Distributed Vehicle Routing Approximation

DISTRIBUTED VEHICLE ROUTING APPROXIMATION

By Akhil Krishnan,B.E

A Thesis Submitted to the School of Graduate Studies in Partial Fulfillment of the Requirements for the degree of Master of Science

McMaster University

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 comparing our algorithm results with ILP solutions and a greedy algorithm.

Acknowledgements

I would like to express my sincere thanks to my Supervisor, Dr.Borzoo Bonakdarpour for his encouragement,his expert guidance and advice through out this thesis.

I would also like to thank the Department of Computing and Software, with their faculty and staff, for their help and support.

Finally, I want to thank my family and friends for their constant support and encouragement during my study at McMaster University.

Contents

List of Figures

Chapter 1

Introduction

With the growing popularity of autonomous systems in the modern world, more and more researchers try to solve complex problems and design algorithms in a distributed manner. These efforts enable us to design autonomous systems which can act together with other autonomous vehicles and solve the problem in hand without having any knowledge of the global state of other autonomous systems. In this thesis, we propose a distributed algorithm to solve a variation of the vehicle routing problem, where a set of distributed vehicles need to make independent decisions to service a set of non-conflicting cities traveling through their respective paths.

1.1 The Vehicle Routing Problem

The *vehicle routing problem* (VRP) 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 serve a set of customers. VRP generalizes the well-known traveling salesman problem (TSP) and the multiple traveling salesman problem (mTSP), where more than one salesman is allowed to be used in the solution [Bektas, 2006]. The VRP was first introduced by [Dantzig and Ramser, 1959]:

> The VRP is defined on a complete graph $G = (V, E)$, where $V = \{0, ..., n\}$ is the set of nodes or cities, and $E = \{(i, j) : i, j \in V\}$ is the arc set. Node 0 represents the *depot* and the rest of the nodes represent the retailers.

The retailers have a certain demand, d_i that must be fulfilled and the cost of traveling between node i and j is defined by a cost c_{ij}.

The objective of the classical VRP is to minimize the overall cost, usually the travel distance, for the whole set of routes. All vehicles are required to start and end at the depot, and the total demand for each route can not exceed the capacity q of the vehicle. Each retailer must be visited once by only one vehicle, and get their demand fulfilled.

There are many variations of the VRP. Examples include cases where

- A number of goods need to be moved from certain pickup locations to other delivery locations;
- At any delivery location, the item being delivered must be the item most recently picked up;
- The delivery locations have time windows within which the deliveries (or visits) must be made;
- The vehicles have limited carrying capacity of the goods that must be delivered;
- There are multiple depots for several vehicles and the vehicles after servicing a set of customers return to the same depot. This variation is called the *multi-depot vehicle routing problem* (MDVRP). Furthermore, a variation of this problem does not require the vehicles to return to their depots. This variation is called the *open multi-depot vehicle routing problem* (OMDVRP).

1.2 Distributed OMDVRP

The original VRP has numerous optimization applications, e.g., in the collection of delivery of goods, waste collection, street cleaning, school-bus routing, routing of salespeople, and dial-a-ride systems. However, with the recent advances in the area of autonomous vehicles (e.g., driverless cars and unmanned aerial vehicles), VRP plays even a more important role in the design and deployment of autonomous vehicles. Such vehicles need to

make local decisions to reach global optimization on-the-fly and distributed fashion. For example, in the context of VRP, the problem is not solved a priori but solved while the vehicles are traveling and servicing cities through communication and making on-the-fly decisions on their respective next steps.

VRP is an NP-complete problem. Solving VRP in an offline and centralized fashion is already challenging. Most research efforts focus on developing heuristics and approximation algorithms and there is, in fact, a large body of work on such techniques. Adding distribution incurs another level of complexity, as there is no entity that has the full view of the system and can take global decisions. As a matter of fact, to our knowledge, there is no work on distributed OMDVRP and this is the problem we investigate in our thesis.

1.3 Thesis Statement

Our research hypothesis is that distributed OMDVRP can be solved effectively and efficiently. By efficient, we mean that one can design distributed algorithms with reasonable approximation bounds. By effective, we mean that these bounds allow us to actually implement and deploy the algorithms in real-world scenarios.

1.4 Contributions

Our goal is to validate our research hypothesis through designing a distributed algorithm that solves VRP and in particular OMDVRP. Given a set A of heterogeneous vehicles and a graph G, in our distributed model of computation (1) the vehicles can communicate using *synchronous* [Lynch, 1996] message passing, (2) the communication network is a complete graph, and (3) every vehicle has a unique *id* and knows the entire graph G and cost functions of all vehicles.

We make the following contributions towards our goal:

Our distributed approximation algorithm is inspired by the technique proposed in [Moscibroda and Wattenhofer, 2005a] to solve the facility location problem and in [Kuhn and Wattenhofer, 2005] to solve the minimum dominating set problem. In particu-

lar, 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 [Jain and Vazirani, 1999], 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 vehicles communicate, take local decisions and move along their routes, the latter stipulates a semi-offline technique, where the vehicles first compute their full routes and then start traveling and servicing.

1.5 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 discusses the related work. Chapter 3 presents the formal problem description of our problem and linear programming transformations. Chapter 4 discusses the designed distributed approximation algorithm and proofs to find the approximation ratio. Chapter 5 discusses the randomized rounding and on-the-fly distributed algorithm along with its implementation. Chapter 6 discusses the simulation results. Finally, in Chapter 7, we make concluding remarks and discuss future work.

Chapter 2

Literature Review

2.1 Multi-Robot Patrolling

The multi-robot patrol application consists of robots coordinating their movements around a patrol route. A patrol route consists of a sequence of GPS coordinates that the robots repeatedly follow, i.e., once a robot reaches the last waypoint, it goes to the first waypoint in the list and repeats its patrol. In this type of problem the robots are required to repeatedly visit nodes and if this constraint is relaxed, we can reduce this problem to m-TSP. Most relevant work in this field is listed below.

- [Portugal and Rocha, 2011] presents a survey on cooperative multi-robot patrolling algorithms. The various strategies proposed is normally based on operational research methods, simple and classic techniques for robot's coordination. The variety of approaches differs in various aspects such as robot type and their decision-making or the coordination and communication mechanisms.

- [Agmon et al., 2011] discusses *multi-robot adversarial patrolling* in which the robots are required to repeatedly visit a target area in a way that maximizes their chances of detecting an adversary trying to penetrate through the patrol path. When facing a strong adversary that knows the patrol strategy of the robots, if the robots use a deterministic patrol algorithm, then in many cases it is easy for the adversary to penetrate undetected (in fact, in some of those cases the adversary can

guarantee penetration). Therefore this paper presents a non-deterministic patrol framework for the robots. A polynomial-time algorithm for determining an optimal patrol under the *markovian strategy* assumption for the robots, such that the probability of detecting the adversary in the patrol's weakest spot is maximized. Since this algorithm uses a probabilistic approach for finding any adversaries, there may be the cases where we may not end up find adversaries all the time. **This approach cannot be used to solve our problem because** these techniques allow robots to take up a virtual area to patrol and these robots cover this virtual area repeatedly. No efforts are taken to reduce the costs of this patrol and also no approximation ratio is found using this approach.

2.2 Multi-Robot Coverage Problem

'Multi-Robot Coverage Problem' (MRCP) is a modern problem, on how to make the coverage of some territory in the best way, using the multiple robots. The most prominent work done in this area are listed below.

- [Kong et al., 2006] discusses the *distributed coverage with multi-robot system*. This algorithm builds on a single robot coverage algorithm using *boustrophedon decomposition.* The robots are initially distributed through space and each robot is allocated a virtually bounded area to cover. The area is decomposed into cells where each cell width is fixed. The decomposed area is represented using an adjacency graph, which is incrementally constructed and shared among all the robots. Communication between the robots is available without any restrictions. The drawback of this paper comes from the fact that designated areas of search are assigned to for every robot and since they use the sweeping algorithm to find the next cell based on utility function, the idea is to cover the most mines prone cell first but they don't make ant attempt to minimize the time taken for coverage.

- [Sheng et al., 2006] presents a multi-robot coordination algorithm to accomplish an area exploration task using limited communication range. This algorithm is

based on a distributed bidding model and map synchronization mechanism. The distributed bidding model is done within the subnet a robot is in and this ensures that the robots stay close to each other. When robots go out of range, the robots share map information when they come within range of each other. The map synchronization module ensures only vital parts of the map are shared between robots and not the entire map. Each robot maintains two maps, a row map table, and a local map. The raw map table consists of all information gathered by all robots and it could overlap. The local map is derived from the raw map using a map fusion mechanism. Each robot acts in an asynchronous manner.

- [Rekleitis et al., 2004] presents *multi-robot team based coverage* with limited communication (line of sight only). The environment is not known in priori but it is assumed to be a static environment. There are two types of agents, *explorers* and *coverers. Explorers* are the two robots tracking endpoints of a slice as it is swept through the free space. Rest of the robots are called *coverers.* This algorithm works on two main ideas. First during the coverage of a single cell, boundaries of a cell are covered by two robots until they are no longer in sight of each other. The *explorer* robots use a break in a line of sight to detect critical points and thus the termination of the cell. Next, to avoid redundant coverage, the teams of robots are divided into two sub-teams once.

- [Zelinsky, 1992] presents an algorithm for path planning to a goal with a mobile robot in an unknown environment. The robot maps the environment only to the extent necessary to achieve the goal. Paths are generated by treating unknown regions in the environment as free space. As obstacles are found, the environment is updated and a new path to the goal is planned and executed. This algorithm uses the distance transform methodology to generate paths for the robot to execute.

This approach cannot be used to solve our problem because these techniques allow robots to take up a virtual area to cover and uses path planning techniques to cover the virtual area and there are no constraints for the robots to fulfill. Mostly these techniques

come close to solving the *multi-depot VRP*.

2.3 Swarm Intelligence

Swarm intelligence according to [Dorigo and Birattari, 2007] " is the discipline that deals with natural and artificial systems composed of many individuals that coordinate using decentralized control and self-organization". The primary focus of this field is to find the collective behavior that results from the local interaction of the individuals in a swarm with each other and the environment. Examples of systems studied by swarm intelligence are colonies of ants and termites, schools of fish, flocks of birds, herds of land animals. The characterizing property of a swarm intelligence system is its decentralized control i.e., the ability to act in a coordinated way without the presence of a coordinator or of an external controller. Many examples can be observed in nature of swarms that perform some collective behavior without any individual controlling the group, or being aware of the overall group behavior.

The two most famous algorithms that uses *swarm intelligence* techniques are *ant colony optimization* (ACO) and *particle swarm optimization* (PSO) respectively.

2.3.1 Ant Colony Optimization (ACO)

Ant colony optimization according to [Dorigo, 2007] " is a population-based meta-heuristic that can be used to find approximate solutions to difficult optimization problems". In ant colony optimization (ACO), a set of software agents called " artificial ants" search for good solutions to a given optimization problem transformed into the problem of finding the minimum cost path on a weighted graph. The artificial ants incrementally build solutions by moving along the graph. The solution construction process is stochastic and is based by a *pheromone model*, that is, a set of parameters associated with graph components (either nodes or edges) whose values are modified at runtime by the ants. ACO has been applied successfully to many classical combinatorial optimization problems, as well as to discrete optimization problems that have stochastic and/or dynamic compo-

nents. The ACO can be applied to various optimization problems. The most interesting and relevant ones are as follows.

- Traveling Salesman Problem. The traveling salesman problem (TSP) can be defined as " Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? ". The process of solving TSP using ACO techniques was shown by [Dorigo et al., 2006].

- m-TSP. According to [Bektas, 2006], " The multiple traveling salesman problem (mTSP) is a generalization of the well-known traveling salesman problem (TSP), where more than one salesman is allowed to be used in the solution". The m-TSP problem can be solved by using ACO techniques as shown by [Junjie and Dingwei, 2006].

- Open VRP. [Li et al., 2009] has shown how the *open VRP* can be solved using a combination of ACO techniques and tabu search techniques. The algorithm designed called the ACO_{ovrp} works as follows. At each iteration first, a set of artificial ants probabilistically build the solutions, exploiting the given pheromone model. Then, the constructed solutions are improved and taken to their local optima using local search. After each ant finishes its solution construction, the information of some 'good' solutions is exploited to update the pheromone trails. The main procedures are repeated until a termination condition is met. This procedure will stop when the number of iterations reaches the maximum allowed number.

  This approach cannot be used to solve our problem because the ACO uses an iterative algorithm which uses solution that is probabilistically chosen and tries to improve them in the next iterations to obtain its best value possible. Also, this algorithm does not use a distributed algorithm to solve the *open VRP*. They solve this problem in a centralized offline manner.

2.3.2 Particle Swarm Optimization (PSO)

Particle swarm optimization (PSO) introduced by [Kennedy, 2011] is a population based stochastic optimization technique inspired by social behavior of bird flocking or fish schooling.

The system is initialized with a population of random solutions and searches for optima by updating generations. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles.

Each particle keeps track of its coordinates in the problem space which is associated with the best solution (fitness) it has achieved so far. This value is called "pbest". Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighbors of the particle. This location is called "lbest". When a particle takes all the population as its topological neighbors, the best value is a global best and is called " gbest ".

The particle swarm optimization concept consists of, at each time step, changing the velocity of (accelerating) each particle toward its pbest and lbest locations (local version of PSO). Like ACO, PSO can be applied to various optimization problems. The most interesting and relevant ones are as follows.

- TSP. The TSP problem can be solved by using PSO techniques as shown by [Shi et al., 2007].

- mTSP. The m-TSP problem can be solved by using PSO techniques as shown by in [Pang et al., 2013].

- Open VRP. Solving the *open VRP* using PCO techniques was proposed by [MirHassani and Abolghasemi, 2011]. Initially, P particles are randomized with a position and velocity. Decode these particles using the decoding algorithm and then evaluate and improve the fitness value of the initial solution to reach the pbest value. Then these details are shared with others, to improve the fitness value to obtain a value closer to gbest. This process is continued until the termination condition occurs.

This approach cannot be used to solve our problem because this approach like the ACO uses an iterative algorithm which uses particles i.e., random solution and tries to improve them in the next iterations. Also, this algorithm does not use a distributed algorithm to solve the *open VRP*. They solve this problem in a centralized offline manner.

Next, we list some of the prominent work done in the area of *open VRP*.

2.4 Cluster First,Route Second (CFRS)

This approach was introduced by [Sariklis and Powell, 2000] to solve the *open VRP* problem based on a *minimum spanning tree with penalties procedure*. This algorithm works in two phases

- Clustering. Clustering procedure has two stages. The first stage initially assigns customers to clusters satisfying the capacity constraint of all vehicles and aims to have as few clusters as possible. In the second stage, the authors attempt to improve the quality of the clusters formed by reassigning customers among them.

- Routing. In the second phase, the authors generate open routes by solving a *minimum spanning tree problem* (MSTP). They use penalties to modify the MSTP solution and iteratively convert infeasible solutions to feasible solutions.

This approach cannot be used to solve our problem because of the following reasons.

- Firstly, this algorithm uses repeated swaps among candidate solutions to guarantee an optimal solution.

- Secondly, they do not use a distributed setting to solve this problem.

- Thirdly, This approach uses backtracking techniques in their second stage, wherein they use penalties to modifying the ordering of visits of customers, which is not true in our case.

2.5 Tabu Search Algorithm (TSA)

Tabu search, created by [Glover, 1986] and formalized in 1989 is a meta-heuristic search method employing local search methods used for mathematical optimization. It allows *local search* (LS) methods to overcome local optima. The basic principle of TS is to pursue LS whenever it encounters a local optimum by not allowing non-improving moves i.e., moving back to previously visited solutions are prevented by the use of memories called tabu lists, that record the recent history of the search. It starts from a random initial solution and successively moves to one of the neighbors of the current solution. Tabu search uses a special short term memory that is composed of previously visited solutions that include prohibited moves. So it gives no permission to revisited solutions and then avoids cycling and being stuck in local optima. During the local search, only those moves that are not tabu will be examined if the tabu move does not satisfy the predefined aspiration criteria. A common aspiration criterion is better fitness, i.e., the tabu status of a move in the tabu list is overridden if the move produces a better solution. [Brandão, 2004] solves the *open VRP* using TSA. The author generates an initial solution using a variety of methods including the nearest neighbour heuristic and the k-tree method. The initial solution is submitted to either a nearest neighbour method or an unstringing and stringing procedure to improve each route. In the tabu search algorithm, there are only two types of trial moves.

- an insert move. This takes a customer from one route and insert it onto another route
- a swap move. This exchanges two customers on two different routes. This process is repeated until the best possible cost is reached.

This approach cannot be used to solve our problem because of the following reasons.

- First of all, this approach does not use a distributed algorithm.
- In our problem, we do not allow vehicles to move back to their previous positions to make improvements to their paths.

- Thirdly, this approach assign initial values to vehicles and processing is done on the initial solution to get the best possible value. This technique cannot be used in our approach.

2.6 Tabu Search Heuristic (TS)

Tabu search is a heuristic method designed to guide methods like local search algorithm, the escape local optima. The basic idea of the heuristic is to begin with an initial solution and at each iteration, a neighborhood of the current solution is created through a different class of moves. Then, the best admissible solution in the neighborhood is selected as the new current solution, then the procedure is repeated until a stopping criterion is satisfied. A move is admissible if it is not in tabu. If it is in tabu, then it still may be admissible if the move produces a solution strictly better than the best solution so far. [Fu et al., 2005] designed a *farthest first heuristic* (FFH) algorithm to solve the *open VRP*. The FFH algorithm starts a new route with the farthest un-routed customer from the depot and tries to add customers to the route until the vehicle is sufficiently full. In the tabu search heuristic, two different nodes (customer or depot, on the same route or different routes) are selected at random and one of four types of neighborhood moves is performed at random.

- node reassignment. Removes the first selected vertex from its current position on the route and insert it to a position before the second selected vertex.
- node swap. Exchanges the positions of the two selected vertices.
- two-opt move. Reverses the order of all the elements between the two selected vertices.
- tails swap. select two customers and swap the tails, that is, perform an exchange from the customer to the end of the route.

The distinctive feature of the TS heuristic is that it uses a simple but powerful neighborhood structure. During the search, the current neighborhood is randomly selected

among four types of neighborhood move, and the tabu length is randomly selected to be between 5-10. By applying a different transformation at each iteration, a larger neighborhood is explored over a few iterations, without the computational burden associated with an extensive search in a unified neighborhood. Simulation results show that there is a strong relationship between the initial solution and the final results. The better the chosen initial solution, the better the chances of TS heuristic returning better final costs.
This approach cannot be used to solve our problem because of the following reasons.

- This approach uses a randomly selected neighborhood technique an also uses random moves for improvement. Hence this cannot be used to prove any bounds or solve the OMDVRP in a distributed setting.

- The results generated is directly proportional to the initial solution. This is more closed to a randomized solution.

2.7 Adaptive Memory Based Tabu Search (BR)

Tabu search (TS) is based on the premise of problem-solving that in order to qualify as intelligent, we must incorporate adaptive memory and responsive exploration. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. The tabu search emphasis on adaptive memory makes it possible to exploit the types of strategies that underlie the best of human problem-solving. A key element of the adaptive memory framework of tabu search is to create a balance between search intensification and diversification. Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Diversification strategies, on the other hand, seek to incorporate new attributes and attribute combinations that were not included within solutions previously generated.
The author [Tarantilis et al., 2004] 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. The components of routes used in the new algorithm are sequences of nodes called bones. The generation of a new solution(routing plan) by BR algorithm is based on the adaptive memory concept. It extracts a sequence of points (called bones) from a set of solutions and generates a route using adaptive memory. If a large number of routes in the set of solutions contains a specific bone, then the authors argue that this bone should be included in a route that appears in a high-quality solution. The *bone route* algorithm has two phases.

- In Phase 1 (pool generation phase).

    1. Generate L Fl solutions;
    2. Improve the quality of L initial solutions by tabu search algorithm;
    3. Insert the routes composing the L solutions in a pool of solutions and
    4. Sort the routes by increasing costs solutions where they belong to.

- In Phase 2 (pool exploitation phase).

    1. promising bones are extracted;
    2. a solution is generated and improved using tabu search and
    3. and the set of solutions is updated.

This approach cannot be used to solve our problem because of the following reasons.

- It uses memory techniques to solve the problem efficiently which is not true in our case. Our problem does not need to use memory techniques.

- Secondly, we cannot prove an approximation ratio of the algorithm using this approach.

2.8 Adaptive Large Neighborhood Search (ALNS)

ALNS is a local search framework in which a number of simple algorithms compete to modify the current solution. In each iteration, an algorithm is chosen to destroy the current solution, and an algorithm is chosen to repair the solution. The new solution is accepted if it satisfies some criteria defined by the local search framework applied at the master level.

In the ALNS framework proposed in [Pisinger and Ropke, 2007], a feasible solution is constructed and then modified. In each iteration, an algorithm is selected to "destroy the current solution and an algorithm is selected to "repair" the solution. An adaptive layer stochastically controls which neighborhoods to choose according to their past performance (score). The more a neighborhood N_i has contributed to the solution process, the larger score π_i it obtains, and hence it has a larger probability of being chosen.

For example, customers can be removed at random from the solution and then reinserted in the cheapest possible route. Several removals and insertion heuristics can be used to diversify and intensify the search. The new solution is accepted if it satisfies the criteria defined by the local search procedure (the authors use simulated annealing).

This approach cannot be used to solve our problem because

- First of all, this approach does not use a distributed algorithm.
- It uses iterative improvements of the solution which cannot be true in a distributed setting, where a vehicle has to make a decision on the go.

Next, we discuss some of the prominent work done in the field of *open multi-depot VRP*

- [Tarantilis and Kiranoudis, 2002] solves the OMDVRP to solve 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) to solve the problem. The LBTA algorithm iteratively searches the solution space guided by a deterministic control parameter called threshold. This

value reveals promising regions for better configurations. The LBTA runs in two phases. The steps involved are given below.

1. Phase 1.
   - Step 1. Initialization. An initial solution is produced by allocating customers to the closest distribution center criss-cross and secondly dispatching to each of them a vehicle.
   - Step 2. A local search is conducted in order to compute the threshold values that represent the list of the algorithm. Local search is blended with a list of moves listed below.
     * 2-Opt move. This is implemented in the case of a single route and multiple routes. This move helps to avoid criss-cross between two edges in a route. It also helps to avoid criss-cross among two different routes.
     * 1-1 exchange move. The 1-1 exchange move swaps two nodes from the same route. In a case of multiple routes, the swapping of nodes takes place between two different routes.
     * 1-0 exchange move. The 1-0 exchange move transfers a node from its position in one route to another position in either in the same route or a different route.

     The type of move is selected employing a stochastic move generation algorithm where the selection follows a uniform distribution. The relative threshold is calculated between the proposed and current solution is calculated. If the threshold is positive and lower than the maximum element in the list, the threshold is inserted in the list. This iterative approach is repeated for each of the moves.
2. Phase 2. The phase 2 of the algorithm works just like phase 1 but in this step, ways are given to escape local minimum points from the exchanges. The algorithm keeps track of values of high list threshold values using which bad

moves are avoided.

- [Habib et al., 2013] solves the OMDVRP problem using UAVs. In the paper, the authors design a path planning algorithm for UAVs using Mixed Integer Linear programming (MILP). There is cooperation among team members as it is helpful to reduce total mission time. The MILP solution gives the UAVs points where they need to go and due to the un-foreseen disturbance in the environments, the UAVS may be forced to apply the path planning again to complete their graph. The complexity of the path planning algorithm is reduced as most of the nodes have been covered and the path planning needs to cover only the uncovered nodes. Then using simulation the algorithm is run on a static and dynamic environments.

This approach does not work for us because we run our distributed algorithm on an instance of the OMDVRP and find the results but the process followed by Tarantilis's paper allows the vehicles to make decisions based on previous results of searches and findings. So this cannot be used in our case. This technique cannot be used to find any bounds in a distributed setting.

2.9 Distributed Approximation

- [Elkin, 2004] discusses the techniques and hardships involved in finding the distributed approximation of various problems. It discusses how to find the lower bound, upper bound in a distributed setting. Also, the rounding algorithm is discussed in detail.

- [Kuhn and Wattenhofer, 2005] presents a new fully distributed approximation algorithm based on LP relaxation techniques for finding the *distributed dominating set*. For an arbitrary parameter k and maximum degree Δ, their algorithm computes a dominating set of expected size $O(k\Delta^{2/k}log\Delta \mid DSOPT \mid)$ in $O(k^2)$ rounds where each node has to send $O(k^2\Delta)$ messages of size $O(log\Delta)$

- [Moscibroda and Wattenhofer, 2005b] finds the approximation ratio of the facility

location problem in a distributed setting. Their algorithm achieves a approximation ratio of $O(\sqrt{k}(m\rho)^{1/\sqrt{k}}log(m+n))$ approximation in $O(k)$ communication rounds where message size is bounded to $O(logn)$ bits. The number of facilities and clients is *m* and *n* respectively, and ρ is a coefficient that depends on the cost values of the instance. Their technique is based on a distributed primal-dual approach for approximating a linear program, that does not form a covering or packing program.

This technique is the one that we are going to use to solve the OMDVRP in a distributed fashion and compute the approximation ratio.

Chapter 3

The Vehicle Routing Problem and Linear Program Transformations

In this section, we present the *vehicle routing problem* (VRP) and its formal description in Section 3.1. Then, Section 3.2 discusses the ILP formulation of the problem. We also present the LP relaxation and the dual LP formulations in Sections 3.3 and 3.4, respectively.

3.1 Problem Definition

VRP has numerous variations. However, its most basic version can be described as follows. According to Laporte [Laporte, 1992]:

> *" The Vehicle Routing Problem (VRP) can be described as the problem of designing optimal delivery or collection routes from one or several depots to a number of geographically scattered cities."*

The variation of VRP we are studying in this thesis is called the *open multi-depot vehicle routing problem* (OMDVRP), wherein a vehicle starts at any one of the depots and are not required to return to any of the depots after servicing the last node in the route. In our version of the problem, we assume that all cities have a constant non-negative demand. We also assume that there is no priority and/or precedence in servicing

the cities. All vehicles may or may not have same servicing and traveling costs. There are also some side constraints that need to be satisfied:

- Demands of all cities including the depots are to be serviced by exactly one vehicle;
- A vehicle has to reach the city before satisfying its demand unless the vehicle is already present in the city (depot), and
- A vehicle can visit a city only once in its entire route. Multiple vehicles can visit the same city if it helps improve the costs.

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 vertex v_0^a called the *depot*. The depot also needs to be serviced. We consider two cost functions:

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

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

- The second is the cost of *traveling* from one city 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 $v_0 = v_0^a$. Let V_a^p be the set of all nodes serviced by a vehicle $a \in A$ in its path p, where $V_a^p \subseteq V$. The total cost of a path $p = v_1 \cdots v_n$ chosen by a vehicle $a \in A$ (denoted $C(a, p)$) is

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

Our version of 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,

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

- And, F respects the following objective

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

  among all possible such functions.

3.2 Transformation to ILP

We now explain the transformation of our version of VRP to integer linear programming. An integer linear program 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 3.1. We now identify the constants, variables, constraints, and optimization objective.

Constants.

- $C_s(a, v)$ is the cost of servicing city $v \in V$ by vehicle $a \in A$. Thus, $C_s(a, v) \geq 0$.
- $C_t(a, (u, v))$ is the cost of travel from city u to city v by vehicle a. Thus, $C_t(a, (u, v)) \geq 0$.

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 $x^a_{(u,v)}$, where
$$x^a_{(u,v)} = \begin{cases} 1 & \text{if vehicle } a \text{ travels from city } u \text{ to city } v \\ 0 & \text{otherwise} \end{cases}$$
- For each vertex $v \in V$ and vehicle $a \in A$, we introduce a binary variable y^a_v, where
$$y^a_v = \begin{cases} 1 & \text{if vehicle } a \text{ services city } v \\ 0 & \text{otherwise} \end{cases}$$
- Finally, for each edge $(u, v) \in E$ and vehicle $a \in A$, we include two integer variables s^a_u, and s^a_v. These auxilliary variables are used for sub-tour elimination constraints (explained below).

Constraints. The set of constrains are as follows:

- All variables are binary. That is, for all $a \in A$ and $(u, v) \in E$, we include:
$$x^a_{(u,v)}, y^a_v \in \{0, 1\} \tag{3.1}$$
- Each vertex is serviced exactly once. That is, for each vertex $v \in V$, we include the following:
$$\sum_{a \in A} y^a_v = 1 \tag{3.2}$$

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

$$\sum_{u \in V} x_{(u,v)}^a - y_v^a \geq 0 \tag{3.3}$$

- We need to ensure that a path enters each node at most once. That is, for each vehicle $a \in A$ and each vertex $v \in V$, we include the following constraint:

$$\sum_{u \in V} x_{(u,v)}^a \leq 1 \tag{3.4}$$

- We need to ensure that a vehicle does not go through a vertex more than once. That is, for each $a \in A$, we include the following:

$$\sum_{v \in V} x_{(u,v)}^a \leq 1 \tag{3.5}$$

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

$$\sum_{u \in V} x_{(u,v)}^a - \sum_{u \in V} x_{(v,u)}^a \geq 0 \tag{3.6}$$

- 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 the vehicle to have a continuous path without any cycles. This constraint is known as *subtour elimination*: for each vehicle $a \in A$ and edge $(u, v) \in E$, where $(u, v) \neq v_0^a$:

$$s_u^a - s_v^a + (n - m) \cdot x_{(u,v)}^a \leq (n - m - 1) \tag{3.7}$$

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

Example a) This constraint associates a positive value s_u^a, s_v^a to all nodes chosen in a path in the order they are traveled and this constraint does not allow a vehicle to choose a path with cycles. In a small graph having 4 nodes, if a vehicle a chooses a path $\langle v_0 v_1 v_2 v_3 \rangle$, where $v_0 = v_0^a$. The corresponding values associated with $v_1 = 0, v_2 = 1, v_3 = 2$. By substituting these values we can see that a vehicle decides to take an edge to reach any previously visited node in its path, the subtour elimination condition does not hold.

Optimization objective.

$$\min \left(\sum_{a \in A} \sum_{v \in V} y_v^a \cdot C_s(a, v) + \sum_{a \in A} \sum_{(u,v) \in E} x_{(u,v)}^a \cdot C_t(a, (u, v)) \right) \tag{3.8}$$

3.3 LP Relaxation

In this section, we show how our ILP constraints are relaxed to form a linear program (LP) constraints. Since LP allows fractional values for $x_{(u,v)}^a, y_v^a$, we cannot use some of the ILP constraints as stated above, since the solution returned from LP may not be feasible for ILP. Hence, some of the constraints are divided by a factor of $m = |A|$. The objective function of LP relaxation remains the same as the objective function of ILP.

The set of constraints are as follows:

- We change Constraint 3.1 to:

$$x_{(u,v)}^a, y_v^a \in \mathbb{R}_{\geq 0} \tag{3.9}$$

- We modify Constraint 3.2 to:

$$\sum_{a \in A} y_v^a = \frac{1}{m} \tag{3.10}$$

- We modify Constraint 3.4 to:

$$\sum_{u \in V} x_{(u,v)}^a \leq \frac{1}{m} \tag{3.11}$$

- We modify Constraint 3.5 to:

$$\sum_{v \in V} x^a_{(u,v)} \leq \frac{1}{m} \tag{3.12}$$

- We modify Constraint 3.7 to:

$$s^a_u - s^a_v + (n-m) \cdot x^a_{(u,v)} \leq \frac{1}{m} \cdot (n-m-1) \tag{3.13}$$

Since we have reduced the value of $x^a_{(u,v)}$ by $\frac{1}{m}$ and also $(n-m-1)$ is reduced by a factor of $\frac{1}{m}$, we need not reduce the s^a_u, s^a_v variables. The subtour elimination constraints will still hold in this case.

Example b) In a small graph having 3 nodes and 2 vehicles. If a vehicle a, chooses a path $\langle v_0 v_1 v_2 \rangle$, then the associated s values are $v_1 = 0, v_2 = 1$. Now if vehicle *a* from v_2 tries to go back to v_1, the subtour constraints would not allow that visit. Here we have $m = 2$. Substituting these values we have for the case of vehicle a going from v_2 to v_1 we have that $1 - 0 + 0.5 \nleq 0$.

3.4 The Dual Linear Program

The motivation behind using a *dual* LP ([Bachem and Kern, 1992]) 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 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.

Given the primal LP defined in Section 3.3, the associated dual LP is the following:

Variables. The set of variables are the following:

- Variable α_v represents Constraint 3.10.
- Variable β_v^a represents Constraint 3.3.
- Variable γ_v^a represents Constraint 3.11.
- Variable τ_u^a represents Constraint 3.12.
- Variable θ_v^a represents Constraint 3.6.
- Variable $\lambda_{u,v}^a$ represents Constraint 3.13.

Constraints. The set of variables are the following:

- Constraint 1: For each $v \in V$ and $a \in A$, we have

$$y_v^a : \alpha_v - \beta_v^a \leq C_s(a, v) \tag{3.14}$$

- Constraint 2: For each $a \in A$ and $(u, v) \in E$

$$x_{(u,v)}^a : \beta_v^a - \gamma_v^a - \tau_u^a - \lambda_{(u,v)}^a + \theta_v^a \leq C_t(a, (u, v)) \tag{3.15}$$

-

$$\alpha_v \text{ is free}, \beta_v^a, \gamma_v^a, \tau_u^a, \lambda_{(u,v)}^a, \theta_v^a \geq 0 \tag{3.16}$$

Optimization objective.

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

In the next chapter, we will present an algorithm that solves the primal LP relaxation and we will provide bounds by use of the dual program.

Chapter 4

Algorithm for Distributed VRP with Fractional Values (Distributed Approximation Algorithm)

Our goal in this chapter is to design an algorithm, where a set of distributed vehicles solve the *open multi-depot vehicle routing problem* (OMDVRP) as formalized in Section 3.1. We will use the LP relaxation and dual LP constraints given in the last chapter and use it to design a *synchronous* distributed algorithm to solve the *open multi-depot vehicle routing problem* (OMDVRP). By incorporating the LP relaxation techniques, we are able to find the approximation ratio of how good our algorithm can perform when compared to the optimal solution from the ILP formulation.

The basic idea of designing our algorithm in this fashion is to prove that in every iteration of the distributed algorithm, all the LP relaxation constraints are satisfied and also the dual constraints are satisfied through out the algorithm. By ensuring this, we are able to find how much our algorithm solution deviates from the optimal solution, which is our approximation ratio.

4.1 Model of Distributed Computation

Given a set A of vehicles and a graph G, our distributed model of computation has the following characteristics:

- The set of distributed vehicles can communicate using *synchronous* [Lynch, 1996] message passing i.e., in each round vehicles send and receive messages synchronously and subsequently engage in internal computation;
- The communication network is a complete graph i.e., each vehicle can communicate with other vehicles regardless of their location in G and they are able to send broadcast messages;
- Each vehicle has a unique *id* and knows the entire graph G and cost functions C_s and C_t of all vehicles and

4.2 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 city:

- A vehicle can decide to service the city if the vehicle finds that it can service the city cheaper (with lesser cost) than any rival vehicle or in fewer communication rounds compared to all other rival vehicles from their current location.

- Otherwise, the vehicle decides to *travel* to a city, from its list of neighboring nodes where it can reach and *service* a city with the least cost possible. A vehicle travels to another city, if it finds a rival vehicle can *service* the city cheaper than the current vehicle 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 city) and the conditions when the vehicle decides to *travel* to another node in Subsections 4.2.1 and Subsection 4.2.2, respectively.

4.2.1 Conditions for Servicing

Step 1 – Decide on vehicle's neighborhood from current location:

Let $loc(a)$ denote the current location of vehicle a. The *neighborhood* for a vehicle from $loc(a)$ (if not serviced) is the set of all simple paths where all the vertices are not yet *serviced* by any vehicle and *traveled* by the current vehicle. If the vehicle's current location is a vertex that has already been serviced, then the neighborhood for this vehicle from its current location is empty.

Definition 4.2.1 *Given a graph $G = \langle V, E \rangle$ and vehicle $a \in A$, the* neighborhood *of a is the following set of simple paths:*

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

where V^p denotes the vertices of path p and $V_p^{\neg s}$ is the set of vertices 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, or, the value of the k, based on the smallest value among the two.

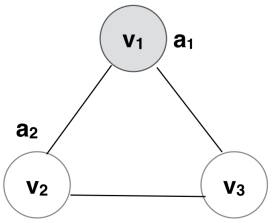


Figure 4.1: Computing the Dynamic Neighborhood (Example 1)

Definition 4.2.2 *Given a graph $G = \langle V, E \rangle$ 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 vertex u from vertex v for any $u, v \in V$ and $|p|$ is the length of path p.

Notice that in Definition 4.2.1, if there exists $a' \in Rival_a$ such that a and a' are in the same vertex, then $N_k^a = \{loc(a)\}$.

All concepts discussed above can be shown using the following examples:

- **Example 1.** In this example we have a graph having three nodes and two vehicles a_1 and a_2 which are placed at vertices v_1 and v_2 respectively as shown in Figure 4.1. Vertex v_1 has been serviced and is denoted by the shaded node. 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 this example we have a graph having three nodes and two vehicles a_1 and a_2 which are placed at vertices v_1, v_2 respectively as shown in Figure 4.2. In this case for $k = 2$, we have a $d(loc(a_1), loc(a_2)) = 1$, $N_2^{a_1} = \{v_1\}$ and $N_2^{a_2} = \{v_2\}$.

- **Example 3.** In this example we have a graph having three nodes and two vehicles a_1 and a_2 and both vehicles are placed at vertex v_1 as shown below in Figure 4.3. In this case we have $k = 2$, and both vehicles a_1 and a_2 have the same location i.e., vertex v_1. Hence, we have $N_2^{a_1} = \{v_1\}$ and $N_2^{a_2} = \{v_1\}$.

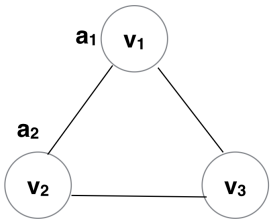


Figure 4.2: Computing the Dynamic Neighborhood (Example 2)

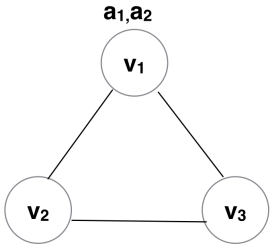


Figure 4.3: Computing the Dynamic Neighborhood (Example 3)

- **Example 4.** In this example, we have a graph having four nodes and two vehicles a_1 and a_2 which are placed at vertices v_1 and v_3 respectively as shown in Figure 4.4. In this case,we take $k = 2$ and $d(loc(a_1), loc(a_2)) = 2$. Hence, 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 possible path from N_k^a:

After we compute N_k^a, a vehicle should decide which path it will take 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:

$$C(N_k^a) = \min \left\{ \frac{\sum_{v \in p} C_s(a, v) + \sum_{(u,v) \in p} C_t(a, (u, v))}{2|p| + 1} \;\middle|\; p \in N_k^a \right\} \tag{4.1}$$

We present an example to show how $C(N_k^a)$ is calculated. In the graph shown in Figure 4.5, we have four nodes and two vehicles with identical *service* and *travel* costs (each edges is labeled by it *travel* cost and *service* costs are shown above vertices). We take $k = 3$ here and vehicle a_1 has its depot at v_1 and a_2 has its depot at v_3.

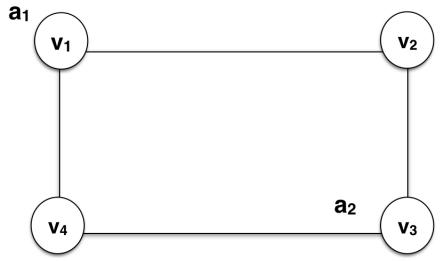


Figure 4.4: Computing the Dynamic Neighborhood (Example 4)

For vehicle a_1, we have $N_3^{a_1} = \{v_1, v_1v_2, v_1v_4\}$ and

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

We denote the path that yields $C(N_k^a)$ by $\pi(N_k^a)$. 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, $\pi(N_3^{a_2}) = v_3v_4$.

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

Each vehicle needs to check if it takes fewer number of communication rounds to *service* all the nodes in the path than other vehicles from their locations. This check is done using the function CheckIfPathFree (see Algorithm 1). Before the call to the function, the value of $\pi(N_k^a)$ is assigned to p' and the function is called with the value of p'. The function

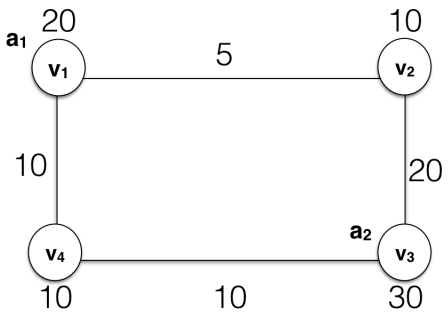


Figure 4.5: Choosing the best path from N_k^a

Algorithm 1 Function CheckIfPathFree for vehicle a.

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

```
1:  if (|p| = 0) then
2:     p' ← p;
3:     if (loc(a) = loc(a')) ∧ (C_s(a, loc(a)) > C_s(a', loc(a)) ∨
4:                              (C_s(a, loc(a)) = C_s(a', loc(a)) ∧ id(a) > id(a'))) then
5:        p' ← ∅;
6:        return p';
7:     end if
8:  else
9:     for (i ← 0; i ≤ n; i++) do
10:       if (2.d(loc(a), v_i) + 1) < d(loc(a'), v_i) then
11:          p' ← p'.⟨v_i⟩;
12:       else
13:          return p';
14:       end if
15:    end for
16: end if
17: return p';
```

CheckIfPathFree takes two arguments as input and returns a path p' or its subset for "success" or returns an $\emptyset$ denoting "failure". The first argument passed to the function is the path chosen by the current vehicle i.e., p' and the second argument passed, is the location of a rival vehicle. If CheckIfPathFree returns p' or its subset, it ensures that no rival vehicle can enter the path $\pi(N_k^a)$ in less than or equal to $2|p'| + 1$ rounds of communication (we will describe how the vehicles communicate later). Note that CheckIfPathFree is called for every rival vehicle of the current vehicle a. This is done to ensure that when a vehicle chooses a path p', the vehicle has the freedom to *service* all the nodes by itself and does not allow any other rival vehicle to enter its computed path.

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.

Functionality of CheckIfPathFree can be broken down into two steps. Let a denote the current vehicle, a' denote a rival vehicle and $\pi(N_k^a)$ be the path chosen by a.

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'.

Note: If CheckIfPathFree returns a path $|p'| < |\pi(N_k^a)|$, the subsequent function calls to CheckIfPathFree is made with the updated path. We now present an example. Figure 4.6 shows a graph of eight nodes and three vehicles a_1, a_2, and a_3 with depots v_1, v_2, and v_5, respectively and 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:

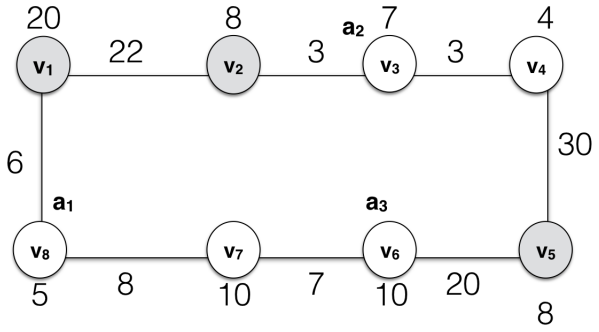


Figure 4.6: CheckIfPathFree: Example 1

- **Vehicle a_1:** Vehicle a_1 assigns the value of $\pi(N_2^{a_1})$ to p' and calls the function CheckIfPathFree with parameters $p' = v_8$ and $loc(a_2) = v_3$. Since $loc(a_1) \neq loc(a_2)$, the function returns v_8. Next, vehicle a_1 calls the function with parameters

$p' = v_8$ and $loc(a_3) = v_6$. Since $loc(a_1) \neq loc(a_3)$, the function again returns v_8. Hence, vehicle a_1 can *service* its node v_8.

- **Vehicle a_2:** Vehicle a_2 assigns the value of $\pi(N_2^{a_2})$ to p' and calls the function CheckIfPathFree with parameters $p' = v_3v_4$ and $loc(a_1) = v_8$. Vehicle a_2 can *service* node v_3 in fewer rounds than its rival vehicles a_1, a_3. Next, vehicle a_2 takes $2|p'| + 1 = 3$ communication rounds to *service* node v_4. Notice that since a_1 has traveled from v_1 to v_8 and is not allowed to *travel* back to v_8, then 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 $d(loc(a_1), v_4) > 2|p'| + 1$. Hence, the function returns v_3v_4. Next, vehicle a_2 calls the function with parameters $p' = v_3v_4$ and $loc(a_3) = v_6$. 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 $d(loc(a_3), v_4) > 2|p'| + 1$. So, the function returns v_3v_4. Hence, vehicle a_2 can *service* the chosen path $p' = v_3v_4$.

- **Vehicle a_3:** Vehicle a_3 assigns the value of $\pi(N_2^{a_3})$ to p' and calls the function CheckIfPathFree with parameters $p' = v_6v_7$ and $loc(a_1) = v_8$. Vehicle a_3 can *service* node v_6 in fewer rounds than its rival vehicles a_1, 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 totality, 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.

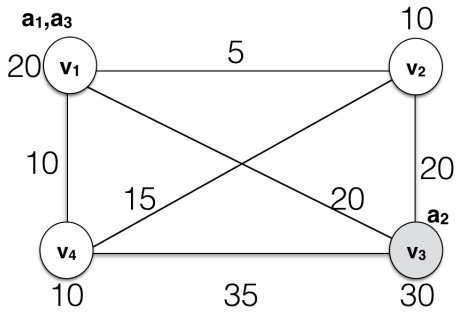


Figure 4.7: Travel Based on $nextbest(a)$ value

4.2.2 Conditions for Traveling

If any of the conditions given above is not satisfied, then a vehicle travels from the 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 that 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 have already been serviced, the vehicle temporarily assigns a large value as the cost of servicing that node. This large value can simply be defined as the following:

$$\rho = \max\left\{C_s(a,v), C_t(a,(u,v)) \mid (a \in A) \ \wedge \ (u,v \in V)\right\} \tag{4.2}$$

Next, we present an example to demonstrate this. In this complete graph we have four nodes and three vehicles with same servicing and traveling costs (see Figure 4.7). Vehicles a_1 and a_3 are placed at node v_1 and vehicle a_2 is currently placed at node v_3 which has been serviced by a_2.

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

$$nextbest(a) = \min\{C_s(a,v) + C_t(a,(loc(a),v)) \mid (loc(a),v) \in E\} \tag{4.3}$$

where v has not been traveled to or from by a. In our example, the available vertices 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 vertex 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*.

4.3 Detailed Description of the Algorithm

The technique of designing distributed approximation algorithms by first computing a fractional solution and rounding them in a second phase has been inspired by [Kuhn and Wattenhofer, 2005]. Our algorithm combines these techniques with ideas from the centralized primal-dual approach in [Jain and Vazirani, 1999]. All vehicles execute the same local algorithm. 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 i.e., the value of $|V|$. When the value of s becomes zero, it signals the termination of the algorithm.

The inner t-loop is initialized by the value of $tassign$ which gives the value 1 either if $N_k^a = \emptyset$ or $p' = \emptyset$ or else gives the value $2|p'| + 1$. Similar to the outer s-loop, the inner t-loop also is 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 decreased 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 $x^a_{(u,v)}, y^a_v, \alpha_v, \beta^a_v, \lambda^a_{(u,v)}, \gamma^a_v, \tau^a_u, \theta^a_v$ are set to zero. 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. The current vehicle starts its work from its depot location; i.e., v_0^a. From its current location, the vehicle finds the minimum of k and distance of its rival vehicles from itself using which it calculates its N_k^a. Then, $C(N_k^a)$ and $\pi(N_k^a)$ are calculated as explained in Section 4.2.1. In lines 14 – 16, a vehicle checks to see if its chosen path is free from other rival vehicles. Based on the path p' returned, the value of $tassign$ is set for the execution of the inner for loop. In the inner for-loop line 26 checks if a vehicle can *service* its chosen path. We now explain the process of *servicing* and *traveling* in two sections:

- **Service Section.** This section includes lines 27 – 41. If the condition in line 26 is satisfied, then a vehicle enters the servicing part of the algorithm. Line 29 checks if the current vehicle satisfies $y_v^a = 0$. If true, the vehicle services the node and assigns $y_v^a \leftarrow \frac{1}{m}$. 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 the 2nd LP constraint; i.e., $\sum_{u \in V} x_{(u,v)}^a - y_v^a \geq 0$. 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 the 2nd LP constraint is not true and hence the initial value of β_v^a is retained. The boolean value *flag* is used to check if a vehicle has serviced a node from its depot location.

  Next, if the algorithm from line 29 returns false, it means that vehicle a has already serviced its current location and executes lines 31 - 40. In these lines, the vehicle moves from its current location to the next node in its chosen path. In this process the value of $x_{(u,v)}^a$ is increased by $\frac{1}{m}$ to denote an edge has been taken. The corre-

sponding γ_v and τ_u are increased by the value of $\Delta x^a_{(u,v)}$ and $-\Delta x^a_{(u,v)}$, respectively. These values are incremented in an equal and opposite way to denote an outgoing edge from a node u to an another node v can also be seen as an incoming edge for v from u. Hence, these values are equal and opposite. Lines 36 , 47 are never executed as N^a_k would not contain already visited edges and also $nextbest(a)$ would not allow the vehicle to *travel* to a node that is already visited. The maximum cost of $C_t(a,(u,v))$ can be bounded by ρ. Hence, the value β^a_v is increased by ρ. Note that $C(N^a_k)$ was normalized by a factor of $2|p'|+1$. Hence, $C(N^a_k)$ if repeatedly sent over $2|p'|+1$ communication rounds would yield the actual costs of the vehicle servicing and travel across all nodes in its path. Hence, the temporary assignment in line 39 is made to ensure $C(N^a_k)$ is send out in each communication round of the *service section* of the code.

- **Travel Section.** This section extends from line 42 to 50. If a vehicle is not allowed or cannot *service* its current location, the vehicle enters the *travel section*. In this section, the value of $nextbest(a)$ is calculated as shown in Section 4.2.2. 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 an 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 giving out details of the vehicle id, current location, cost incurred in the current communication round is sent out and received from all vehicles. The cost incurred in the current communication round (*service* or *travel* costs) is multiplied with the corresponding $x^a_{(u,v)}$ or y^a_v value increased in the current communication round and this information is passed in the broadcast message. The messages received, can be distinguished into two sets, one that contains service messages and one that contains travel messages. From the service message set, the location of all vehicles that sent a service message is updated, values of corresponding rival vehicle's y^a_v are updated, the count of these messages are taken and the cost sent by each vehicle is added up to the correspond-

ing α_v value.

The number of *service* message 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. That way, all vehicles have an updated count of the number of nodes yet to be serviced at all times during the algorithm. From the set of travel message set, 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. 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 θ^a_v is given the value $\frac{1}{m}$ in the last iteration of the s-loop (Line 75). 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 s 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.

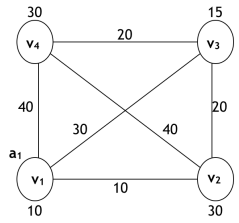
Algorithm 2 Algorithm for vehicle a

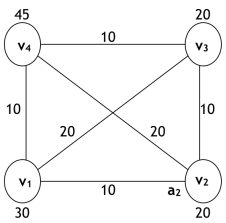
Input: Graph $G = \langle V, E \rangle$, cost functions C_s and C_t of all vehicles, and value of k.
Output: Fractional Value for x^a_{uv}, y^a_v

$x^a_{(u,v)}, y^a_v, \Delta x^a_{(u,v)}, \Delta y^a_v, \alpha_v, \beta^a_v, \Delta\alpha_v, \Delta\beta^a_v, \lambda^a_{(u,v)}, \gamma^a_v, \Delta\gamma^a_v, \tau^a_u, \Delta\tau^a_u, \theta^a_v \leftarrow 0$;
$loc(a) \leftarrow v^a_0$;
$flag \leftarrow true$;
$Rival_a \leftarrow A - \{a\}$;
Let N^a_k, $C(N^a_k)$, ρ, and $nextbest$ be as defined in Definition and Equations 4.2.2, 4.1, 4.2, and 4.3, respectively.
for $s = |V|$ **to** 0 **do**
  $tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$;
  Calculate N^a_k; {*Step 1 in Conditions For Servicing.*}
  if $(N^a_k = \emptyset)$ **then**
    $tassign \leftarrow 1$;
  else
    Calculate $C(N^a_k)$ and $\pi(N^a_k)$; {*Step 2 in Conditions For Servicing.*}
    $p' \leftarrow \pi(N^a_k)$;
    for each $a' \in Rival_a$ **do**
      $p' \leftarrow \mathsf{CheckIfPathFree}(p', loc(a'))$; {*Step 3 in Conditions For Servicing.*}
    end for
    if $(p' = \emptyset)$ **then**
      $tassign \leftarrow 1$;
    else
      $tassign \leftarrow 2|p'| + 1$;
      if $(|p'| \neq |\pi(N^a_k)|)$ **then** Calculate $C(N^a_k)$ for p';
    end if
  end if
  for $t = tassign$ **to** 0 **do**
    $isService \leftarrow false$;
    if $(p' \neq \emptyset) \ \wedge \ (N^a_k \neq \emptyset)$ **then** {*Check if Servicing conditions are met.*}
      $isService \leftarrow true$; {*If Conditions of Servicing are met, Servicing begins.*}
      $v \leftarrow loc(a)$;
      if $(y^a_v = 0)$ **then** $\Delta y^a_v \leftarrow \dfrac{1}{m}$; {*Servicing is done here.* }
        if $(flag \neq true)$ **then** $\Delta\beta^a_v \leftarrow \beta^a_v - C_s(a, v)$;
      else {*Vehicle travels to next node in path.*}
        $u \leftarrow loc(a)$;
        Let v be the node adjacent to u on p' st. $x^a_{(u,v)} = 0$;
        $isService \leftarrow false$;
        $\Delta x^a_{(u,v)} \leftarrow \dfrac{1}{m}$;
        if $(x^a_{(v,u)} > 0)$ **then** $\lambda^a_{(u,v)}$++;
        $\Delta\gamma^a_v \leftarrow \Delta x^a_{(u,v)}; \Delta\tau^a_u \leftarrow -\Delta x^a_{(u,v)}$;
        $\Delta\beta^a_v \leftarrow \rho$;
        Temporarily assign $C_t(a, (u, v)) \leftarrow C(N^a_k)$;
      end if
      $t \leftarrow t - 1$; {*Servicing ends here.*}
    else {*If Conditions for Servicing are not met, Travel starts here.*}
      Let $u \leftarrow loc(a)$;

44: Let v yield $nextbest(a)$; {*Conditions for Traveling: Calculate* $nextbest(a)$.}
45: $\Delta x^a_{(u,v)} \leftarrow \frac{1}{m}$;
46: $\Delta\beta^a_v \leftarrow \rho$;
47: **if** $(x^a_{(v,u)} > 0)$ **then** $\lambda^a_{(u,v)}$++;
48: $\Delta\gamma^a_v \leftarrow \Delta x^a_{(u,v)}; \Delta\tau^a_u \leftarrow -\Delta x^a_{(u,v)}$;
49: $t \leftarrow 0$;
50: **end if** {*Travel ends here*}
51: $\gamma^a_v \leftarrow \gamma^a_v + \Delta\gamma^a_v$;
52: $\tau^a_v \leftarrow \tau^a_v + \Delta\tau^a_v$;
53: $\beta^a_v \leftarrow \beta^a_v + \Delta\beta^a_v$;
54: $y^a_v \leftarrow y^a_v + \Delta y^a_v$;
55: $x^a_{(u,v)} \leftarrow x^a_{(u,v)} + \Delta x^a_{(u,v)}$;
56: **if** $(isService)$ **then** $\mathsf{Send}(id(a), v, \Delta y^a_v.C(N^a_k))$; {*To all vehicles*}
57: **else** $\mathsf{Send}(id(a), (u,v), \Delta x^a_{(u,v)}.C_t(a,(u,v))$; {*To all vehicles*}
58: **end if**
59: $\{(id, v, Y)\}_{A' \subseteq A},\ \{(id, (u,v), X)\}_{A'' \subseteq A} \leftarrow \mathsf{Receive}()$; {$A'$ *is the set of vehicles sent a message on Line 56 and* A'' *is the set of vehicles sent a message on Line 57*}.
60: **for each** $\bar{a} \in A'$ **do**
61: $loc(\bar{a}) \leftarrow v$;
62: **if** $(a \neq \bar{a})$ **then** $y^{\bar{a}}_v \leftarrow \frac{1}{m}$;
63: $\Delta\alpha_v \leftarrow Y$;
64: $\alpha_v \leftarrow \alpha_v + \Delta\alpha_v$;
65: $count \leftarrow count + 1$;
66: **end for**
67: **for each** $a'' \in A''$ **do**
68: $loc(a'') \leftarrow v$;
69: **if** $(a \neq a'')$ **then** $x^{a''}_{(u,v)} \leftarrow \frac{1}{m}$;
70: $\alpha_v \leftarrow X$;
71: $\alpha_v \leftarrow \alpha_v + \Delta\alpha_v$;
72: **end for**
73: **if** $(t = 0 \wedge s - count = 0)$ **then**
74: $v \leftarrow loc(a)$;
75: $\theta^a_v \leftarrow \frac{1}{m}$;
76: **end if**
77: $flag \leftarrow false$;
78: **end for**
79: $s \leftarrow s - count$; {*Value of s is decremented by count.*}
80: **end for**

We now demonstrate how the algorithm works in a step-by-step fashion using an example. We have a complete graph of four nodes and two vehicles a_1 and a_2. Vehicles a_1 and a_2 have their depots at vertices v_1 and v_2, respectively. The algorithm takes four rounds of communication to complete the given problem and we take $k = 2$.



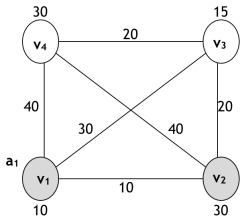


$x^a_{(u,v)}, y^a_v, \Delta x^a_{(u,v)}, \Delta y^a_v, \alpha_v, \beta^a_v,$	
$\Delta\beta^a_v, \Delta\alpha_v, \lambda^a_{(u,v)}, \gamma^a_v,$	
$\Delta\gamma^a_v, \tau^a_u, \Delta\tau^a_u = 0$	(Line 1)
$loc(a_1) \leftarrow v_1$	(Line 2)
$flag \leftarrow true$	(Line 3)
$Rival_{a_1} \leftarrow \{a_2\}$	(Line 4)
$\rho = 45$	(Line 5)
for(**s=4 to** 0 **do**)	(Line 6)
$tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$	(Line 7)
$N_2^{a_1} = \{v_1\}$	(Line 8)
$C(N_2^{a_1}) = 10, \pi(N_2^{a_1}) \leftarrow v_1$	(Line 12)
$p' \leftarrow v_1$	(Line 13)
$v_1 \leftarrow \mathsf{CheckIfPathFree}(v_1, v_2)$	(Line 15)
$tassign \leftarrow 1$	(Line 20)
for(**t=1 to** 0 **do**)	(Line 24)
$isService \leftarrow false$	(Line 25)
$p' \neq \emptyset \wedge N_2^{a_1} \neq \emptyset$	(Line 26)
$isService \leftarrow true$	(Line 27)
$v \leftarrow v_1$	(Line 28)
$\Delta y^{a_1}_{v_1} = 0.5$	(Line 29)
$t \leftarrow 0$	(Line 41)
$\gamma^{a_1}_{v_1} \leftarrow 0$	(Line 51)
$\tau^{a_1}_{v_1} \leftarrow 0$	(Line 52)

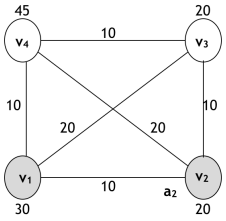
$x^a_{(u,v)}, y^a_v, \Delta x^a_{(u,v)}, \Delta y^a_v, \alpha_v, \beta^a_v,$	
$\Delta\beta^a_v, \Delta\alpha_v, \lambda^a_{(u,v)}, \gamma^a_v,$	
$\Delta\gamma^a_v, \tau^a_u, \Delta\tau^a_u = 0$	(Line 1)
$loc(a_2) \leftarrow v_2$	(Line 2)
$flag \leftarrow true$	(Line 3)
$Rival_{a_2} \leftarrow \{a_1\}$	(Line 4)
$\rho = 45$	(Line 5)
for(**s=4 to** 0 **do**)	(Line 6)
$tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$	(Line 7)
$N_2^{a_2} = \{v_2\}$	(Line 8)
$C(N_2^{a_2}) = 20, \pi(N_2^{a_2}) \leftarrow v_2$	(Line 12)
$p' \leftarrow v_2$	(Line 13)
$v_2 \leftarrow \mathsf{CheckIfPathFree}(v_2, v_1)$	(Line 15)
$tassign \leftarrow 1$	(Line 20)
for(**t=1 to** 0 **do**)	(Line 24)
$isService \leftarrow false$	(Line 25)
$p' \neq \emptyset \wedge N_2^{a_2} \neq \emptyset$	(Line 26)
$isService \leftarrow true$	(Line 27)
$v \leftarrow v_2$	(Line 28)
$\Delta y^{a_2}_{v_2} = 0.5$	(Line 29)
$t \leftarrow 0$	(Line 41)
$\gamma^{a_2}_{v_2} \leftarrow 0$	(Line 51)
$\tau^{a_2}_{v_2} \leftarrow 0$	(Line 52)

$\beta_{v_1}^{a_1} \leftarrow 0$		(Line 53)	$\beta_{v_2}^{a_2} \leftarrow 0$		(Line 53)
$y_{v_1}^{a_1} \leftarrow 0.5$		(Line 54)	$y_{v_2}^{a_2} \leftarrow 0.5$		(Line 54)
$\mathsf{Send}(a_1, v_1, 0.5 \times 10)$		(Line 56)	$\mathsf{Send}(a_2, v_2, 0.5 \times 20)$		(Line 56)
$\{(a_1, v_1, 5), (a_2, v_2, 10)\}$	$\leftarrow$ $\mathsf{Receive}()$		$\{(a_1, v_1, 5), (a_2, v_2, 10)\}$	$\leftarrow$ $\mathsf{Receive}()$	(Line 59)
$loc(a_1) \leftarrow v_1$		(Line 61)	$loc(a_1) \leftarrow v_1$		(Line 61)
$\alpha_{v_1} \leftarrow 5$		(Line 63)	$y_{v_1}^{a_1} \leftarrow 0.5$		(Line 62)
$count \leftarrow 1$		(Line 65)	$\alpha_{v_1} \leftarrow 5$		(Line 63)
$loc(a_2) \leftarrow v_2$		(Line 61)	$count \leftarrow 1$		(Line 65)
$y_{v_2}^{a_2} \leftarrow 0.5$		(Line 62)	$loc(a_2) \leftarrow v_2$		(Line 61)
$\alpha_{v_2} \leftarrow 10$		(Line 63)	$\alpha_{v_2} \leftarrow 10$		(Line 63)
$count \leftarrow 2$		(Line 65)	$count \leftarrow 2$		(Line 65)
$flag \leftarrow false$		(Line 77)	$flag \leftarrow false$		(Line 77)
end-for		(Line 78)	**end-for**		(Line 78)
$s \leftarrow 4 - 2 = 2$		(Line 79)	$s \leftarrow 4 - 2 = 2$		(Line 79)
end-for		(Line 80)	**end-for**		(Line 80)

In this communication round, vehicle a_1 starts from its depot at node v_1. Vehicle a_1 finds the location of its rival vehicle i.e., a_2 which is node v_2. Since in a complete graph all nodes are connected to each other, we always have the maximum size of N_k^a=1. So vehicle a_1 chooses $p' = v_1$. Then it finds out the cost of servicing node v_1 which is 10. Then vehicle a_1 calls $\mathsf{CheckIfPathFree}()$ which returns v_1. This is due to the fact that the vehicles a_1, a_2 currently has different location. So all conditions required to *service* a node is satisfied and hence vehicle a_1 services node v_1. Similarly following the same procedure vehicle a_2 services node v_2. Then after servicing a node, the vehicle sends the product of $\Delta y_v^a \cdot C(N_k^a)$ to its rival vehicle a_2. When a message is received from other rival vehicles, the corresponding α_v values are updated. Also, the count of a number of nodes serviced by all vehicles are taken and the outer for-loop value *s* is decremented by the value of $count$.



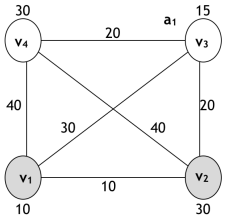
for(**s=2 to** 0 **do**)	(Line 6)
$tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$	(Line 7)
$N_2^{a_1} = \emptyset$	(Line 8)
$tassign \leftarrow 1$	(Line 10)
for(**t=1 to** 0 **do**)	(Line 24)
$isService \leftarrow false$	(Line 25)
$N_2^{a_1} = \emptyset$	(Line 26)
$u \leftarrow v_1$	(Line 43)
v_3 yields $nextbest(a_1)$	(Line 44)
$\Delta x^{a_1}_{(v_1,v_3)} \leftarrow 0.5$	(Line 45)
$\Delta \beta^{a_1}_{v_1} \leftarrow 45$	(Line 46)
$\Delta \gamma^{a_1}_{v_1} \leftarrow 0.5; \Delta \tau^{a_1}_{v_3} \leftarrow -0.5$	(Line 48)
$t \leftarrow 0$	(Line 49)
$\gamma^{a_1}_{v_1} \leftarrow 0.5; \tau^{a_1}_{v_3} \leftarrow -0.5$	(Line 51)
$\beta^{a_1}_{v_3} \leftarrow 45$	(Line 53)
$x^{a_1}_{(v_1,v_3)} \leftarrow 0.5$	(Line 55)
$\mathsf{Send}(a_1, (v_1, v_3), 0.5 \times 30)$	(Line 57)



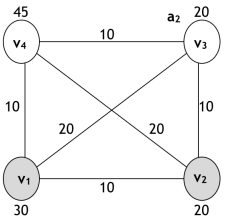
for(**s=2 to** 0 **do**)	(Line 6)
$tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$	(Line 7)
$N_2^{a_2} = \emptyset$	(Line 8)
$tassign \leftarrow 1$	(Line 10)
for(**t=1 to** 0 **do**)	(Line 24)
$isService \leftarrow false$	(Line 25)
$N_2^{a_2} = \emptyset$	(Line 26)
$u \leftarrow v_2$	(Line 43)
v_3 yields $nextbest(a_2)$	(Line 44)
$\Delta x^{a_2}_{(v_2,v_3)} \leftarrow 0.5$	(Line 45)
$\Delta \beta^{a_2}_{v_2} \leftarrow 45$	(Line 46)
$\Delta \gamma^{a_2}_{v_2} \leftarrow 0.5; \Delta \tau^{a_2}_{v_3} \leftarrow -0.5$	(Line 48)
$t \leftarrow 0$	(Line 49)
$\gamma^{a_2}_{v_2} \leftarrow 0.5; \tau^{a_2}_{v_3} \leftarrow -0.5$	(Line 51)
$\beta^{a_2}_{v_3} \leftarrow 45$	(Line 53)
$x^{a_2}_{(v_2,v_3)} \leftarrow 0.5$	(Line 55)
$\mathsf{Send}(a_2, (v_1, v_3), 0.5 \times 10)$	(Line 57)

$\{(a_1,(v_1,v_3),15),(a_2,(v_2,v_3),10)\} \leftarrow$		$\{(a_1,(v_1,v_3),15),(a_2,(v_2,v_3),5)\} \leftarrow$	
Receive()	(Line 59)	Receive()	(Line 59)
$loc(a_1) \leftarrow v_3$	(Line 68)	$loc(a_1) \leftarrow v_3$	(Line 68)
$\alpha_{v_3} \leftarrow 15$	(Line 70)	$x^{a_1}_{(v_1,v_3)} \leftarrow 0.5$	(Line 69)
$loc(a_2) \leftarrow v_3$	(Line 68)	$\alpha_{v_3} \leftarrow 15$	(Line 70)
$x^{a_2}_{(v_2,v_3)} \leftarrow 0.5$	(Line 69)	$loc(a_2) \leftarrow v_3$	(Line 68)
$\alpha_{v_3} \leftarrow 20$	(Line 70)	$\alpha_{v_3} \leftarrow 20$	(Line 70)
$flag \leftarrow false$	(Line 77)	$flag \leftarrow false$	(Line 77)
end-for	(Line 78)	**end-for**	(Line 78)
$s \leftarrow 2-0=2$	(Line 79)	$s \leftarrow 2-0=2$	(Line 79)
end-for	(Line 80)	**end-for**	(Line 80)

In this communication round, vehicle a_1 is in node v_1. This node has already been serviced by a_1 in the previous iteration and now N^a_k from this node is $\emptyset$. Now the vehicle a_1 has a choice of travelling to nodes v_2, v_3, v_4. Since node v_2 has already been serviced by a_2, the vehicle a_1 temporarily assigns the value ρ as the servicing cost for node v_2 by a_1. Now there are more chances for vehicle a_1 to choose nodes v_2 or v_3 based on the $nextbest(a_1)$ value. The $nextbest(a_1)$ value returns 45, which corresponds to node v_3. Hence a_1 travels to node v_3. Vehicle a_2 also performs travel in this iteration and decides to go to node v_3 based on its $nextbest(a_2)$ value. The value of $count$ for this iteration is zero as both vehicles did not *service* any nodes in this round of communication.



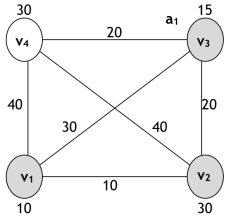
for(**s=2 to** 0 **do**)	(Line 6)
$tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$	(Line 7)
$N_2^{a_1} = \{v_3\}$	(Line 8)
$C(N_2^{a_1}) = 15, \pi(N_2^{a_1}) \leftarrow v_3$	(Line 12)
$p' \leftarrow v_3$	(Line 13)
$v_3 \leftarrow \mathsf{CheckIfPathFree}(v_3, v_3)$	(Line 15)
$tassign \leftarrow 1$	(Line 20)
for(**t=1 to** 0 **do**)	(Line 24)
$isService \leftarrow false$	(Line 25)
$p' \neq \emptyset \wedge N_2^{a_1} \neq \emptyset$	(Line 26)
$isService \leftarrow true$	(Line 27)
$v \leftarrow v_3$	(Line 28)
$\Delta y_{v_3}^{a_1} = 0.5$	(Line 29)
$t \leftarrow 0$	(Line 41)
$\beta_{v_3}^{a_1} \leftarrow 30$	(Line 53)
$y_{v_3}^{a_1} \leftarrow 0.5$	(Line 54)
$\mathsf{Send}(a_1, v_3, 0.5{\times}15)$	(Line 56)



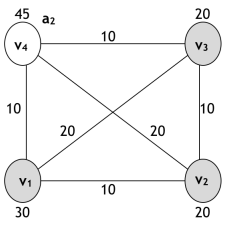
for(**s=2 to** 0 **do**)	(Line 6)
$tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$	(Line 7)
$N_2^{a_2} = \{v_3\}$	(Line 8)
$C(N_2^{a_2}) = 20, \pi(N_2^{a_1}) \leftarrow v_3$	(Line 12)
$p' \leftarrow v_3$	(Line 13)
$\emptyset \leftarrow \mathsf{CheckIfPathFree}(v_3, v_3)$	(Line 15)
$tassign = 1$	(Line 18)
for(**t=1 to** 0 **do**)	(Line 24)
$isService \leftarrow false$	(Line 25)
$p' = \emptyset$	(Line 26)
$u \leftarrow v_3$	(Line 43)
v_4 yields $nextbest(a_2)$	(Line 44)
$\Delta x_{(v_3,v_4)}^{a_2} \leftarrow 0.5$	(Line 45)
$\Delta \beta_{v_4}^{a_2} \leftarrow 45$	(Line 46)
$\Delta \gamma_{v_3}^{a_2} \leftarrow 0.5; \Delta \tau_{v_4}^{a_2} \leftarrow -0.5$	(Line 48)
$t \leftarrow 0$	(Line 49)
$\gamma_{v_2}^{a_2} \leftarrow 0.5; \tau_{v_3}^{a_2} \leftarrow -0.5$	(Line 51)
$\beta_{v_4}^{a_2} \leftarrow 45$	(Line 53)
$x_{(v_3,v_4)}^{a_2} \leftarrow 0.5$	(Line 55)
$\mathsf{Send}(a_2, (v_3, v_4), 0.5 \times 10)$	(Line 57)

$\{(a_1, v_3, 7.5)\}, \{(a_2, (v_3, v_4), 5)\}$	$\leftarrow$	$\{(a_1, v_3, 7.5)\}, \{(a_2, (v_3, v_4), 5)\}$	$\leftarrow$
Receive()	(Line 59)	Receive()	(Line 59)
$loc(a_1) \leftarrow v_3$	(Line 61)	$loc(a_1) \leftarrow v_3$	(Line 61)
$\alpha_{v_3} \leftarrow 27.5$	(Line 63)	$y_{v_3}^{a_1} \leftarrow 0.5$	(Line 62)
$loc(a_2) \leftarrow v_4$	(Line 68)	$\alpha_{v_3} \leftarrow 27.5$	(Line 63)
$x_{(v_3,v_4)}^{a_2} \leftarrow 0.5$	(Line 69)	$loc(a_2) \leftarrow v_4$	(Line 68)
$\alpha_{v_4} \leftarrow 5$	(Line 70)	$\alpha_{v_4} \leftarrow 5$	(Line 70)
$flag \leftarrow false$	(Line 77)	$flag \leftarrow false$	(Line 77)
end-for	(Line 78)	**end-for**	(Line 78)
$s \leftarrow 2-1=1$	(Line 79)	$s \leftarrow 2-1=1$	(Line 79)
end-for	(Line 80)	**end-for**	(Line 80)

In this communication round, both vehicles a_1 and a_2 enter the same location i.e., node v_3. Both vehicles choose N_k^a as $\{v_3\}$ and call CheckIfPathFree(). The function CheckIfPathFree() returns v_3 for a_1 and $\emptyset$ to vehicle a_2. This is because cost of servicing node v_3 by vehicle a_1 is cheaper when compared to a_2. So a_1 services node v_3 and a_2 travels to node v_4. The value of $count$ after this round of communication is 1 and there is only 1 node i.e., node v_4 which is yet to be serviced.



for(**s=1 to** 0 **do**)	(Line 6)
$tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$	(Line 7)
$N_2^{a_1} = \emptyset$	(Line 8)
$tassign \leftarrow 1$	(Line 10)
for(**t=1 to** 0 **do**)	(Line 24)
$isService \leftarrow false$	(Line 25)
$N_2^{a_1} = \emptyset$	(Line 26)
$u \leftarrow v_3$	(Line 43)
$\Delta x^{a_1}_{(v_3,v_4)} \leftarrow 0.5$	(Line 45)
$\Delta\beta^{a_1}_{v_4} \leftarrow 45$	(Line 46)
$\Delta\gamma^{a_1}_{v_3} \leftarrow 0.5; \Delta\tau^{a_1}_{v_4} \leftarrow -0.5$	(Line 48)
$t \leftarrow 0$	(Line 49)
$\gamma^{a_1}_{v_3} \leftarrow 0.5, \tau^{a_1}_{v_4} \leftarrow -0.5$	(Line 51)
$\beta^{a_1}_{v_4} \leftarrow 45$	(Line 53)
$x^{a_1}_{(v_3,v_4)} \leftarrow 0.5$	(Line 55)
$\mathsf{Send}(a_1, (v_3, v_4), 0.5{\times}20)$	(Line 57)



for(**s=1 to** 0 **do**)	(Line 6)
$tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$	(Line 7)
$N_2^{a_2} = \{v_4\}$	(Line 8)
$C(N_2^{a_2}) = 20, \pi(N_2^{a_2}) \leftarrow v_4$	(Line 12)
$p' \leftarrow v_4$	(Line 13)
$v_4 \leftarrow \mathsf{CheckIfPathFree}(v_4, v_3)$	(Line 15)
$tassign \leftarrow 1$	(Line 20)
for(**t=1 to** 0 **do**)	(Line 24)
$isService \leftarrow false$	(Line 25)
$p' \neq \emptyset \wedge N_2^{a_2} \neq \emptyset$	(Line 26)
$isService \leftarrow true$	(Line 27)
$v \leftarrow v_4$	(Line 28)
$\Delta y^{a_2}_{v_4} = 0.5$	(Line 29)
$t \leftarrow 0$	(Line 41)
$\gamma^{a_2}_{v_4} \leftarrow 0$	(Line 51)
$\tau^{a_2}_{v_4} \leftarrow 0$	(Line 52)
$\beta^{a_2}_{v_4} \leftarrow 0$	(Line 53)
$y^{a_2}_{v_4} \leftarrow 0.5$	(Line 54)
$\mathsf{Send}(a_2, v_4, 0.5 \times 45)$	(Line 56)

$\{(a_1,(v_3,v_4),10)\},\{(a_2,v_4,22.5)\}$	$\leftarrow$	$\{(a_1,(v_3,v_4),10)\},\{(a_2,v_4,22.5)\}$	$\leftarrow$
Receive()	(Line 59)	Receive()	(Line 59)
$loc(a_1) \leftarrow v_4$	(Line 61)	$loc(a_1) \leftarrow v_4$	(Line 61)
$\alpha_{v_4} \leftarrow 15$	(Line 63)	$x^{a_2}_{(v_3,v_4)} \leftarrow 0.5$	(Line 69)
$loc(a_2) \leftarrow v_4$	(Line 68)	$\alpha_{v_4} \leftarrow 15$	(Line 63)
$y^{a_2}_{v_4} \leftarrow 0.5$	(Line 62)	$loc(a_2) \leftarrow v_4$	(Line 68)
$\alpha_{v_4} \leftarrow 37.5$	(Line 70)	$\alpha_{v_4} \leftarrow 37.5$	(Line 70)
$v_4 \leftarrow loc(a_1)$	(Line 74)	$v_4 \leftarrow loc(a_2)$	(Line 74)
$\theta^{a_1}_4 \leftarrow 0.5$	(Line 75)	$\theta^{a_2}_4 \leftarrow 0.5$	(Line 75)
$flag \leftarrow false$	(Line 77)	$flag \leftarrow false$	(Line 77)
end-for	(Line 78)	**end-for**	(Line 78)
$s \leftarrow 1-1=0$	(Line 79)	$s \leftarrow 1-1=0$	(Line 79)
end-for	(Line 80)	**end-for**	(Line 80)

In this communication round, vehicle a_2 services node v_4 and vehicle a_1 travels to node v_4. The $count$ in this round of communication is 1 and the value of s becomes 0 denoting that all nodes have been serviced. The α values are $\alpha_1 = 5, \alpha_2 = 10, \alpha_3 = 27.5, \alpha_4 = 37.5$ respectively and the total cost is 80. If the fractional values chosen by the vehicles are rounded to integer values i.e., all fractional increases are given the value 1 and rest are given the value 0, then the total cost is 160.

Next, we will find the approximation ratio of Algorithm 2 based on the following proofs.

4.4 Proof of Correctness and Approximation Bound

Lemma 4.4.1 *Algorithm 2 forms a feasible solution for the LP.*

Proof

- **Constraint 3.10.** The function call to CheckIfPathFree in line 15 ensures that a node is serviced by only one vehicle in any iteration of the t-loop. In line 29, the algorithm increases the y_v^a value by $\frac{1}{m}$. Hence, the $\sum_{a' \in A} y_v^{a'} = \frac{1}{m}$. 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 3.3.** 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 35 or line 45 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 29, 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 $\frac{1}{m}$ before increasing the value of y_v^a by $\frac{1}{m}$ in the next communication round. So this guarantees the fact that if the node is serviced in the current iteration of the loop, the previous iteration would increase the corresponding $x_{(u,v)}^a$ value. 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 3.11, 3.12, 3.13.** A vehicle always chooses a simple path and hence this path satisfies the constraints 3.11, 3.12, and 3.13. Constraints 3.11 and 3.12 are satisfied from the definition of a simple path, which ensure 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 lines 37 and 48, 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 $\frac{1}{m}$. Hence, Constraints 3.11

and 3.12 are satisfied. And also the definition of a simple path ensures there is no sub-tour, which satisfies Constraint 3.13. 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 3.6.** Constraint 3.3 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 3.11, 3.12 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 4.4.2 *At the end of each iteration of the t-loop, the value of the primal and dual objectives are equal. That is,*

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

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

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

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

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

First, observe that in lines 37 and 48, 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 from any $(u, v) \in E$, from the definition of N_k^a and $nextbest(a)$. Hence, lines 36, and 47 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 59 and subsequently all iterations of the two for-loops in lines 60-66 (for Δy values) and 67 - 72 (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 15. 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 4.4.3 *In every iteration of the t-loop, it holds that*

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

Proof We define $\rho = \max\left\{C_s(a,v), C_t(a,(u,v)) \mid (a \in A) \ \wedge \ (u,v \in V)\right\}$ 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 $C_s(a,v) \leq \rho$. 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 $C_s(a,v) \leq \rho$. 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

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

■

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 4.4.4 *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 4.4.1, 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 $\dfrac{1}{m}$ 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 $x^a_{(u,v)}$ increased by $\frac{1}{m}$ 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 vehicle completes servicing all nodes in the path chosen. Hence, in one iteration of the s-loop, at most, we can have two rounds of communications (servicing, travelling) 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 3.15 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 4.4.5 *For all $v \in V - \{v_0^a\}$ and $a \in A$, in all iterations of the s-loop, Constraint 3.14 holds.*

Proof Following Lemma 4.4.4, we have $\alpha_v \leq \rho$. From the algorithm we have that,

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

- **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 4.4.6 *For all $(u,v) \in E$ and $a \in A$, in all iterations of the s-loop, Constraint 3.15 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,

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

- **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 $\frac{1}{m}$. An increase by $\frac{1}{m}$ 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 increrased by $\frac{1}{m}$. An increase by $\frac{1}{m}$ 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 3.15 is trivial. Since we have proved that $\beta_v^a + \theta_v^a \leq C_t(a,(u,v))$ and hence $\frac{\beta_v^a + \theta_v^a}{n} \leq C_t(a,(u,v))$. That is, $\hat{\beta_v^a} + \hat{\theta_v^a} \leq C_t(a,(u,v))$.

Lemma 4.4.7 *For all $v \in V - \{v_0^a\}$ and $a \in A$, in all iterations of the s-loop, Constraint 3.14 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.

Lemma 4.4.8 *The approximation ratio of Algorithm 2 is* $O(n \cdot (\rho)^{1/n}$

Proof ALG can be bounded as

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

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

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

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

■

In the next chapter we introduce the *randomized rounding* algorithm that converts the fractional solutions from Algorithm 2 to integer solutions.

Chapter 5

Randomized Rounding and On-The-Fly Algorithms

5.1 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 probability p, the vehicle that has increased its $\hat{y}^a_v$ fractional value corresponding to a node v gets to service the node and with probability $(1-p)$, the vehicle with the least cost to travel and service among the vehicles that have traveled to this corresponding node is allowed to service the node. We run this algorithm on $total^a$ which is the union of nodes for which the vehicle has traveled to and 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 can calculate the rounded values for a node (even if the current vehicle has not serviced it) and inform other vehicles the value it found and these values are updated by the respective vehicle.

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.

$visited^a \leftarrow \{\}$;
$total^a \leftarrow \{v \in V \mid \hat{y}^a_v > 0\} \cup$
$\{v_i, v_{i+1} \in V \mid (\forall i \in [0, n-1].(v_i, v_{i+1}) \in E \wedge \hat{x}^a_{(v_i,v_{i+1})} > 0 \wedge v_0 \leftarrow v^a_0)\}$;
for each $v \in total^a$ **do**
  $a'' \leftarrow -1$; $cost \leftarrow 0$;
  if $v \notin visited^a$ **then** {*Rounding starts here*}
    if $(\exists a \in A.\hat{y}^a_v > 0)$ **then** $a' \leftarrow a$;
    if $(\exists u \in V.\hat{x}^{a'}_{(u,v)} > 0 \wedge x^{a'}_{vu} \neq 1)$ **then** $x^{a'}_{(u,v)} \leftarrow 1$;
    $p^{a'}_v \leftarrow \min\{1, \hat{y}^{a'}_v \cdot ln(n+m)\}$; {*Probability $p^{a'}_v$ is calculated here*}
    $y^{a'}_v \leftarrow \begin{cases} 1 & \text{with probability } p^{a'}_v. \\ 0 & \text{with probability } 1 - p^{a'}_v. \end{cases}$ {$y^{a'}_v \leftarrow 1$ *with probability* $p^{\vec{a}}_v$ }
    $C_v \leftarrow \sum_{a \in A} \sum_{u \in V} \hat{x}^a_{(u,v)} \cdot C_t(a, (u, v))$; {$C_v$ *is calculated here.* }
    $count \leftarrow \left|\{(u, v) \mid \exists a \in A.\exists u \in V.\hat{x}^a_{(u,v)} > 0\}\right|$;
    $A_v \leftarrow \{a \in A \mid \exists u \in V.\hat{x}^a_{(u,v)} > 0 \ \wedge \ C_t(a, (u, v)) \leq ln(n+m) \cdot C_v\}$;
    if $(count \neq |A_v|)$ **then** $A_v \leftarrow \emptyset$;
    if $a' \in A_v \wedge (y^{a'}_v = 1)$ **then** $a'' \leftarrow a'$;
    else
      $y^{a'}_v \leftarrow 0$;
      $cost \leftarrow \min\{C_s(a, v) + C_t(a, (u, v)) \mid \exists a \in A.\exists u \in V.\hat{x}^a_{(u,v)} > 0\}$;
      Let $\tilde{a} \in A$ yield the value $cost$.
      if $(\exists u \in V.\hat{x}^{\tilde{a}}_{(u,v)} > 0 \wedge x^{\tilde{a}}_{vu} \neq 1)$ **then** $x^{\tilde{a}}_{(u,v)} \leftarrow 1$;
      $y^{\tilde{a}}_v \leftarrow 1$; $a'' \leftarrow \tilde{a}$;
    end if
    $visited^a \leftarrow visited^a \ \cup \ \{v\}$; {*Rounding ends here*}
    else Let a'' be the vehicle, where $y^{a''}_v = 1$.
  end if
  Send(v, a''); {*To all vehicles*}
  $RV \leftarrow$ Receive(); {*From all vehicles*}
  for each $(v', a') \in RV$ **do**
    if $(\exists a \in A.\exists u \in V.\hat{x}^a_{(u,v')} > 0 \wedge x^a_{v'u} \neq 1)$ **then** $x^a_{(u,v')} \leftarrow 1$;
    if $(v' \notin visited^a) \wedge (v' \in total^a)$ **then**
      $y^{a'}_{v'} \leftarrow 1$;
      $visited^a \leftarrow visited^a \ \cup \ \{v'\}$;
    else if $(v' \in visited^a) \ \wedge \exists a''.y^{a''}_{v'} = 1 \ \wedge (id(a') < id(a'')$ **then**
      $y^{a''}_{v'} \leftarrow 0$;
      $y^{\acute{a}}_{v'} \leftarrow 1$;
      else if $(v' \notin total^a)$ **then** $y^{a'}_{v'} \leftarrow 1$;
    end if
  end for
end for

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

- If $v \notin visited^a$ (From Line 5), the following steps are executed.

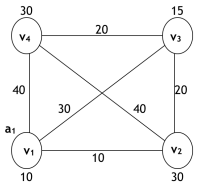
    1. In Line 6 the current vehicle chooses the vehicle id a' which has serviced node v (i.e., $\exists a \in A.\hat{y}_v^a > 0$). In Line 7 the corresponding vehicle's incoming edge is made 1. In Line 8, the fractional value $\hat{y}_v^{a'}$ is multiplied with $\ln(n+m)$ and the value returned is stored as $p_v^{a'}$. In Line 9 with probability of $p_v^{a'}$, the corresponding $y_v^{a'}$ gets the value 1 else it gets the value 0.
    2. In Line 10, let C_v be the cost of travels made to node v by all vehicles. Further in Line 12, let A_v be the set of all vehicles who have travel costs that are within a factor of $\ln(n+m)$ of the travel cost C_v.
    3. A vehicle in Line 11 keeps track of the count of vehicles that has entered the node v. In Line 13 if the size of A_v is equal to count, then the contents of A_v are retained else it is made empty. Then in Line 14 we check, if A_v contains the vehicle id (a') for which $y_v^{a'}$ has been made 1 in Line 9 of the algorithm. If this condition is satisfied the values increased in Lines (7 – 9) are retained.
    4. If either of the above conditions is not satisfied, the value of $y_v^{a'}$ is made 0 in Line 16. In Line 17 the vehicle (with $\hat{x}_{(u,v)}^a > 0$) that returns the minimum cost for traveling to node v and servicing it, is chosen and the corresponding $x_{(u,v)}^a, y_v^a$ is made 1 in Lines 19 – 20. After each node in $total^a$ is processed, the node is added to the visited nodes list $visited^a$.

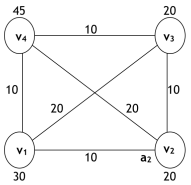
- If $v \in visited^a$ (Line 23) and the node has been already added to $visited^a$, the vehicle id which has increased its corresponding y_v^a is retrieved.

- Then, a message is sent to all vehicles (Line 25) informing them of the vehicle id which has increased its $y_v^a = 1$ for a node in consideration. When a message is received (Line 26), its corresponding $x_{(u,v)}^a$ are made 1 in Line 28. 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 $y_v^a = 1$. 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_v^a values is increased by 1.

Next we present an example (from Page 44) to show the working of *randomized rounding*.





$x^{a_1}_{(v_1,v_3)}, x^{a_1}_{(v_3,v_4)}, x^{a_2}_{(v_2,v_3)}, x^{a_2}_{(v_3,v_4)} \leftarrow 0.5$		$x^{a_1}_{(v_1,v_3)}, x^{a_1}_{(v_3,v_4)}, x^{a_2}_{(v_2,v_3)}, x^{a_2}_{(v_3,v_4)} \leftarrow 0.5$
$x^{a_1}_{(v_1,v_2)}, x^{a_1}_{(v_1,v_4)}, x^{a_1}_{(v_2,v_1)}, x^{a_1}_{(v_2,v_3)} \leftarrow 0$		$x^{a_1}_{(v_1,v_2)}, x^{a_1}_{(v_1,v_4)}, x^{a_1}_{(v_2,v_1)}, x^{a_1}_{(v_2,v_3)} \leftarrow 0$
$x^{a_1}_{(v_2,v_4)}, x^{a_1}_{(v_3,v_1)}, x^{a_1}_{(v_3,v_2)}, x^{a_1}_{(v_4,v_1)} \leftarrow 0$		$x^{a_1}_{(v_2,v_4)}, x^{a_1}_{(v_3,v_1)}, x^{a_1}_{(v_3,v_2)}, x^{a_1}_{(v_4,v_1)} \leftarrow 0$
$x^{a_1}_{(v_4,v_2)}, x^{a_1}_{(v_4,v_3)}, x^{a_2}_{(v_4,v_2)}, x^{a_2}_{(v_4,v_3)} \leftarrow 0$		$x^{a_1}_{(v_4,v_2)}, x^{a_1}_{(v_4,v_3)}, x^{a_2}_{(v_4,v_2)}, x^{a_2}_{(v_4,v_3)} \leftarrow 0$
$x^{a_2}_{(v_1,v_2)}, x^{a_2}_{(v_1,v_3)}, x^{a_2}_{(v_1,v_4)}, x^{a_2}_{(v_2,v_1)} \leftarrow 0$		$x^{a_2}_{(v_1,v_2)}, x^{a_2}_{(v_1,v_3)}, x^{a_2}_{(v_1,v_4)}, x^{a_2}_{(v_2,v_1)} \leftarrow 0$
$x^{a_2}_{(v_2,v_4)}, x^{a_2}_{(v_3,v_1)}, x^{a_2}_{(v_3,v_2)}, x^{a_1}_{(v_4,v_1)} \leftarrow 0$		$x^{a_2}_{(v_2,v_4)}, x^{a_2}_{(v_3,v_1)}, x^{a_2}_{(v_3,v_2)}, x^{a_1}_{(v_4,v_1)} \leftarrow 0$
$y^{a_1}_{v_1}, y^{a_2}_{v_2}, y^{a_1}_{v_3}, y^{a_2}_{v_4} \leftarrow 0.5$		$y^{a_1}_{v_1}, y^{a_2}_{v_2}, y^{a_1}_{v_3}, y^{a_2}_{v_4} \leftarrow 0.5$
$y^{a_1}_{v_2}, y^{a_1}_{v_4}, y^{a_2}_{v_1}, y^{a_2}_{v_3} \leftarrow 0.5$		$y^{a_1}_{v_2}, y^{a_1}_{v_4}, y^{a_2}_{v_1}, y^{a_2}_{v_3} \leftarrow 0.5$
$visited^{a_1} \leftarrow \{\}$;	(Line 1)	$visited^{a_2} \leftarrow \{\}$;
$total^{a_1} \leftarrow \{v_1, v_3, v_4, v_1, v_3\}$	(Line 2)	$total^{a_2} \leftarrow \{v_2, v_3, v_4, v_2, v_4\}$
for($v_1 \in \{v_1, v_3, v_4, v_1, v_3\}$)	(Line 3)	**for**($v_2 \in \{v_2, v_3, v_4, v_2, v_4\}$)
$a'' \leftarrow -1$, $cost \leftarrow 0$	(Line 4)	$a'' \leftarrow -1$, $cost \leftarrow 0$
$v_1 \notin visited^a$	(Line 5)	$v_2 \notin visited^a$
$a' \leftarrow a_1$	(Line 6)	$a' \leftarrow a_2$
$p^{a_1}_{v_1} \leftarrow \min\{1, 0.5{\cdot}ln(6)\}$	(Line 8)	$p^{a_2}_{v_2} \leftarrow \min\{1, 0.5 \cdot ln(6)\}$
$p^{a_1}_{v_1} \leftarrow \min\{1, 0.89\}$	(Line 8)	$p^{a_2}_{v_2} \leftarrow \min\{1, 0.89\}$
$p^{a_1}_{v_1} \leftarrow 0.89$	(Line 8)	$p^{a_2}_{v_2} \leftarrow 0.89$

Random value generated:0.75		Random value generated:0.50
$y_{v_1}^{a_1} \leftarrow 1$	(Line 9)	$y_{v_2}^{a_2} \leftarrow 1$
$C_{v_1} \leftarrow 0$	(Line 10)	$C_{v_2} \leftarrow 0$
$count \leftarrow 0$	(Line 11)	$count \leftarrow 0$
$A_{v_1} \leftarrow \emptyset$	(Line 12)	$A_{v_2} \leftarrow \emptyset$
$a_1 \notin A_{v_1}$	(Line 14)	$a_1 \notin A_{v_1}$
$y_{v_1}^{a_1} \leftarrow 0$	(Line 16)	$y_{v_2}^{a_2} \leftarrow 0$
$cost \leftarrow 10$	(Line 17)	$cost \leftarrow 20$
$y_{v_1}^{a_1} \leftarrow 1$	(Line 20)	$y_{v_2}^{a_2} \leftarrow 1$
$a'' \leftarrow a_1$	(Line 20)	$a'' \leftarrow a_2$
$visited^a \leftarrow \{v_1\}$	(Line 22)	$visited^a \leftarrow \{v_2\}$
Send(v_1, a_1) to a_2	(Line 25)	Send(v_2, a_2) to a_1
Receive(v_2, a_2) from a_2	(Line 26)	Receive(v_1, a_1) from a_1
$y_{v_2}^{a_2} \leftarrow 1$	(Line 35)	$y_{v_1}^{a_1} \leftarrow 1$
end-for	(Line 38)	**end-for**

In this communication round vertices v_1, v_2 were processed by a_1, a_2 respectively. There are no incoming edges to either of these nodes. Hence the $x_{(u,v)}^a$ values are not increased. At the end of rounding in this communication round, vehicle a_1 decides to makes its corresponding y_v^a for node v_1 as 1 and the vehicle vehicle a_2 decides to makes its corresponding y_v^a for node v_2 as 1.

for($v_3 \in \{v_1, v_3, v_4, v_1, v_3\}$)	(Line 3)	**for**($v_3 \in \{v_2, v_3, v_4, v_2, v_4\}$)
$a'' \leftarrow -1, cost \leftarrow 0$	(Line 4)	$a'' \leftarrow -1, cost \leftarrow 0$
$v_3 \notin visited^a$	(Line 5)	$v_3 \notin visited^a$
$a' \leftarrow a_1$	(Line 6)	$a' \leftarrow a_1$
$x^{a_1}_{v_1,v_3} \leftarrow 1$	(Line 7)	$x^{a_1}_{v_1,v_3} \leftarrow 1$
$p^{a_1}_{v_3} \leftarrow 0.89$	(Line 8)	$p^{a_1}_{v_3} \leftarrow 0.89$
Random value generated:0.95		Random value generated:0.90
$y^{a_1}_{v_3} \leftarrow 0$	(Line 9)	$y^{a_1}_{v_3} \leftarrow 0$
$C_{v_3} \leftarrow (30 + 10) \cdot 0.5 \leftarrow 20$	(Line 10)	$C_{v_3} \leftarrow (30 + 10) \cdot 0.5 = 20$
$count \leftarrow 2$	(Line 11)	$count \leftarrow 2$
$A_{v_3} \leftarrow \{a_1, a_2\}$	(Line 12)	$A_{v_3} \leftarrow \{a_1, a_2\}$
$a_1 \in A_{v_1} \wedge y^{a_1}_{v_3} = 0$	(Line 14)	$a_1 \in A_{v_1} \wedge y^{a_1}_{v_3} = 0$
$y^{a_1}_{v_3} \leftarrow 0$	(Line 16)	$y^{a_1}_{v_3} \leftarrow 0$
$cost \leftarrow 30$	(Line 17)	$cost \leftarrow 30$
$x^{a_2}_{v_2,v_3} \leftarrow 1$	(Line 19)	$x^{a_2}_{v_2,v_3} \leftarrow 1$
$y^{a_2}_{v_3} \leftarrow 1$	(Line 20)	$y^{a_2}_{v_3} \leftarrow 1$
$a'' \leftarrow a_2$	(Line 20)	$a'' \leftarrow a_2$;
$visited^a \leftarrow \{v_1, v_3\}$	(Line 22)	$visited^a \leftarrow \{v_2, v_3\}$
Send(v_3, a_2) to a_2	(Line 25)	Send(v_3, a_2) to a_1
Receive(v_3, a_2) from a_2	(Line 26)	Receive(v_3, a_2) from a_1
$x^{a_1}_{v_1,v_3} \leftarrow 1$	(Line 28)	$x^{a_1}_{v_1,v_3} \leftarrow 1$
$x^{a_2}_{v_2,v_3} \leftarrow 1$	(Line 28)	$x^{a_2}_{v_2,v_3} \leftarrow 1$
end-for	(Line 38)	**end-for**

In this communication round vertices v_3 were processed by both a_1, a_2 respectively. Vehicle a_2 finds that vehicle a_1 has increased its corresponding fractional value y^a_v for this node v_3. Hence the vehicle a_2 runs the rounding process on behalf on a_1. At the end of rounding in this communication round, vehicle a_2 decides to makes its corresponding y^a_v for node v_3 as 1.

for($v_4 \in \{v_1, v_3, v_4, v_1, v_3\}$)

$a'' \leftarrow -1, cost \leftarrow 0$

$v_4 \notin visited^a$

$a' \leftarrow a_2$

$x^{a_1}_{v_2,v_4} \leftarrow 1$

$p^{a_2}_{v_4} \leftarrow 0.89$

Random value generated:0.97

$y^{a_2}_{v_4} \leftarrow 0$

$C_{v_4} \leftarrow 15$

$count \leftarrow 2$

$A_{v_4} \leftarrow \{a_1, a_2\}$

$a_1 \in A_{v_1} \wedge y^{a_2}_{v_4} = 0$

$y^{a_2}_{v_4} \leftarrow 0$

$cost \leftarrow 50$

$x^{a_1}_{(v_3,v_4)} \leftarrow 1$

$y^{a_1}_{v_4} \leftarrow 1$

$a'' \leftarrow a_1$

$visited^a \leftarrow \{v_1, v_3, v_4\}$

Send(v_4, a_1) to a_2

Receive(v_4, a_1) from a_2

$x^{a_1}_{(v_3,v_4)} \leftarrow 1$

$x^{a_2}_{(v_3,v_4)} \leftarrow 1$

end-for

for($v_4 \in \{v_2, v_3, v_4, v_2, v_4\}$)

$a'' \leftarrow -1, cost \leftarrow 0$

$v_4 \notin visited^a$

$a' \leftarrow a_2$

$x^{a_2}_{v_2,v_4} \leftarrow 1$

$p^{a_2}_{v_4} \leftarrow 0.89$

Random value generated:0.99

$y^{a_2}_{v_4} \leftarrow 0$

$C_{v_4} \leftarrow 15$

$count \leftarrow 2$

$A_{v_4} \leftarrow \{a_1, a_2\}$

$a_1 \in A_{v_1} \wedge y^{a_2}_{v_4} = 0$

$y^{a_2}_{v_4} \leftarrow 0$

$cost \leftarrow 50$

$x^{a_1}_{(v_3,v_4)} \leftarrow 1$

$y^{a_1}_{v_4} \leftarrow 1$

$a'' \leftarrow a_1$

$visited^a \leftarrow \{v_2, v_3, v_4\}$

Send(v_4, a_1) to a_1

Receive(v_4, a_1) from a_1

$x^{a_1}_{(v_3,v_4)} \leftarrow 1$

$x^{a_2}_{(v_3,v_4)} \leftarrow 1$

end-for

In this communication round vertices v_4 were processed by both a_1, a_2 respectively. Vehicle a_1 finds that vehicle a_2 has increased its corresponding fractional value y^a_v for this node v_4. Hence the vehicle a_1 runs the rounding process on behalf on a_2. At the end of rounding in this communication round, vehicle a_1 decides to makes its corresponding y^a_v for node v_4 as 1. In the next two communication rounds the nodes in $total^a$ are already

present in $visited^a$. Hence this information is retrieved and no rounding happens further in this algorithm.

for($v_1 \in \{v_1, v_3, v_4, v_1, v_4\}$)	(Line 3)	**for**($v_2 \in \{v_2, v_3, v_4, v_2, v_4\}$)
$a'' \leftarrow -1, cost \leftarrow 0$	(Line 4)	$a'' \leftarrow -1, cost \leftarrow 0$
$v_1 \in visited^a$	(Line 23)	$v_2 \in visited^a$
$a'' \leftarrow a_1$	(Line 23)	$a'' \leftarrow a_2$
Send(v_1, a_1) to a_2	(Line 25)	Send(v_2, a_2) to a_1
Receive(v_2, a_2) from a_2	(Line 26)	Receive(v_1, a_1) from a_1
$y_{v_2}^{a_2} \leftarrow 1$	(Line 35)	$y_{v_1}^{a_1} \leftarrow 1$
end-for	(Line 38)	**end-for**

for($v_4 \in \{v_1, v_3, v_4, v_1, v_4\}$)	(Line 3)	**for**($v_4 \in \{v_2, v_3, v_4, v_2, v_4\}$)
$a'' \leftarrow -1, cost \leftarrow 0$	(Line 4)	$a'' \leftarrow -1, cost \leftarrow 0$
$v_4 \in visited^a$	(Line 23)	$v_4 \in visited^a$
$a'' \leftarrow a_1$	(Line 23)	$a'' \leftarrow a_1$
Send(v_4, a_1) to a_1	(Line 25)	Send(v_4, a_1) to a_1
Receive(v_4, a_1) from a_1	(Line 26)	Receive(v_1, a_1) from a_1
$x_{(v_3,v_4)}^{a_1} \leftarrow 1$	(Line 28)	$x_{(v_3,v_4)}^{a_1} \leftarrow 1$
$x_{(v_3,v_4)}^{a_2} \leftarrow 1$	(Line 28)	$x_{(v_3,v_4)}^{a_2} \leftarrow 1$
end-for	(Line 38)	**end-for**

Algorithm 2 from page 44 gave a total cost of 160 whereas the *randomized rounding* algorithm gives us a better solution of 150. Next, we prove that our solution from *randomized rounding* algorithm is a feasible ILP solution and also find the bounds for the *randomized rounding* algorithm.

5.1.1 Proofs and Approximation Bounds of Rounding Algorithm

Lemma 5.1.1 *Algorithm 3 produces a feasible ILP solution.*

Proof

- **Constraint 3.2.** 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 20); 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 14 returns false, and hence only one y_v^a is assigned the value 1 from line 20. 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 30. 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 34 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 3.3.** For every node $v \in V$, the randomized rounding algorithm increases $x_{(u,v)}^a = 1$ in lines (7 or 19 or 28). 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 3.4, 3.5, and 3.7.** From lines (7 and 19 and 28) we can say that there are no sub-tours. Constraints (3.4, 3.5) 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 3.6.** The Constraint 3.3 ensures that a vehicle enters a node before servicing it. Also Constraints 3.4, 3.5 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 5.1.2 *The approximation ratio of Algorithm 3 for integer solution to* $x^a_{(u,v)}, y^a_v$ *is*

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

Proof

- **Case 1.** $(a' \in A_v \wedge (y^{a'}_v = 1))$ (Line 9)

  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 travelling is at most

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

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

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

- **Case 2.** $(a' \notin A_v \vee (y^{a'}_v \neq 1))$ (Line 9)

  From the definition of C_v, A_v we have that

  $$\sum_{a \notin A_v} \sum_{u \in V} \hat{x}^a_{(u,v)} \leq \frac{1}{ln(n+m)} \tag{5.1}$$

  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 $\sum_{u \in V} \hat{x}^a_{(u,v)} - \hat{y}^a_v \geq 0$ 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

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

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

$$\sum_{a \in A} \sum_{u \in V} \hat{x}^a_{(u,v)} \leq 1 \tag{5.2}$$

Equation 5.2 value comes from the fact that $\hat{x}^a_{(u,v)}$ is increased by a value $\frac{1}{m}$ and at most m vehicles can have $\hat{x}^a_{(u,v)} = \frac{1}{m}$. So, in total,

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

Hence we have

$$\sum_{a \in A_v} \hat{y}^a_v \leq \sum_{a \in A_v} \sum_{u \in V} \hat{x}^a_{(u,v)} \leq 1 - \frac{1}{ln(n+m)} \text{ (From } 5.1, 5.2) \tag{5.3}$$

The probability that $(a' \notin A_v \vee (y^{a'}_v \neq 1))$ 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 5.3 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

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

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

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

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:

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

5.2 Distributed On-The-Fly Solution

We can modify Algorithm 2 from the last chapter to solve the *open multi-depot vehicle routing problem* in a distributed on-the-fly fashion. Here now we assign, $x^a_{(u,v)}, y^a_v$ as either 0 or 1 values to mimic the work of ILP directly. We don't prove any bounds in this case, but we just compare results of Algorithm 2 with the distributed *on-the-fly* solution using simulations which are shown in the next section.

We use the same notations from Algorithm 2 here. Here in the *on-the-fly* solution the decision whether to service or travel are taken by the distributed algorithm itself unlike Algorithm 2 where decisions on whether to service or travel are taken by the *randomized rounding*. Since this distributed algorithm can take decisions *on-the-fly*, there is no need for a *randomized rounding* algorithm for this case.

We can modify Algorithm 2 to solve the VRP in an on-the-fly distributed fashion. We basically need take out the processing of dual LP variables $\alpha_v, \beta^a_v, \lambda^a_{(u,v)}, \gamma^a_v, \tau^a_u, \theta^a_v$ and in lines 35, 45, and 69 we increase $\Delta x^a_{(u,v)}$ by value 1 and in lines 29, and 62, we increase Δy^a_v by 1.

Algorithm 4 Algorithm for vehicle a

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

$x^a_{(u,v)}, y^a_v, \Delta x^a_{(u,v)}, \Delta y^a_v, totalcost \leftarrow 0$;
$loc(a) \leftarrow v^a_0$;
$Rival_a \leftarrow A - \{a\}$;
Let N^a_k, $C(N^a_k)$, ρ, and $nextbest$ be as defined in Definition and Equations 4.2.2, 4.1, 4.2, and 4.3, respectively.
for $s = |V|$ **to** 0 **do**
  $tassign \leftarrow 0, p' \leftarrow \emptyset, count \leftarrow 0$;
  Calculate N^a_k; {*Step 1 in Conditions For Servicing.*}
  if $(N^a_k = \emptyset)$ **then**
    $tassign \leftarrow 1$;
  else
    Calculate $C(N^a_k)$ and $\pi(N^a_k)$; {*Step 2 in Conditions For Servicing.*}
    $p' \leftarrow \pi(N^a_k)$;
    for each $a' \in Rival_a$ **do**
      $p' \leftarrow \mathsf{CheckIfPathFree}(p', loc(a'))$; {*Step 3 in Conditions For Servicing.*}
    end for
    if $(p' = \emptyset)$ **then**
      $tassign \leftarrow 1$;
    else
      $tassign \leftarrow 2|p'| + 1$;
      if $(|p'| \neq |\pi(N^a_k)|)$ **then** Calculate $C(N^a_k)$ for p';
    end if
  end if
  for $t = tassign$ **to** 0 **do**
    $isService \leftarrow false$;
    if $(p' \neq \emptyset) \wedge (N^a_k \neq \emptyset)$ **then** {*Check if Servicing conditions are met.*}
      $isService \leftarrow true$; {*If Conditions of Servicing are met, Servicing begins.*}
      $v \leftarrow loc(a)$;
      if $(y^a_v = 0)$ **then** $\Delta y^a_v \leftarrow 1$; {*Servicing is done here.* }
      else {*Vehicle travels to next node in path.*}
        $u \leftarrow loc(a)$;
        Let v be the node adjacent to u on p' st. $x^a_{(u,v)} = 0$;
        $isService \leftarrow false$;
        $\Delta x^a_{(u,v)} \leftarrow 1$;
        Temporarily assign $C_t(a, (u, v)) \leftarrow C(N^a_k)$;
      end if
      $t \leftarrow t - 1$; {*Servicing ends here.*}
    else {*If Conditions for Servicing are not met, Travel starts here.*}
      Let $u \leftarrow loc(a)$;

39: Let v yield $nextbest(a)$; {*Conditions for Traveling: Calculate* $nextbest(a)$.}
40: $\Delta x^{a}_{(u,v)} \leftarrow 1$;
41: $t \leftarrow 0$;
42: **end if** {*Travel ends here*}
43: $y^{a}_{v} \leftarrow y^{a}_{v} + \Delta y^{a}_{v}$;
44: $x^{a}_{(u,v)} \leftarrow x^{a}_{(u,v)} + \Delta x^{a}_{(u,v)}$;
45: **if** ($isService$) **then** Send$(id(a), v, \Delta y^{a}_{v}.C(N^{a}_{k}))$; {*To all vehicles*}
46: **else** Send$(id(a), (u,v), \Delta x^{a}_{(u,v)}.C_t(a,(u,v))$; {*To all vehicles*}
47: **end if**
48: $\{(id, v, Y)\}_{A' \subseteq A},\ \{(id, (u,v), X)\}_{A'' \subseteq A} \leftarrow$ Receive();
49: **for each** $\bar{a} \in A'$ **do**
50: $loc(\bar{a}) \leftarrow v$;
51: **if** $(a \neq \bar{a})$ **then** $y^{\bar{a}}_{v} \leftarrow 1$;
52: $totalcost \leftarrow totalcost + Y$;
53: $count \leftarrow count + 1$;
54: **end for**
55: **for each** $a'' \in A''$ **do**
56: $loc(a'') \leftarrow v$;
57: **if** $(a \neq a'')$ **then** $x^{a''}_{(u,v)} \leftarrow 1$;
58: $totalcost \leftarrow totalcost + X$;
59: **end for**
60: **end for**
61: $s \leftarrow s - count$; {*Value of s is decremented by count.*}
62: **end for**

In the next section, we discuss the implementation of our algorithm in detail.

5.3 Implementation of the Algorithm

5.3.1 Communication Model Of The System:

We design a synchronous algorithm to solve the *open multi-depot vehicle routing problem*.We use a global synchronous broadcast model for our system. The vehicles after each iteration of our algorithm broadcast their current location and costs incurred by the vehicle in the current step of the algorithm. The vehicles do not have an agreement to service particular sets of nodes among each other. The broadcast information just helps other vehicles to know if a new node has been serviced and also to keep track of the total cost incurred thus far by all vehicles. Our algorithm uses a nearest neighbour technique to decide on the possible next path for servicing the unserviced nodes. Even though this

approach is short-sighted, we still are able to design a distributed algorithm to solve the OMDVRP problem which gives a sub-optimal solution in most cases and in some cases even gives us optimal solution. We assume our broadcast to be perfect, in the sense that no bits of information is lost during transit among different vehicles. This assumption is very important, because after each iteration of the algorithm, all vehicles wait for messages from all other vehicles and only after receiving this information they start their next iteration.

Our algorithm is implemented using Java language. We have implemented our algorithm using the TCP/IP protocol. We have implemented both servers and clients in our laptop using the local host. After implementing the algorithm in Java, we run simulations for different graphs of varying size. The communication model is centralized. Every broadcasted message goes through the server to all other vehicles. The server specifically does two jobs to do: One is to allow exactly 'm' vehicles to work in one algorithm and then act as a access point between two or more vehicles for communication. Figure 5.1 below depicts the communication model that has been used.

A vehicle from the message received from a rival vehicle, takes note of the costs incurred and also the node it has serviced in the current iteration. Keeping track of the count of serviced nodes help in the faster completion of the problem. Also it is important to note that based on the information obtained from other vehicles, the current vehicle does not restrict or change its path to a new path. One other use of this broadcasted message is that each vehicle gets to know the current location of all other vehicles at the current iteration. This helps in choosing a path in such a way that the vehicle has the least cost of servicing nodes. The structure of the broadcast message is shown next.

The structure of the broadcasted message contains four important parts namely the *vehicle id, current node, cost incurred* in the current iteration and finally a bit to distinguish if a vehicle has serviced the current node or has traveled to the current node. The vehicle id is a unique id assigned to each vehicle when it connects to the server. This id is used to differentiate between vehicles when a message is being broadcast. This bit is very important as each vehicle waits for messages from all other vehicles before commencing

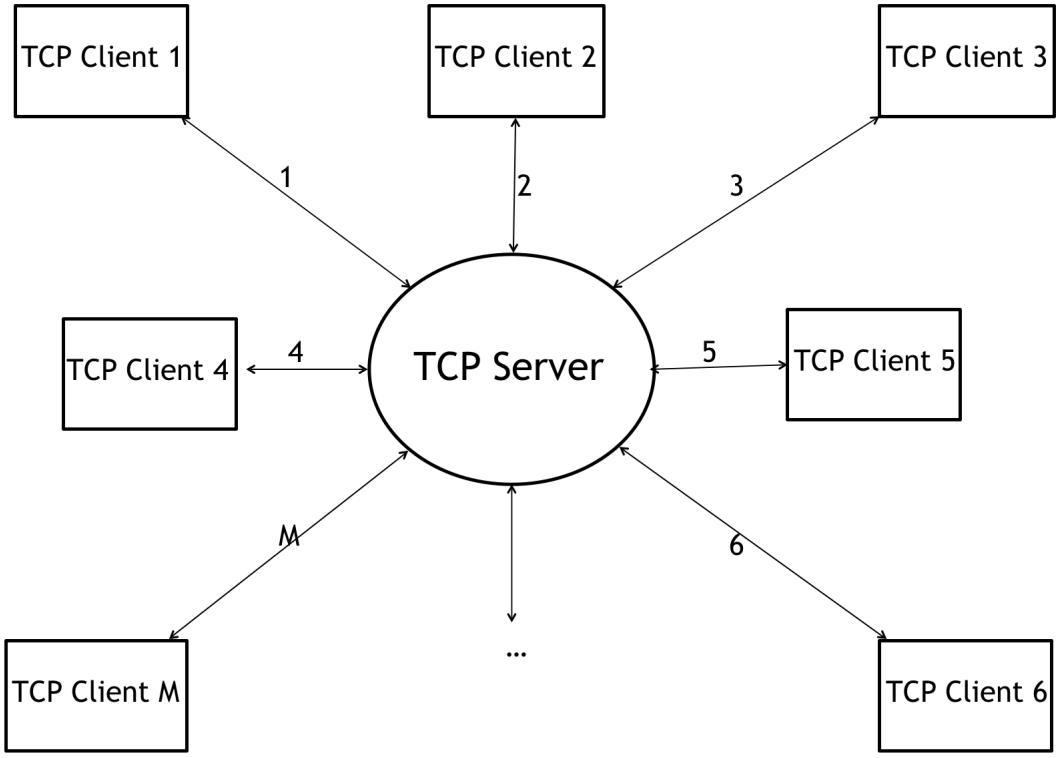


Figure 5.1: Communication Model Of the System

the next iteration. The current node helps other vehicles to find the location of the current vehicle. This, in turn, helps the vehicles in choosing the size of their dynamic neighbourhood. The third bit consists of the cost which is very updated by all vehicles during each iteration of the algorithm. The last and the most important bit of this structure is the bit which differentiates between servicing and travel. If a node has been serviced, all other vehicles take note of the node that has been serviced and assigns temporarily a high cost for servicing the node. Also these nodes are immediately taken out of the dynamic neighbourhood in the next iteration of the loop. Also a vehicles does not assume that a rival vehicle has reached a node, it will surely service it. If the current vehicle has a better cost by taking a particular path, it will take that path not worrying if other rival vehicles are already present in that node.

Next we discuss the structure of the broadcast message for the *randomized rounding* algorithm.

Here we have just two bits of information that has been broadcasted to other vehicles.

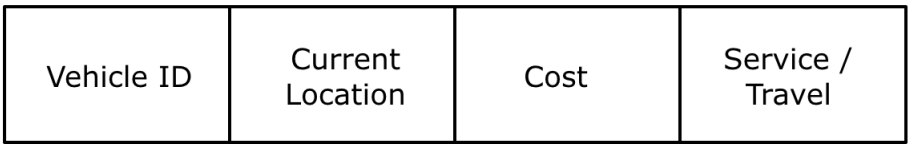


Figure 5.2: Structure of the Broadcasted Message for Distributed Approximation and Distributed On-The-Fly Algorithm

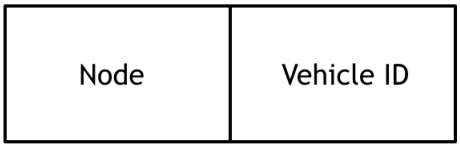


Figure 5.3: Structure of the Broadcasted Message for Randomized Rounding Algorithm

The first bit contains the node for which the *randomized rounding* has been applied to. The second and the most important bit gives the information of the vehicle id that has been assigned to service that particular node. If a vehicle receives a conflicting message from two vehicles i.e., if for the same node, there are two messages with different vehicle ids, then the vehicle only keeps the message which has the lowest vehicle ID for that corresponding node. If such an update happens, the previous message is discarded and the corresponding y_v^a value is made 0.

5.3.2 Computation Model Of The System:

Here we show the computation models used to design both the distributed approximation and *on-the-fly* algorithm to solve OMDVRP. First, we show the computation model used for the distributed approximation solution which combines the distributed approximation along with *randomized rounding*.

1. **Distributed Approximation Algorithm**

   There are three main parts of computation that take place for the distributed approx-

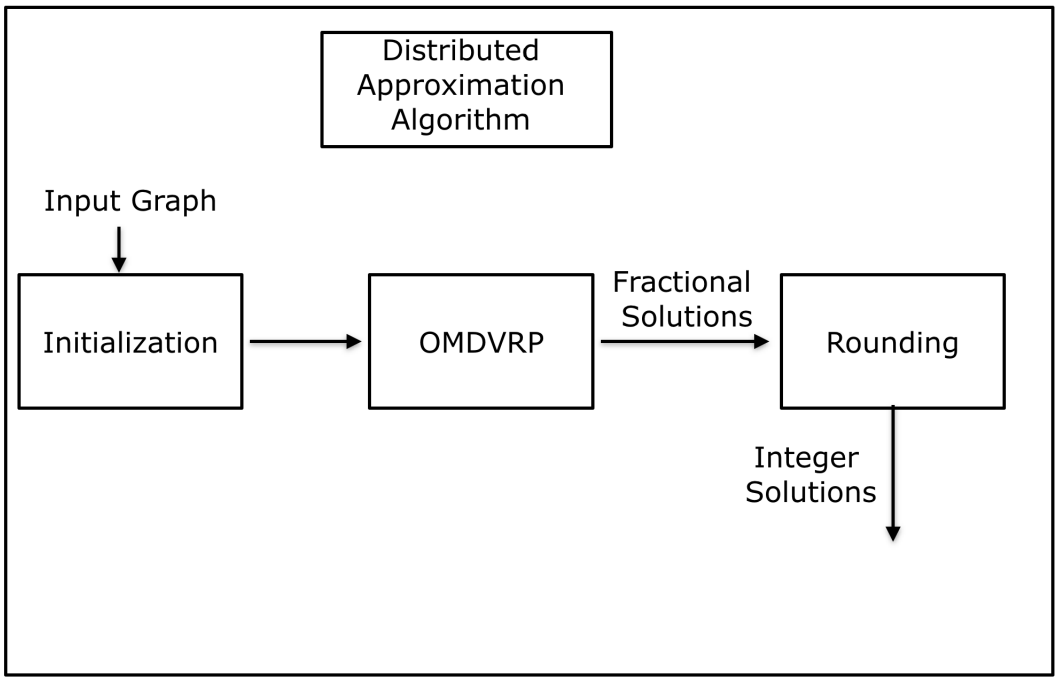


Figure 5.4: Computation Model for Distributed Approximation Algorithm

imation algorithm.

- **Initialization**

  In this step, each vehicle connects to the server and gets associated with a vehicle ID. The vehicles even after getting their respective vehicle IDs wait till the last vehicle has been assigned its vehicle ID. After assigning the last vehicle with an ID, the server sends across the graph containing the travel costs, servicing costs and initial locations of all vehicles. Thus each vehicle holds the graph of all its rival vehicles apart from having its own graph. This information is useful when vehicles need to decide on who has the least cost of servicing a node when both these vehicles are present in the same node.

- **OMDVRP**

  The second and most important step is when all vehicles collectively solve the OMDVRP problem by respecting all the side constraints. After each iteration, broadcast messages are sent to all vehicles informing them about the current location of a vehicles, the nodes serviced or traveled in the current iteration

and also helping all vehicles to have a count of the number of nodes that has been serviced. This step produces the fractional values for servicing or travel across nodes for all vehicles.

- **Rounding**

  Each vehicle runs the *randomized rounding* algorithm once for each node in the complete list of paths and nodes serviced during the distributed approximation algorithm. If a node in the total list has already been processed by some other vehicle, the id of the vehicle who is assigned to service the node stored and updated if necessary.

2. **Distributed *On-The-Fly* Algorithm:**

There are two main parts of computation that takes place for the distributed *on-the-fly* algorithm.

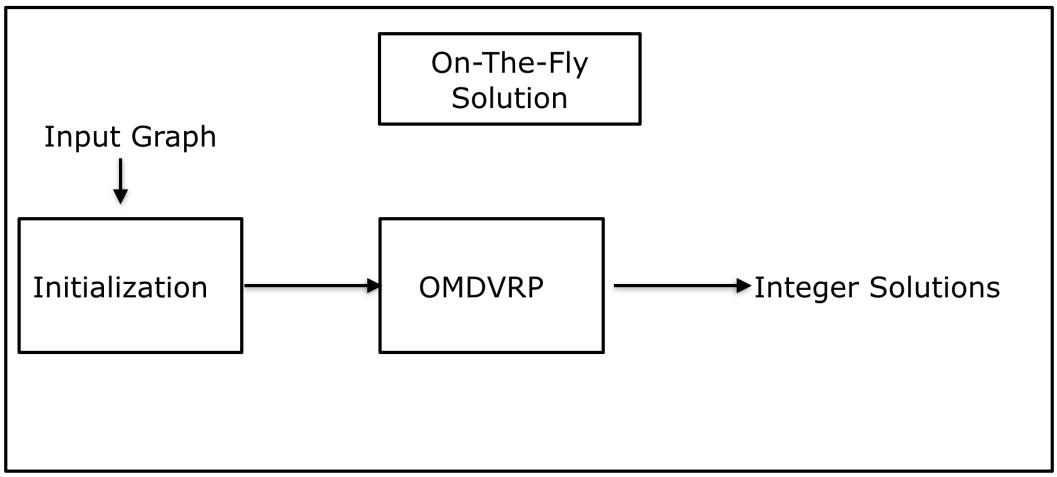


Figure 5.5: Computation Model for Distributed *On-The-Fly* Algorithm

- **Initialization**

  The initialization step is same as the Initialization step for the distributed approximation algorithm.

- **OMDVRP**

  The second and most important step is when all vehicles collectively solve the

OMDVRP problem by respecting all the side constraints. After each iteration, broadcast messages are sent to all vehicles informing them about the current location of a vehicles, the nodes serviced or traveled in the current iteration and also helping all vehicles to have a count of the number of nodes that has been serviced. This step produces the integer values for servicing or travel across nodes for all vehicles.

5.3.3 Implementation of the Algorithm in Java

Our algorithm was implemented using Java. The structure of the program consists of three main parts:

- **Server.java**

  This is the file which keeps track of the number of vehicles and the number of nodes in the graph generated. It also has the list of vehicle IDs to be assigned to each vehicle when it connects to the server. Also it does not allow more that m vehicles to join the server. Each vehicle is assigned a dedicated thread called ClientThread and is invoked using the vehicle ID and the socket details.

- **ClientThread.java**

  This is the file which acts as a server for the clients that connect to the server. All interactions between server and client are done using the ClientThread class. It basically stores the structure of the graph for all vehicles and also stores the details of depots of each vehicles. Its the file that generates the cost of traveling and cost of servicing for vehicles and it can broadcast messages to all vehicle id simultaneously. This is the file that helps us achieve synchronous broadcast.

- **Client.java**

  Its the most important file that depicts the working of our algorithm. It has to connect to the server using the same port number as the server and then receive all details like the structure of the graph, number of vehicles in the experiment and so on. It runs the algorithm once the graph is given as an input from the last

vehicle's ClientThread class. It has to execute the algorithm and also broadcast valid information to all vehicles and also gain valuable information from all other rival vehicles. They must synchronize every step of computation with all other vehicles and collectively complete the OMDVRP problem.

Next we discuss some major modules and functions of our implementation.

The first step in our implementation is to create the Server. This will allow exactly m clients to connect to it and run simulations. The server holds the value of the number of nodes that are used for the current experiment and also the number of vehicles along with unique vehicle ids. The server opens a dedicated ClientThread for every client that connects to it. The ClientThread has two main functions. One is to create the graph to be used for simulation along with servicing and traveling costs for all vehicles and also act as an access point between multiple clients. Next we discuss how these graphs are generated by the ClientThread class.

Our algorithm works well on both complete and cyclic graphs. When designing a complete graph, a random graph is generated and it gets converted to a complete graph using the Dijkstra's all pair shortest path algorithm i.e., if there is no edge between two pair of nodes, the virtual edge is created with shortest distance (using other edges) to travel between those 2 nodes. Also we must ensure a symmetric complete graph is obtained. Next when we design a cyclic graph we must ensure connects to only two nodes i.e., its successor and its predecessor. Again we must ensure the cyclic graph is symmetric in nature.

We have two ways to input data to the clients. The first way is when the ClientThread generates a new graph and uses the standard broadcast feature wherein a ClientThread broadcasts the graph's information to all clients. The second is to use a file from which data can be read into by the ClientThread instead of generating a new graph. Next we discuss the working of our clients which implement the algorithm in them.

Each client after being created connects to the server using the same port number as the server. Then the server responds to the client by sending the client a vehicle id that it uses to differentiate itself from other vehicles. Then after the last client connects tot the

server, the ClientThread sends out data using a synchronous broadcast. The clients upon receiving this information stores the corresponding travel costs and servicing costs. This is done extracting the first two values that has been received. The first bit of information contains the number of vehicles participating in this experiment and the second element gives the number of nodes used in the experiment. Using this information a client creates two arrays i.e., to hold $x^a_{(u,v)}, y^a_v$ values. The maximum size of these arrays are set from the first two bits of information. Then the following bits of information are the location of each vehicle. So the client by using the count of the vehicles present in the algorithm is able to distinguish between the location of vehicles and the costs of travel. Next the set of bits that come in are the travel costs of all vehicles followed lastly by the cost of servicing nodes for all vehicles. After each vehicle receives this message the algorithm begins. The client is able to differentiate costs of other vehicles by the vehicle id presented to it. The client works differently based on the type of the graph.

- **Complete Graphs**

  For a complete graph, the minimum distance from a vehicle to an another vehicle is always 1. Hence the size of neighbourhood is always one. So the clients are not forced to create paths of size more than 1.

- **Cyclic Graphs**

  For cyclic graphs, the vehicles are free to have the size of their neighbourhood less than the distance between the nearest rival and can have a maximum size of m. Thus the vehicles are free to have a dynamic size neighbourhood. This helps in reducing the computations also by a good factor. That is if vehicles are placed far from each other. each vehicle can take a path of maximum size m and after ensuring no vehicle can enter the path chosen in lesser iteration than the current node, each vehicle need not perform any computations for finding a new path till end of processing the last node in the path.

The paths generated is found using the simple breadth-first algorithm. Since we use this path finding algorithm only for cyclic graphs, the breadth-first algorithm works well. For

complex graph structures, we may need to use the A star algorithm to find the paths. The distance between rival vehicles can also be found by the same algorithm. Next after the paths are generated the $CheckIfPathfree()$ function is called which returns a path or its subset if there are no better vehicles to service the nodes in the path better than the current vehicle else it returns an $\emptyset$.

Then after a vehicle performs a service or travel it has to pass this information to all vehicles . This is done using the synchronous broadcast. The last bit of this broadcast message is of greater importance. To denote if a node has been serviced in the current iteration, a vehicle sends the value 6543 and also to denote if a node has been traveled to, the vehicle sends the value 3546. Based on these two distinct values, each vehicle that receives the broadcast information is able to differentiate between a service and a travel. Each vehicle stores a copy of the $x^a_{(u,v)}, y^a_v$ is a separate array. This array is updated every time a vehicle visits a node or services it. If an edge has been traversed by a vehicle the corresponding vehicle's edge entry has been modified to a temporary value. This information is important when it comes to finding the number of steps a rival vehicle takes to reach the path chosen by a current vehicle.

After each round, a message is broadcast by the current vehicle and it also receives broadcast messages from rival vehicles. This information is processed and the corresponding outer loop sentinel value is decremented if a node has been serviced in the current iteration by any vehicle.

In the case of the distributed approximation algorithm, the fractional solutions produced at the end of the algorithm is given to the rounding algorithm as an input. These values are converted to $\{0, 1\}$ based on some probability. For the case of *on-the-fly* solution, this step is not required as the *on-the-fly* solution itself produces a integer solutions for $x^a_{(u,v)}, y^a_v$

Chapter 6

Simulation Results

In this chapter, we present our simulations results comparing the results of ILP solution with a greedy algorithm, distributed approximation algorithm, and on-the-fly solution for *cyclic* and *grid graphs converted to complete* graphs.

6.1 Simulation Results.

6.1.1 Parameters and Metrics of the Simulations.

We list below the parameters that have been changed during simulations:

- location of depots.
- the type of graph i.e., *grid to complete graphs* or *cyclic* graph.
- the type of the vehicles, i.e., from vehicles having same travel time and service time to vehicles having different travel time and service time.

We evaluate the *total cost* metric in our simulations which denote the costs that all vehicles collectively take to finish servicing all nodes in the given graph.

6.1.2 Results Of The Simulations.

- **Comparison of Solutions for Small Grid Graphs converted to Complete Graph.** We study grid-shape graphs with different sizes. To ensure that the grids have

Hamiltonian paths, we convert them 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 designed 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. The graphs comparing simulations results of a 3*3, 5*5, 7*7 grid is shown in Figures 6.1, 6.2, 6.3 respectively.

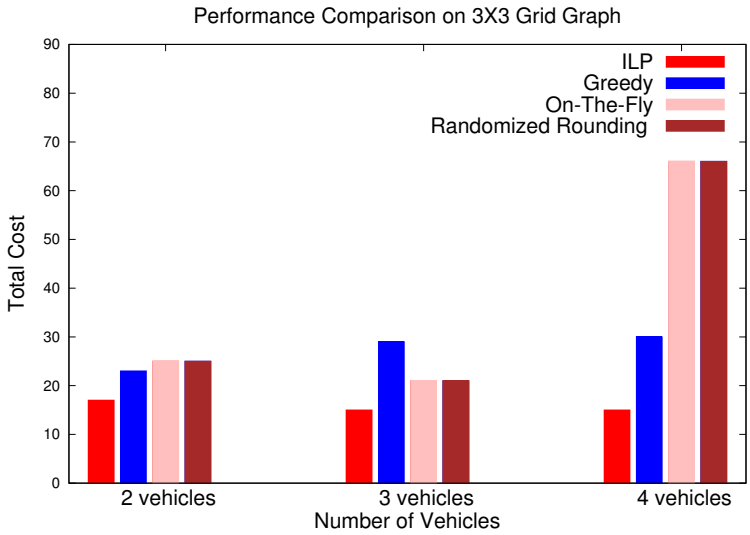


Figure 6.1: Comparison of all Solutions on a 3*3 grid

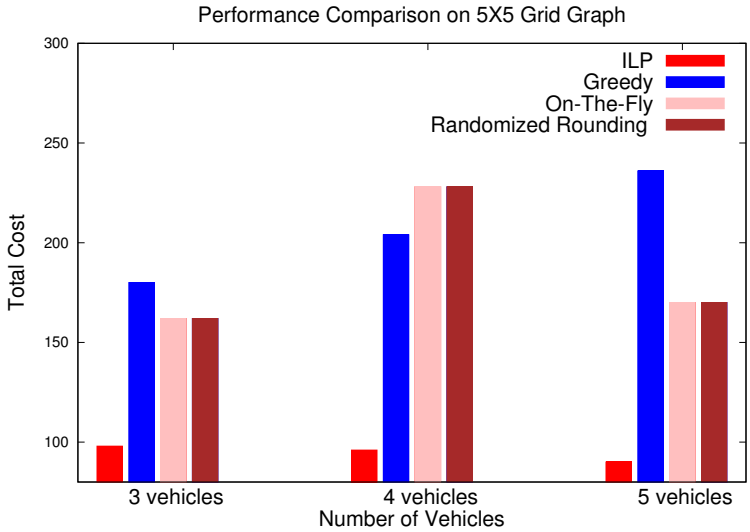


Figure 6.2: Comparison of all Solutions on a 5*5 grid

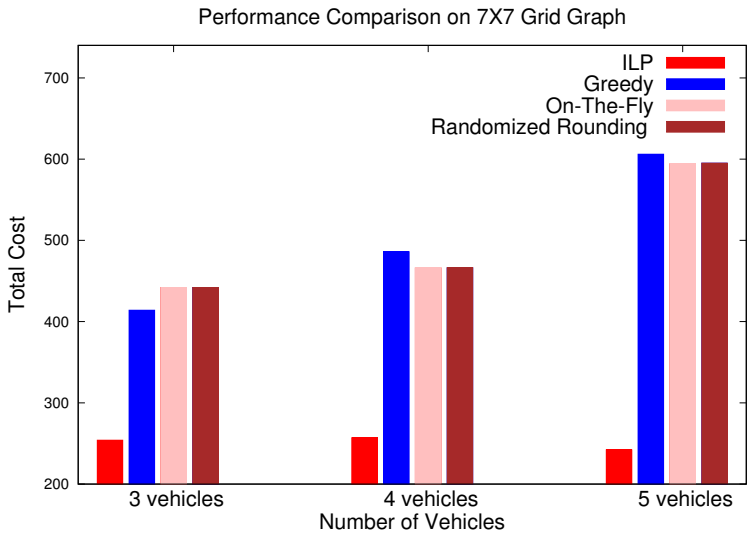


Figure 6.3: Comparison of all Solutions on a 7*7 grid

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 grid, 7*7 grid we assign the cost of service as 2, 4 respectively for all vehicles. All our simulations are run with 95 $\%$ confidence interval.

We acknowledge that the ILP solution outperforms both Algorithm 3 and the on-the-fly solution. 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.

- **Comparison of Solutions for Small Cyclic Graphs.**

  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 1 to travel and all vehicles having the servicing cost as 2. The greedy algorithm designed does not allow vehicles to travel across same nodes. But our distributed approximation algorithm and the on-the-fly

algorithm does allow vehicles to travel to the same node during the execution of the algorithm. For cyclic graphs, if a vehicle decides to choose a path that has already been taken, the vehicle is forced to continue its work in the chosen path whereas this was not the case in grid converted to complete graph case. In grid graphs, if two or more vehicles travel to a same node in any communication round, there are chances that they might not travel to a common node for travel. But for cyclic graphs, this guarantee cannot be given and vehicles might be forced to follow one another until the termination is reached without servicing any nodes in its path. For these reasons, our algorithm does not seem to perform at the same level as these algorithms. The graphs comparing simulations results on a cyclic graph having 12, 24, 48 nodes is shown in Figures 6.4, 6.5, 6.6 respectively.

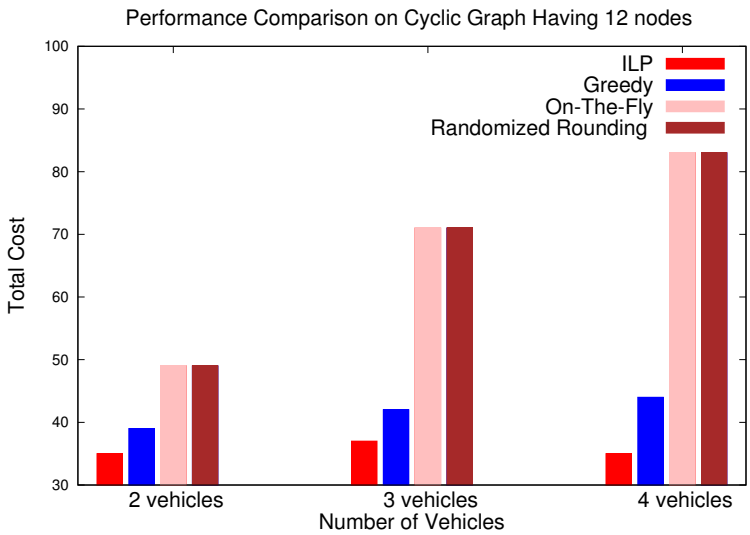


Figure 6.4: Comparison of all Solutions on a Cyclic Graph Having 12 nodes

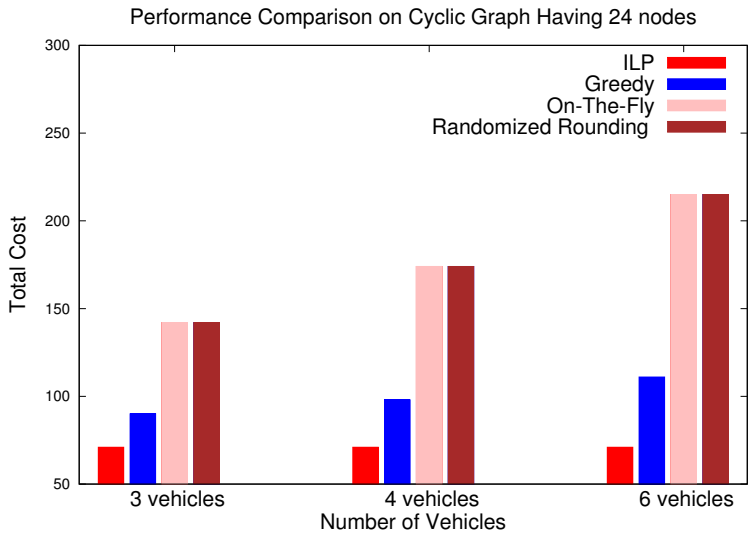


Figure 6.5: Comparison of all Solutions on a Cyclic Graph Having 24 nodes

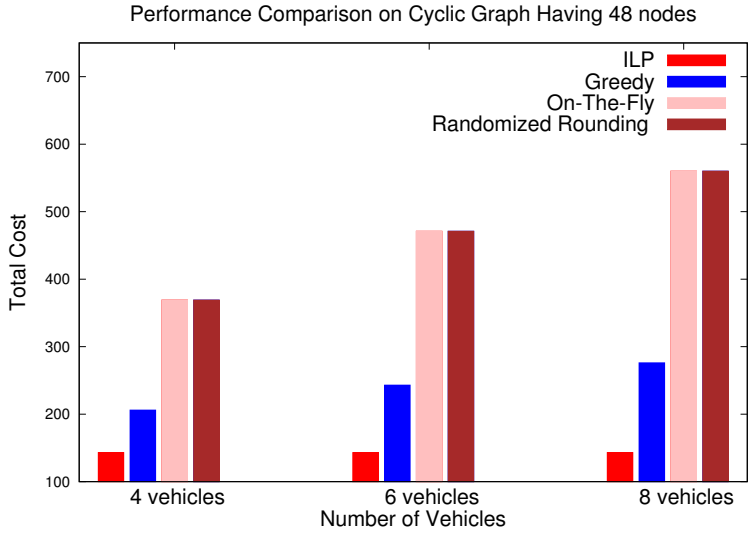


Figure 6.6: Comparison of all Solutions on a Cyclic Graph Having 48 nodes

- **Impact of value of *k*.** Here we only show the comparison results on cyclic graphs. We place the vehicles in depots equally far from each other to find the impact of k. We have the same cost of travel and service for all vehicles as 2. We can see that when k values are increased the total cost is decreased. When $k >= 2$, the total cost is at is minimum. This is because the vehicles are able to decide their best paths with the assigned k-value. So any increase of k value after this yields the same result. So it is better we always choose $k \leq |A|$ to get a better total

Cyclic Graph: Vehicles having same travel and service costs							
Value of k	24 nodes, 4 vehicles	50 nodes, 5 vehicles	120 nodes, 6 vehicles	210 nodes, 7 vehicles	320 nodes, 8 vehicles	450 nodes, 9 vehicles	600 nodes,10 vehicles
k=1	452	975	2400	4235	6480	9135	12200
k=2	452	975	2400	4165	6480	9135	12200
k=3	436	919	2196	3971	6072	8559	11416
k=4	436	919	2196	3971	6072	8559	11416
k=5	-	919	2196	3971	6072	8559	11416
k=6	-	-	2196	3971	6072	8559	11416
k=7	-	-	-	3971	6072	8559	11416
k=8	-	-	-	-	6072	8559	11416
k=9	-	-	-	-	-	8559	11416
k=10	-	-	-	-	-	-	11416

Table 6.1: Impact of k on Cyclic Graphs Having Same Travel and Service Costs for all vehicles.

cost. The tabulated results comparing the performance of the *on-the-fly* algorithm for different values of k are shown in Table 6.1.

Chapter 7

Conclusion and Future Work

7.1 Summary

In this thesis, we studied the problem of distributed open multi-depot vehicle routing problem (OMDVRP). In particular,

- We proposed a synchronous distributed approximation algorithm that solves OMDVRP. The algorithm yielded fractional values for a given instance of the OMDVRP. We have shown that the approximation ratio of this algorithm is

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

  where n is the size of the graph, ρ is the highest service or travel cost across all vehicles. We obtained this ratio using a primal-dual linear programming technique.

- To obtain integer values from the fractional solutions of distributed approximation algorithm, we used (1) a simple rounding technique called *on-the-fly* algorithm, which preserved 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 n is the size of the graph, ρ is the highest service or travel cost across all vehicles and m is the number of vehicles.

- Both algorithms were fully implemented on TCP/IP protocol stack. We reported

the results of simulations that compare the results of ILP, greedy, distributed approximation after rounding and on-the-fly algorithms.

7.2 Future Work

There are numerous open problems for further research:

- **Capacity Constraint.** In this thesis, we assumed that the vehicles have enough capacity to service all the nodes in the graph by itself. But in real-life scenarios, vehicles come with a capacity limit, beyond which they cannot perform a service or a travel. For instance, UASs flight time is bounded by energy limits. An important challenge would be to design a distributed algorithm for the cases where vehicles come with different capacity constraints.

- **Fault-tolerance.** We also assumed that vehicles do not crash or stop their work between their respective paths. One can design algorithms that can tolerate the different type of faults like crash faults, hardware failures, etc. It would be interesting to see how other vehicles cope up with such a situation and service the entire graph.

- **Number of vehicles.** In our thesis, we had fixed the number of vehicles that can take part in the OMDVRP. It would be interesting to find the exact number of vehicles required to produce the optimal solution to service a given graph with a set of costs for each vehicle.

- **Unknown/Stochastic demands** We solved the OMDVRP for the case where cities have constant known demands. Designing an algorithm that solves the case where cities have unknown or stochastic demands using a distributed algorithm would be a very interesting problem.

- **Dynamic graph.** We solved the OMDVRP for fixed graphs. In real time scenarios, certain cities tend to have sudden demands which would be better served in the present instance of the problem. In this case, the graphs are not known fully and

vehicles get to know their demands only after reaching a city. One can design a probabilistic distributed algorithm to solve this problem.

- **Implementation on UASs.** The immediate future work from this thesis is to implement this algorithm on UASs and record its working using different settings like vehicles with different travel speed,servicing speed, different graphs etc.

Bibliography

[Agmon et al., 2011] Agmon, N., Kaminka, G. A., and Kraus, S. (2011). Multi-robot adversarial patrolling: facing a full-knowledge opponent. *Journal of Artificial Intelligence Research*, pages 887–916.

[Bachem and Kern, 1992] Bachem, A. and Kern, W. (1992). Linear programming duality. In *Linear Programming Duality*, pages 89–111. Springer.

[Bektas, 2006] Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219.

[Brandão, 2004] Brandão, J. (2004). A tabu search algorithm for the open vehicle routing problem. *European Journal of Operational Research*, 157(3):552–564.

[Dantzig and Ramser, 1959] Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management science*, 6(1):80–91.

[Dorigo, 2007] Dorigo, M. (2007). Ant colony optimization. *Scholarpedia*, 2(3):1461.

[Dorigo and Birattari, 2007] Dorigo, M. and Birattari, M. (2007). Swarm intelligence. *Scholarpedia*, 2(9):1462.

[Dorigo et al., 2006] Dorigo, M., Birattari, M., and Stützle, T. (2006). Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39.

[Elkin, 2004] Elkin, M. (2004). Distributed approximation: a survey. *ACM SIGACT News*, 35(4):40–57.

[Fu et al., 2005] Fu, Z., Eglese, R., and Li, L. Y. (2005). A new tabu search heuristic for the open vehicle routing problem. *Journal of the operational Research Society*, 56(3):267–274.

[Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549.

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

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

[Junjie and Dingwei, 2006] Junjie, P. and Dingwei, W. (2006). An ant colony optimization algorithm for multiple travelling salesman problem. In *Innovative Computing, Information and Control, 2006. ICICIC'06. First International Conference on*, volume 1, pages 210–213. IEEE.

[Kennedy, 2011] Kennedy, J. (2011). Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer.

[Kong et al., 2006] Kong, C. S., Peng, N. A., and Rekleitis, I. (2006). Distributed coverage with multi-robot system. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2423–2429. IEEE.

[Kuhn and Wattenhofer, 2005] Kuhn, F. and Wattenhofer, R. (2005). Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310.

[Laporte, 1992] Laporte, G. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358.

[Li et al., 2009] Li, X., Tian, P., and Leung, S. C. (2009). An ant colony optimization metaheuristic hybridized with tabu search for open vehicle routing problems. *Journal of the Operational Research Society*, 60(7):1012–1025.

[Lynch, 1996] Lynch, N. (1996). *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA.

[MirHassani and Abolghasemi, 2011] MirHassani, S. and Abolghasemi, N. (2011). A particle swarm optimization algorithm for open vehicle routing problem. *Expert Systems with Applications*, 38(9):11547–11551.

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

[Moscibroda and Wattenhofer, 2005b] Moscibroda, T. and Wattenhofer, R. (2005b). Facility location: distributed approximation. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 108–117. ACM.

[Pang et al., 2013] Pang, S., Li, T., Dai, F., and Yu, M. (2013). Particle swarm optimization algorithm for multi-salesman problem with time and capacity constraints. *Applied Mathematics & Information Sciences*, 7(6):2439.

[Pisinger and Ropke, 2007] Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435.

[Portugal and Rocha, 2011] Portugal, D. and Rocha, R. (2011). A survey on multi-robot patrolling algorithms. In *Technological innovation for sustainability*, pages 139–146. Springer.

[Rekleitis et al., 2004] Rekleitis, I., Lee-Shue, V., New, A. P., and Choset, H. (2004). Limited communication, multi-robot team based coverage. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3462–3468. IEEE.

[Sariklis and Powell, 2000] Sariklis, D. and Powell, S. (2000). A heuristic method for the open vehicle routing problem. *Journal of the Operational Research Society*, 51(5):564–573.

[Sheng et al., 2006] Sheng, W., Yang, Q., Tan, J., and Xi, N. (2006). Distributed multi-robot coordination in area exploration. *Robotics and Autonomous Systems*, 54(12):945–955.

[Shi et al., 2007] Shi, X. H., Liang, Y. C., Lee, H. P., Lu, C., and Wang, Q. (2007). Particle swarm optimization-based algorithms for tsp and generalized tsp. *Information Processing Letters*, 103(5):169–176.

[Tarantilis and Kiranoudis, 2002] Tarantilis, C. and Kiranoudis, C. (2002). Distribution of fresh meat. *Journal of Food Engineering*, 51(1):85–91.

[Tarantilis et al., 2004] Tarantilis, C. D., Diakoulaki, D., and Kiranoudis, C. T. (2004). Combination of geographical information system and efficient routing algorithms for real life distribution operations. *European Journal of Operational Research*, 152(2):437–453.

[Zelinsky, 1992] Zelinsky, A. (1992). A mobile robot exploration algorithm. *Robotics and Automation, IEEE Transactions on*, 8(6):707–717.