Route Planning for Long-Term Robotics Missions

Christopher Alexander Arend Tatsch
WVU, christophertatsch@gmail.com

Route Planning for Long-Term Robotics Missions

Christopher Alexander Arend Tatsch

ROUTE PLANNING FOR LONG-TERM ROBOTICS MISSIONS

Christopher A. A. Tatsch

Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

Department of Mechanical and Aerospace Engineering

Morgantown, West Virginia
2020

Keywords: Robotics, VRP, optimization, meta-heuristic, autonomy, planning

ABSTRACT

Route Planning for Long-Term Robotics Missions

by Christopher A. A. Tatsch

Many future robotic applications such as the operation in large uncertain environment depend on a more autonomous robot. The robotics long term autonomy presents challenges on how to plan and schedule goal locations across multiple days of mission duration. This is an NP-hard problem that is infeasible to solve for an optimal solution due to the large number of vertices to visit. In some cases the robot hardware constraints also adds the requirement to return to a charging station multiple times in a long term mission. The uncertainties in the robot model and environment require the robot planner to account for them beforehand or to adapt and improve its plan during runtime. The problem to be solved in this work is how to plan multiple day routes for a robot where all predefined locations must be visited only a single time and at each route the robot must start and return to the same initial position while respecting the daily maximum operation time constraint.

The proposed solution uses problem definitions from the delivery industry and compares various metaheuristic based techniques for planning and scheduling the multiple day routes for a robotic mission. Therefore the problem of planning multiple day routes for a robot is modeled as a time constrained Vehicle Routing Problem where the robot daily plan is limited by how long the robot with a full charge can operate. The costs are modeled as the time a robot takes to move among locations considering robot and environment characteristics. The solution for this method is obtained in a two step process where a greedy initial solution is generated and then a local search is performed using meta-heuristic based methods. A custom time window formulation with respect to the theoretical maximum daily route is presented to add human expert input, priorities or expiration time to the planned routes allowing the planner to be flexible to various robotic applications.

This thesis also proposes an intermediary mission control layer, that connects the daily route plan to the robot navigation layer. The goal of the Mission Control is to monitor the robot operation, continuously improve its route and adapt to unexpected events by dropping waypoints according to some defined penalties. This is an iterative process where optimization is performed locally in real time as the robot traverse its goals and offline at the end of each day with the remaining vertices.

The performance of the various meta-heuristic and how optimization improves over time are analysed in several robotic route planning and scheduling scenarios. Two robotic simulation environments were built to demonstrate practical application of these methods. An unmanned ground vehicle operated fully autonomously using the presented methods in a simulated underground stone mine environment where the goal is to inspect the pillars for structural failures and a farm environment where the goal is to pollinate flowers with an attached robotic arm. All the optimization methods tested presented significant improvement in the total route costs compared to the initial Path-Cheapest-Arc solution. However the Guided Local Search presented a smaller standard deviation among the methods in most situations. The time-windows allowed for a seamless integration with an expert human input and the mission control layer, forced the robot to operate within the mission constraints by dynamically choosing the routes and the necessity of dropping some of the vertices.

I dedicate this thesis to my parents Helvio and Adriana, and to my sister Andressa who always supported me.

Acknowledgments

I wish to express my deepest gratitude to **Dr. Yu Gu** for being a great advisor. I want to thank you for all the support and guidance provided during my research. For encouraging me to explore my ideas, helping me grow as a person and expanding my horizons.

Additionally, I want to thank all the members of the **Interactive Robotics Lab** for their friendship and support. It was great pleasure to have met you and worked together for the past couple of years.

Also, I want to acknowledge the support provided in part by the United States Department of Agriculture (USDA) National Institute of Food and Agriculture (NIFA) Project: 2017-67022-25926 and the Alpha Foundation Project: AFC820-69, making this research possible. I want to acknowledge the DARPA sub-T for the underground tunnel models used in the simulation.

Contents

List of Figures

List of Tables

Acronyms

ACO	Ant Colony Optimization
CVRP	Capacitated Constraint Vehicle Routing Problem
DARPA	Defense Advanced Research Projects Agency
DCVRP	Distance Constrained Vehicle Routing Problem
DVR	Dynamic Vehicle Routing
DWA	Dynamic Window Approach
GA	Genetic Algorithm
GLS	Guided Local Search
GNSS	Global Navigation Satellite System
IMU	Inertial Measurement Unit
MDP	Markov Decision Process
MILP	Mixed Integer Linear Programming
OR	Operation Research
PRM	Probablistic Roadmap
RRT	Rapidly Exploring Random Trees
ROS	Robotics Operating System
SA	Simulated Annealing
STRANDS	Spatio-Temporal Representations and Activitiesfor Cognitive Control in Long-Term Scenarios
TS	Tabu Search
TSP	Travelling Salesmen Problem
VRP	Vehicle Routing Problem
VRPB	Vehicle Routing Problem with Backhauls
VRPSPD	Vehicle Routing Problem with Simultaneous Pickup and Delivery
VRPTW	Vehicle Routing Problem with Time Windows
WVU	West Virginia University

Chapter 1

Introduction

Defining a set of goal locations and making decisions on which place to visit next are challenging tasks for an autonomous robot. It must combine its decision making and planning modules to come up with an answer. Applications where a prior map is available can offer an initial knowledge about the robot goals beforehand. Deciding which location to visit next depend on knowing the cost to visit that goal and constraints from the robot and environment. An optimization over the order in which these goals are visited will reduce the total cost of completing the mission. This research effort aims to study the modeling of this problem and optimization methods that can be used to satisfy the requirements of a variety of long-term robotic missions.

Some examples of autonomous robotic applications that would benefit from an improved planning and decision making includes inspection in dangerous environments such as on offshore oil platforms [1] where it is necessary to do regular machinery and infrastructure inspection on a very remote location and where an optimized planning method would improve efficiency in a repetitive process. Other example is in underground mines where some autonomous tasks such as excavating and simultaneous localization and mapping (SLAM) already exist [2][3], and a framework with initial offline route plan and online decision making would contribute to integrate these autonomous capabilities into a robotic system able to operate entirely autonomously inside the mine, where communication is very limited.

In agriculture the environment is designed by humans, such as having crops aligned in rows with a determined interval in between. These semi-structured environment allows robotics applications such as precisely removing weed [4], flower pollination [5] and fruit picking [6]. However these robots can be inefficient if they just drive through the multiple rows following a lawn mower behavior. Since farms can be a very large environment and robots have battery and carrying capacity limitation they will be required to go to a base station multiple times to either load/unload or recharge and then go back to the location where they were. In this scenario there might be a much more efficient plan for the robot that considers these constraints than just that simplistic driving behavior.

A requirement for these mobile autonomous robots is being able to autonomously traverse the terrain. This is a combination of planning, mapping, navigation and controls. Planning is a very broad concept in the robotics community. It can vary from a decision making structure for generating the next goals to a high dimension trajectory planner for a manipulator pick and place task. In this manuscript the interest is to study the first, where in an autonomous mission many goals must be generated and a planner must balance risks and benefits to ensure a smooth operation.

During a long-term mission these requirements are stressed even further. Since a robot will need to plan its operation for days, months or years, accounting for robot resources. It needs to be able to choose among multiple goals, to be robust and deal with uncertainties and unexpected situations that might arise, and adapt to this with re-planning. In some situations learning and planning might also overlap, where robots will improve their plan as they get more knowledge about their operations. The different robot architectures also influence how a planner is designed. There are centralized architectures where all the robot actions comes from a Finite State Machine or Markov Decision Process or a more decentralized structure like the ones inspired by Intelligence without Representation [7] where sub-modules work independently and decisions are made locally.

In many robotics applications, the goals and locations that a robot should visit are roughly known before the robot is deployed and cost or reward can be estimated early on. This information, combined with constraints from the robot and environment can be used to find a plan that maximizes the reward or that minimize the time the robot takes to visit or inspect all these goals.

However, minimizing the cost among a large number of nodes is not trivial. Even considering a very simplified scenario with known cost between all nodes and a simple travel salesman problem (TSP) formulation where a robot would start from a place, visit all nodes, and come back to that place is not tractable. This is an NP-hard problem where the number of routes can be defined by the gamma function $\Gamma(n, 0)$, where n is the number of nodes. As an example, if considering 20 locations there are 2,432,902,008,176,640,000 possible routes and for a thousand nodes there are 4×10^{2564} possible routes.

The long-term autonomous robotics applications also require more elaborate problem definition. In these scenarios some robots are tasked to visit many more locations than it is possible on a single battery charge, and therefore the robot should autonomously go back to a base station for recharging in between goals. Other objective is that a planner should be able to smoothly integrate external information to its plan, such as a human expert input instruction on what time some location should be visited.

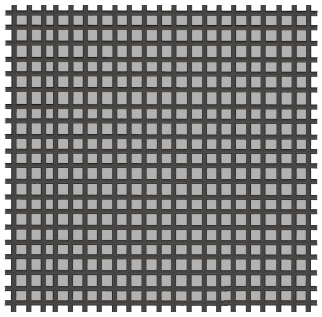
Long-term robot planning in a large environment is complex, there are many challenges and considerations for choosing a planning strategy such as: (*i*) optimally choosing among the many goals is not tractable; (*ii*) defining the cost is subjective, the goal can be just minimizing the time, or additional parameters that are important to a specific application can be included; (*iii*) deciding what constraints should be included in the problem formulation is important; (*iv*) determining if a complete plan for the mission is necessary or just being able to choose what to do next is enough; (*v*) information might be unreliable; (*vi*) how to use all the data is challenging, there might be not enough or too much data; (*vii*) when the planning algorithm is structured in layers, what level of details should be accounted for in each of the layers, (*viii*) deciding if re-planning is necessary, and if yes, on what layers and how to know when it is needed.

A common planning approach is to have the planner divided in multiple level of layers [8],[9], which might include a first layer for choosing and generating the next goals for a robot to visit; a second layer for obtaining a path between the current position and the goal by using discrete or sampled based methods and that might include information such as maps from the environment, sensors data and beliefs; and a third layer for dealing locally with robot and environment dynamics to generate trajectory commands that follows the path. This research effort aims to study and present methods for solving the first layer of choosing and ordering the way-points to visit for a long-term autonomous mission respecting the constraints and optimizing time.

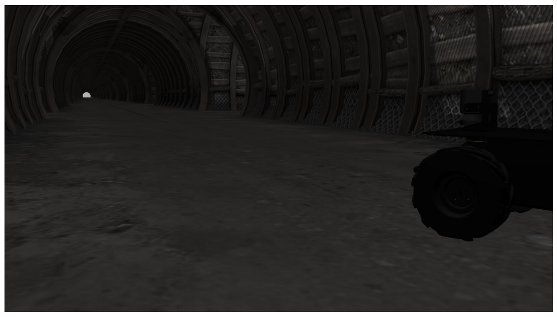
1.1 Problem Statement

The goal of this work is to address the problem of planning on scenarios such as Figure 1.1 where there is a very large environment with many locations to visit. In that figure an underground mine simulation where all the columns must be inspected for structural integrity which demands multiple days of operation from a robot. This requires an elaborated problem definition, where there are considerations such as having the robot return to a base station for recharging, having priorities for some of the nodes and uncertainties from robot behaviors and the environment.

This optimization problem can be described by the graph $G = (V, A)$ where an example is shown in Figure 1.2. The V is the entire set of vertices $V = (v_0, v_1, ..., v_n)$ and $A = \{(v_i, v_j) : i \neq j\}$ is the set of arc costs, where a_{ij} is the associated non negative arc cost from vertex i to vertex j. The

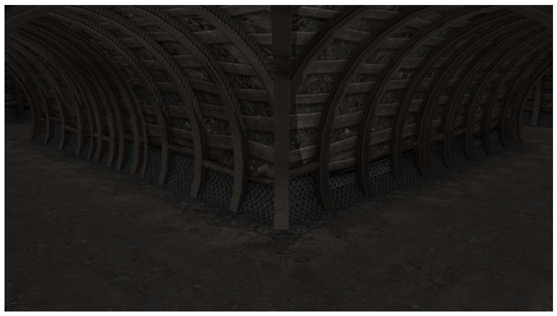


(a) Mine overview



(b) Tunnels and robot

(c) Obstacles



(d) Columns to inspect

Figure 1.1: Underground mine Gazebo simulation environment

x_{ij} is a binary value, $x_{ij} \in \{0, 1\}\ \forall\, i, j \in V$, where its value is 1 when the arc from i to j is part of the solution and zero otherwise. The work presented on this manuscript aims to minimize the traversing cost function in Equation (1.1).

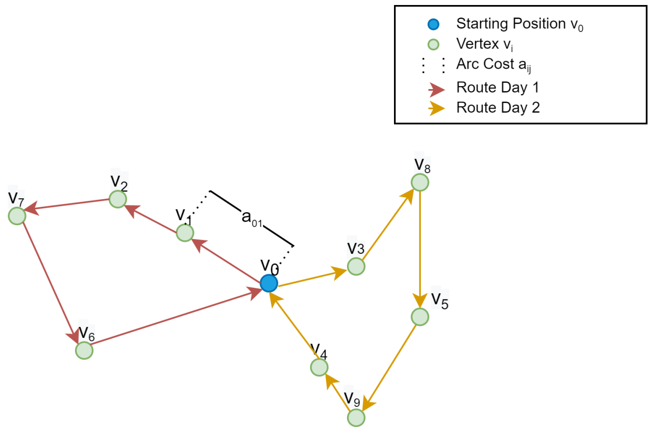


Figure 1.2: Example graph for the problem definition

$$L = \sum_{i \in V} \sum_{j \in V} a_{ij} x_{ij} \tag{1.1}$$

The constraints associate to x_{ij} are:

$$\sum_{i \in V} x_{ij} = 1 \; \forall j \in V \setminus \{0\} \tag{1.2}$$

$$\sum_{j \in V} x_{ij} = 1 \; \forall i \in V \setminus \{0\} \tag{1.3}$$

Where equations (1.2) and (1.3) state that each vertex is visited one time with the exception from the robot starting position vertex of index 0.

$$\sum_{i \in V} x_{i0} = D \tag{1.4}$$

$$\sum_{j \in V} x_{0j} = D \tag{1.5}$$

The equations (1.4) and (1.5) state that the number of times a robot leaves the base station should be equal to the number of times it returns to the base station, and that this is equal to D, that is the planned number of deployments or days of operation.

Other constraint for the problem is Equation (1.6). Where each individual route r_i is an ordered subset of vertices from V and $C(r_i)$ represents the total arc cost of visiting all the vertices in that route. For each day or deployment the route must respect a time constraint related to the maximum robot operation time $T = (t_0, ..., t_n)$.

$$C(r_i) < t_i \tag{1.6}$$

Time window constraints can be associated to the vertices V. When an initial and finish time $[s_{initial}, s_{finish}]$ is added as a parameter to a vertex v_i, this vertex must be visited within that time window inside the route.

The low-level mapping, motion planning, and obstacle avoidance modules are considered available for dealing with local hazards and condition throughout this work. The global map assumption applies to many practical applications because on outside locations there are satellite imagery and digital elevation maps, indoor locations are mostly man-made and therefore the building blueprints would be a good description of the environment, and even in other planets there are topographical maps available for a robotic mission [10]. Perfect localization is used to calculate the arc costs, and it is assumed that the mobile robot is able to localize itself, navigate the environment and reach the way-points by fusing the information from its sensors, including local sensors such as Inertial Measurement Unit and Wheel Encoders, and external sensors such as beacons, GNSS, etc. The arc-costs are known and constant for generating the initial multiple routes and calculated using driving time estimation, this can vary from a simplistic robot average speed to a forecast from a 3D physical simulation from the robot driving on the environment from point A to B. Uncertainty is only considered in execution time, where during the execution of a single route the arc-costs from that route might not be the same value as it was planned for and some adjustment on that route such as dropping some of the vertices to be able to return to the starting position might be required to respect the problem constraints.

Contributions of this work include a method for planning and scheduling multiple day routes for long term robotic missions; comparison among metaheuristics for the local search in that problem; a method to add human expert input, priorities or expiration time for the planning routes; and a mission control layer for monitoring navigation and improving the routes during runtime. Applications of this work include inspection of large environments such as stone-mine and office building security; agricultural applications such as weed control, pollination and fruit picking; robot delivery; among others.

The presented framework was tested on multiple goal configuration and optimization scenarios and on two robotics physical simulation environments that represent a realistic application of the presented work on long term autonomous missions. Chapter 2 will cover related and background work on planning and scheduling for long term autonomy and on vehicle routing problem. Chapter 3 explains the technical approach used to solve the problem, the built simulation environment and integration with robot navigation. Chapter 4 provide the experiments motivation, design and results obtained, and Chapter 5 is a conclusion about the proposed methods.

Chapter 2

Background

This chapter provides a review on background work related to the goals of this thesis. Section 2.1 is an analysis on long term autonomous robot projects and how their goals, schedule and planning are defined, and Section 2.2 is a study on vehicle routing problem and solvers that is part of the proposed approach to solve the problem of choosing goals and routes on a long term robotic mission.

2.1 Planning and Scheduling for Long Term Autonomy

There are long term robotic examples in various domains such as space, marine, air, field, road, and service. In each domain the level of autonomy and the area where it is applied is different because of environment variability, diversity, dynamics, interaction and cooperation with other robots and humans, how cost and how critical they are, and the duration of each mission [11]. Scheduling goals and planning is part of an autonomy architecture that might also include navigation and mapping, perception, knowledge representation, reasoning, interaction and learning.

Some robots are able to operate for long term in indoor environment. PR2 from Willow Garage is an office robot which was able to autonomously ran an office marathon (26.22 miles) on a 30h mission [9], this was accomplished with a combination of predefined goals in the office, an A* global planner on an obstacle costmap and a local planner using Dynamic Window Approach [12]. Policies for recovery behavior were added to deal with unexpected situations, such as entrapment. Because the objective was just to maximize driving distance, the goals were simple. To drive around the office environment multiple laps in a circuit like configuration and return to a base station when low on battery.

In the Spatio-Temporal Representations and Activities for Cognitive Control in Long-Term Scenarios (STRANDs) [8] project the robots are able to operate in office environments for days interacting with humans and performing tasks. STRANDs was developed for two scenarios, one is a security task where the robot patrols the environment performing regular checks and generating alerts for unusual events. The other is an elderly care facility where robot supports staff and patients by providing information to them. Long term planning capabilities are built on top of the ROS core navigation system that uses similar algorithm implementation from the PR2 robot. A Monitored Navigation layer observes the execution of the standard ROS Navigation and iterates through a set user defined list of recovery behaviors based on the type of failure. Generating goals for the planner is done overnight during charging time by an adaptive topological navigation layer where an Markov Decision Process (MDP) is used to produce a policy for a goal and the duration based on previous logged data. Therefore the robot is able to learn and adapt from the collected data by improving the estimated navigation time and maximizing the number of services provided.

The CoBots [13][14] is another long term project, that similar to the problem that is being addressed in this thesis, offers a method for planning and scheduling tasks and goals. CoBots are designed to operate in office spaces and are able to accomplish tasks such as Deliver-message, transport, escort and telepresence. The tasks in this project are requested by humans over the network or through an onboard tablet, and a scheduler layer [15] is used to check if the task is feasible and then dynamically sorts the order using mixed-integer linear programming (MILP). Planning and navigation includes a human-centered symbiotic approach where robots will ask humans for help when they would not know the location or the action to take.

Dynamic User Task Scheduling [15] is a mixed-integer programming technique for solving robotic task planning in situations where there are overlapping time windows in the tasks. In [16] a mixed-integer programming method is proposed for solving single robot task planning for a set of tasks with temporal constraints and considering schedules from multiple human users, and the commercial CPLEX MIP solver is used to obtain the solution for a set of 40 tasks problem. Task Scheduling with Interval Algebra [17] is another method for addressing the robotic task scheduling problem, this method heuristically order the task pairs aiming to maximize the time available for the tasks, and to minimize the sum of completion time.

Projects such as [18] provide some example of planning and scheduling architecture to be used in an industrial environment, where the mission planning method proposed for multiple parts feeding uses a genetic algorithm heuristic for generating the sequence of tasks by minimizing waiting time and total traveling time from a mobile robot. A mission control layer is used to feed this generated task sequence into the robot in real time, where it monitors for disruption in the production, that takes priority over the generated task sequence.

In [19], a mixed integer programming method for planning tasks for an heterogeneous fleet of robots is presented. Multi robot task scheduling is also proposed in [20], where heuristics are proposed on a coalition level without the requirement of robot specification, in this work robots are required to cooperate tightly, sharing different capabilities among a fleet of heterogeneous robots to accomplish the tasks using the proposed Interfere Assign and Min Interfere methods.

Field robots require planning and scheduling for long term operation in an environment less constrained than the office and industry. A goal of this thesis is to present a method that is able to plan routes under this complex environment characteristics. The challenge is planning the long sequence of locations to visit for these robots to operate for long term with multiple deployments in a large environment. The planners should also be flexible to consider the particular aspects of each application, where robots might have some priorities given by a human expert or be required to adjust its plans due to uncertainty. These robots usually operate in scenarios where the rough terrain requires to model the environment in three dimensions and the challenges to drive in this conditions and the large environment make it crucial to plan considering robot energy and the requirement to recharge the robot. There are energy aware planning methods developed using energy cost models for different types of ground vehicles, such as [21][22] that presents planners for finding a more energy efficient path on uneven terrains and [23] that studies the deployment of robots when energy and timing constraints are considered based on power model calculated at different speeds. These planners solve for a more energy efficient path, that can be used by other algorithms as the cost values when defining the sequence of goals that a robot should have on a mission.

Some of the field robotics applications are coverage problems, where the robot is required to traverse the entire environment. An example of planning for this problem considering the limited battery capacity of the robot is presented on [24], where the proposed method is divided in two steps, an offline part assuming full-knowledge of the environment and uses the shortest path potential function, the saddle curves, the corridors induced by saddle curves, and the coverage path split to generate the plan; and the online part that uses information calculated on the offline part to generate the coverage for an unknown environment.

Agricultural robots are a good example of field robots. SwagBot [25] is an omnidirectional robot designed for autonomous weed detection and spraying, soil sampling, pasture analysis, biomass estimation, and livestock monitoring on uneven terrain. This robot plans its route by solving a travel salesman problem (TSP) over the energy costs of visiting its predefined goals. The costs are generated with a combination of Probablistic Roadmap (PRM) and the energy cost of motion for this omnidirectinal robot presented in [26]. Limitation of this method is that the asymmetric TSP only generates a route from the starting position covering all the vertices and returning to the same position. This is only a single deployment of the robot on a single battery charge, that can't be extended for multiple days of operation. Another issue is that it does not support for uncertainties that might happen during run-time.

2.2 Vehicle Routing Problem

The vehicle routing problem (VRP) has been an important field of study since the 60s with [27], where the traditional problem involve simultaneously determining the routes for multiple vehicles from a depot to a number of customer locations and returning back to the depot without exceeding capacity constraints from each vehicle. This is extremely important for all kind of delivery tasks such as package delivery, postal services, public transport routes, etc where the goal is to minimize distance traveled or time while maintaining or possibly increasing service quality.

VRP is a combinatorial optimization problem, that can be described by a graph $G = (V, A)$, where $V = (v_0, v_1, ..., v_n)$ are the vertices set and $A = \{(v_i, v_j) : i \neq j\}$ are the arcs set, where a non negative cost c_{ij} is associated with each arc and represent the travel cost for going from vertex v_i to v_j and C is the vehicle capacity constraint. The problem must be solved under the following constraints: (i) each location is visited exactly one time; (ii) each vehicle must start and end its rout at the depot v_0; (iii) the sum of the cost of the vertices visited in a route cannot exceed the vehicle capacity C.

Extensive research has been accomplished on vehicle routing problems where many variation from the basic problem formulation have been created to address the requirements from the delivery industry. According to [28], problem formulation includes: Capacitated Constraint VRP (CVRP) [29] where each vehicle has a maximum capacity and each customer location has a demand associated and the total demand for each route cannot exceed the total vehicle capacity; Distance Constrained VRP (DCVRP) [30] where the capacity limitation for each vehicle is a maximum distance that it can travel and therefore each route has maximum distance limitation; VRP with Time Windows (VRPTW) [31] that is an extension from the CVRP where each customer location has also a time interval $[a_i, b_i]$ constraint where a delivery is only allowed on each location during its time windows and where the minimization parameter is time, the distance cost is defined as time unit and waiting for delivery is also allowed; VRP with Backhauls (VRPB) [32] where the customer location vertices are divided in two subsets, one subset are linehaul customers requiring products to be delivered and the other are backhaul customer requiring a product to be picked up, and in this formulation the added constraint is that all linehaul customers must proceed the backhaul customers in each route; VRP with Simultaneous Pickup and Delivery (VRPSPD) [33] where each customer vertices are divided in a pair of origin O_i and destination D_i and they have associated demands to be picked d_i and demand to be delivered p_i, on this problem added constraints are: (i) the load of each vehicle on a route must always be non negative (ii) for each customer D_i, when different from the depot, the customer O_i must be served on the same route and before customer D_i. Other problem variations include multiple depots, dropping visits and other combinations from previous described formulation.

Due to being a NP-hard problem, solving for the optimal solution is not trivial and only possible for graphs with a small number of nodes. In [34] a minimum K-Trees algorithm is presented and proven to solve for the optimal solution on specific network configurations with up to 71 nodes. For more general or larger networks metaheuristic optimization based approaches are used for local search the routes to present best known solution for the problems. This metaheuristic optimization include the use of Genetic Algorithms (GA) [35], Ant Colony Optimization (ACO) [36][37], Tabu Search (TS) [38][39], Simulated Annealing (SA) [40][41], Greedy Approach [42] and Guided Local Search (GLS)[43][44].

On [37] a solution for the VRP using ACO is presented, on this each ant constructs a vehicle route that visits each customer; next customer is updated based on a probabilistic equation that considers pheromone value and distance between nodes, the pheromone trail is updated both locally by simulating an evaporation behavior and globally by updating the best route among the feasible ones. Besides the standard ACO implementation, route improvement strategies are also used on this implementation, this includes 2-opt heuristic where pairwise exchange of nodes is done over routes from individual vehicles and candidate list where only a limited number of candidate nodes are evaluated to find the next node on a route (e.g. 1/4 of total nodes). Both this strategies where evaluated on the paper with different values for the candidate list. The use of single ACO and multiple ACO was also evaluated. Results are presented comparing the different configuration on the ACO as a probabilistic distribution for the problems. On [45] the ACO for VRP is improved by randomly including mutation operations where nodes on different routes are swapped and optimized locally with 2-opt heuristic resulting in a new solution that is not very far from the original. Results are presented for 14 problems on the OR-Library [46] that includes problems with number of nodes from 50 to 199 and where results are very similar to the known best solution which at the time were provided by multiple implementations of the Tabu Search algorithm.

The Greedy Approach presented in [42] is a simple local search strategy with improved heuristics for problems with time windows and capacity constraints. The heuristics used are: (*i*) the previously presented 2-opt heuristic, (*ii*) the relocate operator that moves a node location from one node to another, (*iii*) the exchange operator that swaps nodes from different routes and (*iv*) the cross operator that swaps entire portion of routes between nodes. To minimize the objective function a greedy approach using the steepest descent is used, there all the costs to move to neighborhoods are calculated and the one that reduces the maximum cost is chosen, all the heuristics are applied to attempt to reduce the cost, the states that violate any of the problem constraints are discarded and a final solution is given when the heuristics does not present any more improvement to the solution. The results from using this algorithm are usually within 10% of the best known solution, advantage of this is that it ensures repeatability because it does not use any random strategy, however it can be slow due the requirement for calculating all the costs each time-step.

The simulated annealing [41] is a metaheuristic approached inspired on the annealing process used on metallurgy. Starting from an initial feasible solution for a VRP problem, and given an initial temperature and a cooling schedule, every step a random update is generated using heuristics similar as the ones used on the Greedy Approach by [42]. If the candidate solution is valid and better than the current it automatically updates the state. If it is valid but worse than the current best it updates with a probability $e^{\Delta/T}$, where Δ is the difference between the states, and T is the current temperature. The temperature is updated according to the cooling schedule. This process is repeated until the temperature reaches a determined value or a number of steps is reached.

The Tabu Search [47] is a metaheuristic search method based on memory structures called tabu list, that is a list of forbidden behavior, where a set of strategies (Forbidden Strategy, Freeing Strategy and Short-Term Strategy) decide what goes in and out of the list, where on VRP the previously visited locations are added to it. The evaluation function used on the search, in a similar way as it is done on SA, can produce either the best improvement or the least non-improvement, where the tabu list forbid the search to come back a previous best result allowing the search to local optimality. Tabu Search presents some of the best known solution on benchmark tests such as [46] and [48].

Other algorithm that excel on benchmark tests is the Guided Local Search [44], that is a memory-based approach that augments the cost function and penalise terms based on how far they are from previous local minima. This algorithm starts by using some other local search method such as Greedy Approach to get into a local minima, there a penalty function is implemented that penalizes a spread of features around the current solution, the local search is done on the set of penalized features until the algorithm reaches some stopping criteria. This method allows the search to leave local minima points. When this approach was tested on [48] dataset, it was able to find 13 new best solution for its problems and consistently outperforms other methods on longer routes with fewer vehicles. On classes with shorter routes it is outperformed by TS implementations.

New VRP formulations for different applications are currently being studied. In [49] a problem formulation that is designed on a mix of Drones and Truck scenario is presented. In this problem drones are allowed a simple unitary move, from truck to a node and back to the truck, and the truck is supposed to go from depot, consumer locations, skipping the ones visited by the drone, and back to depot, where the idea is that the solution is improved when the drone is doing a delivery on one location and the truck is already driving directly to the new location. This is currently being presented on a very small network with only few nodes and claiming a reduction of delivery completion time by 75% on scenarios that the drone is faster than the truck.

Other current field of research on VRP are dynamic formulations of the problem. In [50] the Dynamic Vehicle Routing (DVR) problem is defined to plan for policies and queuing theory, policies studied include $single-vehicle\,Divide\,\&\,Conquer$, $multi-vehicle\,Divide\,\&\,Conquer$, $The\,No\,(Explicit)\,Communication$, $Nearest-Depot\,Assignment$ among others. Other possibility for solving the Dynamic Problem on vehicle routing is to combine static routing with sequential re-optimization algorithms.

Chapter 3

Technical Approach

This chapter describes the technical approach for addressing the problem of long term planning for robotics missions. The contribution of this thesis is presenting a method for modeling robotics planning and scheduling problems as a Vehicle Routing Problem and integration of that solution planner into a robotics framework. The proposed method, as shown on Figure 3.1, divides the planner in two parts. The first is a mission planner and scheduler responsible for generating all the routes for the entire mission given robot, environment and mission constraints, and the list of goals. The second part is the execution of the planned routes during run-time, where a mission control layer will interact with the navigation layer, generating the next way-points and monitoring the execution of the mission.

Section 3.1 provides fundamentals on how VRP are solved in two steps, generating an initial solution given the problem constraints and optimizing that solution using local search metaheuristic methods. Section 3.2 presents how long term robotic planners can be modeled as a VRP, Section 3.3 describes the mission control layer and how VRP formulation is part of that layer. The Section 3.4 describes the developed robotic simulation environments, the robot and navigation algorithms used, and the integration of the VRP route planner with the robot navigation.

3.1 Fundamentals of Vehicle Routing Problem

The Vehicle Routing Problem as defined in Section 2.2 is a mathematical description of the delivery problem, where the objective is to minimize the travelled distance or the time. The constraints are associated with the vehicle carrying capacity, picking something up before delivering, loading and unloading time and delivery time windows. This formulation is suitable for the planning and

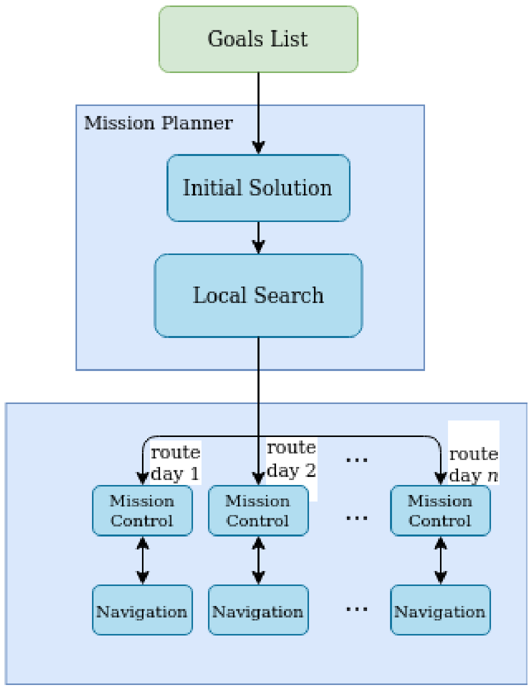


Figure 3.1: Planning diagram for long-term robotics missions

scheduling for long term robotic missions, where the multiple vehicles can be seen as a multi-day plan for robots, where they must depart and return to their initial position at each day.

Existing solutions for the VRP are modular where calculating the cost is independent from generating the solution and optimization. Initial solution methods are shown in Section 3.1.1. Improving the initial routes is accomplished by performing local search using metaheuristic based optimization algorithms in Section 3.1.2, where algorithms with different characteristics are presented. As an implementation consideration, the robot starting position is going to be represented with index zero throughout this thesis.

3.1.1 Initial Solution

The initial solution for the Robotic Routing Planner using Vehicle Routing Problem definition follows the constraints defined in Section 1.1 problem statement. Equation (3.1) and (3.2) state that each vertex should be visited one time only with the exception of the starting position v_0. Equations (3.3) and (3.4) state that the robot should, at each route, start from the initial position and return to that same position at the end of the route and that this is going to be repeated through the total number of routes D. The last constraint is that the total time cost of each route should be less than the daily vehicle constraints $T = (t_0, ..., t_n)$. Route costs are calculated using the matrix of arc costs. This matrix indices represent the arc cost from one vertex to another, if the arc cost $x_{ij} = x_{ji} \; \forall \; j, i$ this matrix is a symmetric matrix. All the elements in this matrix diagonal are equal to zero, because costs for staying in the same place are not considered as it is forbidden to repeat vertices on a route. Section 3.2.1 will present methods to generate this matrix for robotics problems.

$$\sum_{i \in V} x_{ij} = 1 \; \forall \, j \in V \setminus \{0\} \tag{3.1}$$

$$\sum_{j \in V} x_{ij} = 1 \; \forall \, i \in V \setminus \{0\} \tag{3.2}$$

$$\sum_{i \in V} x_{i0} = D \tag{3.3}$$

$$\sum_{j \in V} x_{0j} = D \tag{3.4}$$

The first algorithm considered for obtaining the initial solution is the **Path-Cheapest-Arc** [51], which is also used as the standard initial solutions for VRP in the Google Operations Research Tools [51]. This algorithm is a greedy iterative process and Algorithm 1 shows its pseudo-code. Starting from empty routes and respecting that each vertex can only be added a single time with the exception of the initial position and that the robot has the constraint for maximum operation time in each daily route this method always selects the valid vertex with less cost as the next vertex in a route. The custom implemented algorithm was designed to ensure repeatability for running the experiments, therefore the function that returns the vertex with minimum distance from another vertex was developed so it always returns the same vertex in cases where there are multiple vertices at the same minimum distance. In the algorithm pseudo-code, the *route_distance* is the current total cost of a route; the *distance(i,j)* evaluates the arc cost between vertex i and j; *route[-1]* represents the last element on the route; *append* adds a vertex to a route and *remove* removes a vertex from the remaining vertices list.

Algorithm 1: Path-Cheapest-Arc

```
Routes = [];
Vertices = V // Set of all vertices to visit with exception of starting position
$T = \{t_0, ..., t_n\}$ // Max daily route cost
for $t$ in $T$ do
    route=[];
    while True do
        if route_distance(route)+distance(0,route[-1]) $< t$ then
            if route_distance(route)+distance(0,route[-1])+min(route[-1], Vertices) $< t$ then
                route.append(vertex(min(route[-1],Vertices)) // Add closest vertex to the
                    current last vertex on the route
                Vertices.remove(vertex(min(route[-1],Vertices)))
            else
                route.append(0) // return to initial position
                Routes.append(route);
                False;
            end
        else
            False;
        end
    end
end
$check\_routes(Route, Vertices)$
```

A second algorithm considered for generating initial solutions is the **Path-Cheapest-Arc with Random Vertices** that is a combination of the first algorithm with a probability of randomly assigning a vertex to a route. As shown in Algorithm 2, the algorithm uses Path-Cheapest-Arc with a small chance that a random valid vertex would be added at each step. The algorithm implementation is similar to Algorithm 1, but a *random()* function provides 95% chance of applying path-cheapest-arc and 5% chance of adding a random vertex at each iteration. The random vertex is chosen from the Vertices list using the method *choice()*, if it is a valid choice it is added to the route and removed from the Vertices list. When running multiple times this algorithm can present an improved initial solution, however it does not ensure repeatability and when iterating over multiple times, a very inefficient optimization is actually being performed.

Algorithm 2: Path-Cheapest-Arc+Random approach

```
Routes = [];
Vertices = V // Set of all vertices to visit with exception of starting position
$T = \{t_0, ..., t_n\}$ // Max daily route cost
for $t$ in $T$ do
    route=[];
    while True do
        if route_distance(route)+distance(0,route[-1]) $<$ $t$ then
            if Random()<0.95 then
                if route_distance(route)+dist(0,route[-1])+min(route[-1],Vertices) $<$ $t$ then
                    route.append(vertex(min(route[-1],Vertices)) // Add closest vertex to
                       the current last vertex on the route
                    Vertices.remove(vertex(min(route[-1],Vertices)))
                else
                    route.append(0) // return to initial position
                    Routes.append(route);
                    False;
                end
            else
                new_vertex = choice(Vertices);
                if route_distance(route)+dist(0,route[-1])+dist(route[-1],new_vertex) $<$ $t$ then
                    route.append(new_vertex) // Add Random vertex to the route
                    Vertices.remove(new_vertex)
                end
            end
        else
            False;
        end
    end
end
check_routes(Route, Vertices);
```

3.1.2 Local Search

Local Search is the process of improving an initial solution. This is used because it is infeasible to perform optimization in the entire global search space. A general definition of local search is that of an iterative process that starts by defining a neighborhood for the current solution and choosing the best solution within that neighborhood while some criteria is not met. The criteria includes time limits and maximum number of steps and solutions, including failures and solution convergence.

Optimization methods were chosen based on characteristics and performance in general VRP formulations reviewed in the background Section 2.2. The metaheuristics are described following the implementation from Google Operation Research Library [51] which is also used throughout the experiments of this thesis. Some methods are common for all the presented metaheuristic algorithms for solving the robotics planning VRP:

- **StopCriterion**: This method defines the criteria to exit the optimization. Three things are considered, the solution tolerance that is the number of solutions allowed without any improvement on the objective value, the number of iterations of the algorithm and the time. This method is also able to catch interruptions, and therefore the user can manually force to leave optimization without losing the data.
- **Fitness**: this method evaluates how good a proposed solution is, this is the inverse of the total cost for the planned routes.
- **LocalSearchOperator**: this method is used to generate and explore a neighborhood around an input solution. These are constructed using two techniques: the 2-opt where 2 vertex are swapped from the current solution and the shuffle operation where an arbitrary number of vertices are permuted within a route. The number of neighbors and number of operations used to generate the neighbors is a parameter to be defined on initialization of this method and depends on the kind of optimization used. The Simulated Annealing and Guided Local Search only require one valid candidate solution per iteration, whereas on Greedy Descent, Tabu Search a list of valid candidates is necessary. An *IsValid* method is implemented inside this method to check if the candidate solution follows all the problem constraints. In addition to that, filters are used to improve speed and remove illegal operation such as not allowing to shuffle the starting and end position at each route.

The **Greedy Descent** [51] pseudo-code shown in Algorithm 3 is an approach that makes the best locally optimal choice at each iteration by choosing the best candidate generated from the

LocalSearchOperator method. This local search is simple and converges fast, but is not able to avoid local minimum and was chosen because it is a good comparison with other approaches that should be able to avoid these local minimum.

Algorithm 3: Greedy Descent

```
// Initialize the optimization
current_best = initial_solution;
k=0;
while not StopCriterion() do
    candidates = LocalSearchOperator(current_best);
    candidate = candidates[0] if fitness(candidate) > fitness(current_best) then
        current_best = candidate;
    end
    k = k+1;
end
```

Simulated Annealing [51], as presented in Section 2.2, is a metaheuristic optimization method based on the annealing metallurgic process. The pseudo-code is shown in Algorithm 4, where the main idea is that it is possible to avoid local minimum by accepting a candidate solution that is not better than the current solution. Solutions that are better than the current solution are always accepted. However, there is an acceptance probability P for taking a solution that is worse than the current solution. This acceptance probability reduces as the temperature decreases and the process converges to its best solution.

Besides the *StopCriterion* and *LocalSearchOperator* previously described, this algorithm implementation uses the following methods: **Temperature(k)** that returns the ratio of the initial temperature over the number of iterations k; **Energy(x)** returns the energy value of a proposed solution x (same as the fitness method); **Random()** returns a float between 0 and 1 and $P(e, e_{test}, t)$ is the acceptance probability method, if $e > e_{test}$, $P(e, e_{test}, t) = 1$ else $P(e, e_{test}, t) = exp(-(e_{test} - e)/t)$.

Algorithm 4: Simulated Annealing

```
// Initialize the optimization:
current_solution = initial_solution;
e = Energy(current_solution);
t = temperature0;
k = 0;
while not StopCriterion() do
    t = temperature(k);
    candidates = LocalSearchOperator(current_solution);
    candidate = candidates[0];
    $e_{test} = Energy(candidate)$
    if $P(e, e_{test}, t) > Random()$ then
        current_solution = candidate;
        e = Energy(current_solution);
    end
    k = k+1;
end
```

The **Tabu Search** [51] is a metaheuristic algorithm based on lists of forbidden actions where for some period of time it disfavors some of the locations to visit. The presented implementation is divided in two phases the diversification mode where it looks for different features and the intensification mode where it seeks similar features. The optimization will oscillate between this two steps to avoid being trapped in local optimum and also explore regions close to promising solutions.

The algorithm uses two lists to accomplish that two phase outcome, the **Forbidden List** that is a list of candidate solutions that are not allowed and the **Keep List** that is a list of candidate solutions that should be kept. Elements in these lists are of type struct where each value has the candidate solution with the routes and a stamp. This stamp is updated in the lists at each iteration of the algorithm, and is used by the *UpdateLists* method to add and remove values from there. The lists are queues where values are added to the end and retrieved at the beginning, this way it is not required to iterate over all values in the list to update the list. Tenure time is the number of iterations that a value should be kept on a list, there are two tenure times Forbidden Tenure Time and Keep Tenure Time. The **UpdateLists** method uses these tenures to performs 4 operations. It updates the stamp on current members of the lists; adds new values to the keep list; removes values that have expired from the Keep List and adds them to the Forbidden List; and removes values that have expired from the Forbidden List. The process of allowing new values and forbidding old values is what alternates between the intensification and diversification phase.The TS only adds constraints for the optimization, therefore the aspiration criterion will accept a solution, no matter what, if it is better than the current best solution.

The optimization pseudo-code is shown in Algorithm 5, where the search should start after a local optimum is reached by the greedy descent. In an iterative process it selects neighbors from the *Keep List* solutions, selects the best candidate not in the *Forbid List* from the neighbors, checks if this best candidate is the best known solution and updates both list using the *UpdateLists* method with the best candidate solution.

Algorithm 5: Tabu Search

```
// Initialize the optimization
best_solution = initial_solution;
keep_tabu_list = [];
forbid_tabu_list = [];
keep_tenure = 5 ; // number of iterations value
forbid_tenure = 10;
k = 0
while not StopCriterion() do
    neighborhood = LocalSearchOperator(keep_tabu_list);
    best_candidate = neighborhood[0];
    for candidate in neighborhood do
        if not forbid_tabu_list.contains(candidate) and fitness(candidate)>
         fitness(best_candidate) then
            best_candidate = candidate; // get best solution from the neighborhood
        end
    end
    if fitness(best_cadidate)> fitness(best_solution) then
        best_solution = best_candidate; // best candidate is the best solution
    end
    UpdateLists(best_candidate) ; // adds the best candidate to the keep list and
     updates both lists
    $k++$
end
```

The **Guided Local Search** [51] is a metaheuristic optimization based on penalties to avoid local minimum and plateaus. The idea is that by penalizing some repeated features from a local optimum it can reach other parts of the search space. Penalties are implemented using an indicator function, where $I_i(x) = 1$ if feature i is part of solution x and $I_i(x) = 0$ otherwise. In the routing problem these features are the arc costs, where c_{ij} represents the arc cost from going to vertex i to vertex j in the solution, p_{ij} is a counter that represents the number of occurrences of some arc cost, u_{ij} is the utility function $u_{ij}(x) = I_i(x) * c_{ij}(x)/(1 + p_i)$ where features will be penalized, but less penalized if they appear more often because features that appear often on local optimum might be a part of a good solution.

The $g(x)$ is the augmented objective function $g(x) = \sum_{(i,j)} c_{ij} + \lambda \sum_{(i,j)} I_{ij}(x) * p_{ij} * c_{ij}$. Where λ is the penalty factor, a small value will tune the search for intensification and a large value will induce diversification of the solution. The pseudo-code is shown in Algorithm 6, where it is implemented using two methods, the **ObjectiveFunction** takes a candidate solution, lambda and penalties as input and applies the augmented objective function g using the indicator function $I(x)$ and returns the calculated new total cost with penalties. The **UpdatePenalties** method gets the penalties p and the local best solution as input and calculates the utility function u. Then it updates the penalty values p for the features where a maximum utility function u was obtained.

At each iteration of the algorithm, while the stopping criterion is not met, the candidate solution is obtained with the *LocalSearchOperator* current local best solution; the candidate solution and current local best solution are augmented using the *ObjectiveFunction* and compared; if greater the local best is updated with the candidate solution; then it checks if the candidate solution is the current best overall solution; penalties p are updated using the *UpdatePenalties* method and the current local best solution.

Algorithm 6: Guided Local Search

```
// Initialize the optimization
best_solution = initial_solution;
local_best = initial_solution;
k=0;
lambda = 0.1;
p = [0, ...,0];
while not StopCriterion() do
    candidate = LocalSearchOperator(local_best)[0];
    if ObjectiveFunction(candidate, lambda, p) > ObjectiveFunction(local_best, lambda, p)
     then
        local_best = candidate;
    end
    if fitness(candidate) > fitness(best_solution) then
        best_solution = candidate;
    end
    p = UpdatePenalties(p, local_best);
    k = k+1;
end
```

3.2 Vehicle Routing as a Robotic Planner

This section presents proposed methods on how to integrate the VRP solutions as a robotics planner and scheduler. The proposed method to choose constraints for VRP Robotics Planning is different than what is used in the delivery application. In the delivery application the number of vehicles available is what define the design of the problem. However for robotics planning the primary limitation is the robot operation time or energy. The maximum number of days for a mission is a flexible constraint, the mission planner layer will generate the minimum possible number of routes. The number of days is used as a boundary for checking if the problem parameters are viable. If they are too strict and it is not possible to generate an initial solution within the defined number of days it will return an error informing it is not possible to obtain a solution with the specified robot operation time and number of days.

Section 3.2.1 describes how to generate the arc costs for the robotic planner considering the information available about the robot, the mission and the environment. This is a modular and independent process from generating the initial solution and optimization

The variations of VRP formulation also provide some flexibility for the robotic route planning. In Section 3.2.2 a method to add priorities and expiration to some of the tasks is proposed. These allows for an expert human input to the routes and the addition of new constraints that were not previously considered.

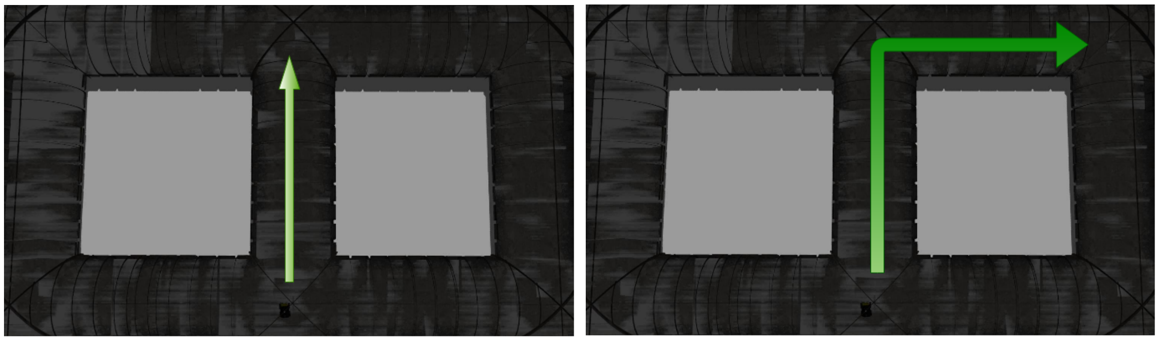
3.2.1 Arc Costs

The search algorithms employed on this work uses a matrix of costs among all the vertices as input that represent time costs. Therefore there are many possibilities on how to calculate arc-costs. The requirements from algorithm implementation are that arc-cost values must be integers and that all the matrix must be calculated beforehand for algorithm efficiency.

A traditional approach is to get a distance between two nodes using distance metrics such as Euclidean, Manhattan, Mahalanobis, Hamming, among others [52]. For calculating this distance the only information required by the problem is the position in the world of each of the vertices location to be visited. When problems are modeled as time dependent such as in this robotic application where the constraint is the maximum operation time per deployment or in problems with time window constraints the distances cost can be converted to time cost by multiplying it with the robot average speed.

An option that provides a more complete cost estimation is to use a physical simulation. A method to get these costs is assigning the robot to traverse between way-points in a Gazebo simulation environment and obtaining the robot driving time to the goal. This considers many variables throughout the simulation, the robot dynamics that consider the forces acting on the robot and acceleration produced and its interaction with a 3 dimensional environment. There the cost matrix can be generated by simulating the robot traversing among all possible vertices in the problem, where regions that are not traversable will get an infinite cost. This simulator provides the possibility to simulate faster then real time and multiple simulations running in parallel. However, even with this resources, using this approach is only feasible for problems with a very small number of locations to visit.

A possible solution to that problem is getting probabilistic data on how a robot traverse a determined type of environment using the robotics physical simulation, and then apply that value



(a) Driving straight (b) Driving and yurning

Figure 3.2: Generating arc costs from underground mine simulation environment

across all the arc costs in the matrix. In Figure 3.2 an example using the mine environment and the Husky robot is shown. Details about the simulation, the custom navigation used and parameters of the robot are described in Section 3.4. In this example the tunnel environment is like a city block, where most of the sections have similar distances. Figure 3.2 (a) shows the path used for calculating the average time for straight drive through a section of the tunnel. The robot drove 18 meters for 100 times at the average driving time of 19.63 seconds and standard deviation of 1.68 seconds. Figure 3.2 (b) shows the robot goal as the diagonal of the current position in the tunnel, it has to drive one section of the tunnel turn and drive through another section. Two routes are possible and the robot navigation does not have previous knowledge of the map. The robot traversing time is calculated 100 times to obtain the average time of 32.43 seconds and standard deviation of 6.34 seconds. One of the reasons that the driving time in (a) is not half of (b) is because accelerating and stopping are significant part of the traversing time. Also the large deviation value for (b) is that turning in a skid steered robot can be erratic when in maximum speed. The average driving time costs obtained in this demonstration can then be applied for all the vertices to obtain the cost matrix.

3.2.2 Time Window

In addition to the constraint on the limited time of operation of a robot at each deployment, other constraint defined in the problem statement in Section 1.1 is the time window associated to each location. Some vertices might be required to be visited before, due some expiration, and some vertices should not be visited at the beginning because they are not ready. Other consideration is to allow human expert input on how the routes should be generated for a determined application. The proposed idea on how to add priorities to some of the vertices is to use a custom Time Window formulation for the VRP. An initial and finish time is associated to each of the vertices as a constraint parameter. When planning the routes, the vertices must be visited within their time window.

Adding time windows does not require many changes on previous algorithms. The pseudo-code in Algorithm 7 shows how to generate the initial solution for this problem. A starting time s_i is applied for each planned route, where the deployment of the robot is delayed to start at time $s_i = i * MAX\_ROUTE\_TIME$, therefore the time windows are assigned with respect to the robot maximum operation time. As a standard all vertices should be initialized with a time window. When no time window is explicitly assigned, it is going to be from zero to the maximum time allowed for the entire mission. Each route starts at time s_i, the vertices are initially sorted on a list by finish time and starting time window. Then additional *if* statement is added to the Path-Cheapest-Arc algorithm to check if there are any vertices available with time windows within the current route time using the *IsTimeWindowAvailable* method. If there are no vertex with a time window available it will waits until a time window is available or return to the starting position and end that daily route. If it finds an available time window, the windows are sorted for choosing the vertex with the smaller time window remaining using the *GetTimeWindowVertex* method, if more than one smaller window with same size is available it selects the closest node with smaller window.

Algorithm 7: Path-Cheapest-Arc with Time Windows

```
Routes = [];
Vertices = V // Set of all vertices to visit with exception of starting position
TW = TimeWindows // Set of time windows associated to the vertices
$T = \{t_0, ..., t_n\}$ // Max daily route time
$S = [s_0, ..., sn]$ // Starting Route Time
index = 0;
for $t$ in $T$ do
    route=[];
    current_time = S[index];
    while True do
        if IsTimeWindowAvailable() then
            vertex_to_visit = GetTimeWindowVertex(V, TW);
            if current_time+distance(route[-1], vertex_to_visit)+distance(vertex_to_visit, 0)<t then
                route.append(vertex_to_visit) // Add the vertex with the smaller remaining
                    time window to the end of the route
                Vertices.remove(vertex_to_visit) // Remove that vertex to the list of remaining
                    vertices to visit
                current_time += distance(route[-1],vertex_to_visit) // Update route current time
            else
                route.append(0) // Return to initial position
                Routes.append(route);
                index += 1;
                False;
            end
        else
            if current_time+distance(route[-1],0) < t then
                current_time += 1 // Wait one second
            else
                route.append(0) // Return to initial position
                Routes.append(route);
                index += 1;
                False;
            end
        end
    end
end
```

For the optimization using the Local Search algorithms, the new constraints are added to the *LocalSearchOperator* method to not allow an illegal operation of moving a vertex outside its time window and to the IsValid method to check if a proposed candidate solution is valid. The other change is that routes are also calculated in series, beginning at the starting time s_i and the route maximum operation time being defined with respect to the starting time of each route.

3.3 Mission Control

The Mission Control layer is inspired by the Monitored Navigation layer from the STRANDs [8] project, where there is a layer for monitoring the robot navigation and ensuring that the robot completes its tasks when possible. The proposed approach is to use VRP problem formulation as the Mission Control layer during runtime. The idea, as shown in the diagram in Figure 3.3, is that, with a custom problem definition, it is possible to have optimization running on real time that is capable to deal with unexpected events. The proposed method uses a single route problem, where the starting position is the current robot position $v_{current}$ and the end of the route is the initial position v_0. The single day route was chosen to reduce computational cost of the optimization allowing the operation in runtime. It is impossible to find the initial solution and perform local search in large multiple day routes in runtime. The Mission Control layer starts with a daily route that was solved in Section 3.1.2 as the input. Then it keeps solving for that route as it monitors runtime parameters. These parameters are the remaining time for current route, the remaining vertices from the single day route and the robot remaining operation time. The VRP formulation used is a VRP with dropping visits where, in the situation that it is impossible to visit all the remaining vertices from the single day route within the robot remaining operation time, it will choose to drop some of the vertices on the route.

To solve the VRP with dropping visits, the vertices are implemented as structures called disjunction, each optional disjunction can only be visited a maximum of a single time, with $p + \sum_{i \in Disjunction} ActiveVertex(i) = 1$, where p is a boolean variable and if no vertex on the disjunction is visited p must be set to one, then a penalty $p * penalty$ is added to the objective function. The penalty must be a value larger than the cost of visiting the vertices to force the robot to visit all viable vertices. Homogeneous penalty was used in all the vertices .

The objective function is shown in Equation (3.5), where the total cost to minimize during optimization is the sum of the arc costs on the route plus the penalty costs for not visiting the optional Disjunctions. The V is the entire set of vertices $V = (v_0, v_1, ..., v_n)$ and a_{ij} is the associated

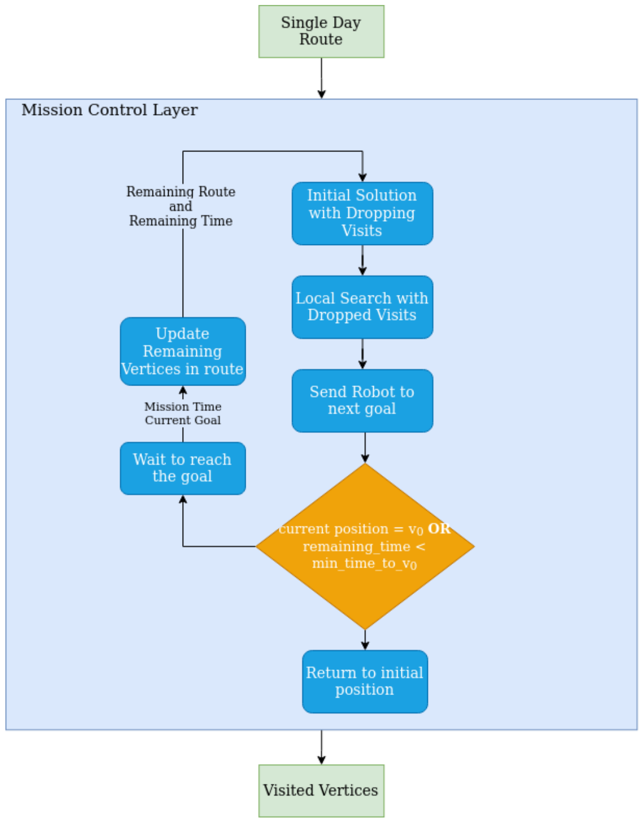

Single Day Route
Mission Control Layer
Remaining Route and Remaining Time
Initial Solution with Dropping Visits
Local Search with Dropped Visits
Update Remaining Vertices in route
Send Robot to next goal
Mission Time Current Goal
Wait to reach the goal
current position = v0 OR remaining_time < min_time_to_v0
Return to initial position
Visited Vertices

Figure 3.3: Mission control layer diagram

non negative arc cost from vertex i to vertex j. The x_{ij} is a binary value, $x_{ij} \in \{0, 1\} \; \forall \, i, j \in V$, where its value is 1 when the arc from i to j is part of the solution and zero otherwise. The p_i is a binary value that if vertex v_i is dropped it is set to one and zero otherwise. The *penalty* is the cost of dropping that vertex. The same algorithms as in Section 3.1.2 are applied to solve the Local Search with a modified *LocalSearchOperator* method that considers dropping visits for creating candidate solutions.

$$L = \sum_{i \in V} \Big(\sum_{j \in V} a_{ij} x_{ij} + p_i * penalty \Big) \tag{3.5}$$

The Mission Control Layer is an iterative process. It starts from the first goal generated in the local search and sends that goal location as a command to the robot navigation. Once that goal is reached the list of vertices to visit in the route is updated and that goal becomes the current starting position $v_{current}$ for the next optimization with the remaining time that is available on the route as the time constrain. This process is repeated until a stopping criterion is reached. There are two stopping criterion, one is the route was completed and that the robot reached the starting vertex v_0, and the other is that the remaining time to return to starting position v_0 is less than the remaining available time in the route plus a constant margin.

3.4 Simulation Environment and Integration

Simulation environments were built in the Gazebo simulator for testing the application of the developed framework in long term robotic operation. The simulations use a modified Husky base [53] with a 32 Channel LiDAR model, the robot is spawned in a known location and true world localization is used.

To integrate the Long Term Route Planning algorithms with the simulation environment, robot Navigation is implemented using move base framework [54], in a similar way as to other long term robotics projects presented in Section 2.1. The robot navigation diagram is shown in Figure 3.4. The core components are (*i*) *global planner* that generates a global 2D path between 2 vertices based on a *global costmap*, (*ii*) *local planner* that generates a smaller path to a point inside the global path and considers *local costmap* information and robot dynamics and (*iii*) *recovery behavior* that is an emergency action that the robot will take to recover from an unexpected situation. The core operation depends on *local costmap* that is a 2D map generated directly from 3D Lidar sensor data that inflates obstacles; *global costmap* that is a 2D static map that is either imported from the simulation environment or built with simultaneous localization and mapping (SLAM); and *odometry* that on this experiments is the perfect value from simulation. The *move base* will output velocity commands to the robot base wheel controller inside the simulation environment. This can be easily transferred to a real robot that uses velocity values as the input to control its motors.

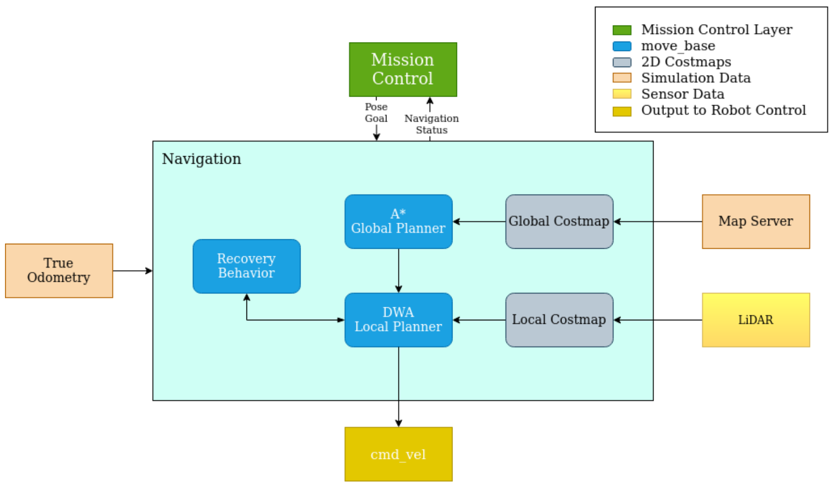


Figure 3.4: Robot navigation diagram

A standard implementation of move base global planner is $A*$ [55], that is used to create the robot global 2D path using the global costmap data. The local planner used is the Dynamic Window Approach (DWA) [12] that discretely samples for the robot control space (dx, dy, $dtheta$) and performs a simulation from the current robot state to predict what happens when different velocities are applied. Then each trajectory is scored based on parameters that are defined in the planner configuration and the highest scoring velocity is chosen as the command velocity output. The recovery behavior used is a turn in place maneuver.

The input for the *move base* is a pose goal, that is $\{x, y, z\}$ position and $\{x, y, z, w\}$ quaternion orientation. When no goal orientation is provided, some value must be chosen as the input for the robot navigation. A practical approach is to use the direction from the current to the goal position. The heading is calculated from current position $\{x_{current}, y_{current}\}$ to the robot next goal $\{x_{goal}, y_{goal}\}$, where $\theta = atan2((y_{goal} - y_{current}) \div (x_{goal} - x_{goal}))$. Then Equations (3.6), (3.7), (3.8) and (3.9) convert the heading to quaternion values.

$$x = 0 \tag{3.6}$$

$$y = 0 \tag{3.7}$$

$$z = sin(\theta/2) \tag{3.8}$$

$$w = cos(\theta/2) \tag{3.9}$$

Two physical simulation environments were built in the Gazebo simulator for testing the mission planner and mission control layer in robotic operation. Both environments were created based on projects that are current being developed at WVU Robotics and that require a long term route planning and scheduling for autonomous operation. These are large environments where there are more locations to visit than a real robot would be capable on a single deployment.

3.4.1 Stone Mine Environment

This simulation, shown in Figure 1.1, was built for the Gazebo simulator using open-source models from the DARPA SubT simulation challenge [56] and is based on a real stone mine located on Lake Lynn, PA. This simulation is a flat environment with a single entrance and 400 pillars configured in a city block style. More pillars can be easily added if required. The standard simulation has no obstacles and has a size of 820 x 800 meters, where each pillar is a square with approximately 18 x

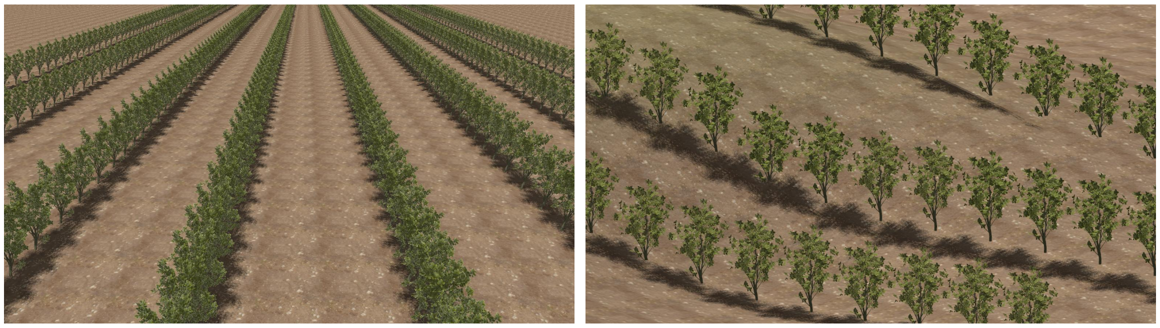

(a) Farm 2D rows (b) Farm 3D rows

Figure 3.5: Farm Gazebo simulation environment

18 meters in size. Small randomness is added to the pillar size, therefore they are not all entirely equal.

This simulation was built based on a stone mine inspection project where a robot team must autonomously inspect every pillar inside an underground mine for assessing its structural integrity. In this project, inspection should be performed by an UAV that is connected by a tether to the ground vehicle, and therefore the robot must be able to park close to each of the pillars on the mine. The proposed Mission Planner and Scheduler and Mission Control methods will contribute to this project by optimizing the sequence of parking goals that the robot should follow on a multiple day mission, monitoring robot operation and improving routes and plans autonomously in case of unexpected events. Other contribution is the possibility of a mine engineer to assign inspection priority to some of the pillars using the Time Window method presented in Section 3.2.2.

3.4.2 Farm Pollination Environment

This simulation was based on the project [5], where the goal is to pollinate bramble flowers using a ground vehicle with a robotic arm attached to it. The simulation environment, as shown in Figure 3.5, is an outdoor farm environment and is implemented in Gazebo. Two terrain version were built: one is a 50 x 50 meters terrain with elevations that were extracted from real world DEM; the other is a 100 x 100 meters flat terrain.

The tree and flower models used were open-source 3D CAD models that were transformed to Gazebo format. The tree is $1m^3$ in dimension and the flowers are attached to it. The trees are displaced in rows 4 meters apart. An invisible wall that is detectable by the LiDAR is added in each row for obstacle avoidance. Since the robot should not be allowed to cross those lines with exception of the area at the end of the rows.

Chapter 4

Results

Experiments were designed to test particularities of applying VRP solutions to the long term robotics planning problem with the following goals:

- Analyse how good a solution is by comparing the local search metaheuristic optimization methods presented in 3.1.2, as well as the improvement compared to the initial solution methods.
- Find how the metaheuristics perform on different scenarios. This include node configurations that are designed based on possible robotics applications, different mission lengths and added priorities that can mimic a human input that partially constrains the plan to have some nodes on specific days.
- Learn how precise these methods are by obtaining statistical data such as repeatability and standard deviation from the various scenarios that were considered.
- Analyse the possibility of the existence of multiple optimal solutions, explore situations where this might happen, such as on symmetric and evenly spaced node locations where there are more than one vertex at the same distance from each node.
- Explore how optimization methods perform as a function of time. This is an important knowledge for applications that have a resource or time limitation for planning; in cases where there are a very large number of vertices; and also in situations where there is a high uncertainty such as the prior map not being entirely accurate with a blocked section or a sub-set of the vertices not being reachable and re-planning is needed during runtime.
- Explore a scenario with an outside depot location that simulates a robot going in an area for exploration and coming back at every deployment. This is different than what is presented on

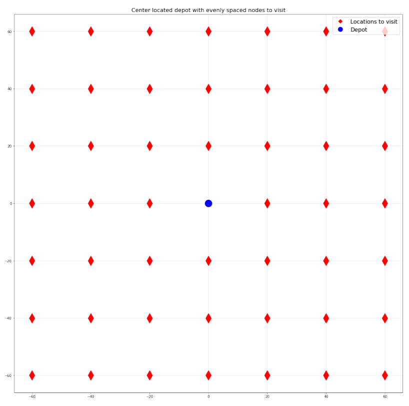
most of delivery scenarios data where a VRP problem is designed with a central depot and tasked to find routes for multiple vehicles. In the proposed testing configuration the plan is to analyse how the algorithm behaves when there is a part of the path that will be traversed on every robot deployment although not explicitly defined in the route plan that allows for each goal to be visited only a single time.

- Visually analyse the routes that were generated. There is a possibility that for some scenarios a human might get insights to how an optimal or best know solution is generated by an algorithm.

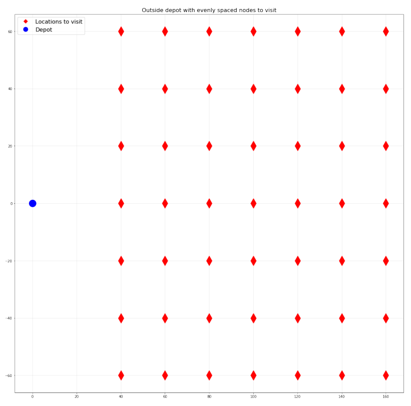
4.1 Routing Experiments

These experiments aim to test the VRP metaheuristic optimization algorithms on scenarios that are likely to be encountered in robotics applications. The initial solution uses the custom path-cheapest-arc method showed in 3.1.1 that enables to systematically obtain the same solution for a specific vertices configuration. The Operations Research Tools library [51] is used as the implementation for the metaheuristics algorithms in Section 3.1.2. All the vertices and arc costs are defined as integers to improve algorithms performance by only using integer operation. The number of vertices is limited by how indexing is done inside the algorithm, which is defined as a type short integer, and, therefore, $32,767$ is the maximum number of vertices that it can be solved for. All the experiments were performed in the same computer, with an Intel Core i7-7700K CPU @ 4.20GHz and 32GB of RAM memory.

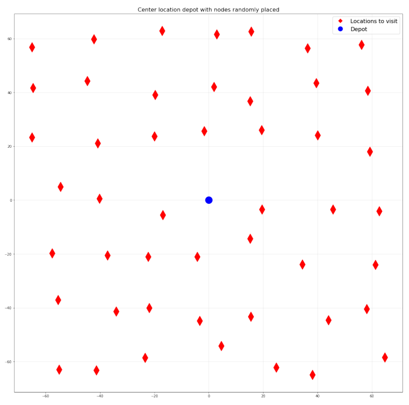
In these experiments vertices are defined as position on the 2D plane and they are unitless. In real world applications these values can be seen as length, energy or time vertices. The Manhattan Distance is used as arc costs among the vertices. The maximum operation time constraint for the problems were chosen in such a way that the initial solution using path-cheapest-arc would generate routes that cover all the defined days of operation. When the local search is performed the number of routes that leave the starting position might decrease. Routes from starting position to starting position are allowed by the problem definition for obtaining a solution but are disregarded because they have zero cost. Figure 4.1 show some of the test configurations, where the blue dot represent the starting position and the red diamonds represent the vertices to be visited.



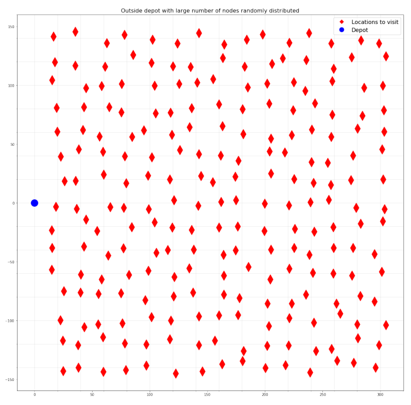
(a) Vertices evenly spaced and center depot



(b) Vertices evenly spaced and outside depot



(c) Vertices randomly spaced and outside depot



(d) Vertices randomly spaced and outside depot

Figure 4.1: Vertices configuration for VRP problems

4.1.1 Route Analysis on Evenly Distributed Vertices and Center Starting Position

Figure 4.1 (a) presents vertices configurations for an environment where there is a single depot localized in the center. There are 48 vertices in this configuration and the distance between adjacent vertices are 20 on both the x and y directions, resulting on 20 being the shortest distance from the depot and 120 being the longest. The problem in this scenario was modeled for planning a maximum of 5 days and a maximum distance of 350. The initial total route distance calculated with path-cheapest-arc is $1,520$ which is close to the maximum total distance allowed of $1,750$ (5×350). All the four algorithms were able to repeatedly converge to the same solution of $1,160$ in less than five seconds, an improvement of 23.6% in the solution. The routes found using each of the local search methods were also identical, although because the map is symmetric they can be flipped on the origin, demonstrating that there are multiple best solution routes in this scenario.

Figure 4.2 presents the routes found by the optimizations where each of the routes covers a different direction. Only four routes are seen and the last route does not leave the starting position because after optimization it was possible to visit all the vertices in only four days. The number of days of a mission is a flexible constraint where the constraint sets the maximum allowed number of days for the route planner. But the local search will minimize the number of routes to the minimum number of routes that respects the other problem constraints. The optimal solution for a problem without constraints on driving time would be the TSP solution where all the vertices are visited in the same day.

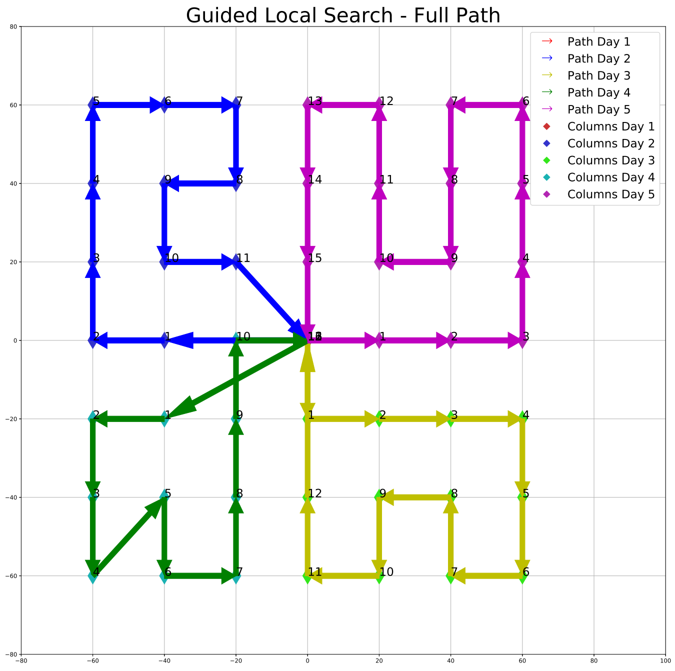


Figure 4.2: Planned routes for 5 days with evenly distributed vertices and center starting position

4.1.2 Route Analysis on Evenly Distributed Vertices and Outside Starting Position

The vertices configuration is presented in Figure 4.1 (b), where there are 49 vertices disposed similar as in the first problem, but the single starting location is outside of the vertices, forcing all the routes to leave the depot in the same direction, causing overlap. On this problem, the maximum number of days was set to five and distance to 500. The total initial route distance using the path-cheapest-arc is 2,320 and again all the optimization methods were able to repeatedly converge to the same solution of 1,600 total route distance in less than five seconds that is a 31.8% improvement. Figure 4.3 shows the routes found, where all the vertices were able to be visited in four days, one of the planned routes only visited three vertices that were located far from starting position and another route visited 20 vertices that is 40.8% of the locations.

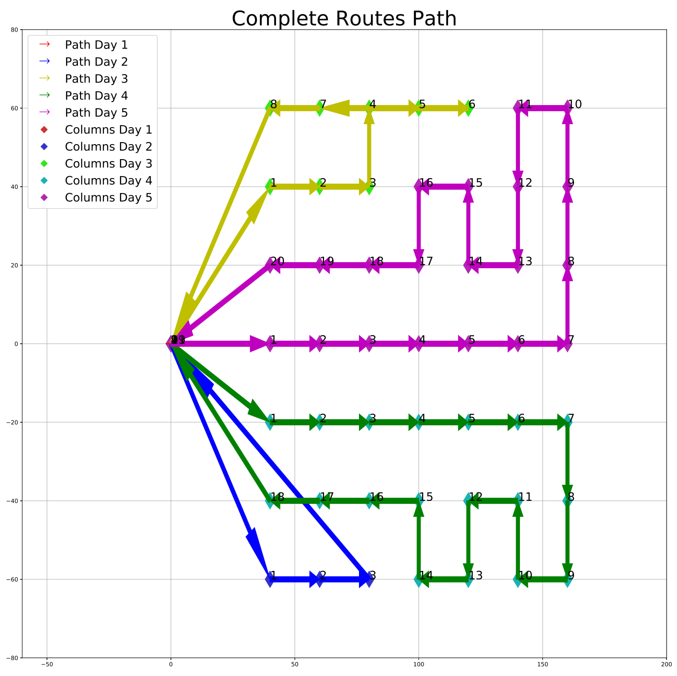


Figure 4.3: Planned routes for 5 days with evenly distributed vertices and outside starting position

4.1.3 Route Analysis on 48 Random Vertices and Center Starting Position

In this problem, the vertices configuration is the Figure 4.1 (c), where the 48 vertices were first disposed on a grid as in first problem and then a random noise of ± 5 was added to each vertex location. The problem constraints were defined as 375 the maximum route distance and the maximum number of days is five. The initial route distance of 1549 was obtained using path-cheapest arc and the local search was performed 100 times for each of the methods. Results are shown in

Table 4.1: Total route distance result for 48 random vertices and center starting position - average, max, min and standard deviation for each of the methods

Optimization Method	Average	Max	Min	Standard Deviation	Avg Improvement(%)
GLS	1231.71	1247	1225	5.58	20.48
Simulated Annealing	1236.00	1298	1225	8.48	20.20
Greedy Descent	1231.60	1236	1225	5.38	20.49
Tabu Search	1231.60	1236	1225	5.38	20.49

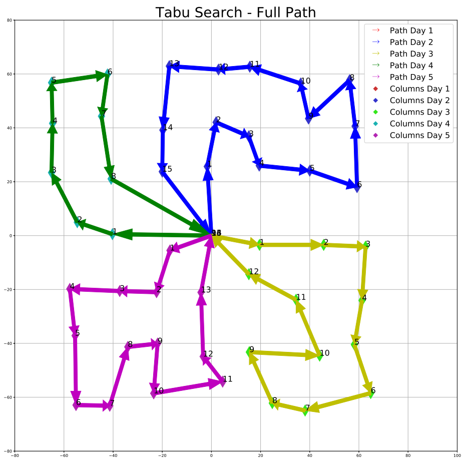
Table 4.1 where all the algorithms were able to reach the best solution of $1,225$, an improvement of 20.92% from the initial solution. However the simple randomness added to the vertices location made optimization methods reach local minimum sometimes, stopping criterion time was set to a large number, but optimization exited due to the other stopping criterion of being unable to find a better solution for so many times. Small difference in performance were found in this scenario. The SA performed worst with the largest standard deviation value and average solution. The greedy descent and TS had the best performance.

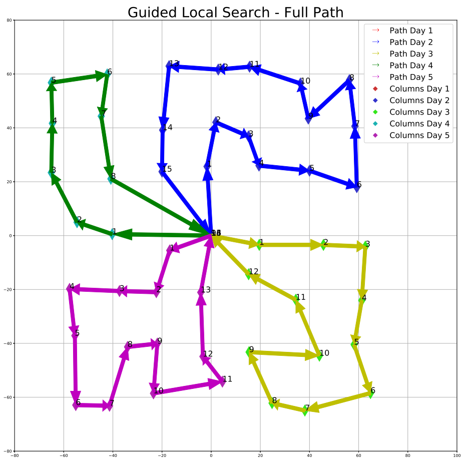
Figure 4.4 shows the routes for the different methods, where the route shown for GLS and SA was one with the total route distance of $1,236$ and for the Greedy Descent and TS it was the one with total route distance of $1,225$. They differ in one vertex close to the starting position that is assigned to a different day route and this results in the different total distances.

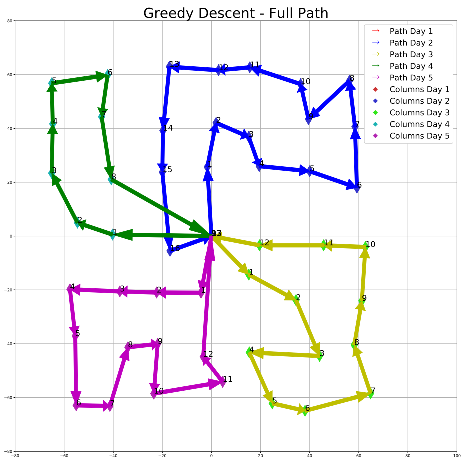
4.1.4 Route Analysis on 225 Random Vertices with Outside Starting Position

This problem, as shown in Figure 4.1 (d), is a combination of the first three problems, where the vertices are distributed on a grid like configuration, starting position located outside and small randomness added to each vertex position. Other change is that the number of vertices to visit increase to 225. Constraints are that there is five days to visit all the vertices and a maximum daily distance of $1,750$. The total initial solution distance cost using path-cheap-arc is $7,355$. The local search was performed 100 times for each of the methods and results for the optimization are presented on Table 4.2, where all the methods were able to reach the best solution of $5,409$, that is an improvement of 26.25% in the total route solution. There is a small difference in performance of the algorithms where TS has a smaller deviation and better average improvement with respect to the initial solution compared to the other methods.

Some planned routes are shown in Figures 4.5, 4.6, 4.7, 4.8 for TS, GLS, SA and Greedy Descent respectively. These routes are significantly different from the others, but the total distance traversed


(a) Tabu Search


(b) Guided Local Search


(c) Greedy Descent

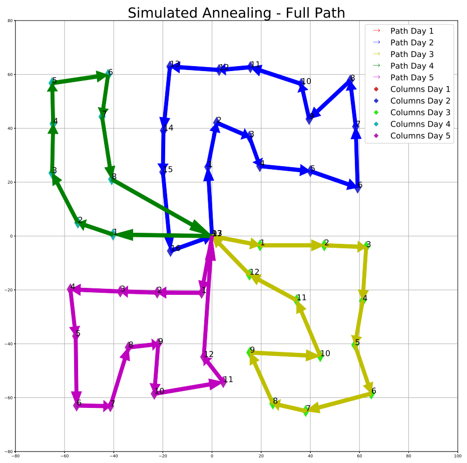

(d) Simulated Annealing

Figure 4.4: Planned routes for 5 days with random vertices and center starting position - (a) Tabu Search, (b) Guided Local Search, (c) Greedy Descent, (d) Simulated Annealing

is similar for each of them.

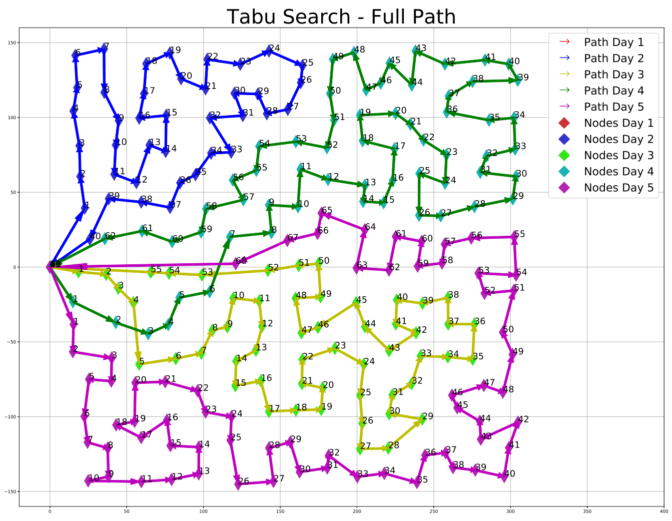


Figure 4.5: Tabu search planned routes for 5 days with 225 random vertices with outside starting position

4.1.5 How Optimization Improves Over Time

In this problem the goal is to learn how optimization improves the solution over time. The scenario where this was tested includes a starting position located outside the locations to visit, and 900 vertices are placed on a grid with a small randomness added to each vertex position in a similar configuration as to Figure 4.1 (d).

Given an initial solution obtained from the path-cheapest-arc and using time constraint of $6,300$ and the maximum number of deployments days of five, the SA, GLS, TS and Greedy Descent were evaluated ten times each and optimization time as stopping criterion. Figure 4.9 presents the average and standard deviation total routes distance over time, where there is a sharp decrease in the total route cost on the first few seconds and all the methods converging to the same solution after some

Table 4.2: Total route distance result for 225 random vertices with outside starting position - average, max, min and standard deviation for each of the methods

Optimization Method	Average	Max	Min	Standard Deviation	Avg Improvement(%)
GLS	5,499.10	5,708	5,409	67.43	25.22
Simulated Annealing	5,499.84	5,708	5,409	67.30	25.22
Greedy Descent	5,499.10	5,708	5,409	67.49	25.22
Tabu Search	5,498.72	5,708	5,409	63.79	25.24

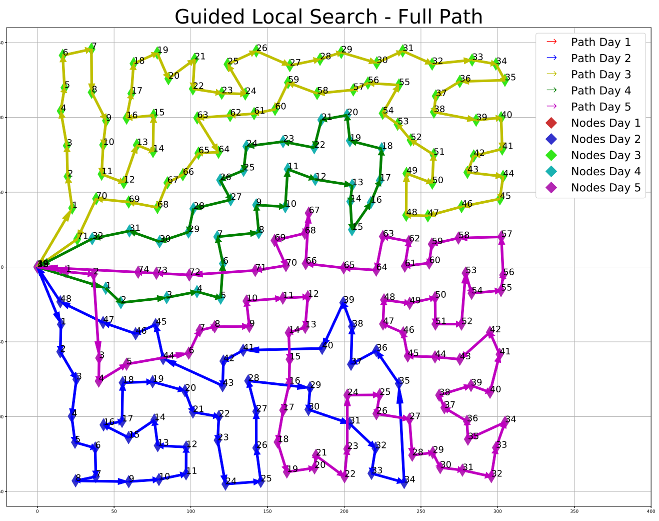


Figure 4.6: Guided Local Search planned routes for 5 days with 225 random vertices with outside starting position

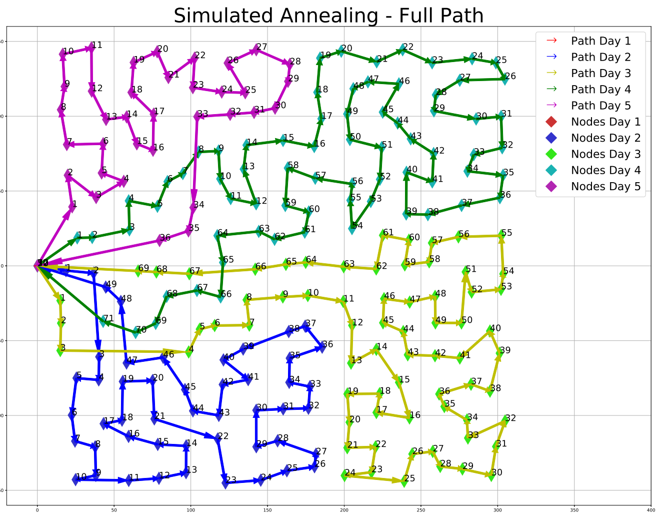


Figure 4.7: Simulated Annealing planned routes for 5 days with 225 random vertices with outside starting position

time. Deviation grows for each method before it starts converging. Total distance for the initial solution is 29,974 and the solution after optimization is 23,116 which is an improvement of 22.87% in less than 100 seconds.

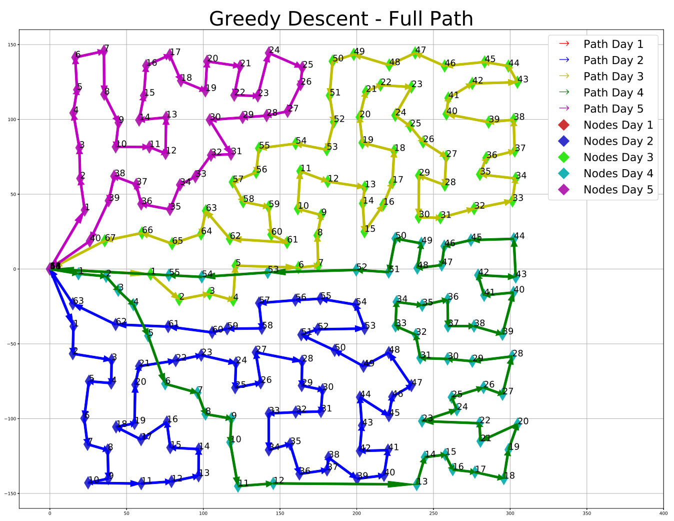


Figure 4.8: Greedy Descent Planned Routes planned routes for 5 days with 225 random vertices with outside starting position

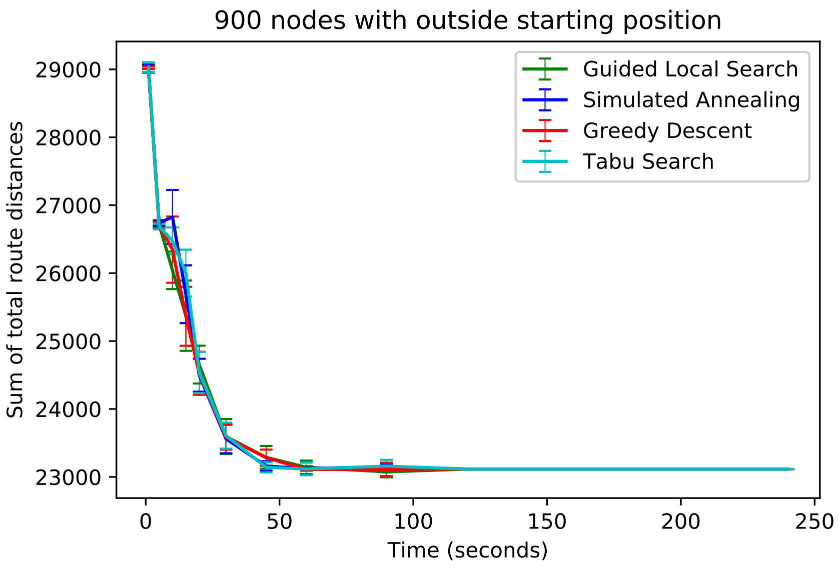


Figure 4.9: Optimization performance over time for 900 vertices and 5 days route planning

However robots might have many goals in a mission and there are situations that is infeasible to converge to a solution using the presented methods. When increasing the number of nodes to $2,500$ and keeping a similar configuration of the vertices distribution, five days maximum operation and a $17,000$ maximum distance per route, the optimization will not converge in a small amount of time.

Table 4.3: Total route distance results for 2500 nodes and 5 vehicles at time 600 seconds - average, max, min and standard deviation for each of the methods

Optimization Method	Average	Max	Min	Standard Deviation	Avg Improvement(%)
GLS	58,123.2	59,770	58,212	245.85	20.54%
Simulated Annealing	58,034.2	59,878	58,566	240.00	20.66%
Greedy Descent	58,268.4	59,226	57,990	504.07	20.34%
Tabu Search	58,475.0	59,572	58,022	333.73	20.05

In this experiment the local search starts from an initial solution using path-cheapest-arc with the total route distance of $73,148$ and each optimization method is evaluated ten times for the period of 600 seconds.

Figure 4.10 shows how results improve over time with local search and Table 4.3 presents the optimization results at the time 600 seconds. The greedy descent resulted in the best total distance solution of $57,990$ and was able to reach these better solutions faster than others at time 360. However no improvement was made from time 360 to 600, where it is fair to infer that a local optimum was reached. This algorithm also has the worst standard deviation values at 504. The SA algorithm kept improving and was able to reach the best average solution at time 600 with a 20.66% improvement compared to the initial solution.

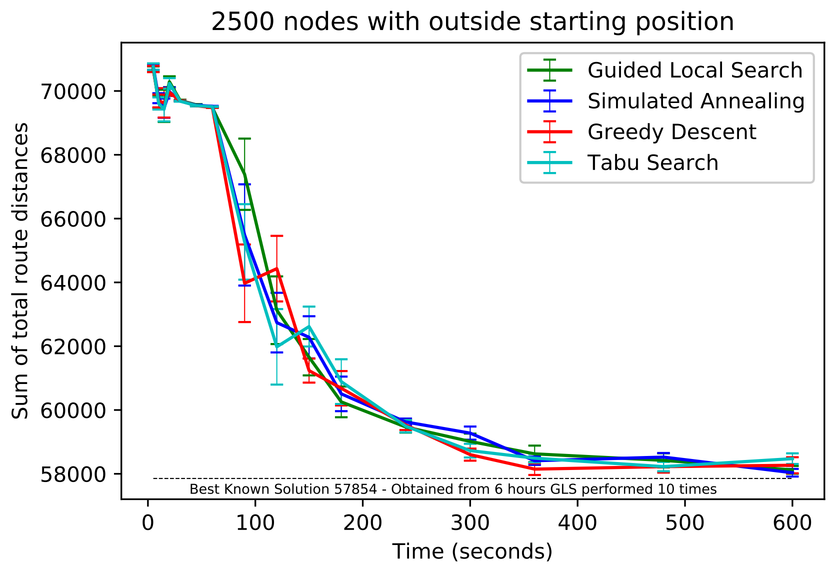


Figure 4.10: Optimization performance over time for 2500 vertices and 5 days route planning

The previous example demonstrated how local search performs when the robot has to traverse long routes, where at each route the robot would visit 500 locations on average. Other scenario is spreading the mission constraint to have more days and a smaller distance constraint. This example

Table 4.4: Total route distance results for 2500 nodes and 30 vehicles at time 600 seconds - average, max, min and standard deviation for each of the methods

Optimization Method	Average	Max	Min	Standard Deviation	Avg Improvement(%)
GLS	85787.4	83832	89378	2023.59	37.14%
Simulated Annealing	85743.3	83686	87384	1179.79	37.17%

uses the same 2,500 vertices location, but with the new constraints that the robot has thirty days to complete the mission and a more strict distance constraint of 4,750 per route. Initial solution total route distances using path-cheapest-arc was 136,476. Figure 4.11 present how optimization improves the total route distances over time and Table 4.4 present the average, maximum, minimum, standard deviation and average improvement with respect to the initial solution. Both GLS and SA improved around 37% which is a much larger improvement in the solution for the scenario where routes were planned for thirty days compared to the same vertices location and five days route.

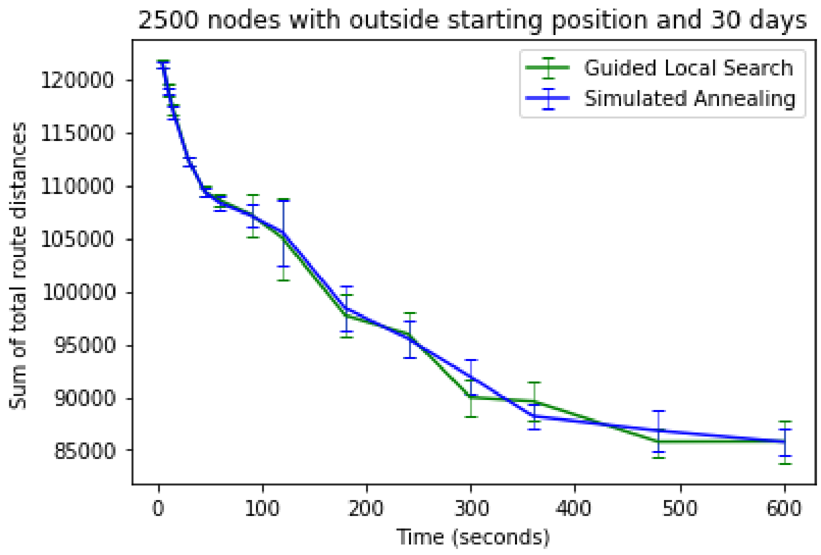


Figure 4.11: Optimization performance over time for 2500 vertices and 30 days route planning

4.1.6 VRP with Time Windows for Robotics Planning

The purpose of this experiment is to analyse how manually assigning some kind of priority or constraint to some of the goal locations can affect the planned routes. Figure 4.12 shows the added priorities to some of the nodes, where the vertices within the red rectangle have a time window priority of [0,1500] and therefore must be visited on the first day and the vertices within the green rectangle have a time window of [2500,4500] and therefore must be visited at the end of day two or at the beginning of day three.

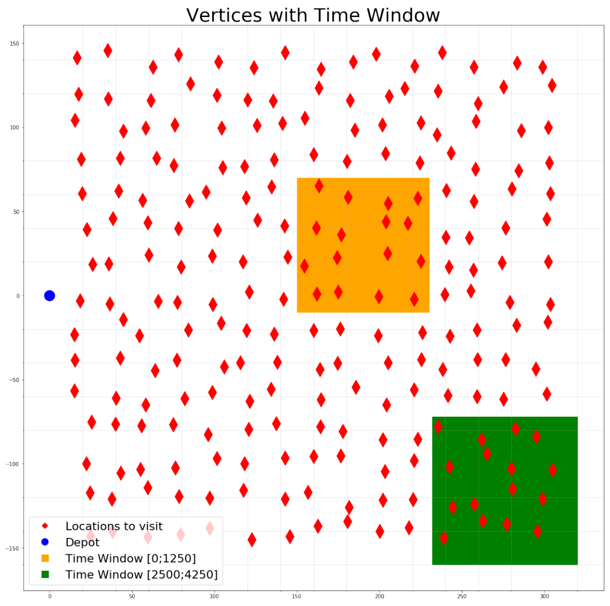


Figure 4.12: Vertices location and time windows

VRP with Time Windows and delayed robot deployment implementation is used to assign the priorities to some of the vertices. All costs are modeled as time, and vertices within the selected rectangles regions have a time window constraint added to it, 0 to $1,250$ seconds to the vertices in the orange rectangle and $2,500$ to $4,250$ seconds in the green rectangle. The robot is modeled to leave the starting position at times multiple of the maximum route time constraint, in this problem at time 0, 1750, 3500, 5250 and 7000 seconds. The time constraint is defined as the time difference between leaving and returning to the depot. The problem uses the same constraints as in Problem 4 with five days and maximum total distance of $1,750$ for each route. The initial solution was calculated using the modified path-cheapest-arc for time windows that checks for remaining nodes within required time windows. The total initial routes time was $7,588$ seconds and the routes are shown on Figure 4.13 where the vertices within the orange rectangle were visited at the beginning of day one and the ones within the cyan rectangle were visited in the second part of the day two following the defined time window constraints.

The local search was evaluated 100 times for each method and the results are presented in Table 4.5, where almost all the tests converged to a solution with total route time of $5,614$ seconds and very small standard deviation. This is a 26% with respect to the initial solution. A reason for that is by having more constraints to a problem there are fewer valid solutions and it was easier for the optimization methods to converge to that solution. The routes obtained by the different methods best solution are the same and are shown in Figures 4.14 where the rectangles represent regions with

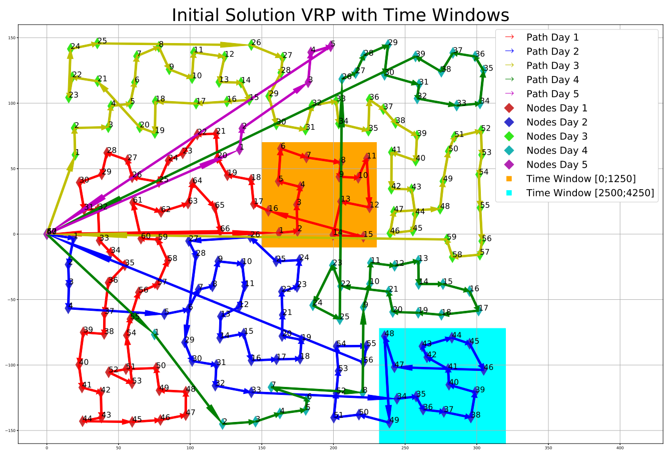


Figure 4.13: Initial solution routes for 5 days with time windows

Table 4.5: Total route distance results for 225 vertices with time windows - average, max, min and standard deviation for each of the methods

Optimization Method	Average	Max	Min	Standard Deviation	Avg Improvement(%)
GLS	5614.50	5664	5614	4.97	26.00%
Simulated Annealing	5614.14	5628	5614	1.39	26.01%
Greedy Descent	5614.49	5663	5614	4.88	26.00%
Tabu Search	5614.16	5630	5614	1.59	26.01%

added priorities. A downside of this method is that the assigned time windows must be sufficient soft to be able to generate a valid initial solution and problems that are very strict are not solvable.

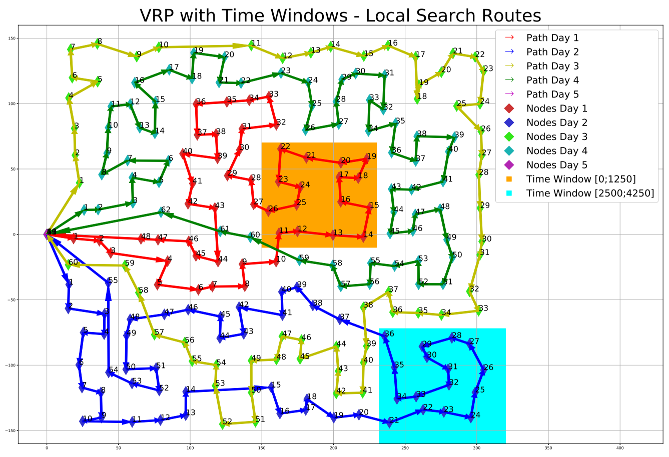


Figure 4.14: Local search planned routes - 5 days with time windows

4.2 Mine environment

The goal of this experiment is to demonstrate how VRP long term route planning integrates with a robot by using the underground mine environment simulation presented in Section 3.4 to inspect a region with 40 columns (5x8) as shown on Figure 4.15. The starting position is represented in blue and the regions that the robot should visit in green. The Husky robot base was used in the simulation, where the navigation and driving parameters were tuned due the perfect IMU and wheel interaction with the environment. Arc cost estimation was done using the robot average driving time method presented in Section 3.1.1 with the tuned robot, where the time used was the average time for driving a straight line across one section of the mine which is equal to ≈ 20 seconds (19.63).

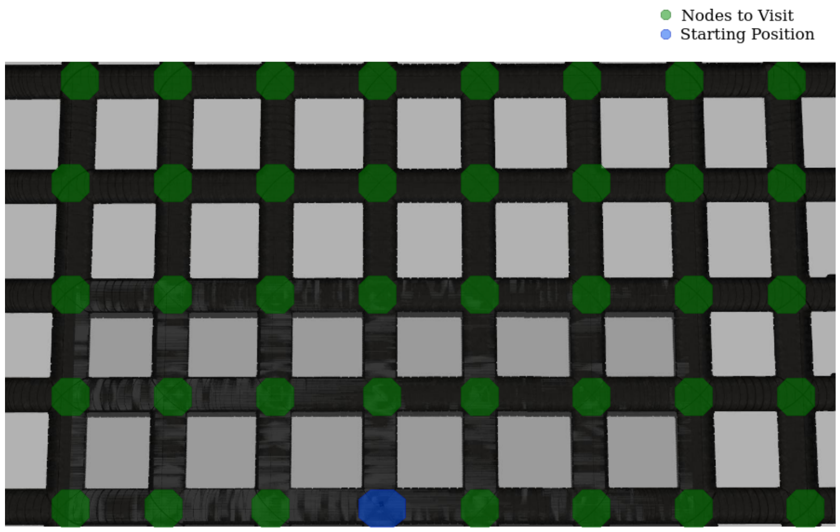
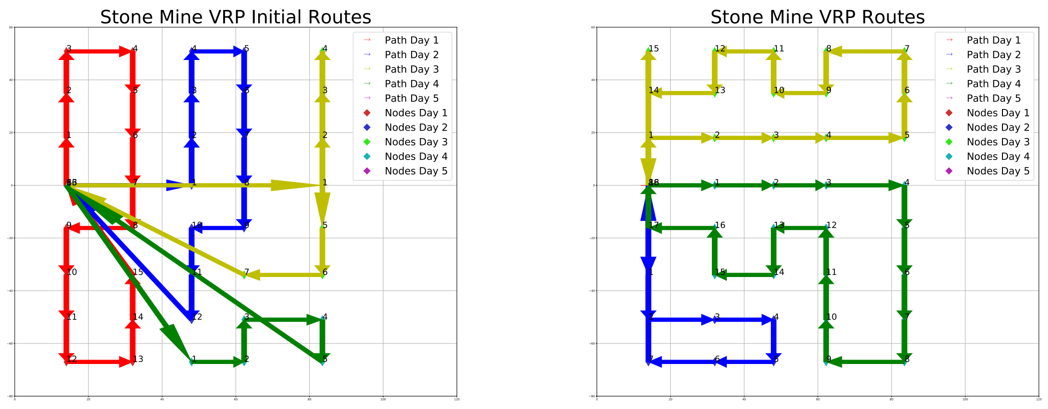


Figure 4.15: Underground mine goals

4.2.1 VRP

A difference when using vehicle routing as a robotic planner is the integration with the robot problem. The matrix of time costs is used to obtain the initial solution and local search. Then the generated route order is transformed to vertices coordinates. Which are converted to position and orientation goals as explained in Section 3.4.

The VRP was solved for a four days mission, the maximum time allowed for each route was 380 seconds. This maximum allowed time constraint was set so the initial solution yield complete routes for the four days. The initial solution generated using path-cheapest-arc has a total route time of 1,440 seconds with all the routes having 360 seconds. After optimization is performed all methods converge to a total route of 960 seconds, with two routes with 360 seconds, one route with 240



(a) Routes initial solution

(b) Routes after local search is performed

Figure 4.16: Comparison VRP routes generated for underground mine Experiment

seconds and the last route is zero. Figure 4.16 present a comparison between the routes generated on the initial solution and after local search was performed.

4.2.2 Robot Simulation

In this experiment the Husky robot was required to follow the optimization generated routes. The robot drives inside the gazebo simulation underground mine environment through all the way-points in the routes. The A* is used as the global planner to get the path between goals and the DWA is used as the local planner that controls robot driving to that goal. As a goal is reached the next one is fed into the global path planner.

While driving, the robot is also exploring and mapping the mine environment with the LiDAR data as shown in Figure 4.17. No prior cost map is provided and re-planning is done at a constant rate of $0.1Hz$ for the A* global planner and at $10Hz$ for the DWA local planner to deal with the new map information and driving uncertainty.

The robot took 257.01 seconds to complete the first route, 512.19 seconds to complete the second route and 513.48 to complete the third day route. This was more than the maximum time allowed in the problem definition. This happened because the arc costs chosen were very optimistic, they did not consider the turning maneuvers and situations where robot heading and goal direction were 180° apart requiring additional time to complete those sections of the route.

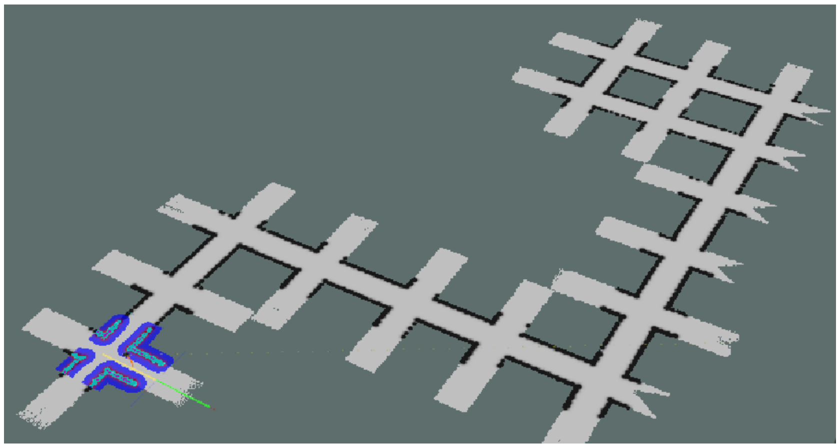


Figure 4.17: Exploration of the underground mine while driving

4.2.3 Robot Simulation with Mission Control

In this experiment, the Mission Control layer addresses the issue of violating the maximum daily operation time presented in the last subsection. This layer monitors the operation of the robot in runtime, by constantly re-planning the current route and dropping some vertices if necessary to respect the daily route time constraint. When a vertex is dropped from a route a constant homogeneous penalty is added to the loss function. In this problem the VRP routes for the entire mission are also re-planned at the end of each day. The generated routes are then sorted by time with the longest being the next route that the robot must follow.

With the mission control added to the simulation, the following times were obtained: 350.95 seconds for first route, 378.50 seconds for the second route 259.60 seconds for the third route and 52.74 for the new fourth route. The vertex v_1 was dropped from the first route, then routes were re-planned for the remaining three days. In the next route three vertices (v_4, v_{14}, v_{22}) were dropped. The remaining vertices were re-planned for the last two days where all the vertices were within a single day route, but again the vertex v_1 was dropped for the last day of robot operation. All the routes were visited throughout the four days mission respecting the limited time of robot operation and the real routes taken by the robot are shown in Figure 4.18.

4.3 Farm Environment

As presented in Section 3.4, the farm environment is composed of tree rows that must be pollinated by a robotic ground vehicle with a robotic arm mounted to it. In this experiment the Husky robot receives both position and orientation goals to visit. Figure 4.19 shows the poses that the robot must

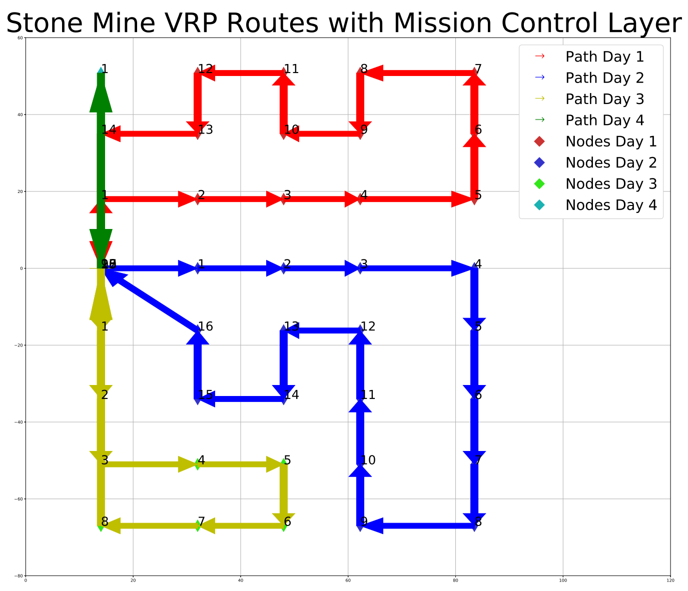


Figure 4.18: Real routes traversed by the robot in the underground mine with mission control layer

reach to start the flower pollination process in each tree. Beginning every route from the starting position, the robot task is to visit five rows of plants with six trees in each row. Because a tree must be visited by both sides, there are 60 locations to visit.

A difference from the previous experiment in the mine environment (Section 4.2) is how the cost matrix is calculated for this scenario. Equation (4.1) and (4.2) were formulated to address the constraint that the robot must drive through rows, where there are trees to pollinate in both sides of each row, and that the robot can only move to another row by first going to one of the edges of the current row. Equation (4.1) shows the arc cost a_{ij} for visiting a goal location in a different row. The arc cost is the sum of the y coordinate from current and goal position multiplied by a constant c_1, the absolute distance between rows in the x direction multiplied by a constant c_2, and a turning cost c_3. Equation (4.2) is used when the location goals are in the same row. The arc cost a_{ij} is the absolute distance between vertices in the y direction plus a turning constant c_4. Constant values were set according to driving data from the robot in the simulation.

$$a_{ij} = (y_i + y_j) * c_1 + |x_i - x_j| * c_2 + c_3 \tag{4.1}$$

$$a_{ij} = |y_i - y_j| * c_1 + c_4 \tag{4.2}$$

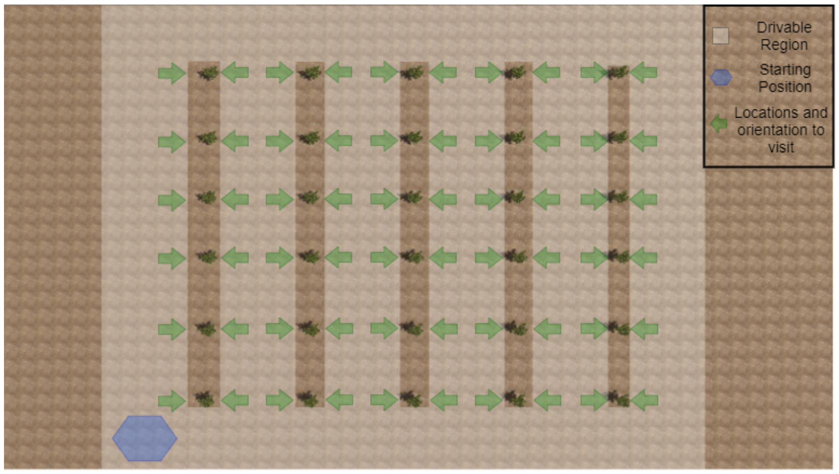


Figure 4.19: Farm goals

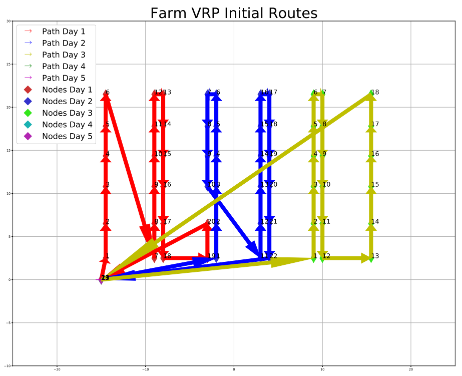
4.3.1 VRP

The constraints for this problem is that the robot must visit the 60 locations within five days with a maximum daily route time of 720 seconds. The routes obtained in the initial solution using path-cheapest-arc are shown in Figure 4.20 (a), where the total route time is $2,019$ seconds. The local search solution is shown in Figure 4.20 (b), where all the methods converged to this solution and the total route time was $1,999$ seconds, which is an improvement of 1.0% of the initial solution.

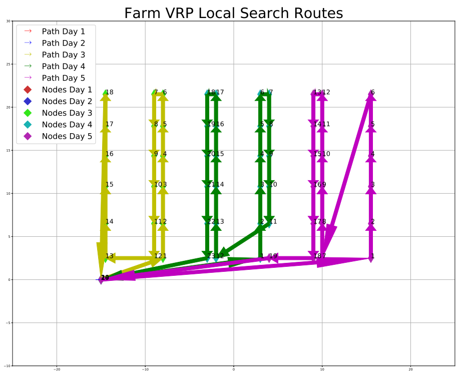
4.3.2 Robot Simulation with Mission Control

The routes were executed in the Gazebo simulation using the proposed mission control layer with the Guided Local Search metaheuristic and dropping visits. Due to the a high number of turning maneuvers required during execution of the routes which are not very consistent in the skid steered robot, there is a difference between planned and the executed routes. The routes traversed by the robot are shown in Figure 4.21. The robot took five days to reach all the goals, and the routes time were 703.16, 689.75, 694.33, 568.81 and 13.73 seconds respectively, those are within the maximum route time limit of 720 seconds. The total time was $2,669.78$ seconds which is worse than the estimated time by the VRP local search.

The developed system with robot experiments in the presented simulation environment is demonstrated in this video: `https://youtu.be/iQwJFOGEhjY`



(a) Routes initial solution



(b) Routes after local search is performed

Figure 4.20: Comparison VRP routes generated for farm experiment

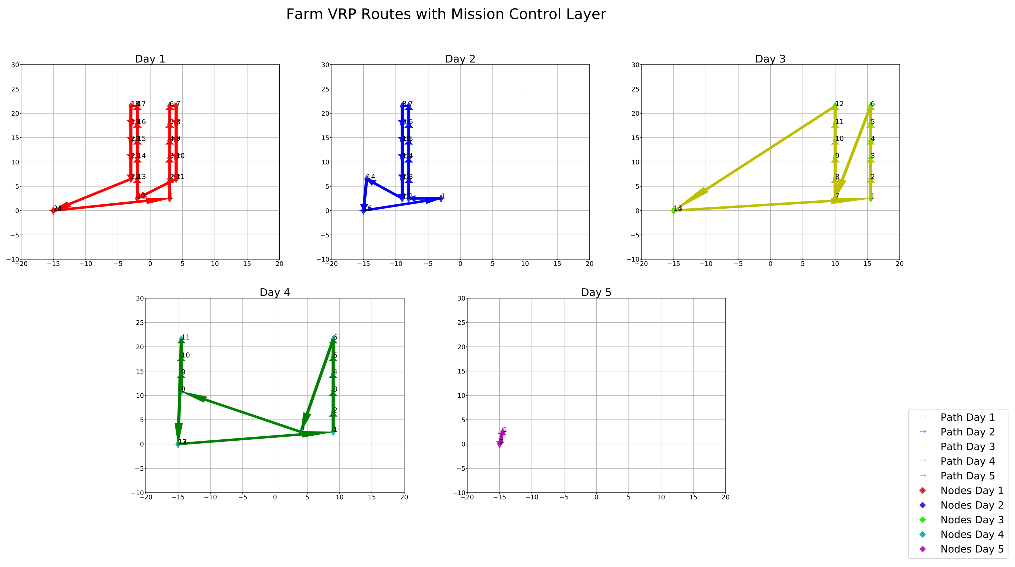


Figure 4.21: Real routes traversed by the robot in the farm with mission control layer

Chapter 5

Conclusion and Future Work

This thesis presented a method for planning and scheduling multiple days routes for robotic missions and adjusting and improving these routes during robot operations. This was accomplished by adapting solutions from the delivery industry to that problem. Constraints to this problem are related to robot hardware limitation and modeled as time. The goal is to minimize the total route time. The cost estimation is modular and independent from the optimization part and a solution using robot physical simulation was presented that is easily applied to other robots or environments.

The proposed solution to obtain the multiple routes plan was a 2 step process where an initial greedy solution is generated following the problem constraints and then improved using metaheuristic optimization. Guided Local Search (GLS), Greedy Descent, Simulated Annealing (SA) and Tabu Search (TS) were tested in scenarios that are envisioned in robotics application. These methods presented similar average improvement with respect to the initial solution. Greedy Descent presented a larger standard deviation value that is explained due to the lack of means to avoid local optimum. GLS, TS and SA presented comparable results in the many different scenarios with a slightly better performance for the GLS.

The proposed use of time windows constraints with respect to theoretical maximum daily route of the robot proved to be an innovative and simple way to add human expert input, priorities and expiration time to the multiple route planner and scheduler without disrupting its operation. This also added flexibility to the to planner to be applied to more robotics application situations.

The mission control layer was a reliable way to ensure that the robot respects its constraints while following the generated routes. Optimization is performed iteratively in the current robot route, and the algorithm is able to monitor robot operation, continuously improve the route and adapt it by dropping visits when unexpected events happened, ensuring that the robot will respect its time constraints and return to the starting position. Homogeneous penalties were added for dropping the visits. Other distribution of penalties can be used to prioritize dropping some routes over others, and might be useful depending on how critical is the safe return of the robot in an application. The routes obtained with the local search methods for the mine environment where the vertices displacement is in a city block like configuration presented a significant improvement compared to the greedy initial solution. However for the farm environment, where the robot must drive through rows, the improvement was very small.

The generated routes can also be used as a multi robot planner for independent tasks where you can simply assign the multiple day planned routes to multiple robots. An heterogeneous fleet of robot can also be used by assigning different daily constraints to the list of constraints. However this method currently does not allow for cooperation among robots. Future work includes developing a structure incorporating elements from VRP with Backhauls and VRP with Pickup and Delivery, where a vertex will require to be visited by more than one robot at the same time and to allow the robots to work together to complete their task.

Future work includes the deployment of the presented method in real robotic missions. This will include applying the mission planner and control layers in the robot for underground stone mine pillar inspection project and in the farm environment for pollination.

Other subsequent work is learning goals during operation time and integrating them to the mission planner and mission control layer. Additional application is the development of a modified Time Window VRP formulation together with the dropping visit capabilities and use it to solve a coverage problem, where time windows would be defined as a gradient through the entire map and penalties to drop visit would dynamically change inside a small region as the robot explores that area, prioritizing the robot to move to unexplored regions.

Bibliography

[1] ANYbotics AG, “Worlds First Autonomous Offshore Robot,” 2018, https://www.anybotics.com/worlds-first-autonomous-offshore-robot/, last checked on 2020-05-23.

[2] Nuchter, A., Surmann, H., Lingemann, K., Hertzberg, J., and Thrun, S., “6D SLAM with an application in autonomous mine mapping,” *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, Vol. 2, IEEE, 2004, pp. 1998–2003.

[3] Marshall, J. A., Bonchis, A., Nebot, E., and Scheding, S., “Robotics in mining,” *Springer handbook of robotics*, Springer, 2016, pp. 1549–1576.

[4] Underwood, J. P., Calleija, M., Taylor, Z., Hung, C., Nieto, J., Fitch, R., and Sukkarieh, S., “Real-time target detection and steerable spray for vegetable crops,” *Proceedings of the International Conference on Robotics and Automation: Robotics in Agriculture Workshop, Seattle, WA, USA*, 2015, pp. 26–30.

[5] Strader, J., Nguyen, J., Tatsch, C., Du, Y., Lassak, K., Buzzo, B., Watson, R., Cerbone, H., Ohi, N., Yang, C., et al., “Flower Interaction Subsystem for a Precision Pollination Robot,” *arXiv preprint arXiv:1906.09294*, 2019.

[6] Anderson, N., Underwood, J., Rahman, M., Robson, A., and Walsh, K., “Estimation of fruit load in mango orchards: tree sampling considerations and use of machine vision and satellite imagery,” *Precision Agriculture*, Vol. 20, No. 4, 2019, pp. 823–839.

[7] Brooks, R. A., “Intelligence without representation,” *Artificial intelligence*, Vol. 47, No. 1-3, 1991, pp. 139–159.

[8] Hawes, N., Burbridge, C., Jovan, F., Kunze, L., Lacerda, B., Mudrova, L., Young, J., Wyatt, J., Hebesberger, D., Kortner, T., et al., "The strands project: Long-term autonomy in everyday environments," *IEEE Robotics & Automation Magazine*, Vol. 24, No. 3, 2017, pp. 146–156.

[9] Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., and Konolige, K., "The office marathon: Robust navigation in an indoor office environment," *2010 IEEE international conference on robotics and automation*, IEEE, 2010, pp. 300–307.

[10] Smith, D. E., Zuber, M. T., Frey, H. V., Garvin, J. B., Head, J. W., Muhleman, D. O., Pettengill, G. H., Phillips, R. J., Solomon, S. C., Zwally, H. J., et al., "Mars Orbiter Laser Altimeter: Experiment summary after the first year of global mapping of Mars," *Journal of Geophysical Research: Planets*, Vol. 106, No. E10, 2001, pp. 23689–23722.

[11] Kunze, L., Hawes, N., Duckett, T., Hanheide, M., and Krajník, T., "Artificial intelligence for long-term robot autonomy: A survey," *IEEE Robotics and Automation Letters*, Vol. 3, No. 4, 2018, pp. 4023–4030.

[12] Fox, D., Burgard, W., and Thrun, S., "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, Vol. 4, No. 1, 1997, pp. 23–33.

[13] Veloso, M., Biswas, J., Coltin, B., Rosenthal, S., Kollar, T., Mericli, C., Samadi, M., Brandao, S., and Ventura, R., "Cobots: Collaborative robots servicing multi-floor buildings," *2012 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2012, pp. 5446–5447.

[14] Veloso, M. M., Biswas, J., Coltin, B., and Rosenthal, S., "CoBots: Robust Symbiotic Autonomous Mobile Service Robots." *IJCAI*, 2015, p. 4423.

[15] Coltin, B., Veloso, M., and Ventura, R., "Dynamic user task scheduling for mobile robots," *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[16] Booth, K. E., Tran, T. T., Nejat, G., and Beck, J. C., "Mixed-integer and constraint programming techniques for mobile robot task planning," *IEEE Robotics and Automation Letters*, Vol. 1, No. 1, 2016, pp. 500–507.

[17] Mudrova, L. and Hawes, N., "Task scheduling for mobile robots using interval algebra," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 383–388.

[18] Dang, Q.-V., Nielsen, I., Bøgh, S., and Bocewicz, G., “Modelling and scheduling autonomous mobile robot for a real-world industrial application,” *IFAC Proceedings Volumes*, Vol. 46, No. 9, 2013, pp. 2098–2103.

[19] Atay, N. and Bayazit, B., “Mixed-integer linear programming solution to multi-robot task allocation problem,” 2006.

[20] Zhang, Y. and Parker, L. E., “Multi-robot task scheduling,” *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 2992–2998.

[21] Ganganath, N., Cheng, C.-T., and Chi, K. T., “A constraint-aware heuristic path planner for finding energy-efficient paths on uneven terrains,” *IEEE transactions on industrial informatics*, Vol. 11, No. 3, 2015, pp. 601–611.

[22] Ganganath, N., Cheng, C.-T., Fernando, T., Iu, H. H., and Chi, K. T., “Shortest path planning for energy-constrained mobile platforms navigating on uneven terrains,” *IEEE Transactions on Industrial Informatics*, Vol. 14, No. 9, 2018, pp. 4264–4272.

[23] Mei, Y., Lu, Y.-H., Hu, Y. C., and Lee, C. G., “Deployment of mobile robots with energy and timing constraints,” *IEEE Transactions on robotics*, Vol. 22, No. 3, 2006, pp. 507–522.

[24] Shnaps, I. and Rimon, E., “Online coverage of planar environments by a battery powered autonomous mobile robot,” *IEEE Transactions on Automation Science and Engineering*, Vol. 13, No. 2, 2016, pp. 425–436.

[25] Wallace, N. D., Kong, H., Hill, A. J., and Sukkarieh, S., “Motion Cost Characterisation of an Omnidirectional WMR on Uneven Terrains,” *IFAC-PapersOnLine*, Vol. 52, No. 22, 2019, pp. 31–36.

[26] Wallace, N. D., Kong, H., Hill, A. J., and Sukkarieh, S., “Energy Aware Mission Planning for WMRs on Uneven Terrains,” *IFAC-PapersOnLine*, Vol. 52, No. 30, 2019, pp. 149–154.

[27] Clarke, G. and Wright, J. W., “Scheduling of vehicles from a central depot to a number of delivery points,” *Operations research*, Vol. 12, No. 4, 1964, pp. 568–581.

[28] Toth, P. and Vigo, D., *The vehicle routing problem*, SIAM, 2002.

[29] Ralphs, T. K., Kopman, L., Pulleyblank, W. R., and Trotter, L. E., “On the capacitated vehicle routing problem,” *Mathematical programming*, Vol. 94, No. 2-3, 2003, pp. 343–359.

[30] Li, C.-L., Simchi-Levi, D., and Desrochers, M., "On the distance constrained vehicle routing problem," *Operations research*, Vol. 40, No. 4, 1992, pp. 790–799.

[31] Cordeau, J.-F. and Groupe d'études et de recherche en analyse des décisions (Montréal, Q., *The VRP with time windows*, Groupe d'études et de recherche en analyse des décisions Montréal, 2000.

[32] Toth, P. and Vigo, D., "VRP with backhauls," *The vehicle routing problem*, SIAM, 2002, pp. 195–224.

[33] Min, H., "The multiple vehicle routing problem with simultaneous delivery and pick-up points," *Transportation Research Part A: General*, Vol. 23, No. 5, 1989, pp. 377–386.

[34] Fisher, M. L., "Optimal solution of vehicle routing problems using minimum k-trees," *Operations research*, Vol. 42, No. 4, 1994, pp. 626–642.

[35] Potvin, J.-Y. and Bengio, S., "The vehicle routing problem with time windows part II: genetic search," *INFORMS journal on Computing*, Vol. 8, No. 2, 1996, pp. 165–172.

[36] Dorigo, M. and Di Caro, G., "Ant colony optimization: a new meta-heuristic," *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, Vol. 2, IEEE, 1999, pp. 1470–1477.

[37] Bell, J. E. and McMullen, P. R., "Ant colony optimization techniques for the vehicle routing problem," *Advanced engineering informatics*, Vol. 18, No. 1, 2004, pp. 41–48.

[38] Glover, F. and Laguna, M., "Tabu search," *Handbook of combinatorial optimization*, Springer, 1998, pp. 2093–2229.

[39] Gendreau, M., Hertz, A., and Laporte, G., "A tabu search heuristic for the vehicle routing problem," *Management science*, Vol. 40, No. 10, 1994, pp. 1276–1290.

[40] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by simulated annealing," *science*, Vol. 220, No. 4598, 1983, pp. 671–680.

[41] Chiang, W.-C. and Russell, R. A., "Simulated annealing metaheuristics for the vehicle routing problem with time windows," *Annals of Operations Research*, Vol. 63, No. 1, 1996, pp. 3–27.

[42] Prosser, P. and Shaw, P., "Study of greedy search with multiple improvement heuristics for vehicle routing problems," 1996.

[43] Voudouris, C. and Tsang, E., "Function optimization using guided local search," *University of Essex, Technical Report CSM-249, Colchester, UK*, 1995.

[44] Kilby, P., Prosser, P., and Shaw, P., "Guided local search for the vehicle routing problem with time windows," *Meta-heuristics*, Springer, 1999, pp. 473–486.

[45] Yu, B., Yang, Z.-Z., and Yao, B., "An improved ant colony optimization for vehicle routing problem," *European journal of operational research*, Vol. 196, No. 1, 2009, pp. 171–176.

[46] Beasley, J. E., "OR-Library: distributing test problems by electronic mail," *Journal of the operational research society*, Vol. 41, No. 11, 1990, pp. 1069–1072.

[47] Osman, I. H., "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of operations research*, Vol. 41, No. 4, 1993, pp. 421–451.

[48] Solomon, M. M., "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, Vol. 35, No. 2, 1987, pp. 254–265.

[49] Wang, X., Poikonen, S., and Golden, B., "The vehicle routing problem with drones: Several worst-case results," *Optimization Letters*, Vol. 11, No. 4, 2017, pp. 679–697.

[50] Bullo, F., Frazzoli, E., Pavone, M., Savla, K., and Smith, S. L., "Dynamic vehicle routing for robotic systems," *Proceedings of the IEEE*, Vol. 99, No. 9, 2011, pp. 1482–1504.

[51] van Omme, N., Perron, L., and Furnon, V., "or-tools user's manual," Tech. rep., Google, 2014.

[52] Black, P. E., *Dictionary of algorithms and data structures*.

[53] ClearpathRobotics, "Husky Unmanned Ground Vehicle Robot," 2012, https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/, last checked on 03.11.2020.

[54] Marder-Eppstein, E., "move_base," 2012, http://wiki.ros.org/move$_base, last checked on$03.11.2020.

[55] Hart, P. E., Nilsson, N. J., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100–107.

[56] OpenRobotics, "Darpa Subterranean Grand Challenge," 2019, https://subtchallenge.world/home/, last checked on 03.11.2020.