we have

$$\Delta L.H.S = \sum_{a\in A}\sum_{v\in V}\Delta y_v^a \cdot C_s(a,v) + \sum_{a\in A}\sum_{(u,v)\in E}\Delta x_{(u,v)}^a \cdot C_t(a,(u,v))$$

$$\Delta R.H.S = \frac{1}{m}\Bigg(\sum_{v\in V}\Delta\alpha_v - \sum_{a\in A}\sum_{v\in V}\Delta\gamma_v^a - \sum_{a\in A}\sum_{u\in V}\Delta\tau_u^a - (n-m-1)\sum_{a\in A}\sum_{(u,v)}\Delta\lambda_{(u,v)}^a\Bigg)$$

Let's consider the case of R.H.S. First, observe that in line 36, we have $\Delta\gamma_v^a = -\Delta\tau_u^a$, taken to travel from any pair of vertices u and v. So for any vehicle $a \in A$, the sum $\sum_{v\in V}\Delta\gamma_v^a = -\sum_{u\in V}\Delta\tau_u^a$ and hence the sum across all vehicles must be equal as well. Hence, we have

$$\sum_{a\in A}\sum_{v\in V}\Delta\gamma_v^a = -\sum_{a\in A}\sum_{u\in V}\Delta\tau_u^a$$

Also, we claim $\Delta\lambda_{(u,v)}^a = 0$. This is because our algorithm ensures that a vehicle cannot travel u to v and from v to u form any $(u,v) \in E$, from the definition of N_k^a and $nextbest(a)$. Hence, lines 26, and 35 will not be executed in the algorithm. From the definition of N_k^a the algorithm ensures that a path is chosen only if all the edges in the path are not traveled before by the current vehicle. Hence, a vehicle will not *travel* through the same edge twice. Also, the definition of $nextbest(a)$ does not allow a vehicle to visit a node that has already been visited by it.

$$\therefore \Delta R.H.S = \frac{1}{m}\sum_{v\in V}\Delta\alpha_v$$

At the end of each round, each vehicle sends a message containing either the value of $(\Delta y_v^a \cdot C(N_k^a))$ or $(\Delta x_{(u,v)}^a \cdot C_t(a,(u,v)))$ to all vehicles. After receiving these messages in line 44 and subsequently all iterations of the two for-loops in lines 45-51 (for Δy values) and 52 - 57 (for Δx values), we will have

$$\frac{1}{m}\sum_{v\in V}\Delta\alpha_v = \sum_{a\in A}\sum_{v\in V}\Delta y_v^a \cdot C(N_k^a) + \sum_{a\in A}\sum_{u\in V}\Delta x_{(u,v)}^a \cdot C_t(a,(u,v))$$

Now, observe that for any $a \in A$ and $v \in V$, we have

$$C(N_k^a) = \left(\frac{1}{2|p'|+1}\cdot\left(\sum_{v\in p'}C_s(a,v) + \sum_{(u,v)\in p'}C_t(a,(u,v))\right)\right)$$

where p' is the path chosen to service and travel by vehicle a in line 12. Since we have that a vehicle can *service* only one node in each iteration of the t-loop, we take the value of $(2|p'|+1)$ as 1 and take $C_t(a,(u,v))$ as zero. Hence, for every node v,

$$C(N_k^a) = C_s(a,v)$$

In total we have,

$$\frac{1}{m}\sum_{v\in V}\Delta\alpha_v = \sum_{a\in A}\sum_{v\in V}\Delta y_v^a \cdot C_s(a,v) + \sum_{a\in A}\sum_{u\in V}\Delta x_{(u,v)}^a \cdot C_t(a,(u,v)) = \Delta L.H.S$$

■

Lemma 3: In every iteration of the t-loop, it holds that

$$C(N_k^a) \le \rho$$

Proof: We define $\rho = \max\Big\{C_s(a,v), C_t(a,(u,v)) \mid (a \in A) \wedge (u,v \in V)\Big\}$ and N_k^a generates a set of paths p starting from length 0 to a maximum value less than k. For the case when $|p| = 0$, we have that $C(N_k^a)$ would have its servicing cost of its current location to be $C_s(a,v) \le \rho$. For the case where $|p| > 0$, $C(N_k^a)$ is normalized by a factor of $2|p'|+1$. When a vehicle chooses a path of length $|p| > 0$, all paths of length starting from zero are listed and the minimum cost among these is chosen as $C(N_k^a)$. We have shown that for the case $|p| = 0$, we have $C_s(a,v) \le \rho$. Since we take the minimum value among all the generated paths starting from length 0 as $C(N_k^a)$, its always the case that we have

$$C(N_k^a) \le \rho$$

■

In the next lemma, we bound the $\Delta\alpha_v$ that each vehicle receives in one iteration of the s-loop for any node $v \in V$. Let us define $\Delta\alpha_v(s)$ as the sum of $\alpha_v(s)$ values received for a node v in a single iteration of the s-loop by a vehicle a, having a maximum of k *rounds of communication* of the t-loop.

$$\therefore \Delta\alpha_v(s) = \sum_{v\in p'}\Delta\alpha_v$$

and let $\sigma_v(s)$ be the defined as the sum of all $(\Delta y_v^a \cdot C(N_k^a) + \Delta x_{(u,v)}^a \cdot C_t(a,(u,v)))$ values received for a node v by a

vehicle a for a single iteration of the s-loop having a maximum of k rounds of communication of the t-loop.

$$\sigma_v(s) = \sum_{v \in p'} (\Delta y_v^a \cdot C(N_k^a) + \Delta x_{(u,v)}^a \cdot C_t(a,(u,v)))$$

$$= \underbrace{\sum_{v \in p'} (\Delta y_v^a \cdot C(N_k^a))}_{\sigma_v^1(s)} + \underbrace{\sum_{v \in p'} \Delta x_{(u,v)}^a \cdot C_t(a,(u,v))}_{\sigma_v^2(s)}$$

We have that

$$\Delta \alpha_v(s) = \sigma_v(s)$$

Lemma 4: In each iteration of s-loop, for any vehicle $a \in A$ and node $v \in V$, we have

$$\sigma_v(s) \leq \rho$$

Proof:

From Lemma 1, we have shown that Algorithm 2 satisfies all constraints of the LP at all times and hence we have that, all vehicles can visit a node at most once and only one among those vehicles can *service* a particular node. Hence, we can bound the $\sigma_v(s)$ values with this information.

We distinguish two cases:

- **Case 1.** Algorithm 2 ensures that only one vehicle services a node v at most once. When a vehicle services a node the condition $C(N_k^a) \leq \rho$ is satisfied. Hence,

  $$\sigma_v^1(s) \leq \sum_{v \in p'(s)} \Delta y_v^a \cdot \rho$$

  We have the value of y_v^a increased by $1/m$ and we can bound the value of $C(N_k^a)$ by the value ρ

  $$\sigma_v^1(s) \leq \left(\frac{1}{m}\right) \cdot \rho$$

- **Case 2.** Algorithm 2 also ensures that a vehicle travels to a node v at most once. We have the value of $x_{(u,v)}^a$ increased by $1/m$ and we can again bound the cost of $C_t(a,(u,v))$ by ρ from the definition of ρ. Hence,

  $$\sigma_v^2(s) \leq \left(\frac{1}{m}\right) \cdot \rho$$

  For a given iteration of the s-loop, we can have Case 1 or Case 2 or both can be true.

- **Case 1 and 2 are simultaneously true.** This scenario comes when a vehicle chooses a path of length > 0 and decides to *service* all nodes. So for all nodes except the starting node, we have that a vehicle travels to that node before servicing the node. Before servicing a node the CheckIfPathFree function is called and this function ensures that no other vehicles can enter the nodes in the path before the current vehicle completes servicing all nodes in its chosen path. Hence, in one iteration of the s-loop, at most we can have two rounds of communications (servicing, traveling) involving the node v.

  $$\sigma_v(s) = \sigma_v^1(s) + \sigma_v^2(s)$$

  $$\sigma_v(s) \leq \frac{2 \cdot \rho}{m}$$

  Now, recall that in VRP, we have $m \geq 2$. Hence,

  $$\sigma_v(s) \leq \rho$$

- **Either Case 1 or 2 is true.** In this case, either a vehicle services the node or travels to an another node. That is,

  $$\sigma_v(s) = \sigma_v^1(s) \quad \text{or} \quad \sigma_v(s) = \sigma_v^2(s)$$

  This implies that,

  $$\sigma_v(s) \leq \left(\frac{1}{m}\right) \cdot \rho$$

  In the same iteration of the s-loop, we can have at most have $m-1$ vehicles entering the same node. So, in each iteration of the t-loop, when the s-loop terminates, we have

  $$\sigma_v(s) \leq \left(\frac{1}{m}\right) \cdot m \cdot \rho = \rho$$

  Hence, in both cases, we can bound

  $$\sigma_v(s) \leq \rho$$

  Therefore, we can bound

  $$\alpha_v(s) \leq \rho$$

■

For our dual solution to be feasible, Constraint 15 must be satisfied. That is, for all $a \in A$ and $(u,v) \in E$

$$\beta_v^a - \gamma_v^a - \tau_u^a - \lambda_{(u,v)}^a + \theta_v^a \leq C_t(a,(u,v))$$

The dual solution produced by our algorithm does not exhibit this feasibility property. This is because we cannot guarantee the value $\beta_v^a + \theta_v^a$ is $\leq C_t(a,(u,v))$ for all $(u,v) \in E$ for a fixed vehicle a. We can only guarantee this constraint satisfies for the edges (u,v) which was taken by a vehicle a in any s-loop. However, we can at least show that the degree of infeasibility is bounded. Specifically, it holds that if we only consider the sum of the increases of the $\beta_v^a - \gamma_v^a - \tau_u^a - \lambda_{(u,v)}^a + \theta_v^a \leq C_t(a,(u,v))$ for every node, in a single iteration of the s-loop, it fulfills the desired property.

Lemma 5: For all $v \in V - \{v_0^a\}$ and $a \in A$, in all iterations of the s-loop, Constraint 14 holds.

Proof: Following Lemma 4, we have $\alpha_v \leq \rho$. From the algorithm we have that,

$$\beta_v^a = \begin{cases} 0 & \text{if vehicle } a \text{ services a node } v_0^a. \\ \rho - C_s(a,v) & \text{if vehicle } a \text{ services a node } v. \\ \rho & \text{if vehicle } a \text{ travels to a node } v \text{ to } u. \end{cases}$$

- **Case 1.** $\beta_v^a = 0$, when a vehicle services its depot. Hence, the condition $\alpha_v \leq C(N_k^a)$ is satisfied and we have

  $$\alpha_v - \beta_v^a \leq C(N_k^a) = C_s(a,v) + C_t(a,(u,v))$$

In this case, a vehicle services from the depot. Hence, we take $C_t(a,(u,v)) = 0$ and we have:

$$\alpha_v - \beta_v^a \leq C_s(a,v)$$

- **Case 2.** $\beta_v^a = \rho - C_s(a,v)$. Hence,

$$\alpha_v - \beta_v^a \leq \{\rho - \rho\} + C_s(a,v) = C_s(a,v)$$

- **Case 3.** $\beta_v^a = \rho$. Hence,

$$\alpha_v - \beta_v^a = \{\rho - \rho\} = 0 \leq C_s(a,v)$$

■

Lemma 6: For all $(u,v) \in E$ and $a \in A$, in all iterations of the s-loop, Constraint 15 holds.

Proof: Algorithm 2 assigns $\gamma_v^a = -\tau_u^a$ and $\lambda_{(u,v)}^a = 0$ for all edges (u,v) taken for travel. So we just need to prove

$$\beta_v^a + \theta_v^a \leq C_t(a,(u,v))$$

From Algorithm 2 we have,

$$\beta_v^a = \begin{cases} 0 & \text{if vehicle } a \text{ services a node } v_0^a. \\ \rho - C_s(a,v) & \text{if vehicle } a \text{ services a node } v. \\ \rho & \text{if vehicle } a \text{ travels to a node } v \text{ to } u. \end{cases}$$

- **Case 1.** $\beta_v^a = 0$, when a vehicle services its depot. θ_v^a is not defined for this case. Hence, we have

$$\beta_v^a \leq C_t(a,(u,v))$$

- **Case 2.** $\beta_v^a = \rho - C_s(a,v)$.
In this case, a vehicle travels to a node v and also services it. So we can write the value of β_v^a as $C_t(a,(u,v)) - C_s(a,v)$. Hence, we have

$$\beta_v^a \leq C_t(a,(u,v))$$

θ_v^a is increased by $1/m$. An increase by $1/m$ would not affect the $C_t(a,(u,v))$ because by the definition of

$$C_t : A \times E \to \mathbb{Z}_{\geq 0}$$

Hence, we have $\beta_v^a + \theta_v^a \leq C_t(a,(u,v))$

- **Case 3.** $\beta_v^a = \rho$. In this case, a vehicle travels to a node v and without servicing it. So we can write the value of β_v^a as $C_t(a,(u,v))$. Hence, we have

$$\beta_v^a \leq C_t(a,(u,v))$$

Again θ_v^a is increased by $1/m$. An increase by $1/m$ would not affect the $C_t(a,(u,v))$. Hence, we have $\beta_v^a + \theta_v^a \leq C_t(a,(u,v))$.

■

Having bounded the degree of dual infeasibility in the two previous lemmas, we can now establish the approximation ratio of the algorithm using the laws of LP duality. Specifically, we prove that the dual feasibility is violated only by a factor $O(n \cdot (\rho)^{1/n})$ and hence, when dividing by α_v, β_v^a and θ_v^a by suitably large values, we obtain a feasible solution $\hat{\alpha_v}$, $\hat{\beta_v^a}$ and $\hat{\theta_v^a}$.

Let us define $\hat{\alpha_v} = \frac{\alpha_v}{n(\rho)^{1/n}}$, $\hat{\beta_v^a} = \frac{\beta_v^a}{n}$, $\hat{\theta_v^a} = \frac{\theta_v^a}{n}$. We show that $\hat{\alpha_v^a}$, $\hat{\beta_v^a}$ and $\hat{\theta_v^a}$ form a feasible solution to the dual LP. The feasibility of Constraint 15 is trivial. Since we have proved that $\beta_v^a + \theta_v^a \leq C_t(a,(u,v))$ and hence, we have $\frac{\beta_v^a + \theta_v^a}{n} \leq C_t(a,(u,v))$. That is, $\hat{\beta_v^a} + \hat{\theta_v^a} \leq C_t(a,(u,v))$.

Lemma 7: For all $v \in V - \{v_0^a\}$ and $a \in A$, in all iterations of the s-loop, Constraint 14 holds.

Proof:

First, observe that

$$\hat{\alpha_v} - \hat{\beta_v^a} =$$

$$= \sum_{s=1}^{n} \frac{\alpha_v(s)}{n(\rho)^{1/n}} - \sum_{s=1}^{n} \frac{\beta_v^a(s)}{n}$$

$$= \frac{1}{n}\left(\sum_{s=1}^{n}\left(\frac{\alpha_v(s)}{(\rho)^{1/n}} - \beta_v^a(s)\right)\right)$$

So we need to prove the following:

$$\frac{\alpha_v(s)}{(\rho)^{1/n}} - \beta_v^a(s) \leq C_s(a,v)$$

- **Case 1.** $\beta_v^a = 0$, when a vehicle services its depot. Hence, the condition $\alpha_v \leq C(N_k^a)$ is satisfied and we have

$$\frac{\alpha_v(s)}{(\rho)^{1/n}} - \beta_v^a \leq C(N_k^a) = C_s(a,v) + C_t(a,(u,v))$$

In this case, a vehicle services from the depot. Hence, we take $C_t(a,(u,v)) = 0$.

$$\therefore \frac{\alpha_v(s)}{(\rho)^{1/n}} \leq \frac{C_s(a,v)}{\rho^{1/n}} \leq C_s(a,v)$$

- **Case 2.** $\beta_v^a = \rho - C_s(a,v)$.

$$\frac{\alpha_v(s)}{(\rho)^{1/n}} - ((\rho) - C_s(a,v))$$

$$\leq (\rho)^{(1-1/n)} - \rho + C_s(a,v) \leq C_s(a,v)$$

- **Case 3.** $\beta_v^a = \rho$.

$$\frac{\alpha_v(s)}{(\rho)^{1/n}} - (\rho)$$

$$\leq (\rho)^{(1-1/n)} - (\rho) \text{ is a negative value}$$

$$\therefore \frac{\alpha_v(s)}{(\rho)^{1/n}} - \beta_v^a \leq C_s(a,v)$$

By this lemma, we have that each term of the sum is bounded by $C_s(a,v)$. Therefore, we have $\hat{\alpha_v} - \hat{\beta_v^a} \leq \frac{n \cdot C_s(a,v)}{n} \leq C_s(a,v)$

■

Let OPT be the optimal value and ALG be the value computed by Algorithm 2. By the LP duality theorem, the sum of $\hat{\alpha_v}$ values is a lower bound of OPT.

Theorem 1 The approximation ratio of Algorithm 2 is

$$O(n \cdot (\rho)^{1/n}).$$

Proof: ALG can be bounded as

$$ALG = \sum_{a \in A} \sum_{v \in V} y_v^a \cdot C_s(a,v) + \sum_{a \in A} \sum_{(u,v) \in E} x_{(u,v)}^a \cdot C_t(a,(u,v))$$

$$= \frac{1}{m} \sum_{v \in V} \alpha_v$$

$$\leq n(\rho)^{1/n} \sum_{v \in V} \hat{\alpha_v}$$

$$= n(\rho)^{1/n} \cdot OPT$$

■

Lemma 8: Algorithm 3 produces a feasible ILP solution.
Proof:

- **Constraint 2.** For every node $v \in V$, the vehicle that processed the node v, increases its corresponding $y_v^a = 1$ from the lines (9 or 25); i.e., For any node v, if the value is made $y_v^{a'}$ is made 1 with probability $p_v^{a'}$ and $a' \in A_v$, only one $y_v^{a'}$ is assigned the value 1 from line 9. Similarly, for any node v, if y_v^a is assigned 0 with probability $1 - p_v^{a'}$, then the condition check in Line 16 returns false, and hence, only one y_v^a is assigned the value 1 from line 25. From the messages received if node $(v \notin visited^a \wedge v \in total^a)$, the corresponding vehicle's y_v^a is made 1 from line 39. Also, if there is an update in the vehicle id which has serviced the node, the previous y_v^a is made 0 and then the newly updated y_v^a is made 1 from line 43 else the same y_v^a value is retained. Hence, in all cases a node is serviced exactly once. So, we have that $\sum_{a \in A} y_v^a = 1$.
- **Constraint 3.** For every node $v \in V$, the randomized rounding algorithm increases $x_{(u,v)}^a = 1$ in lines (6 or 23 or 36). In these 3 lines, the value $x_{(u,v)}^a$ is made 1 only if $\hat{x}_{(u,v)}^a > 0$; i.e., if the vehicle has taken an edge to reach node v. Only one vehicle is allowed to *service* a node, as shown in the previous constraint and we have $\sum_{a \in A} y_v^a = 1$. Hence, the condition $\sum_{u \in V} x_{(u,v)}^a - y_v^a \geq 0$ is always true.
- **Constraints 4, 5, and 7.** From lines (6 and 23 and 36) we can say that there are no sub-tours. Constraints (4, 5) are satisfied as the $x_{(u,v)}^a$ values are increased by 1 only if the corresponding fractional values are increased (i.e.,$\hat{x}_{(u,v)}^a > 0$) from Algorithm 2.
- **Constraint 6.** The Constraint 3 ensures that a vehicle enters a node before servicing it. Also Constraints 4, 5 ensure that there are at most only one incoming and outgoing edge for each node v. Also Algorithm 2 guarantees that a vehicle does not exit a node without traveling to it. Hence, its always the case that the sum of incoming edges to a node is greater than or equal to the number of outgoing edges from the node. Hence, this constraint is also satisfied.

■

Theorem 2: The approximation ratio of Algorithm 3 is

$$O(n \cdot (\rho)^{1/n} \cdot log(n+m)).$$

Proof:

- **Case 1.** $(a' \in A_v \wedge (y_v^{a'} = 1))$ (Line 16)
By the definition of A_v the *travel* costs are at most

$$C_t(a,(u,v)) \leq \ln(n+m) \cdot C_v$$

It follows that, the total cost for traveling is at most

$$\ln(n+m) \sum_{a \in A} \sum_{(u,v) \in E} \hat{x}_{(u,v)}^a \cdot C_t(a,(u,v))$$

We have from line 8 of Algorithm 3, the value of y_v^a is 1 with the probability $\min\{1, \hat{y_v^a} \cdot \ln(n+m)\}$. The expected cost of *service* is bounded by the value

$$\ln(n+m) \sum_{a \in A} \sum_{v \in V} \hat{y}_v^a \cdot C_s(a,v)$$

- **Case 2.** $(a' \notin A_v \vee (y_v^{a'} \neq 1))$ (Line 16)
From the definition of C_v, A_v we have that

$$\sum_{a \notin A_v} \sum_{u \in V} \hat{x}_{(u,v)}^a \leq \frac{1}{\ln(n+m)} \tag{20}$$

for if not, C_v would be larger.
Algorithm 2 guarantees that, for all $a \in A$, and $v \in V - \{v_0^a\}$, we have $\sum_{u \in V} \hat{x}_{(u,v)}^a - \hat{y}_v^a \geq 0$ i.e., it guarantees that *service* happens only once at a node and hence only the number of travels can increase and not the number of services for any node. Therefore, we have that

$$\sum_{a \in A_v} \hat{y}_v^a \leq \sum_{a \in A_v} \hat{x}_{(u,v)}^a$$

and also we can claim that for $v \neq v_0^a$ (where v_0^a represents depot location of vehicle a)

$$\sum_{a \in A} \sum_{u \in V} \hat{x}_{(u,v)}^a \leq 1 \tag{21}$$

Equation 21 value comes from the fact that $\hat{x}_{(u,v)}^a$ is increased by a value $\frac{1}{m}$ and at most m vehicles can have $\hat{x}_{(u,v)}^a = \frac{1}{m}$. So, in total,

$$\sum_{a \in A} \sum_{u \in V} \hat{x}_{(u,v)}^a \leq m \cdot \left(\frac{1}{m}\right) \leq 1$$

Hence, we have

$$\sum_{a \in A_v} \hat{y}_v^a \leq \sum_{a \in A_v} \sum_{u \in V} \hat{x}_{(u,v)}^a \leq 1 - \frac{1}{\ln(n+m)} \quad (\text{ From } 20, 21) \tag{22}$$

The probability that $(a' \notin A_v \vee (y_v^{a'} \neq 1))$ happens is at most

$$q_a = \prod_{a \in A_v} (1 - p_a)$$

From Means Inequality we have, let $A \subset \mathbb{R}^+$ be a set of positive real numbers. The product of the values in A can be upper bounded by replacing each factor with arithmetic mean of elements of A:

$$\prod_{x \in A} x \leq \left(\frac{\sum_{x \in A}}{|A|} \right)^{|A|}$$

$$\therefore q_a \leq \left(\frac{\sum_{a \in A_v} (1 - p_a)}{n + m} \right)^{n+m}$$

$$\leq \left(1 - \frac{\ln(n+m) \sum_{a \in A_v} \hat{y}_v^a}{n+m} \right)^{n+m}$$

From Equation 22 we have

$$q_a \leq \left(1 - \frac{\ln(n+m)}{n+m} \left(1 - \frac{1}{\ln(n+m)} \right) \right)^{n+m}$$

$$\leq \left(1 - \frac{\ln(n+m) + 1}{n+m} \right)^{n+m}$$

From Means Inequality we have, for $n \geq x \geq 1$, we have

$$\left(1 - \frac{x}{n} \right)^n \leq e^{-x}$$

$$\therefore q_a \leq \left(1 - \frac{\ln(n+m) + 1}{n+m} \right)^{n+m}$$

$$\leq e^{-\ln(n+m)-1} = \frac{1}{e(n+m)}$$

Combining all results together, the total expected cost $\mu = E[ALG]$ is

$$\mu \leq \ln(n+m) \left(\sum_{a \in A} \sum_{v \in V} \hat{y}_v^a \cdot C_s(a, v) + \sum_{a \in A} \sum_{(u,v) \in E} \hat{x}_{(u,v)}^a \cdot C_t(a, (u, v)) \right)$$

$$\mu \leq \ln(n+m) \left(\sum_{a \in A} \sum_{v \in V} \hat{y}_v^a \cdot C_s(a, v) + \sum_{a \in A} \sum_{(u,v) \in E} \hat{x}_{(u,v)}^a \cdot C_t(a, (u, v)) \right) + \frac{n}{e(n+m)} \left(\sum_{a \in A} \sum_{v \in V} \hat{y}_v^a \cdot C_s(a, v) + \sum_{a \in A} \sum_{(u,v) \in E} \hat{x}_{(u,v)}^a \cdot C_t(a, (u, v)) \right)$$

$$\leq (\ln(n+m) + O(1)) \cdot n \cdot \rho^{1/n} \cdot OPT$$

Since we have $\ln(a) = \frac{log(a)}{log(e)}$, we obtain

$$\mu = E[ALG] \leq (log(n+m)) \cdot n \cdot \rho^{1/n} \cdot OPT$$

Hence, in total from Algorithms 2 and 3 the approximation ratio is the following:

$$O(n \cdot (\rho)^{1/n} \cdot log(n+m))$$

■