Abnormality Management in Spatial Crowdsourcing for Multi-skilled Workers Assignment

by

Srinandan Kota

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Goce Trajcevski, Major Professor

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

NOMENCLATURE

VRPTW	Vehicle Routing Problem with Time Windows
GLS	Guided Local Search

ACKNOWLEDGMENTS

I would like to thank my major professor, Dr. Goce Trajcevski, for giving me an opportunity to work under his guidance and be a part of this research project. I am grateful to him for seeing potential in me, commending me on my work, constructive criticism, and being patient when my performance wasn't at my best.

In addition, I would also like to thank my friends, colleagues, the department faculty and staff for making my time at the Department of Computer Engineering, and Iowa State University a wonderful experience. I want to also offer my appreciation to students working on similar problems in this research area at the university and numerous other well-wishers who have provided feedback and information that helped me complete this project.

Last but not the least I would like to thank my parents, and my brother who have always been a constant source of support and motivation to me throughout my journey.

ABSTRACT

Crowdsourcing is dependent on a number of skilled workers who are needed to accomplish spatial tasks. This has been an active area of research and is gaining wide popularity now. Most of these tasks can be completed online due to convenience, but this method fails when there is a need of completing a task at actual physical locations. This has led to a new area called Spatial crowd sourcing [1] that consists of location-specific tasks that require people who can accomplish them to physically be at specific locations. The tasks which require specific skillsets, completion times or other constraints are matched with workers who can meet these constraints and complete them. In this report we consider a situation where the jobs are at different locations with sequential sub-tasks, each with time and skill constraints, and are to be completed within the given interval by workers who have those required skills and are dispersed. The aim is to finish a majority of tasks in the environment before a final cap time given the constraints of this environment. First workers are assigned to tasks appropriately so that each worker has the skill needed to complete each of the tasks allocated. After the assignment is complete, a variant of the vehicle routing problem called vehicle routing problem with time windows (VRPTW) is used to assign these workers the paths and visiting times that they need to follow to reach specific task locations and finish them within the required time intervals. The vehicle routing problem with time windows (VRPTW) is a generalization of the vehicle routing problem where the service of a customer can begin within the time window defined by the earliest and the latest times when the customer will permit the start of service [2]. We also consider the case when a worker cannot reach a particular task location in an abnormal situation and perform a re-assignment that does not need to re-assign tasks to all workers and is faster. By following these approaches, we aim to create a technique that can be applied to many real-world problems in the spatial crowd-sourcing environment with such practical events.

CHAPTER 1. OVERVIEW

Most tasks in the real world are time bound and require particular skills in order to complete them by the end of a day. Also, there may be many workers who can finish these tasks and they can start from different locations. There also may be scenarios where a worker cannot finish a task assigned to him. An important solution to this problem will need to consider efficient assignments of tasks to workers for such situations. We try to provide solutions to these problems and also suggest a technique which allows us to perform a re-assignment without having to perform complete assignments of all tasks to all workers when a worker notifies that he cannot perform a task. This leads to less load on the server platform.

1.1 Spatial Crowdsourcing

The term ‘Crowdsourcing’ was first introduced by Jeff Howe in a Wired magazine article titled ‘The Rise of Crowd-sourcing’ in June 2006 as follows: “Crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call.” It can be performed by individuals or groups. This field has wide ranging applications in the field of data mining and software engineering. Some of the terms used to represent crowdsourcing is social computing, crowdsensing, crowd computing, crowd wisdom, smart mobs and mass collaboration. Typically, this happens online and there are many commercial and academic platforms for this field of computing. An example is Amazon Mechanical Turk (AMT) that publishes tasks which can be completed by people. The workers who receive the tasks or request the tasks do not usually have to be in the same location. However, there are many tasks that require workers to be at a particular location. This requirement and the availability of mobile phones have led to a new form of crowdsourcing called spatial crowdsourcing.

Spatial Crowdsourcing is an emerging field in crowdsourcing which needs workers to move to a particular location to perform the task. This type of crowd wisdom needs people to gather, analyze and disseminate geographical with or without social information. In this a requester can commission workers to perform spatial tasks (tasks related to geographical location and time). There are many projects by companies and governments in this area like the Open Street Map, Google MapMaker and Wikimapia.

1.2 Challenges of Spatial Crowdsourcing

These are some of the most common issues that a crowdsourcing system faces-

1) Task formulation: It is very important to formulate tasks and this deals with questions like if the task needs to be well defined and if each larger task needs to be split into smaller tasks.
2) Task assignment: The issue here is to assign a task to a worker or workers with some constraints like minimum budget allocation, reliability or stringent time requirements.
3) Incentive mechanism: Some tasks are reward driven and there needs to be an incentive-based design to motivate workers to perform tasks. These can involve money, prestige or other rewards.
4) Scalability: The design of such a system has to be scalable regardless of the number of tasks and workers.
5) Quality: The data has to be of some acceptable quality as needed by the design. For example, malicious and poor-quality data need to be removed from the system. The data also needs to be reliable and compare against the ground truth.
6) Privacy: It also important to protect some users' privacy by not sharing this data with other users.

In addition to these there are several other issues like-

1) Location awareness: This is one of the most important characteristics of the spatial crowdsourcing system and the user movement is involved in the system. Some systems check if the users have really reached the task location.
2) Workers path selection: The users need to travel to an event place and perform the tasks and we need to calculate the best paths for the workers and schedule the task sequence. Classic Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP) are commonly used to solve this issue. In VRP, all workers start from the same location and the number of workers is fixed. This is the technique we will be using for our implementation.
3) Datasets: Only few real-world data sets are available for these systems. There are no real-world data sets that can be directly used for spatial crowdsourcing. These data sets can be obtained by modifying some available datasets so that it can be applied to and be used to solve the spatial crowd sourcing problem.

1.3 Taxonomy

This is the general taxonomy of spatial crowdsourcing which helps us identify the future applications as discussed in [1]-

A. Worker model
    a. The spatial crowdsourcing model consists of human workers. Each worker can be represented by set of attributes like an identifier, geographical coordinates, user's proficiency level, expertise or any other relevant metric. Usually the workers can be classified into reward seeking workers and voluntary workers. Reward seeking

workers perform tasks to gain money or commodities and voluntary workers usually perform without rewards.

B. Task model

a. Each task is associated with a set of attributes like an identifier, deadline of the task which means the task has real-time constraints, location of the task, number of workers, incentive and other relevant parameters. The task can be further divided into single worker required tasks and multiple workers required tasks.

Once the task model and worker model are selected then we can consider task assignment and worker selection. There are two models here-

b. Server assigning tasks. The server assigns each task to workers based on the task and worker locations according to the system optimization goals such as minimizing the total travel distance. This often leads to global optimization for the system.

c. Worker selecting tasks. The server publishes the various tasks online and the worker chooses the tasks based on his preference.

In both the models, the workers' travelling routes which usually consists of the tasks and their locations must be determined. This problem becomes even more prominent when multiple tasks are assigned to workers. As the tasks are not in the same location, we need to avoid unnecessary travelling between locations by assigning paths and by scheduling tasks.

C. Response model

a. Different spatial crowdsourcing applications pose different tasks. Workers may contribute different types of data such as categorical data, continuous data, or multimedia data.

D. Optimization goal

There are usually two different focuses, one is the workers perspective and the other is the system's perspective.

a. From a worker's perspective, the goal is to maximize the total reward which may be anything that we discussed earlier in this report. To achieve this, a worker may seek as many tasks as possible on his travelling path, then the workers compete to complete these tasks. To reduce the cost, the worker may choose the best route to accomplish all the tasks, so task scheduling and path selection need to be jointly considered in the task selection.

b. From a system's perspective, the idea is to maximize the number of assigned tasks.

    i. Maximize task coverage. To achieve this, the server first collects all the locations of the workers and then tries to maximize the overall number of assigned tasks. The workers are considered to be of the same expertise level and then this is reduced to a matching problem. Other systems assign scores to each worker based on expertise levels. Higher scores are given to workers who match the expertise. Due to the nature of travelling involved, task assignment problem often needs to consider task scheduling.

ii. Minimize system costs. The total cost can be defined as the incentives paid or the travelling distances of all the workers. It also possible to use budget to maximize task coverage.

iii. Maximize data quality. A number of strategies can be applied to maximize data quality. This again depends on the definition of data quality.

iv. Minimize number of tasks with missed deadlines. Spatial tasks usually have time constraints and usually the tasks need to be completed before the deadlines. Task scheduling and path selection becomes important here.

If the system is selecting the tasks then it determines the locations of all the workers and then assigns the tasks to the workers and if the workers are choosing the tasks then the selection is based on the maximum reward he/she can achieve by completing the tasks in the system. These assignments utilize the information from worker model and task model. This report will focus on some of the taxonomies mentioned here which are applicable for our implementation and will give reasons for the choice made, propose a more complex scenario with the possibility of re-assignment of tasks to workers when a worker cannot finish a task assigned to him and provide a solution to it which can be extended to more realistic problems.

1.4 Vehicle Routing problem

The routing and scheduling of vehicles form an important part of the distribution and the transportation systems. Vehicle routing involves the design of a set of minimum cost routes, originating and terminating at a central depot, for vehicles which serve a set of customers with known demands. Each customer is serviced once and all customers are assigned to vehicles such that the given capacities are not exceeded. This is the general vehicle routing problem and many variants of this technique exist. One such variant is vehicle routing problem with time windows

(VRPTW) which adds the complexity of allowable delivery times, or time windows. In this, the service of customers (usually pick-up or delivery of goods) can begin within the time window defined by the earliest and the latest times depending on the customer's requirements. Time windows are common to problems faced by organizations that work on fixed time schedules. Some of the most popular approaches in this technique are dynamic programming algorithms to obtain integer optimal solutions with time window constraints, column generation approaches for set partitioning formulations of several VRPTW variants, approximation algorithms for VRPTW and few other recently used ones which have known to give optimal results are local search algorithms for routing in VRPTW problems [3]. We use the last one because of its efficiency.

1.5 Sub-problem of Spatial Crowd Sourcing

The problem space of spatial crowd sourcing is very large as introduced here in the report. Here the focus will be on addressing the challenges like task formulation, assignment and worker path selection. The solution in this report uses the server assigning task model with the goal of finishing a majority of tasks given a final cap time, fixed set of workers and tasks. It also tries to minimize the total time spent by each of the workers in the path followed by him/her. The assumption here is that by minimizing the time spent on a route, the worker saves his time still allowing many of the tasks to be completed within their required time windows. The workers are all considered to be of equal expertise levels and the service times are associated with the tasks which means a task is completed in the same time by any worker who is assigned the task. The report will first sequentially assign the tasks to workers based on matching skill description of the workers and tasks. We use GLS of VRPTW to solve the issue of routing and scheduling of workers to tasks with time windows in the environment. We also suggest and demonstrate a technique that performs a re-assignment that does not need complete re-assignment of tasks and workers when some

worker notifies that a task assigned to him cannot be finished in the environment, still allowing the system to complete a majority of the tasks.

CHAPTER 2. PROBLEM DEFINITION

In this section, the report provides some of the terminology used in the spatial crowdsourcing system and parameters in our environment.

2.1 Spatial Crowdsourcing system

Here we will introduce the concept of the environment and states in the system which are referred to frequently in this report. These concepts are important when we are modelling the real-word problem.

Definition 1. Environment: The entire space of workers and jobs at different instances of time consists of the environment and all the tasks in these jobs will be removed from the environment at a certain cap time at which time they become unavailable. The goal is to complete as many tasks as possible before this final time while satisfying the time requirements of these tasks given a fixed set of jobs and workers.

Definition 2. Skill sets: In this work we assume, that $\psi = \{(s_1, comp_1), (s_2, comp_2)\ldots, (s_k, comp_k)\}$ is the universe of skills in the environment which consists of a set of tuples of skills and the times required to perform tasks requiring those skills. The first parameter s_i $(1 \leq i \leq k)$ in every tuple from this set denotes the skill and each skill is associated with a completion time $comp_i$ $(1 \leq i \leq k)$ denoted by the second parameter in each tuple.

Definition 3. State: The state of the environment is any particular time instant in the environment is when any worker starts moving from his location after completing his task or when he reaches a location and stops to do any work or otherwise.

The concept of environment, skill sets, and state are used to explain our scenario and can be extended further to define new parameters in the environment. Each of these changes from one state in the environment to another are considered to be evolutions which change as time proceeds in the environment. If the initial assignment of tasks to workers do not change in the environment as time proceeds, then these evolutions are deterministic. Now if at any particular instant of time if a moving worker stops abruptly and notifies other users that he cannot complete a particular task because of delays in traffic or any other event then this is a state that was not expected in the system. This is an abnormal state in the evolution of the states in the environment and a fast re-assignment among tasks and workers related to this particular task need to happen while checking some things like if the workers are currently not working on a task. Evolutions of state from this abnormal state have to be determined again.

2.2 Time constrained tasks in the spatial environment

This section describes the jobs and their sequential tasks in the environment. The implementations we provide tries to complete a majority of the fixed number of tasks completed given a final cap time and a set of fixed workers.

Definition 4. Jobs: Let J= $\{j_1, j_2 \ldots, j_m\}$ be a set of jobs in the environment. Each job j_i ($1 \leq i \leq m$) is comprised of sequential tasks τ_j. That is, task τ_p cannot be started unless task τ_q is done where $q < p$. Each τ_j can be written as a triplet of the form $< l_j, (\delta_j,\delta_k), s_j >$ where l_j is the location of the task represented by x and y co-ordinate in a two dimensional plane, δ_j indicates the start time by which the servicing of the task needs to start and δ_k indicates the time at which it needs to be completed and s_j is the skill required to finish this task and it is a skill from the set ψ with a completion time associated with it.

The jobs and their tasks in the environment each have a unique ID and the number of jobs and tasks in the system are fixed. The workers need to arrive at the task location and finish servicing them in the time window specified for a task. Each task also requires only one skill and has a time of completion associated with it. If a task is allocated to a worker and completed, then it is removed from the environment. All these tasks will be removed from the environment at a final cap time t_f beyond which these tasks will be unavailable in the environment.

2.3 Multi skilled workers

We consider multi-skilled workers in our environment. Each of these workers have one or multiple skills from ψ and can perform services for tasks that require his/her skills.

Definition 5. Workers: Let $W = \{w_1, w_2\ldots.., w_n\}$ be a set of workers in the environment. Each worker w_i $(1 \leq i \leq n)$ is recognized by a set of tuples of the form $<l_i(t), S_i, speed_i>$,where $l_i(t)$ is the location of the worker at timestamp t, S_i is a subset of skills that he/she has from ψ and $speed_i$ is the travelling speed of the worker which determines his travel time given the distance between locations.

In this report, the speed of all the workers are assumed to be the same and all workers start from different locations. So, the travel times of workers depends mainly on the distance between the locations of workers and tasks. The number of workers is assumed to be a constant in the environment and each worker has a unique ID associated with him to locate him.

2.4 Trajectory

Trajectories have information like the route, the order in which tasks are to be completed and the earliest arrival and latest departure times of each location the worker visits in his route. Each worker has a start location in the trajectory which does not have any service time and does no work

at that location. A worker after completing his tasks returns back to his starting location in his assigned trajectory.

Definition 6. At any instant of time in the environment, the workers are assigned tasks from different jobs in the form of a sequence (τ_0, τ_1, τ_2.... τ_n, τ_0) such that each worker has a set of tasks formed < τ_1, τ_2.... τ_n> that the worker needs to travel to and finish servicing in order. τ_0 is the starting position of a worker which requires no servicing. Here a worker starts from a location τ_0, moves to task τ_1 and then τ_2 and his last task will be τ_n. He then returns to his start location. Also, each task τ_i $(1 \leq i \leq n)$ in the sequence has tuple of times (arr_i, $dept_i$) where arr_i represents the earliest arrival time at the task location and $dept_i$ represents the latest departure time from the task location. The location τ_0 usually has the same arr_i and $dept_i$ or has $dept_i$ as the final cap time and there is no service time associated with the location. The initial assignment of tasks is done in such a way that a maximum number of the tasks in the environment will be completed by the final time t_f. When there is an abnormal change in the environment such as defined previously, there needs to be a reassignment among the workers who are not busy and could have reached this task location, their uncompleted tasks, the worker who could not perform the task and his/her unfinished tasks such that new trajectories are assigned to these workers. The goal is to still complete a majority of the initially assigned tasks in the environment before t_f. The worker who cannot complete this task which causes abnormality is not assigned this task in the new trajectory. In other words, only a part of these workers and tasks are affected by this abnormal change in state and we avoid a complete re-assignment of tasks and workers when it occurs in the environment.

CHAPTER 3. EXAMPLE SCENARIO

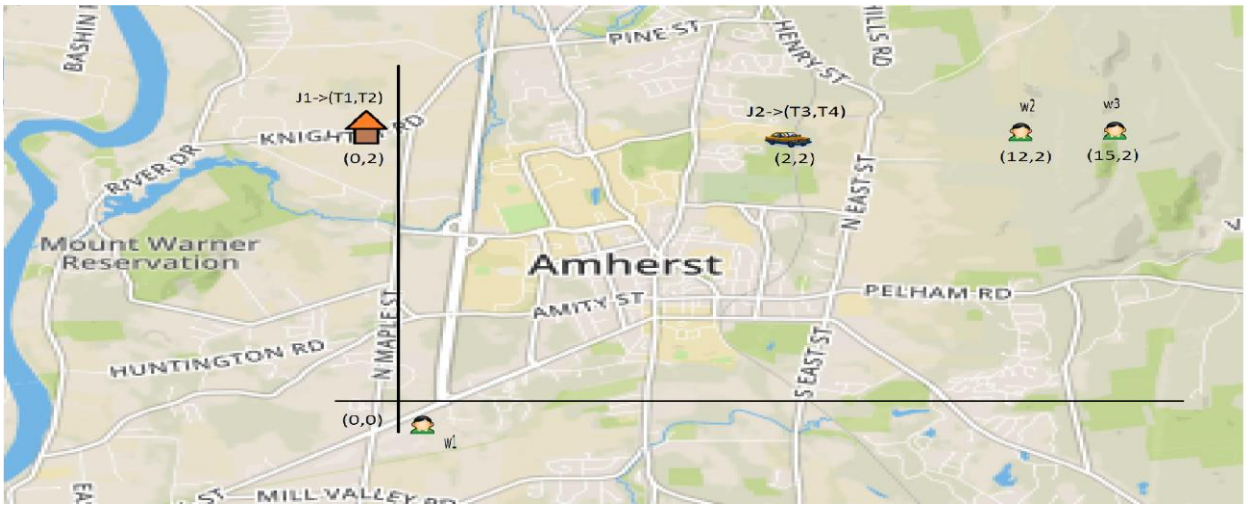

Figure 3.1 - *Location of jobs and workers against Ahmerst city backdrop*

Consider a scenario in spatial crowdsourcing against the city backdrop of Amherst in Massachusetts in Fig. 3.1, where a user wants to restructure a house and another user wants to post tasks related to the maintenance of a car. However, the job of restructuring this house has sub-tasks like repairing the house, which requires a skill (repair), and painting which requires another skill (paint). Similarly, the job related to the maintenance of the car may have sub-tasks like washing, which requires a skill (wash), and cleaning which requires another skill (clean). We consider the co-ordinates in the scenario to be relative to each other for ease of description of events. Each of these tasks have service time associated with it as shown in Table 3.3. There is a final cap time of t = 50 before which majority of tasks posted in the environment need to be completed and there are many skilled workers that can accomplish one or some of these tasks. All workers with a particular skill take the same amount of time to complete a task requiring that skill.

Table 3.1 - *Jobs, their descriptions and sub-tasks*

Job ID	Job Description	Sub-tasks
J1	House-restructuring	T1, T2
J2	Car maintenance	T3, T4

Table 3.2 - *Worker/Task Skills*

Worker/Task	skill key set
w1	a1, a3
w2	a2
w3	a3, a4
T1	a1
T2	a2
T3	a3
T4	a4

Table 3.3 - *Description of Skills and time required to complete them*

Skill key	Skill	Service time
a1	repair	5
a2	paint	3
a3	wash	5
a4	clean	5

Table 3.4 - *Task, Location and validity period*

Task id	(x, y) co-ordinate	Time window- (time to start, time to end)
T1	0, 2	0, 10
T2	0, 2	10, 15
T3	2,2	0, 20
T4	2,2	20, 30

In this scenario, let the user post a spatial job with ID - J1 with sub-tasks having IDs - T1 and T2, and another spatial job with ID - J2 with sub-tasks having IDs - T3 and T4, as shown in Fig. 3.1, in the spatial crowdsourcing system, which requires a set of skills (given in Tables 3.2 and 3.3). The jobs IDs, descriptions and their tasks are shown in Table 3.1. The location of these

tasks and windows of time to arrive and complete these tasks are shown in Table 3.4. In Fig. 3.1, there are three workers, w1, w2 and w3, each of whom has a different set of skills as given in Table 3.2. For example, worker w1 has the skill set repair and wash.

Table 3.5 - *Worker ID, Worker Initial Location, Travelling speed*

Worker	Current location - (x, y) co-ordinate at time t = 0	Speed of travelling (Mph)
w1	(0,0)	1
w2	(12,2)	1
w3	(15,2)	1

In addition, each worker has a current location represented at a specific time and a travelling speed as shown in Table 3.5. Moreover, all workers have the same moving velocities of one unit in this example scenario which makes the travelling time the same as his distances between tasks and worker locations. The workers start from different locations and all the workers are assumed to depart from their visited locations from their latest possible departure times associated with that location for ease of explanation. If they complete early, they are assumed to wait at their current location in this scenario. To accomplish the spatial job J1 (i.e., repair and paint), the spatial crowdsourcing platform needs to select a best subset of workers from w1, w2 and w3, such that the union of their skill sets can cover the required skill set of task T1 and T2, and, moreover, workers can travel to the location of T1 and T2 under the constraints of time windows (earliest arrival and latest departure times) and service them. For example, we can assign task T1 to worker w1 and task T2 to worker w2 whose skills can cover all the required skills of J1 and they can satisfy the time requirements of this task. Similarly, task T4 is assigned to w3. T3 which can be assigned to w1 and w3 is assigned to w1 as he can complete this task within the time window and is available first. Here once a task is assigned to a worker then it is removed from the spatial crowd sourcing platform to avoid duplicate assignment.

Table 3.6 - *Important events and descriptors at different time instants in the environment*

Time instant	State	Worker w1 travel path	Worker w2 travel path	Worker w3 travel path	Worker w1 trajectory	Worker w2 trajectory	Worker w3 trajectory	Tasks not assigned	Tasks completed	Description of Events
t = 0	-	(0,0)	(12,2)	(15,2)	{w1, T1, T3, w1}	{w2, T2, w2}	{w3, T4, w3}	-	-	All workers are assigned initial trajectories
t = 0	1	Moving towards (0,2)	Moving towards (0,2)	Moving towards (2,2)	{T1, T3, w1}	{T2, w2}	{T4, w3}	-	-	All workers move to their next tasks; Previous tasks or locations are removed from trajectories
t = 2	2	(0,2)	Moving towards (0,2)	Moving towards (2,2)	{T3, w1}	{T2, w2}	{T4, w3}	-	-	Worker w1 reached and started working on task T1
t = 10	3	Moving towards (2,2)	Moving towards (0,2)	Moving towards (2,2)	{T3, w1}	{T2, w2}	{T4, w3}	-	T1	Worker w1 moving towards next task in his trajectory
t = 11	3a	(1,2)	Moving towards (0,2)	(4,2)	-	{T2, w2}	-	T3, T4	T1	Worker w1 cannot reach task T3, Reassignment for w1, w3, T3 and T4
t = 12	4	(1,2)	(0,2)	Moving towards (2,2)	-	{w2}	{T3, T4, w3}	-	T1	Worker w1 does not have any tasks and reaches start location, worker w3 is assigned new trajectory and w2 is not affected and reaches Task T2
t = 13	5	(1,2)	(0,2)	(2,2)	-	{w2}	{T4, w3}	-	T1	Worker w2 is working on his task T2, w3 reaches task T3 and starts working on it
t = 15	6	(1,2)	Moving towards (12,2)	(2,2)	-	{w2}	{T4, w3}	-	T1, T2	Worker w2 starts moving towards his start location
t= 18	6	(1,2)	Moving towards (12,2)	(2,2)	-	{w2}	{T4, w3}	-	T1, T2, T3	Worker w3 finishes task T3
t = 20	6	(1,2)	Moving towards (12,2)	(2,2)	-	{w2}	{w3}	-	T1, T2, T3	Worker w3 starts task T4
t = 25	6	(1,2)	Moving towards (12,2)	(2,2)	-	-	{w3}	-	T1, T2, T3, T4	Worker w3 finishes task T4
t = 27	7	(1,2)	(12,2)	(2,2)	-	-	{w3}	-	T1, T2, T3, T4	Worker w2 reaches his start location
t = 30	8	(1,2)	(12,2)	Moving towards (4, 2)	-	-	{w3}	-	T1, T2, T3, T4	Worker w3 starts towards his start location
t = 32	9	(1,2)	(12,2)	(15,2)	-	-	-	-	T1, T2, T3, T4	Worker w3 reaches his start location; No more state changes in the environment
t =50	9	(1,2)	(12,2)	(15,2)	-	-	-	-	T1, T2, T3, T4	Final cap time reached

The state changes and a brief description of important events at different significant time instants are shown in Table 3.6 and the trajectories are represented by IDs of locations from which their location and times can be determined. This table gives information about the change of states, description of events, travelling path of workers, trajectories of workers and information about completed tasks. The states are a metric that can be used to detect an abnormality and perform a re-assignment in our work as explained already. It occurs when a worker starts from a location or stops at a location to perform a task or otherwise and it is used to explain this example scenario. The evolutions of states and the events that followed till State 3 and after it would have been deterministic if the initial assignments described above were fixed. However, at time t=11 after leaving from his current task location at time t = 10, w1 notifies all the constituents of the system that he cannot do task T3 as he has faced a problem due to traffic. This is an abnormal change in the state of the system as Worker w1 stops abruptly. This is State 3a of the environment, which was not present in the states determined earlier, and there needs to be a reassignment of tasks to workers related to this unfinished task as described in Section 2 such that a majority of the tasks in the environment are completed within their time windows using algorithms mentioned in Section 5. Now a re-assignment between workers w1, w3 and tasks T3, T4 need to happen. Tasks T3 and T4 are reassigned to worker w3 as he has the skillsets required to perform these tasks, can satisfy their time requirements and is currently not working on a task and is not busy. Worker w1 is not assigned task T3 in this re-assignment and as he has no other tasks left to do, he returns to the start location of his new trajectory. Worker w2 is not considered for this re-assignment as he cannot perform this task and retains his initial trajectory. There is a fast re-assignment at this point. At time t=32 all workers would have returned to their starting locations of their current trajectories. There are no more state changes till the final cap time of the environment.

CHAPTER 4. FRAMEWORK

The framework we propose uses two main stages. In the first stage we keep a record of all the tasks a worker can perform based only on the skillsets. The second stage then assigns the trajectories (order of the task, times and the route) to each worker in a sequence. We also consider reassignment. Some of the common terms used to describe this are defined in Table 4.1

Table 4.1 - *Symbols and Descriptions*

Symbols	Descriptions
s_i	Skill with ID – i
$comp_i$	Completion time associated with skill s_i
ψ	Universe of skillsets containing s_i and $comp_i$
τ_i	Sequential task i
j_i	Job with ID - i comprised of a number of sequential tasks τ_j
J	Set of Jobs J
w_i	Worker with ID - i
L	List of task lists of all workers
L_i	list of tasks that each worker w_i can complete based on skillset
M_i	Trajectory that each worker w_i is assigned
M	List of trajectories
F	Set of features
$\boldsymbol{\lambda}$	penalty factor
c	cost vector
c_i	cost of feature i
f_i	Indicator function for feature i ϵ F f_i (S) = 1 if the feature i is in solution S, and 0 otherwise.
p	Penalty vector
p_i	Integer number of times feature i has been penalized
(x_i, y_i)	Notification co-ordinates
τ_i'	Unreachable task tuple
W'	Workers who are not currently working and can perform task τ_i'
w_n	Worker who sends notification
L_n	Task list of w_n
w_i'	Worker in W'
t	Time at which notification is received
M_i'	Current trajectory of worker w_i'

CHAPTER 5. FRAMEWORK COMPONENTS

The implementation is divided into main components like assigning tasks to workers based on skillsets, assignment of routes to workers using GLS for VRPTW and a quick re-assignment of a small subset of workers and tasks when a worker cannot reach a task. A brief discussion is given in this section for each of the components. The last component performs a fast assignment that does not displace other workers, who are not related to or cannot do the notified task, and their tasks. This leads to fast assignment times even in an event of abnormality in the environment.

5.1 Assignment of tasks to workers based on skillsets

Algorithm 1. Initialization of tasks to workers based on skillsets

Input: Jobs J, Workers W
Output: Task lists of all workers L

1. **For** each worker w_i in W:
2. **For** skill s_i in skillset of w:
3. If skill s_i is in skill required by any task τ_i in J:
4. Add τ_i to L_i

This is the first stage of our implementation where the jobs and workers are inputs of the algorithm and the output is a list of lists where each list is associated with a worker and consists of the all the tasks in the environment that this worker can complete based only on skillsets. Line 1 iterates through each worker, and for each worker we iterate through each skill of this worker in Line 2 and we check for all the tasks in Job J which requires this skill in Line 3. We then add this task to the task list L_i of this worker in Line 4. This gives us a set of workers and a list of all the tasks a worker can perform in the environment.

5.2 Assignment of trajectories

Algorithm 2. Trajectory assignment

Input: Workers W, List of task lists L
Output: Trajectories of all workers M

1. **For** each worker w_i in W:
2. Apply **GLS of VRPTW** for tasks in L_i and store them in trajectory M_i of Worker w_i
3. Remove tasks τ_i from task list L which is present in M_i

This is the part of our implementation where workers are assigned tasks in an order which satisfies the time requirements of these tasks while trying to complete a majority of tasks in the environment. The input to this algorithm is the set of workers W and list of tasks L and outputs are an initial set of trajectories which each worker is assigned in the environment. Line 1 traverses through each worker in the worker set W. The modified GLS for VRPTW is applied to each task list of a worker in Line 2. This assigns an order in which tasks need to be completed and a route that the worker needs to follow, satisfying the time requirements of the sequential tasks, and outputs a list of ordered tasks with suggested visiting times for each worker. The trajectory is then updated to a set M that stores all the worker IDs and their respective current trajectories. Finally, we remove all occurrences of tasks in L which are present in the obtained trajectory in Line 3 to avoid duplicate assignment of tasks.

5.3 Guided Local Search for Vehicle Routing Problem

We apply a method called Guided Local Search for VRPTW to obtain a trajectory for a worker by treating him as a vehicle and the task locations as the customer locations. The starting point of a worker in the current trajectory is the depot location of the vehicle.

5.3.1 General guided local search

Algorithm 3. GLS for VRPTW
Input: Set of features F, Cost Vector c, Penalty factor λ
Output: $\mathbf{S^*}$

```
1.  p:=O
2.  S:= InitialSolution()
3.  S:= LocalSearch(S,p)
4.  S*:=S
5.  While not StoppingCondition() do
6.      f:= ChoosePenaltyFeatures(S,p)
7.      forall g in f do
8.          pg:= pg+1
9.      S:= LocalSearch(S,p)
10.     if O(S) < O(S*) then
11.         S*:=S
12. S*:= LocalSearch(S*,O)
13. RETURN S*
```

Guided local search (GLS) is a metaheuristic which is based on penalties. It moves out of a local minimum by penalizing particular solution features that according to it should not appear in a near-optimal solution. It uses a modified objective function, augmented with a set of penalty terms on these features. The local search method is then invoked to improve this function and we use the default method provided in [4]. This cycle of local search and update to penalty term can be repeated frequently. GLS tracks penalties applied via p.

Assuming O(S) is the original objective function for the problem, GLS defines an augmented objective function: $O'(S) = O(S) + \lambda \Sigma_{i \in F} f_i(S) p_i c_i$ and requires a local search procedure that minimizes it. The LocalSearch(S,p) that performs a local search, starting at solution S based on the improvement with objective function O' is provided by the user.

GLS provides a function called ChoosePenaltyFeatures(S,p) which takes a solution and the current penalties and returns the set of features to be penalized. GLS penalizes the most costly

features in the current solution, weighted by the number of times the feature has already been penalized. GLS chooses the features i ϵ F for which $c_i/(p_i+1)$ is the largest among the feature in S. Usually only one feature is selected.

This is assuming InitialSolution() and StoppingCondition() exist. A brief explanation of this algorithm is given in [5].

5.3.2 Application of GLS for VRPTW to obtain trajectories

In the Guided Local Search for VRPTW problem we have the following descriptors,

1. Feature set F: Time windows are penalized
2. Feature Costing: We assume the cost c_a to be the service time of the location and travel time to next location
3. Penalty factor λ: 0.1
4. InitialSolution() is the default solution and LocalSearch(S,p) is the algorithm implemented by default in the software suite in [4]. More about the details of the use of this algorithm in our implementation is discussed in Section 6.

For each run of the algorithm we try to minimize the amount of time spent by a worker in the route assigned to him. This finds a path for a worker while trying to minimize all the travel and service times of a worker in his current route.

5.4 Fast re-assignment of trajectories

Algorithm 4. doFastAssignment

Input: Trajectories M, Notification co-ordinates (x_i, y_i), unreachable task τ_i', Task list L, Worker w_n

Output: Task List L

1. Find travel time and notification time t from (x_i, y_i) and M
2. W' = Find all workers who are not currently working and can perform task τ_i'
3. Remove tasks τ' completed by workers in W' till the notification time instant t and all tasks completed by workers in W not in W' from L

4. Remove $\tau_i^{'}$ from task list L_n
5. Update new locations for all workers in $W^{'}$ at notification instant t using travel time
6. For each worker $w_i^{'}$ in $W^{'}$
7. Apply **GLS for VRPTW** for tasks in his task list $L_i^{'}$ and obtain current trajectory $M_i^{'}$
8. Remove tasks in $M_i^{'}$ from L

This is the algorithm which is called when there is a notification from a worker indicating that he cannot reach a task in his initial trajectory. It needs initial trajectories obtained from algorithm 4. This algorithm is designed to be called when there is an abnormal state change in the environment which was discussed in Section 2.1. The input to the algorithm is the initial trajectories of workers, the co-ordinates at which the notification was received, the unreachable task of the worker who sent the notification, this worker's tuple which describes him in the environment and Task lists L. The output is L with the uncompleted tasks in it. In line 1 we find the current notification time based on the notification co-ordinates, the co-ordinates of the last location the worker visited who sends the notification and the co-ordinates of next location which he cannot reach using linear algebra and this gives us the current environment time and the travelling time required to reach the notification location. We find all the workers who can do this task based on simple modification and iteration on task list L, the current environment time and iterations on initial trajectories M as indicated in line 2. In Line 3 we remove all completed tasks by workers in W' from L till notification time t and we also remove all completed tasks by other workers not in W' from L using information from their initial trajectories in M to avoid duplicate assignments. In line 4 we remove unreachable task $\tau_i^{'}$ from task list L_n of worker w_n so that it is not assigned to him. We then update all the new locations of workers in $W^{'}$ in line 5 using the current notification time instant, the travelling time and the initial trajectories of these workers. A new location is assigned to these workers which are between their last visited location before the

current environment time and the next location they were supposed to visit after the current environment time in their initial trajectories. In lines 6-7, we iterate through each of the workers in W' and obtain new trajectories for them for the remaining tasks in L, from their new locations and current environment time t. In line 8, we remove the completed tasks in the new trajectories from Task lists L. This gives us new trajectories for workers which were considered for re-assignment and the tasks left unfinished. The other workers retain their initial trajectories obtained from Algorithm 4. These algorithms are simulated in our implementation on a few test cases and results are obtained.

CHAPTER 6. EXPERIMENTAL SETUP AND TESTS

The implementation was tested on a fixed set of user defined workers and tasks modified based on VRPTW datasets so that it can be reproduced for analysis. The experiments were conducted on a x64-based PC (16 GB, 512 GB SSD), running Windows 10 Home edition. The entire implementation was done using Python 3.6 and the Google OR-Tools software suite (v6.10) which is a binary distribution and can be found in [4]. All the data pre-processing to modify the datasets was done using pandas library in Python. The reason we chose this software suite and language is because of the availability of open source packages and support.

6.1 Datasets

The experiments were conducted using real data sets used for Vehicle Routing Problem with Time Windows given in the Transportation Optimization Portal of SINTEF Applied Mathematics [6]. We used our implementation to test our example scenario mentioned in Section 3. For a larger test set we tried it out with an instance of 100 customers location in Solomon's VRPTW benchmark problem named rc207. Our next test was to use 1000 customer VRPTW rc2_10_10 instance in the Gehring & Homberger benchmark. The information necessary for our implementation were customer number, their position co-ordinates, ready time, due date and service time. We prepare the data using pandas. We assigned a skill and an id based on the customer number for each customer giving us jobs in the environment. We assign a task ID for each task in these jobs. The number of workers were experimented with until we found a combination of workers that would finish all the tasks for both normal state changes and the abnormal state change. Each worker was associated with two to four skills and each task was associated with one skill from the set "repair", "wash", "clean", "paint", "dry" and "build".

We tested our implementation for both the normal state changes in the environment and when there is an abnormal state in the environment. All the important results and the parameters selected for the simulations are noted in this report in Section 6.2.

6.2 Implementation details

The implementation has a two-stage approach. We first assign all the tasks to workers based on skillsets only and then apply GLS for VRPTW for each worker using the google or-tools software suite [4] to obtain a set of distinct tasks and their suggested visit times , the order in which they need to be completed and the route in which a worker has to travel, which is his trajectory. We treat each worker as a moving vehicle so that we can apply a variant of VRPTW algorithm called GLS of VRPTW. GLS is generally the most efficient metaheuristic for vehicle routing [7]. The arc costs are defined to be the travelling times and service times of a worker, which is utilized by the solver in the google or-tools software suite to minimize the time spent by a worker in his current route. The software suite needs a data model as input. The data model consists of an array of travel times between locations, time windows of each locations, number of locations, number of vehicles and a depot location. In our case we have used a distance matrix instead of a time matrix as speed of the worker is the same and is one unit. The distance matrix for each worker is the distances between every task which he can do to every other task that he can do and distance between his current position to every other task he can do. To find the distances between locations represented by (x, y) co-ordinates, we have made use of Manhattan distances. The time windows are the requested time of visits at a task location. Initially when a worker starts form his location, we assign a time window of (0,0) indicating that he starts at time t = 0 from his location. The number of locations for a worker depends on the number of tasks he performs at any point in time. The number of vehicles is one as we consider one worker at a time and the depot location is the current

starting location of a worker. We limit the search of every GLS of VRPTW algorithm run till a fixed number of solutions are obtained as shown in Tables 6.1 and 6.2 and the best among them is treated as the final trajectory of a worker. This is the stopping condition we use. For each task assigned to a trajectory of a worker we remove it from the main task set. This is done to avoid duplication of task assignments. Every worker has his own start location in a trajectory which is his current location and he reaches back to this start location after completing all his assigned tasks. We also add time constraints to the workers so that they have a waiting time and a final cap time which is the same as the final cap time of our environment. The waiting condition allows some buffer for workers to wait on tasks if they finish a task earlier than the required time. We create dimensions which keep track of quantities that accumulate over a vehicle's route. Here we keep track of the time spent on a route by a worker. The task and worker location are also assigned a solution window which forces the worker to visit and service the locations in that time window. We also handle routing problems that have no feasible solutions and allow dropping of visits. To do this we assign what is known as penalties for all task locations. All these methods used have been explained in google or-tools software suite documentation and we have adapted it to our problem to make sure that our implementation scales for large amounts of workers and tasks. The GLS algorithm is run on every worker by treating him as a vehicle and by following the above methods to prepare our data so that it can be input to the software suite to obtain the results. After we obtain the trajectories for the environment without any abnormality, we make a note of all these trajectories. We then pick a task from a trajectory assigned to a worker and make it an unreachable task in our implementation. We use his ID and fix appropriate notification co-ordinates to be the co-ordinates after his last completed task. These are the parameters used to test our implementation for the abnormal situation. Our simulation times for the complete assignment and fast re-

assignment is recorded in this report. The solutions obtained are the trajectories which have worker and task IDs, their time intervals of arrival and departure times and the order in which they are performed along with other results which describe the simulation. The worker and task IDs in the trajectories are representative of the location which the worker has to visit and the time intervals indicates the times in which they have to visit the location associated with that ID to meet the time requirements associated with that location and stay on schedule. The algorithms discussed in Section 5 are implemented using the google or-tools software suite.

6.2.1 State changes with no abnormality

Our implementation was first tested for the situation where all the state changes in the environment is known and do not change. In other words, all workers complete all the tasks in the environment without any interference. The implementation details use the same framework components as described in section 5. The important results obtained for a few test cases and the parameters in the google or-tool software suit selected for the same are shown in Table 6.1. The first test of our implementation is shown in the second row in Table 6.1 and it simulates the example scenario with normal state changes described in Section 3. Its sample output from the simulation is shown in Appendix A.

Table 6.1 – *Simulation parameters and results for normal state changes in the environment*

Test No.	Description of Data set used for demonstration	No of Tasks	No of workers	Percentage of completed tasks	Final Cap time	Stopping condition for VRPTW (Based on number of solutions returned by Guided Local Search)	Simulation time in milliseconds for assignment of all trajectories to workers
1	Example Scenario from Section 3	4	3	100	50	1	15.628
2	rc207 instance from Solomon's VRPTW benchmark	100	10	100	1200	5	73.802
3	rc2_10_10 instance in the Gehring & Homberger benchmark	1000	70	100	7500	7	15512.269

6.2.2 State changes with an abnormality

Our implementation was then tested for the situation where there is an abnormality in the environment. In other words, when a worker cannot complete a task in the environment then there is a re-assignment of a small subset of the tasks and workers such that we can attempt to finish the unfinished task too. There is a rerouting of workers who can perform this task and are currently not working on any task. To test this, we picked a task from a trajectory of a worker obtained from each of the test cases in Table 6.1 and made it unreachable along with the co-ordinates at which the notification was received and the worker who sends this notification. The new results were noted. For now, the user defines the notification parameters like the task which cannot be reached, the worker who is sending this notification and the co-ordinates at which the notification is sent for every run of our implementation. From this and his last visited location co-ordinates and visiting times we determine the possible travel time of the worker to the notification co-ordinates and the time at which the notification may have been sent. From these times and the initial trajectories obtained without abnormal state changes, we find and update new starting locations, between location visits around the current environment time, of all the workers who are currently not working and can perform this task. We run the GLS of VRPTW algorithm to obtain new trajectories for these workers and the tasks that they can complete from this time instant and new starting co-ordinates. The worker who sends the notification is not assigned this task which caused the abnormal state in his new trajectory. We get the new trajectories of these workers. All the other workers have their initial trajectories. We record our approximate simulation time needed for this re-assignment. The main results and the parameters used to obtain them are shown in Table 6.2. A complete simulation output on the first test case of Table 6.2 is shown in Appendix B. The explanation of the simulation results is given in Appendix C.

Table 6.2 – *Simulation parameters and results for abnormal state changes in the environment*

Test No.	Description of Data set used for demonstration	No of Tasks	No of workers	ID of worker who sends notification, ID of unfinished task, x and y co-ordinates at which notification is received, Time instant of notification	Percentage of completed tasks	Final Cap time	Stopping condition for VRPTW (Based on number of solutions returned by Guided Local Search)	Simulation time in milliseconds for fast re-assignment
1	Example Scenario from Section 3	4	3	1, J2t1, (1,2), 11	100	50	1	3.024
2	rc207 instance from Solomon's VRPTW benchmark	100	10	2, J75T1, (24,75), 280	100	1200	5	12.966
3	rc2_10_10 instance in the Gehring & Homberger benchmark	1000	70	1, J5T1, (254,254), 710	100	7500	7	3674.287

CHAPTER 7. CONCLUSION

In all the existing spatial crowd sourcing solutions that have been proposed till now, there are only solutions for normal assignment of tasks to workers given constraints like minimizing the distance traveled, servicing tasks with time windows, budget and many others. They assume that workers finish tasks assigned to them. But none of the works consider the case when a worker cannot reach the task assigned to him. This report proposes an approach to solve the normal worker and task assignment problem given some of these constraints and an approach to solve this problem with an abnormality. The workers are assigned a trajectory which is a route, an order and the times in which the worker has to visit task locations to complete these tasks before a final cap time. The report then suggests a technique which does a fast re-assignment that does not require complete re-assignment of tasks to all workers when some worker cannot reach a task assigned to him or during an abnormal state change. This solution helps in finding new trajectories very quickly and still completes a majority of tasks. It is a solution to some of the problems that are faced by workers and people who post jobs in the crowd-sourcing platforms. Even though this report demonstrates the effectiveness of the solution proposed, there are further optimizations that can be done like minimizing the cost of travel of a worker by allocating a budget, allowing workers to start and reach their desired locations and many more such optimizations to provide a more robust solution to these practical problems in spatial crowdsourcing platforms. We can also improve the accuracy even further and test our implementation for larger data sets. This report proposes a simple and innovative solution to a possible problem in the spatial crowd sourcing platforms and sets a base for further improvements along this line.

REFERENCES

[1] Zhao, Yongjian & Han, Qi. (2016). Spatial Crowdsourcing: Current State and Future Directions. IEEE Communications Magazine. 54. 102-107. 10.1109/MCOM.2016.7509386.

[2] Desrochers, Martin & Desrosiers, Jacques & M Solomon, M. (1992). A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. Operations Research. 40. 342-354. 10.1287/opre.40.2.342.

[3] Nasser A. El-Sherbeny, Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods, Journal of King Saud University - Science, Volume 22, Issue 3, 2010, Pages 123-131, ISSN 1018-3647, https://doi.org/10.1016/j.jksus.2010.03.002.

[4] "Google OR-Tools," Google, [Online]. Available: https://developers.google.com/optimization/. [Accessed 4 6 2019].

[5] Kilby, Philip & Prosser, Patrick & Shaw, Paul. (2002). Guided Local Search for the Vehicle Routing Problem. 10.1007/978-1-4615-5775-3_32.

[6] "Transportation Optimization Portal," SINTEF Applied Mathematics, [Online]. Available: https://www.sintef.no/projectweb/top/. [Accessed 4 6 2019].

[7] "Guided Local Search," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Guided_Local_Search. [Accessed 4 6 2019].

[8] Yongxin Tong, Jieying She, Bolin Ding, Lei Chen, Tianyu Wo, and Ke Xu. 2016. Online minimum matching in real-time spatial data: experiments and analysis. Proc. VLDB Endow. 9, 12 (August 2016), 1053-1064. DOI: http://dx.doi.org/10.14778/2994509.2994523

[9] P. Cheng, X. Lian, L. Chen, J. Han and J. Zhao, "Task Assignment on Multi-Skill Oriented Spatial Crowdsourcing," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 8, pp. 2201-2215, 1 Aug. 2016.

[10] "FIND A BUS STOP," Greyhound, [Online]. Available: https://locations.greyhound.com/us/massachusetts. [Accessed 7 April 2019].

APPENDIX A. SIMULATION OF SCENARIO WITH NORMAL STATES

```
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/SRINANDAN KOTA/.spyder-py3/myproject/creative_component/
WJT_Scenarios/WJT_Scenario_Optimized_RouteTime/AnyLocation_Small/
WJT_Scenario_Normal_States.py', wdir='C:/Users/SRINANDAN KOTA/.spyder-py3/myproject/
creative_component/WJT_Scenarios/WJT_Scenario_Optimized_RouteTime/AnyLocation_Small')
Normal routing for workers without notifications


Skills in this environment->(Skill:Time)
{'repair': 5, 'paint': 3, 'wash': 5, 'clean': 5}


Worker information in this environment->(Initial Location, Skills, Speed)
{1: [(0, 0), ['repair', 'wash'], 1], 2: [(12, 2), ['paint'], 1], 3: [(15, 2), ['wash',
'clean'], 1]}


Job information in this environment->(Job_id:[Task_id:[Location, Time interval, skill]])
{'J1': {'J1t1': [(0, 2), (0, 10), 'repair'], 'J1t2': [(0, 2), (10, 15), 'paint']}, 'J2':
{'J2t1': [(2, 2), (0, 20), 'wash'], 'J2t2': [(2, 2), (20, 30), 'clean']}}


In the time interval(t1,t2), t1 indicates earliest arrival time and t2 indicates latest
departure time of a worker so that worker can meet his task schedules

Route for worker with id = 1:
 1 Time(0,0) -> J1t1 Time(2,10) -> J2t1 Time(9,20) -> 1 Time(18,50)


In the time interval(t1,t2), t1 indicates earliest arrival time and t2 indicates latest
departure time of a worker so that worker can meet his task schedules

Route for worker with id = 2:
 2 Time(0,0) -> J1t2 Time(12,15) -> 2 Time(27,50)


In the time interval(t1,t2), t1 indicates earliest arrival time and t2 indicates latest
departure time of a worker so that worker can meet his task schedules

Route for worker with id = 3:
 3 Time(0,0) -> J2t2 Time(20,30) -> 3 Time(38,50)



Elapsed time for assignment
15.966000000000001


All tasks completed
```

APPENDIX B. SIMULATION OF SCENARIO WITH ABNORMAL STATES

```
In [10]: runfile('C:/Users/SRINANDAN KOTA/.spyder-py3/myproject/creative_component/
WJT_Scenarios/WJT_Scenario_Optimized_RouteTime/AnyLocation_Small/
WJT_Scenario_Special_States.py', wdir='C:/Users/SRINANDAN KOTA/.spyder-py3/myproject/
creative_component/WJT_Scenarios/WJT_Scenario_Optimized_RouteTime/AnyLocation_Small')
Reloaded modules: ortools, ortools.constraint_solver, ortools.constraint_solver.pywrapcp,
ortools.constraint_solver._pywrapcp, ortools.constraint_solver.routing_enums_pb2,
ortools.constraint_solver.routing_parameters_pb2,
ortools.constraint_solver.solver_parameters_pb2
Routing for workers but when a worker cannot reach a task

Scenario with state change that is not previously determined


Skills in this environment->(Skill:Time)
{'repair': 5, 'paint': 3, 'wash': 5, 'clean': 5}


Worker information in this environment->(Initial Location, Skills, Speed)
{1: [(0, 0), ['repair', 'wash'], 1], 2: [(12, 2), ['paint'], 1], 3: [(15, 2), ['wash',
'clean'], 1]}


Job information in this environment->(Job_id:[Task_id:[Location, Time interval, skill]])
{'J1': {'J1t1': [(0, 2), (0, 10), 'repair'], 'J1t2': [(0, 2), (10, 15), 'paint']}, 'J2':
{'J2t1': [(2, 2), (0, 20), 'wash'], 'J2t2': [(2, 2), (20, 30), 'clean']}}


Printing initial trajectories for workers


2 (0,0)->J1t2 (12,15)->2 (27,50)


Notifictaion received about task J2t1  by worker 1

Workers considered for rerouting because of this task J2t1
[1, 3]

Printing updates for workers who have been affected by this notification

Worker 1 issued a  notification about J2t1 at (1,2) and at time t = 11

Intial trajectory of worker 1 till time t = 11
1 (0,0)->J1t1 (2,10)->

Displaying updates for workers who can perform this task at current time

Updates of Current worker 3 till time t = 11
Left from location of 3
Worker 3 will be available

Updating locations for these workers and getting their statuses at time t = 11

Worker 3 is currently not busy
He is at (3, 2)
Updated his current location
```

```
His new printed trajectories will start from current environment time as he is rerouted to
check if he can perform the unfinished task

Worker 1 cannot do task J2t1
He is at (1, 2)
Updated his current location
His new printed trajectories will start from current environment time as he is rerouted to
perform other tasks

Workers who can perform this task and were initially idle
No such worker is idle

Printing rest of trajectories for workers who were considered for re-assignment

New trajectories start from time t = 11
They include tasks completed after time t= 11

Displaying rest of the trajectories

Route for worker with id = 1:
 1 Time(11,11) -> 1 Time(11,50)


Route for worker with id = 3:
 3 Time(11,11) -> J2t1 Time(12,20) -> J2t2 Time(20,30) -> 3 Time(26,50)



Elapsed time for re-assignment of tasks and workers
3.024


All tasks completed
```

APPENDIX C. DETAILS OF THE SIMULATION

1. The simulation results displayed give information about the set of skills, worker information (ID, Current location, skills, speed), jobs (Job ID, Task ID) which has tasks (Task ID, Location, Time interval, skill), trajectories of workers and other important information which help us to follow the simulation results.
2. A trajectory of a worker is displayed for each worker.
3. The trajectories of workers are represented by a worker ID or task ID and the possible times in which the worker can visit that location next to them and the order in which it needs to be followed represented by '->'. This task after symbol '->' is the next task that needs to be visited. The location of tasks and workers can be obtained from the ID at any point in time and we are representing trajectories by IDs as it is visually more appealing.
4. For the abnormal state all the workers who are not re-assigned because of the notification from a worker retain their initial trajectories and they are displayed initially.
5. Details about the notification like the ID of the worker who sent it, the ID of the task which cannot be performed by this worker and the notification time are displayed in the simulation.
6. The initial trajectory of the worker who sends the notification till the notification time instant is displayed indicating he has followed that trajectory and completed those tasks. Similarly updates on workers who can perform this task are then displayed till the notification time. If a capable worker is busy, then he retains his initial trajectory and it is displayed indicating those tasks are completed.
7. For the worker who sends the notification and all the workers who are re-assigned, their new locations are updated. The new locations are between their last visited and next to be

visited location in their initial trajectories so that they can start from new co-ordinates. The new locations of the worker are updated to be the co-ordinates that they might be around during the current environment time and not their previous starting positions of initial trajectories. This information is displayed in the simulation.

8. Now the rest of the trajectories for workers who have the locations updated are displayed in the same pattern as before, starting after the notification time instant indicating they complete the tasks after this time instant and from their current updated starting locations.
9. The approximate time needed for the simulation of total assignment and re-assignment is displayed in these results.