Discrete Path Planning Strategies for Coverage and Multi-robot Rendezvous

by

Neil Mathew

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2013

Abstract

This thesis addresses the problem of motion planning for autonomous robots, given a map and an estimate of the robot pose within it. The motion planning problem for a mobile robot can be defined as computing a trajectory in an environment from one pose to another while avoiding obstacles and optimizing some objective such as path length or travel time, subject to constraints like vehicle dynamics limitations. More complex planning problems such as multi-robot planning or complete coverage of an area can also be defined within a similar optimization structure. The computational complexity of path planning presents a considerable challenge for real-time execution with limited resources and various methods of simplifying the problem formulation by discretizing the solution space are grouped under the class of discrete planning methods. The approach suggests representing the environment as a roadmap graph and formulating shortest path problems to compute optimal robot trajectories on it. This thesis presents two main contributions under the framework of discrete planning.

The first contribution addresses complete coverage of an unknown environment by a single omnidirectional ground rover. The 2D occupancy grid map of the environment is first converted into a polygonal representation and decomposed into a set of convex sectors. Second, a coverage path is computed through the sectors using a hierarchical inter-sector and intra-sector optimization structure. It should be noted that both convex decomposition and optimal sector ordering are known NP-hard problems, which are solved using a greedy cut approximation algorithm and Travelling Salesman Problem (TSP) heuristics, respectively.

The second contribution presents multi-robot path-planning strategies for recharging autonomous robots performing a persistent task. The work considers the case of surveillance missions performed by a team of Unmanned Aerial Vehicles (UAVs). The goal is to plan minimum cost paths for a separate team of dedicated charging robots such that they rendezvous with and recharge all the UAVs as needed. To this end, planar UAV trajectories are discretized into sets of charging locations and a partitioned directed acyclic graph subject to timing constraints is defined over them. Solutions consist of paths through the graph for each of the charging robots. The rendezvous planning problem for a single recharge cycle is formulated as a Mixed Integer Linear Program (MILP), and an algorithmic approach, using a transformation to the TSP, is presented as a scalable heuristic alternative to the MILP. The solution is then extended to longer planning horizons using both a receding horizon and an optimal fixed horizon strategy.

Simulation results are presented for both contributions, which demonstrate solution quality and performance of the presented algorithms.

Acknowledgements

I would like to express my gratitude, appreciation and respect for my two exceptional supervisors, Dr. Steven Waslander, and Dr. Stephen Smith who gave me the opportunity to work on such a relevant, interesting and addicting research area. Their constant encouragement, drive, and mentorship has made me stronger as a person, researcher, academic, and student.

My life at the Waterloo Autonomous Vehicles Lab would not have been nearly as interesting without all my lab mates and colleagues, past and present, who made this experience memorable. I would like to thank my roommates and fellow researchers Arun and Carlos whose collective advice on robots, relationships and the universe has made me a wiser individual and without whom living in Waterloo wouldn't be half as enjoyable. Thanks to PJ, Yan and Rahul who were always ready to discuss the next billion dollar idea. A special thanks to Andy, who worked tirelessly with me through long hours of frustration and fleeting moments of bliss when algorithms worked as expected. Finally, thanks to James, Kevin, Sirui, Adeel, Mike S., Mike T., Nima, and Sid, who were always excited to discuss new research problems and wildly ambitious solutions to them. Their drive and passion for research inspired me in my own work and many an enlightened conversation was shared over the smell of coffee and solder in the lab.

I would also like to thank the University of Waterloo Robotics team, for all the work that went into the competitions we entered over the years, and the great memories that came along with them.

Finally, I would like to thank my mother, father and family whose unconditional support and confidence have been the pillars of all my endeavours so far and will continue to do so long after this thesis.

Dedication

This thesis is dedicated to my parents who have consistently been the optimal solutions to the complexities in my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Advances in autonomous mobile robotics research have demonstrated their potential for adoption in a wide range of applications that are too hazardous, inaccessible or repetitive for a human operator. Examples of such situations are search and rescue operations, surveillance, environmental monitoring and exploration, wherein a single or coordinated team of mobile robots must fulfil certain mission objectives that entail navigating through and interacting with the environment in varied capacities of sensing and actuation. In recent years, mobile robots have already had a significant impact in a number of these application domains.

Search and rescue robots have been used in many recent tragedies, such as the 2001 World Trade Centre collapse [40], Hurricane Katrina in 2005 [39], and the 2011 Tohoku tsunami and earthquake [19]. Currently, the majority of robots used for search and rescue missions are teleoperated, which are effective to an extent, but fall short in their ability to reduce the workload of the field operator in time-critical disaster scenarios [34]. Vehicle autonomy in such scenarios can vastly enhance rescue efforts of human operators. Similarly, robots capable of autonomous exploration and surveillance are of importance in tasks such as climate change monitoring, forest resource monitoring, wildlife population monitoring [14] and biogeological surveying [57], as improvements in vehicle autonomy allow for scientific data collection in settings such as the deep sea, volcanoes, and other locations which are unsuitable for human exploration. Autonomous exploration capabilities are also desirable for planetary exploration missions as teleoperation becomes cumbersome due to large transmission delays.

The autonomous robots performing these tasks are generally Unmanned Aerial Vehicles (UAVs), Unmanned Ground Vehicles (UGVs, Rovers) or Unmanned Underwater Vehicles

(UUVs), known as *agents* that conduct missions in known or unknown environments. As the capabilities of these robots grow it will also be possible to deploy heterogeneous teams of autonomous agents with different capabilities to collaborate on missions that are too complex for a single robot. Regardless of agent or environment, the goal of any autonomous operation is to specify a mission objective as a high-level task, such as surveillance or exploration and enable the robot to interpret it into a set of low-level functions of motion, sensing and actuation to accomplish the task. This thesis focusses on autonomous motion planning, which is an active area of research that addresses some of the most significant challenges in the current sphere of mobile robotics.

Autonomous motion planning involves compiling high-level mission tasks into low-level motion control to enable a robot to traverse through its environment. At the most basic level, motion planning requires a robot to perceive its environment or *configuration space*, identify drivable regions and obstacles, localize itself within the environment, and compute a path from one configuration to another while avoiding obstacles and possibly optimizing a certain objective function subject to constraints like vehicle dynamics. In the case of an unknown environment, perception and localization are accomplished by a robot using a suite of sensors such as laser scanners, stereo cameras, Inertial Measurement Units (IMU) and GPS to build a representational map of its surroundings and simultaneously localize within the generated map. Mapping and localization are out of the scope of this work and the thesis assumes a fully or partially known map, as well as a robot pose, to be provided as inputs to the motion planner. Let us first characterize path planning problems and overview some of the major concepts within this framework.

1.1 The Motion Planning Problem

The prototypical task for motion planning is to compute a feasible path for a robot from one configuration to another while avoiding obstacles [9]. For a mobile robot, in addition to avoiding obstacles the planner may be required to optimize certain objectives such as computing a path of the shortest length, shortest time or lowest energy consumption from one point to another while satisfying constraints determined by vehicle dynamics or the environment. Planning objectives can vary widely based on the robot, environment or application and each new objective presents a unique set of challenges that the field of motion planning seeks to solve. In fact, the motion planning problem is not limited to simply computing a trajectory from one configuration to another. Especially in mobile robotic missions such as exploration, surveillance or search and rescue, the overarching task is to compute a path to explore, surveil or consistently monitor an area. In such scenarios, the motion planner is faced with objectives like complete coverage over all areas in the environment or visiting certain areas periodically with specified frequencies. Further, if a task is to be performed collaboratively by a team of robots, a central or distributed motion planner must compute paths for multiple robots to interact with each other as well as the environment and the scope of the planning problem may expand to task assignment or scheduling along with motion planning to successfully manage the mission.

Motion planners are not only characterized based on planning objectives, but also on the path constraints imposed by the robot or the properties of the planning algorithm. A robot that is free to move instantaneously in any direction in its environment presents no motion constraints to the planner and is known as *omnidirectional*. On the other hand, a *kinodynamic* robot such as a differential drive rover or a quadrotor imposes kinematic or dynamic constraints that must be incorporated into the motion plan. Once the objectives and constraints of the optimization have been defined, an algorithm to compute the path must be developed. Based on the properties of this algorithm, a motion planner can be characterized as *offline*, if the plan is computed in advance, assuming a known model of the environment, or *online* if the plan is dynamically updated during execution. For example, a robot mapping an unknown environment must dynamically update its trajectory using a cycle of sensing, planning and execution to account for new map information. In some cases motion planners can be structured hierarchically with a high-level offline planner and a lower-level online component to improve robustness to uncertainties in the environment model.

Given the focus on real-time execution and dynamic replanning, an important consideration in path-planning is computational resources. An algorithm to find an optimal solution could be *constant*, *polynomial* or *exponential* in runtime or memory usage with a growth in problem complexity. Naturally, exponential time algorithms do not scale well with problem size and often, a polynomial approximation to the optimal solution is computed to speed up execution at the cost of optimality. Such algorithms are known as *approximation algorithms*, *heuristics* or *meta-heuristics* and will be investigated in more detail in Section 1.4. At the cost of optimality, planning complexity can also be reduced by simplifying the problem formulation. Since computing a path in the continuous domain involves solving an optimization over a very large solution space of robot trajectories, we can reduce the solution space by discretizing the problem formulation. This involves discretizing the map representation and enables the use of graph-based techniques to compute an optimal solution.

1.2 Discrete Planning

As the complexity of path planning grows, due to a growth in the size and dimensionality of the configuration space or number of constraints imposed on the optimization, computation in the continuous domain becomes impractical. The goal of discrete planning is to enable rapid path computation with a discrete representation of the planning problem using integer programming and graph-based algorithms to compute optimal solutions.

Discretization of the problem is achieved by abstracting the environment on a graph G where the vertices in V represent discrete locations in the environment and the edges in E represent traversable paths between then. The cost function c defines the cost of travel (i.e. distance, time, energy consumption) along each edge. The robot is constrained to travel along the roadmap graph G and the optimal path plan is now computed using shortest path optimizations on the graph.

Given an environment with obstacles, a discrete map representation can be generated using three main methods.

(i) Occupancy Grid: A gridded map of the environment where each square is marked as free space or in an obstacle. The graph represents traversable grids as vertices and the adjacency relationships between them as edges.

(ii) Visibility Graph: A polygonal representation of the environment where the robot is constrained to move along edges of a graph, the vertices of which are defined by polygonal vertices in the environment and the edges represented by collision free paths between them.

(iii) Sampling Based Roadmap: The vertices of the graph are generated by randomly or uniformly sampling the free space of the environment and the edges define a roadmap of collision free paths between them. Often, the sampling of points is done in a manner that satisfies complex vehicle dynamics constraints as in the methods of Probabilistic Roadmaps (PRM) [23] and Rapidly Exploring Random Trees (RRT) [30].

Given a roadmap graph G and a simple navigation from a start vertex to a goal vertex in the graph, the shortest path can be computed using well established graph search algorithms like Breadth First Search, A* or Dijkstra's algorithm. For more complex planning objectives such as visiting all or a subset vertices in the environment the shortest path problem must be formulated as an optimization to find the minimum cost ordering of vertices in the robots path.

Finding a shortest path through all vertices of a graph is a known NP-Hard problem called the Travelling Salesman Problem (TSP) (defined in Section 1.4), and most complex planning problems such as coverage or exploration that involve visiting a set of vertices in a graph can be modelled as variants of the TSP. Multi-robot path planning problems can be similarly represented as TSP variants such as the Multiple Travelling Salesman Problem (MTSP), Multiple Generalized Travelling Salesman Problem (MGTSP) or the Vehicle Routing Problem (VRP), and their applications are explored in Chapter 3. Since the TSP and its variants are known NP-Hard problems, computing the optimal MILP solutions to these problems is not scalable to large problem sizes and generally polynomial approximation algorithms or heuristics are developed to compute near-optimal solutions with minimal computational effort.

Another approach to solve TSP variants involves transforming the problem into a TSP instance through a graph transformation, such that the optimal solution to the TSP can be used to construct the optimal solution to the original problem. This stems from the property that any NP-Hard problem can be polynomially transformed into any other NP-Hard problem and is generally used to benefit from the large body of established exact, approximate and heuristic algorithms developed to solve the TSP. The work in this thesis explores transformations like Metric Closure and the Noon-Bean Transformation to solve GTSP instances and also presents a novel adaptation of the Noon-Bean method for MGTSP solutions to Multi-robot path planning problems.

In general, the discrete path planning framework involves discretizing the problem by building a roadmap graph of the environment, formulating the shortest path problem as an optimization on the graph and solving it using a number of graph-search or optimal routing problems. Section 1.3 motivates the two main discrete planning problems studied in this thesis and presents each of their contributions within this framework that will be elaborated in Chapters 2 and 3.

1.3 Contributions

This thesis presents two main contributions under the framework of discrete path planning that concern two different classes of planning problems. The first contribution addresses complete coverage of an unknown environment by a single omnidirectional ground rover and the second presents a multi-robot path planning strategy for optimally scheduled rendezvous between UAVs conducting persistent surveillance and UGVs functioning as mobile charging stations.

In both problems, a discrete environment representation is used to generate a graph-based roadmap on which a path or set of paths must be computed. The problem is formulated as a MILP optimization and heuristic algorithms are presented to compute near-optimal solutions with less computational effort than an optimal solver. Algorithms are validated in runtime and solution quality against optimal solutions and existing approaches.

1.3.1 Coverage Path Planning

The objective of coverage is to generate a path for a robot to pass a given sensor footprint over all areas of a search environment with a guarantee that no areas will be left unvisited. The main cost metric of the optimization is the total path distance required to completely cover an area with the sensor footprint that sweeps along the path. As the terrain is not known ahead of time, the solution should not require full map knowledge and should allow replanning as the location is explored.

The approach to coverage planning in this work is to use a convex decomposition to reduce the environment to a set of convex regions that can be searched using a simple sweep pattern as illustrated in Figure 1.1. Further, the notions of sensor based robust coverage and dynamic replanning are investigated and a heuristic algorithm to compute coverage solutions that outperform existing approaches is presented. The main contributions of this work are the following.

(i) The 2D drivability map is approximated using a polygonal representation of free space that is then decomposed into a set of convex sectors.

(ii) The coverage problem is defined as finding the shortest length path that covers every searchable sector of the environment.

(iii) The polygonal environment is further abstracted on visibility graphs representing the searchable sectors, their vertices and the adjacency relationships between them. The robot is constrained to move along edges of the graph between sector vertices.

(iv) The problem is approached in two-step hierarchical manner. First, the optimal order of sectors to visit is computed using a Hamiltonian Path solution on the sector graph. Next, the path is refined by finding the best entry-exit points and sweep pattern directions in each sector.

(v) The developed path planning framework is extended to online planning scenarios where a partially known map is consistently updated during plan execution.

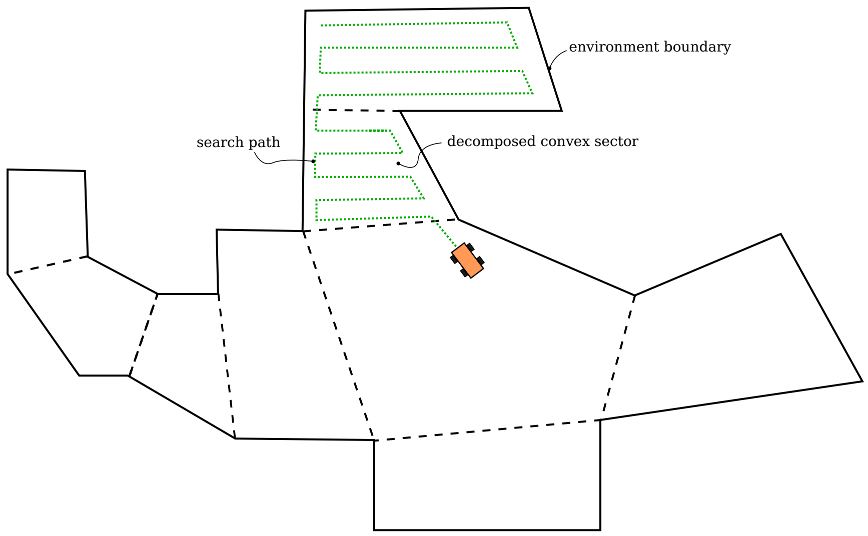


Figure 1.1: Illustration of the coverage planning problem

(vi) Results that demonstrate the feasibility of real-time execution and the quality of the resulting path are presented and compared to existing coverage approaches.

1.3.2 Multi-robot Rendezvous for Autonomous Recharging

Coordinated teams of autonomous robots are often proposed as a means to continually monitor changing environments in applications such as air quality sampling [10], forest fire or oil spill monitoring [5, 58] and border security [26]. These surveillance tasks generally require the robots to continuously traverse the environment in trajectories designed to optimize certain performance criteria such as quality or frequency of sensor measurements taken in the region [11, 51, 47].

The challenge with using autonomous robots in persistent tasks is that mission durations generally exceed the run time of the robots, and in order to maintain continuous operation they need to be periodically recharged or refuelled. In accordance with current persistent surveillance literature, we consider the case of a team of UAVs monitoring an environment in planar trajectories as illustrated in Figure 1.2. The proposal is to introduce a separate team of dedicated charging robots that can autonomously dock with the UAVs and recharge them periodically over the duration of the mission. The approach to the problem is to position multi-robot persistent recharging in the space of graph-based optimal path planning problems formulated using mixed integer linear programs. Based

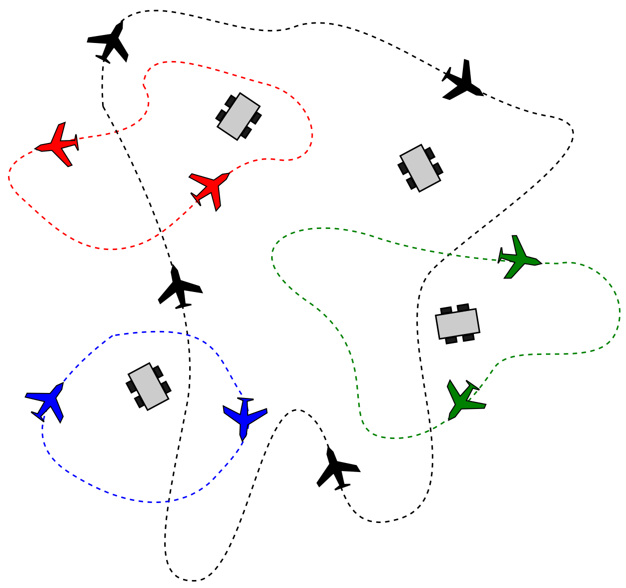


Figure 1.2: A sample scenario with coordinated UAVs performing a persistent surveillance task in planar trajectories. We introduce a team of four ground robots capable of docking with and recharging the UAVs.

on certain enabling assumptions, the problem is formulated as finding optimal paths on the ground for the charging robots to meet the working robots along their trajectories and recharge them as needed. The main contributions are the following.

(i) The idea of multi-robot recharging in dynamic persistent tasks using dedicated teams of mobile charging stations is formally introduced.

(ii) A graph-based abstraction of the problem is presented and defined as a one-in-a-set path problem on a partitioned directed acyclic graph (DAG). The problem is proved to be NP-hard even on a DAG.

(iii) The problem is formulated as a Mixed Integer Linear Program (MILP) and special properties that distinguish it from Travelling Salesman Problem (TSP) and Generalized Travelling Salesman Problem (GTSP) formulations are investigated.

(iv) An algorithmic solution is proposed using novel modifications to existing methods developed for solving the GTSP.

(v) Based on the developed path planning framework the recharging problem is extended to longer planning horizons using both, receding horizon and fixed horizon strategies.

1.4 Primer on Relevant Graph Theory

The following is a review of relevant graph-theory and fundamental optimal path computation problems that comprise the problem formulations and algorithms employed throughout this work.

A *graph* G is represented by (V, E, c), where V is the set of *vertices*, E is the set of *edges* and $c : E \to \mathbb{R}$ is a function that assigns a cost to each edge in E. The number of vertices in G is given by the cardinality of the set V and is denoted $N = |V|$. In an undirected graph, each edge $e \in E$ is a set of vertices $\{v_i, v_j\}$. In a directed graph each edge is an ordered pair of vertices (v_i, v_j) and is assigned a direction from v_i to v_j.

A *partitioned graph* is a graph G with a partition of its vertex set into R exhaustive and mutually exclusive sets $(V_1, \ldots, V_R)$ such that (i) $V_i \subseteq V$ for all i, (ii) $\cup_i V_i = V$, and (iii) $V_i \cap V_j = \emptyset$ for all $i \neq j$.

A *path* in a graph G, is a subgraph denoted by $P = (\{v_1, \ldots, v_{k+1}\}, \{e_1, \ldots, e_k\})$ such that $v_i \neq v_j$ for all $i \neq j$, and $e_i = (v_i, v_{i+1})$ for each $i \in \{1, \ldots, k\}$. The set V_P represents the set of vertices in P and by definition $V_P \subseteq V$. Similarly a *tour* or *cycle* T is a closed path in the graph such that $v_1 = v_{k+1}$. Finally, a *directed acyclic graph* (DAG) is a directed graph in which no subset of edges forms a directed cycle. With this we can define the following key problems.

Definition 1.4.1 (Hamiltonian Path/Tour)**.** A Hamiltonian Path in a graph G is a path P that visits every vertex in G exactly once. Similarly, a Hamiltonian tour T is a closed Hamiltonian Path.

Problem 1.4.2 (The Hamiltonian Path Problem)**.** Given a complete, undirected graph G, does G contain a Hamiltonian path?

Problem 1.4.3 (Travelling Salesman Problem (TSP))**.** Given a complete graph G, find a Hamiltonian tour T such that total cost of $\sum_{e \in E_T} c(e)$ is minimized, where E_T is the set of edges in T. A symmetric TSP is computed on an undirected graph. Similarly, an asymmetric TSP is obtained on a directed graph.

The TSP can be formulated as a MILP and solved using a variety of exact, approximate and heuristic solvers. In this work we use the IBM CPLEX Optimization Studio to compute exact solutions. Since the TSP is an NP-hard problem, it is intractable to compute exact solutions for large problem instances. In such cases, heuristic algorithms may be used to obtain near-optimal solutions with significant reductions in run time. One of the best known algorithms is the Lin-Kernighan heuristic [31] implemented as the Concorde LinKern TSP solver and an adaptation proposed by Helsgaun [20] implemented as the Lin-Kernighan-Helsgaun (LKH) TSP solver. While these heuristics do not have proven guarantees on sub-optimality, they have been empirically shown to often produce solutions within 2% of the optimal [1].

There are a number of TSP variants that extend the problem by imposing additional constraints on the graph optimization. The variant employed in this work is the Generalized Travelling Salesman Problem (GTSP). Let us first define a *One-in-a-set Path*.

Definition 1.4.4 (One-in-a-set Path/Tour)**.** A One-in-a-set Path in a partitioned graph G with a vertex partition $(V_1, \ldots, V_R)$, is a path P that visits a single vertex in every vertex set $V_i \subset G$ exactly once. Similarly, a One-in-a-set tour T is a closed One-in-a-set path.

Problem 1.4.5 (Generalized Travelling Salesman Problem (GTSP))**.** Given a partitioned complete graph G, find a One-in-a-set tour T such that the total cost $\sum_{e \in E_T} c(e)$ of T is minimized, where E_T is the set of edges in T.

The TSP can be seen as a special case of the GTSP where all vertex sets have a cardinality of one [43]. There are a number of approaches to solve GTSP instances developed in literature. Exact algorithms using Lagrangian relaxation and Branch and Cut methods have been proposed by Noon and Bean [42] and Fischetti et al. [48]. Improvement heuristics [22], and meta-heuristics like genetic algorithms [52] and ant colony optimizations [60] have also been proposed.

Another approach, proposed by Noon and Bean [43], involves transforming (or reducing) a GTSP instance into an equivalent TSP instance through a graph transformation, such that the optimal solution to the TSP can be used to construct the optimal solution to the original GTSP. The benefit of such a transformation is that it allows the vast body of existing TSP approaches to be applied to the transformed problems. While such an approach may not necessarily outperform a specialized GTSP algorithm, it provides a platform to verify optimality and compare results of complex routing problems on the foundation of well-studied TSP approaches.

The GTSP can be extended to compute multiple One-in-a-set tours originating at multiple depots as defined in Problem 1.4.6.

Problem 1.4.6 (Multiple Generalized Travelling Salesman Problem (MGTSP))**.** Consider a complete partitioned graph G with vertex partition $(V_0, V_1, \ldots, V_R)$, where V_0 consists of M start-depots. Find a collection of M paths, one for each start depot, which collectively visit each vertex set $(V_1, \ldots, V_R)$ exactly once, with minimum total cost.

In an MGTSP, since all start-depots may not be used in the solution, the decision of the number of routes chosen for the optimal solution is implicit in the formulation. The objective of the optimization can be chosen to minimize the total sum of path costs *(min-sum objective)*, or to minimize the maximum individual path cost *(min-max objective)*.

Chapter 2

Coverage Path Planning

The notion of coverage investigated in this work involves using two sensor modules, namely mapping and vision based object detection to navigate and search an environment for samples. The mapping module generates a 2D drivability map using a laser scanner with a large sensor footprint of $75m$ radius. The path planner must generate a path to search the environment as identified by the map by passing a smaller camera sensor footprint of approximately $3m$ radius over every point in the search space. The planning process must be consistently updated as new map and visual information becomes available.

The path planning problem is solved in two main stages. The first stage consists of transforming the map into a polygonal representation and decomposing it into a set of smaller polygons (sectors). The second stage involves generating an optimal coverage path through all the internal sectors. The two stages are each formulated as separate NP-hard problems and algorithms are presented to achieve improvements over related work in coverage planning literature.

2.1 Related Work

Solutions to the *coverage planning problem* have been presented in a variety of application domains, including search and rescue [38], domestic vacuum cleaning robots [27], and automated milling tool path generation [2]. A common solution from the robotics domain is the Frontier Exploration method [59], which assumes no knowledge of the map at initialization. Instead, a map is incrementally built via sensor feedback while commanding the robot to constantly drive toward the nearest boundary of unexplored space, defined as a frontier. The primary shortcoming of such an approach is the high probability of erratic overlapping paths that lead to excessive redundant sensor sweeps.

If the problem is relaxed by assuming a known map, a more efficient approach may be taken. For instance, Arkin [2] proposes a methodical "zig-zag" sweeping pattern along a single major axis, applied across the entire map. The problem is formulated as a graph in which a vertex is created for each sweep collision with an obstacle or map boundary. A collision-free Eulerian tour of the graph can then be found. The heuristic solution presented results in a total path length that is bounded by 2.5 times the optimal. A major drawback of this approach is the need to circumnavigate obstacle and environment boundaries more than once.

The inefficiencies presented by a constant sweep direction can be mitigated using a method proposed by [38]. The authors propose a decomposition of the map into smaller sectors as it allows for variations in the sweep direction within each sector to better suit map geometry and minimize overall path length [38]. The map is converted into a polygonal representation, and partitioning is performed at major vertices. The decomposed sectors are then represented as a graph and the shortest coverage path through all sectors is computed using a TSP solution. The major assumption, however, is that no obstacles exist in the search space. The assumption simplifies the decomposition problem to one which can be solved using existing polynomial approximation algorithms [18, 24].

In practice, obstacles and non-convex boundaries in the polygonal representation of an environment can block lines of sight from a sensor. Therefore, any sectors derived from a decomposition must also be convex to guarantee sensor coverage. Given this additional constraint, the coverage path planning problem can be transformed into the Watchman Problem [44], which finds the shortest path for a guard to fully monitor an area containing obstacles.

The Watchman problem is an NP-hard problem with its computational complexity dependent upon the number of regions present. One method of mitigating run-time in a real-time system is to reduce the number of regions required to represent the environment, the minimum of which is called the Optimal Convex Decomposition (OCD). Unfortunately, the OCD of a polygon with obstacles (or *"holes"*) is itself an NP-hard problem [25].

To date, there have been various methods of sub-optimal decompositions. The Trapezoidal cut method [6] slices the region in a constant direction from each obstacle and boundary vertex to form convex trapezoidal partitions. However, the large aspect ratios and often small areas make sweeping within such partitions inefficient. Another choice is Delaunay decomposition [8], which forms small triangles around every vertex, optionally followed by aggregation algorithms. However, many more regions than are necessary are

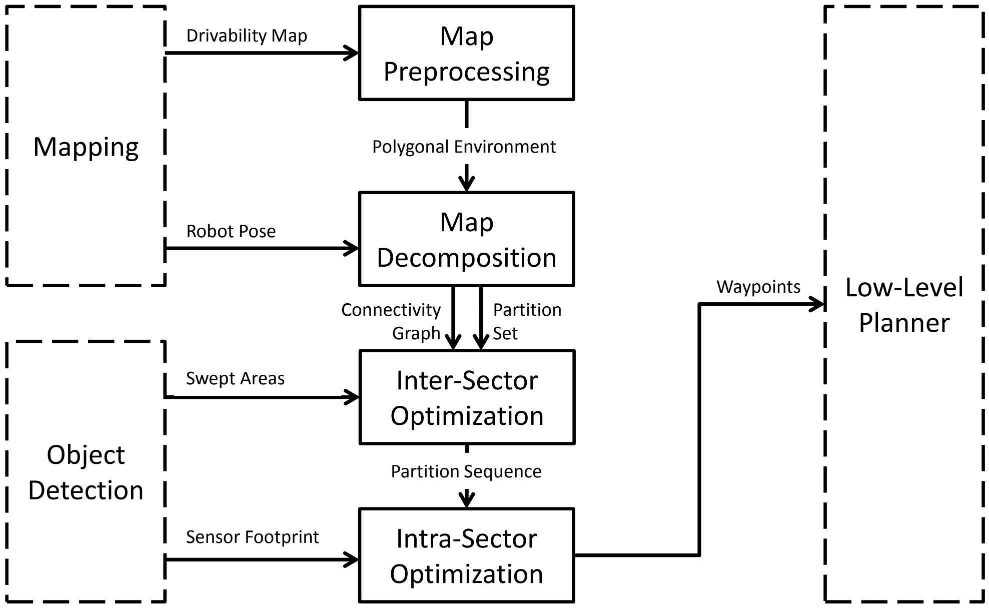


Figure 2.1: Coverage planning algorithm flowchart.

generated with Delaunay decomposition, which again leads to impractical search geometries.

More recently, a greedy decomposition approach [56] displays functionality similar to OCD but extends the capability to polygons with holes. The algorithm presents an approximation scheme which has been demonstrated to consistently outperform the Trapezoidal or Delaunay methods.

2.2 Coverage Planning Approach

The coverage planning approach is depicted in Figure 2.1, and consists of four main steps. The first step is *map pre-processing*, which collects inputs from the two perception modules of the robot. A 2D drivability map and the current robot pose are obtained from the mapping module while the sensor footprint is obtained from the object detection module. The map is then pre-processed into a polygonal representation using an edge-tracing method to define boundaries. In the *map decomposition* step, the greedy cut algorithm is applied to decompose the polygonal map into a set of convex sectors to be explored. Next, the *inter-sector path optimization* step represents the adjacency relationships between the convex sectors on a graph-based roadmap. The problem of finding an optimal order of sectors to visit is an NP-hard problem that is transformed to a TSP and solved using the well-known Lin-Kernighan heuristic algorithm [31]. Finally, the *intra-sector path optimization* refines the complete coverage path, given the sequence of sectors to traverse from the previous step. The sweep pattern orientation as well as the points of entry and exit points in each sector are computed to minimize the path length.

An important feature of coverage planning algorithms is the need to guarantee complete coverage in a changing map. The drivability map is frequently updated as new information becomes available to the robot and it is not sufficient to simply re-generate a path at every map update. The re-planning process retains a memory of previously visited regions to avoid re-traversals and the update frequency is limited to one sector at a time. The output of the path planning process is a set of waypoints that can be executed by low-level obstacle avoidance planners such as Wavefront [3], trajectory roll-out, or other graph based planning techniques. The choice of the low-level planner is correlated with the dynamics of the robot. However, since the particular application requires the robot to operate at low speeds, both differential steer and swerve drive robots are able to execute straight line paths designed using a simple A* graph based path planner.

2.3 Map Preprocessing

In map preprocessing, obstacle boundaries are dilated and converted into vector representations using Suzuki's border-tracing algorithm [54]. The boundary contours are then simplified, to reduce the number of vertices at the cost of shape fidelity, since the eventual polygon decomposition stage is sensitive to the number of vertices in the system. The result is a set of polygons, that represent the obstacle boundaries.

We then apply the Ramer Douglas Peucker (RDP) algorithm for boundary vertex reduction. In a review by Nguyen [41], RDP was found to be over twice as fast as the next best candidate in a comparison with incremental, line regression, RANSAC, Hough Transform, and Expectation-Maximization techniques. The RDP algorithm works by removing points less than a configurable orthogonal distance away from a line joining a start and end vertex. The remaining vertices are then iteratively used to form new lines toward the end vertex until all points have been inspected. The end result is control over the amount of detail along an obstacle border while maintaining the overall obstacle shape. It is assumed that the sensor footprint of the robot will cover any small areas no longer present due to smoothing.

2.4 Map Decomposition

Given a polygonal representation of the search space, the map decomposition problem can be stated as follows. Let an environment $\mathcal{E} \subset \mathbb{R}^2$ be bounded by a simple polygon, S, consisting of a set of vertices V and a set of directed edges E.

Each edge e, represented by the ordered pair, $(v_i, v_j) \in E$, is defined as a line segment between two vertices v_i and v_j where $i \neq j$. The vertices and edges in polygon S are listed in a clockwise (CW) fashion. The furthest Euclidean distance between any two vertices in S is defined as ξ. Obstacles (holes) are denoted by simple polygons that are internal to S. The set of all holes in the system is defined as H and each hole, h_k is indexed by $k \in \{1, \ldots, n_H\}$, where n_H is the cardinality of the set H. Each obstacle is also defined by a set of directed edges, but stated in a counter-clockwise (CCW) manner. The interior of the boundary excluding the obstacles represents the search space to be covered.

Let the interior angle between two adjacent edges of a polygon be defined as ψ_i, occurring at the vertex of intersection, v_i. The set of non-convex vertices (NCV) in the environment is defined by

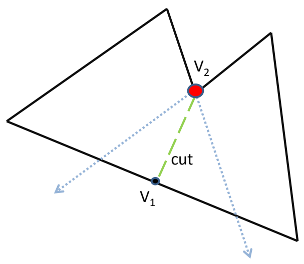
$$V_{\text{NCV}} = \{v_i \subset V : (v_i \in S \ \wedge \ \psi_i > \pi) \ \vee \ (v_i \in H \ \wedge \ \psi_i < \pi)\} \tag{2.1}$$

Greedy cut decomposition algorithm

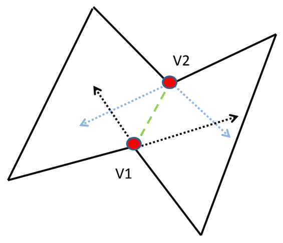
The greedy cut decomposition is an approximation algorithm to the optimal convex decomposition of a polygon [56]. A sequential cut-based approach is taken which incrementally segments the map until all its constituents are convex. The main premise is that convex partitions can be formed by adding cuts emanating from each NCV, thereby dividing a non-convex region into convex sectors [7].

A cut from an NCV to an existing edge or another cut is referred to as a *single cut*, defined as an added edge to the system. It has been demonstrated that at most, a cut may eliminate two NCVs, one at each endpoint [7]. Such two-NCV cuts are referred to as *matching cuts*. One condition is that all cuts must exist within the NCVs' cones of bisection, defined as an area bounded by the projection of the two edges emanating from an NCV.

The algorithm greedily searches all NCVs in the system to make matching cuts first, followed by single cuts for unmatched NCVs. Since greedy cuts are made until all NCVs are eliminated from the system, there is no pre-planning or merging of partitions during the cut process. Instead, the combined set of all cut and boundary edges is used to trace



(a) Single Cut



(b) Matching Cut

Figure 2.2: Two types of cuts possible in a non-convex polygon. Here, red dots represent an NCV. Green dashed lines represent an added cut edge. Dotted lines represent each vertex's cone of bisection.

the resultant partition boundaries. Starting with an arbitrary edge, a loop may be initiated and extended with an element within the edge set which forms the tightest convex turn with the previous edge. Edges are incrementally removed from the set once connected, and retracing a path back to the original start point signifies the completion of one sector. The process is repeated until no edges remain in the set. After cuts are applied to outstanding NCVs, the resulting edges define a set of n_S convex polygonal sectors.

The algorithm operates in $O(n \log n)$ time for the total number of NCVs in the matching cut and polygon identification stages, while a remaining single-cut stage runs in $O(n|E|)$, with the number of remaining unmatched NCVs. Alternative NCV-elimination methods discovered in literature are either slower (Greene's optimal Dynamic Programming algorithm which operates in $O(n^4)$ time [18]), or results in too many polygons (the swept-line approximation method at $O(n \log n)$ time [18].)

The purpose of convex decomposition is to generate sectors of the map that locally guarantee line of sight within each sector area. While this property is necessary to generate feasible sweep patterns in any direction, Section 2.8 shows why this is important for online planning to limit the replanning frequency to one sector at a time with the assumption that the convex sector closest to the robot is fully mapped prior to executing coverage. In order to make this assumption the size of each sector must be limited to the confidence range of the mapping sensor. Figure 2.3 illustrates one method to accomplish this by constraining the greedycut algorithm to a gridded environment using polygon clipping operations.

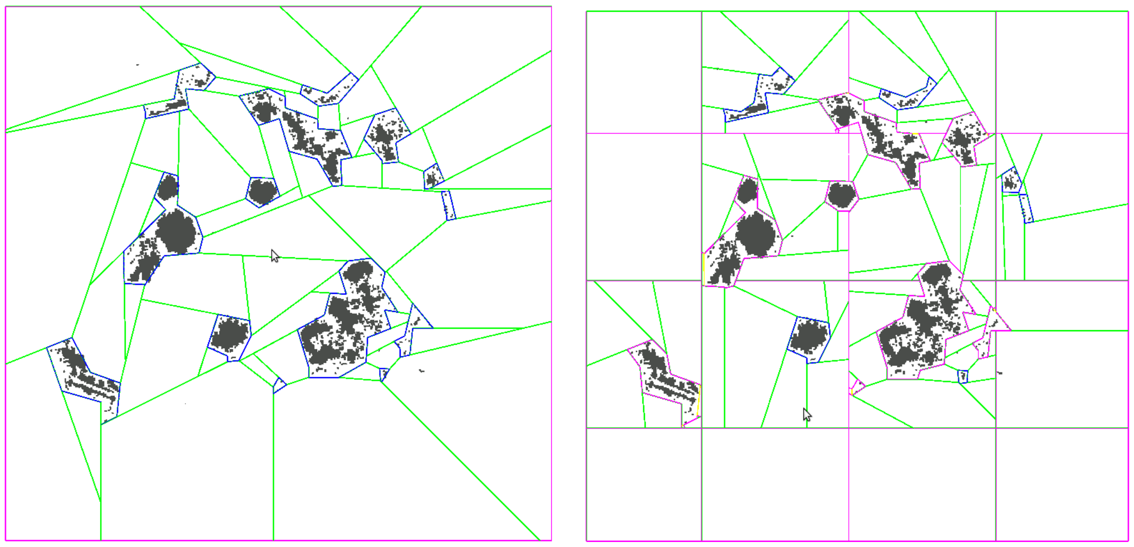

(a) Simple Greedycut decomposition (b) Gridded Greedycut decomposition

Figure 2.3: Comparison of gridded and simple Greedycut decomposition of a test environment.

2.5 Path Generation

Given the set of sectors to cover, the task of the path planner is to compute an optimal path P for the robot through the sectors such that sensor coverage is guaranteed over the entire searchable area. Each sector is entered and exited at points along its boundary and is searched using a *"zig-zag"* sweep coverage pattern. The optimality of the path depends on the order in which sectors are visited, the structure of the coverage pattern within each region, and the entry and exit points used to link successive sectors. The solution approach involves a discrete problem formulation for which graph-based path planning algorithms can be applied.

2.5.1 Problem Formulation

After the map decomposition step, the boundary polygon, S, is decomposed into a set of n_S convex polygonal sectors. Each sector, indexed by s_i where $i \in \{1, \ldots, n_S\}$, represents a subset of the total traversable area to be covered by the robot such that $\cup_{i=1}^{n_S} \alpha(s_i) = \alpha(\mathcal{E})$, where $\alpha : \mathbb{R}^2 \to \mathbb{R}$ is a function that represents the area of a polygon.

Let each sector s_i where $i \in \{1, \ldots, n_S\}$, be represented by a set of vertices V_{s_i}. The complete set of vertices in the map, $V = \cup_{i=1}^{n_S} V_{s_i}$ is the union of n_S collectively exhaustive but non-mutually exclusive sets. It is clear that connected sectors share vertices. If sectors s_i and s_j are not connected, $V_{s_i} \cap V_{s_j} = \emptyset$. If they are connected, $V_{s_i} \cap V_{s_j} \neq \emptyset$ for $i \neq j$. The discrete formulation of the path generation problem enables an efficient graph-based approach to compute a search path. The approximation scheme employed to compute the search path is dependant on a number of simplifications to the problem formulation that are detailed as follows.

Computing a path that passes through an entry and exit vertex in each sector vertex set V_{s_i} in the polygonal map presents a problem with a high level of computational complexity. Similar problems, like the Generalized Vehicle Routing Problem (GVRP) [17],which involves finding a path passing through one vertex in each of many vertex sets, have been previously studied in literature. However, for the coverage problem, defining sector sweep coverage costs between every pair of intra-sector and inter-sector vertices is computationally expensive and unsupportable, given the temporal constraints of real time operation. The problem can be mitigated by splitting the process into two smaller sub-problems, namely an inter-sector path optimization and an intra-sector path optimization.

Inter-sector path costs are minimized by finding a path that travels only between adjacent sectors, through shared vertices between them. In the case where two consecutive sectors are not adjacent, the path travels along collision free lines between sector vertices. Restricting inter-sector paths in this manner is deemed suitable as it is equivalent to applying visibility graphs in polygonal maps, which have been shown to determine shortest paths between any two vertices on the map [12]. The polygonal vertices and connecting edges in the map decomposition generate a similar visibility graph over the environment.

Intra-sector sweep path costs can be minimized by optimally structuring the sweep path within a sector, given a sector's entry and exit points. By applying pre-determined search patterns, sweep paths can be easily projected over large areas, and costs calculated rapidly for real-time path optimization.

The goal of the path generation problem formulation is to design a path such that for all $i \in \{1, \ldots, n_S\}$, the robot enters each polygonal sector s_i at a vertex $v_a \in V_{s_i}$, covers the sector using a sweep coverage pattern and then leaves at a vertex $v_b \in V_{s_i}$ to continue its path.

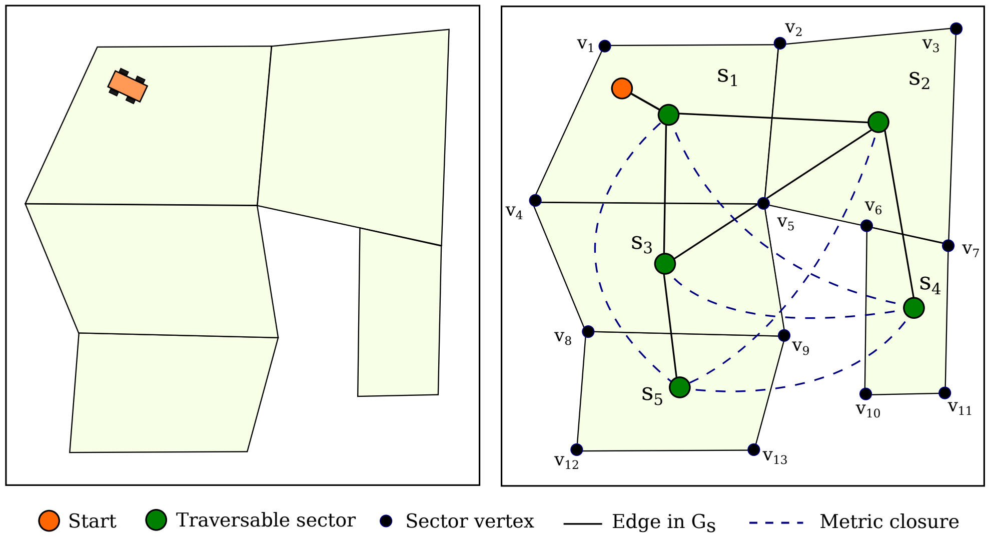


Figure 2.4: Graph based formulation to optimize the inter-sector path generation.

2.6 Inter-sector Path Optimization

An unweighted, undirected *sector graph* $G_s = (V_s, E_s)$ is defined, where V_s is the set of vertices representing each of the n_S traversable sectors in $\mathcal{E}$. Each vertex $v_i \in V_s, i \in \{1, \ldots, n_S\}$ represents a sector s_i. An edge e is added, between vertices v_i and v_j, representing sectors s_i and s_j if, for $i \neq j$, $V_i \cap V_j \neq \emptyset$ (i.e. the sectors have at least one common vertex).

The sector graph G_s is constructed under the assumption that the robot, when restricted to travel between adjacent sectors, will minimize the inter-sector travel costs. This is a reasonable simplification that allows a significant amount of edge pruning, given the assumption that every sector in the map is connected via a common vertex to at least one other sector. Figure 2.4 shows the construction of G_s over the polygonal sectors in the map decomposition. Assuming the robot only travels through shared vertices, the optimization over G_s can be described as the *Sector order decision problem.*

Problem 2.6.1 (Sector Visit Order Problem)**.** Consider a graph G_s in which V_s represents each of the n_S traversable sectors in the environment. Find a path P in G_s starting at v_0 that visits every vertex in V_s atleast once such that the total path cost $\sum_{e \in E_P} c(e)$ is minimized.

The optimization problem, as described, belongs to a known class of NP-hard problems, that can be solved using a transformation to the Hamiltonian Path Problem. Since the graph G_s is connected, but not complete and thereby not guaranteed to be Hamiltonian, a reduction to the Hamiltonian path can be obtained using *metric closure* whereby for each pair of vertices v_i and v_j that are not connected in G_s, a temporary edge that represents the best shortest path between the vertices is computed using a shortest path algorithm such as A* or Djikstras [12]. The problem can then be solved using a variety of commercially available exact and heuristic traveling salesman problem (TSP) solvers [31].

2.6.1 Metric Closure Method

The sector graph G_s is transformed, using a metric closure as described in this section, into a graph on which the solution to the Hamiltonian path will provide the solution to the sector order visit problem.

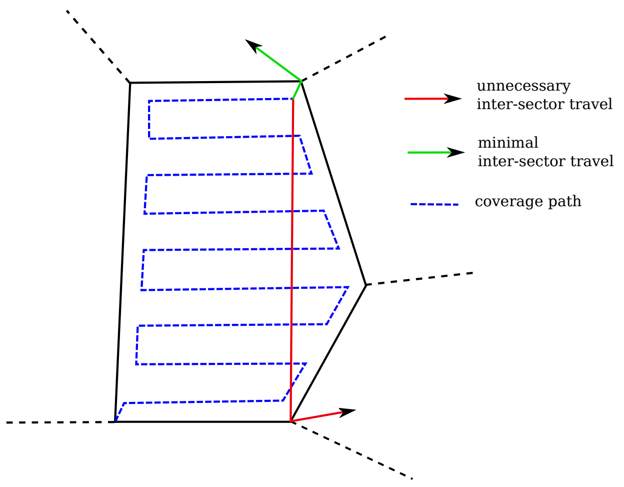
First, the *vertex graph* $G_v = (V_v, E_v, c_v)$ is defined, where V_v is the total set of all vertices in the polygonal decomposition of $\mathcal{E}$, E_v is the set of edges where an edge (v_i, v_j) exists if there is a direct, collision free path between the two vertices. The cost function c_v on edge (v_i, v_j) is the Euclidean distance between the vertices. This graph forms the basis of inter-sector route planning in the robot. The robot is constrained to move only along the edges defined on graph G_v during all inter-sector travel.

Next, the *augmented graph* $G_{sv} = (V_{sv}, E_{sv}, c_{sv})$ is defined, where the set of vertices V_{sv} inherits directly from the vertex set V_s of the sector graph G_s. All edges in E_s are copied into E_{sv}. For every pair of sectors s_i and s_j, where $i \neq j$ and $e \notin E_s$, a supplementary edge is added and the edge costs are defined using the function $c_{sv}(e)$ as follows.

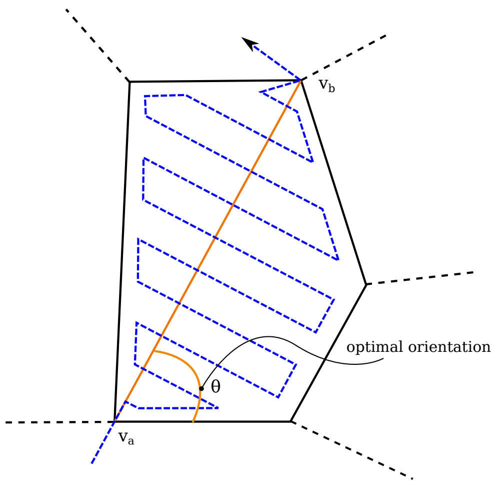
For each edge e,

$$c(e) = \begin{cases} 0, & \text{if } e \in E_s, \\ \tau(s_i, s_j), & \text{if } e \notin E_s. \end{cases} \tag{2.2}$$

where $\tau(s_i, s_j)$, defined as the shortest route cost between sectors s_i and s_j is calculated on graph G_v, by finding $v_l \in s_i$ and $v_k \in s_j$, such that they minimize the smallest Euclidean distance between the two sectors. Now the cost $\tau(s_i, s_j)$ between v_l and v_k is computed on G_v using an A* search and assigned to $c(e)$ on G_{sv}. The graph G_{sv} thus defined is a complete graph that can be used to compute the Hamiltonian path through the sectors using the Concorde LinKern solver which is a freely available implementation of the Lin-Kernighan Heuristic [31]. The resulting path provides an order of sectors for the robot to visit, given the defined cost metric.



(a) Intra sector route optimization problem



(b) Intra sector route optimization solution

Figure 2.5: Intra-sector sweep coverage optimization.

2.7 Intra-sector Route Optimization

The optimal sector visit order was computed as a Hamiltonian path with the assumption that in transitioning between adjacent sectors, the robot will pass through a common vertex. When faced with non-adjacent sectors, it will follow the shortest path through their two closest vertices. Note that since adjacent polygons can share more than a single vertex, the chosen entry and exit points in each sector will significantly affect the final cost. Ideally, the path must seamlessly transition between sectors with minimal unnecessary travel.

In optimizing the coverage path within a sector, a sweep pattern is generated within a convex polygon, as shown in Figure 2.5. For a pair of entry and exit points denoted by v_l and v_k, a sweep pattern, with a pitch is equal to the diameter of the sensor footprint of the robot, is generated. The orientation of the sweep is chosen to minimize the total sweep path length within the sector.

2.7.1 Sweep Graph Construction

From Figure 2.4, it is observed that the shortest route through all sectors travels through entry and exit vertices which are shared between adjacent sectors in the Hamiltonian path.

By formulating the intra-sector cost metric between the vertices, the problem space for the intra-sector path optimization can be defined.

A *sweep* graph $G_a = (V_a, E_a, c_a)$ is defined, where V_a is the set of vertices, E_a is a set of directed edges between them, and c_a is the cost function for the edges in E_a. The attributes of the graph are detailed as follows:

Vertices: Define $n_S - 1$ disjoint vertex sets, $V_{a_1}, \ldots, V_{a_{n_S-1}}$. The set V_{a_i} for $i \in \{1, \ldots, n_S - 1\}$ is given by the set of all shared vertices between sector s_i and s_{i+1} that are indexed according to the sector visit order. Shared vertices between every pair of sectors are defined independently and if the same vertex is shared between more than two sectors, it is repeatedly defined in the formulation. In the case where consecutive sectors s_i and s_j do not share any vertices, a dummy vertex is added to the empty set V_{a_i}. Any path through the graph is implicitly forced to pass through this dummy vertex. Once the path optimization is complete, the dummy vertex is replaced by a set of way points comprising the shortest path between the two disconnected sectors.

Another vertex set V_{a_0} is defined to contain the start position v_0 of the robot in the map. The total vertex set in graph G_a is then $V_a = V_{a_0} \cup V_{a_1} \cup \ldots \cup V_{a_{n_S-1}}$.

Edges: We add a directed edge e between vertex v_l and v_k where $v_l \in V_{a_i}$ and $v_k \in V_{a_j}$ for some $i, j \in 1, \ldots, n_S - 1$, to E_a if $j = i + 1$. The edge e represents a transition from the entering vertex v_l to the exiting vertex v_k in the sector s_i. v_k is subsequently the entering vertex in sector s_j.

Edge Costs: Each edge $e \in E_a$ between vertices v_l and v_k, is associated with a cost $c_a(e)$ that is computed based on the distance traversed by the complete intra-sector coverage path within sector s_i. The cost includes the sweeping pattern as well as travel to and from the entry and exit points. A zero cost is assigned to any edge for which either v_l or v_k is a dummy vertex.

As a simple illustrative example, Figure 2.6 shows the graph G_a constructed for the problem defined in Figure 2.4. The constructed graph is a partitioned directed acyclic graph, with a partition of vertices given by $(V_{a_0}, V_{a_1}, \ldots, V_{a_{n_S-1}})$ representing the shared inter-sector vertices. The graph is multipartite in that there are no edges between vertices in the same vertex partition. Since edges are only defined between vertices of consecutive sectors, this graph construction lends itself to a shortest path search to find the optimal traversal route from the start position to the final sector.

The A* shortest path search algorithm is now implemented on G_a and the solution is an ordered list of vertices in G_a that have a direct mapping to vertices in G_v on the polygonal map. In the A* solution path $P_a := \{v_0, v_1, v_2, \ldots, v_{n_S}\}$, every pair of consecutive vertices

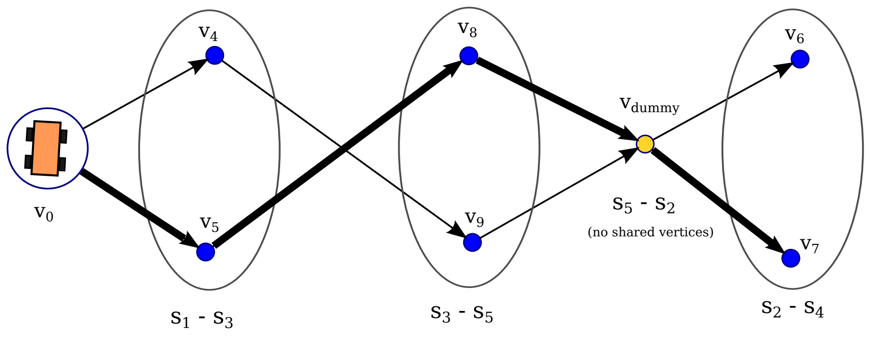


Figure 2.6: Graph construction for intra-sector path optimization. v_0 is the start position of the robot. The path vertices chosen are in the order $\{v_0, v_5, v_8, v_{\text{dummy}}, v_7\}$. Consecutive sectors s_5 and s_2 have no shared vertices.

are an entry-exit pair for a visited sector. In the case where a $v_{\text{dummy}(i)}$ occurs in the path, it is replaced with the shortest path between the two sectors s_i and s_{i+1} with a cost of $\tau(s_i, s_j)$ as calculated on G_v. The intermediate waypoints between the two sectors are then injected into the dummy vertex placeholder. The path P_a is transformed into the final solution path P by injecting intermediate waypoints generated by the sweeping path in each sector s_i between vertices v_i and v_{i+1} indexed according to P_a. Using the above mentioned methods, the final path obtained for the example in Figure 2.4 is illustrated in Figure 2.7.

2.8 Dynamic Re-planning

The planning algorithm described thus far is capable of generating a compete coverage path over a static drivability map. However, in a partially known search environment the planner must be able to dynamically adapt to a changing map and re-plan a coverage path as more information about the environment becomes available. Of particular concern in this situation is determining the re-planning frequency and retaining a memory of previously visited regions of the map to minimize redundant search.

The approach used to solve this problem is consistent with the overarching planning method. Given that within each convex sector, every point lies within the line of sight of the laser scanner, it can be safely assumed that the sectors near the robot contain no large occlusions in the current map. Therefore a path plan for each sector can be executed without a global re-plan. Once each sector is searched, the robot decomposes the map with new information while blocking out sectors already visited. A new coverage path can

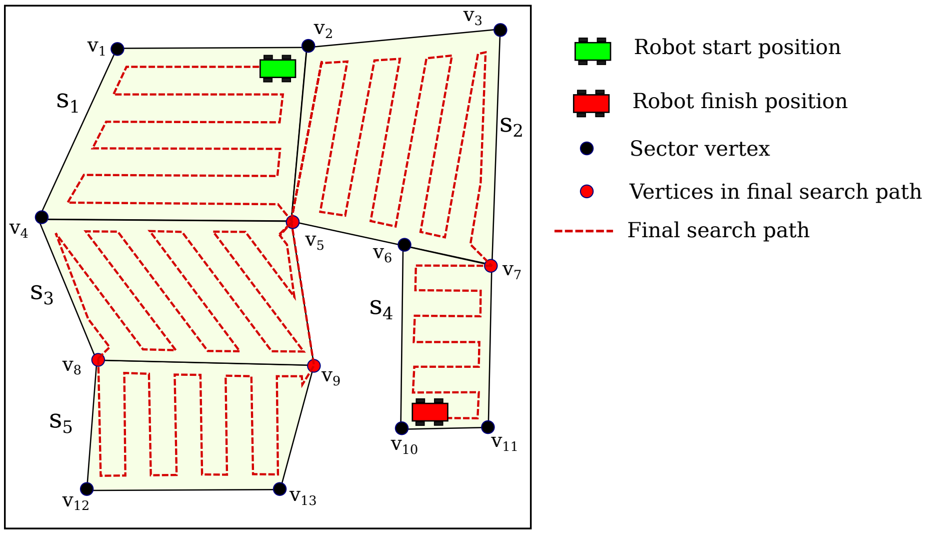


Figure 2.7: Final coverage path: The Sector visit order generated is $\{x_1, x_3, x_5, x_2, x_4\}$. The path travels through inter-sector vertices in the order $\{v_5, v_8, v_9, v_5, v_7\}$. Observe that v_{dummy} between sectors s_5 and s_2 is replaced with the waypoints $\{v_9, v_5\}$.

now be planned to continue the search. This continues until all regions in the map are searched.

2.9 Results

To spur innovation and expand on the state of the art, NASA has developed a Centennial Challenge for rover algorithm design, with a prize of $1.5 million USD. The first NASA Sample Return Robot Challenge (NSRRC) [46] was held in June of 2012 at Worchester Polytechnic Institute, and a second competition took place in June 2013. The challenge requires competitors to develop an autonomous rover weighing less than 80 kg, capable of searching an unknown 80,000 m^2 area for 10 known samples in under 2 hours, at a maximum speed of 2 m/s. Some reference features from the environment are provided, as is a topographical map of the area, but neither GPS nor magnetometer measurements are to be relied on. The test platform designed for this competition is depicted in Figure 2.8.

The objective of coverage is to generate a path to pass a given sensor footprint over all the free space of a search area using a sensor with a larger footprint to generate a map of the environment. Given the dependency on the sensor footprint, it is clear that the navigation

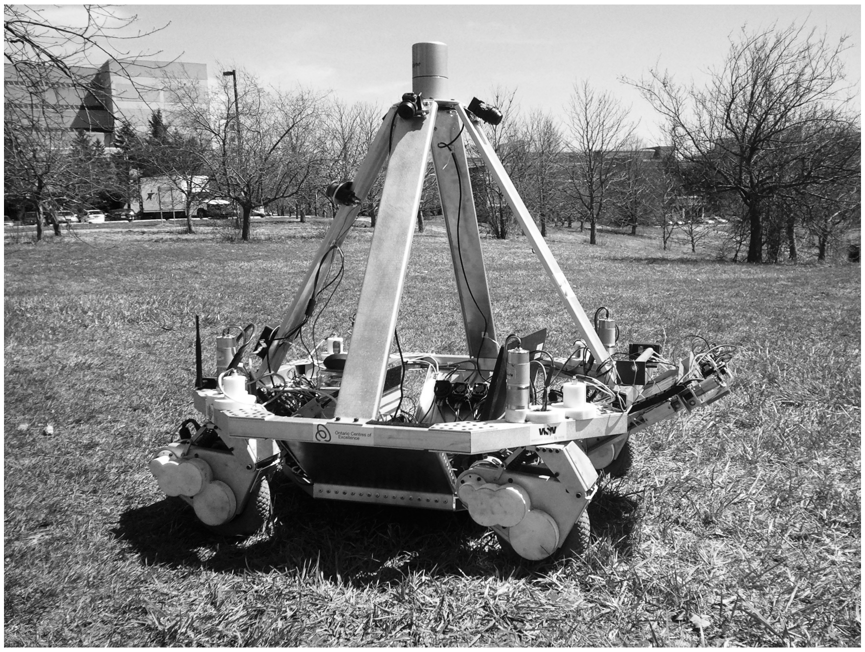
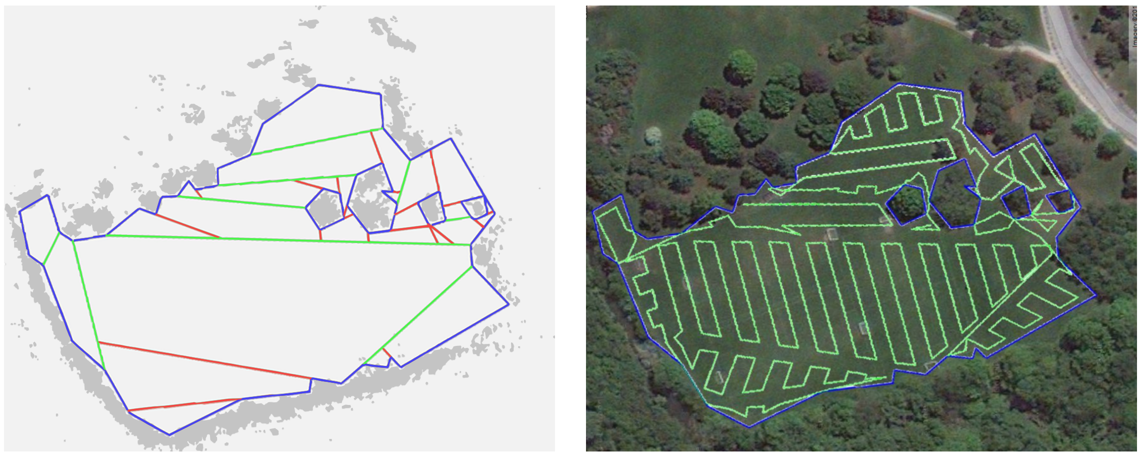


Figure 2.8: University of Waterloo rover for the NSRRC 2013.

algorithms cannot be developed in isolation from object detection, as the design of the coverage path is inherently related to the maximum range at which the object detection can reliably perform. Further, the performance of the planning and mapping algorithms are heavily coupled, as the planned path directly affects the consistency and density of features in the map, and the consistency of the map directly affects the replanning behaviour of the planning algorithm. Thus, the development of a complete autonomous rover system requires careful considerations for each of the algorithms to ensure the overall effectiveness of the system

The experimental results for path planning are presented in two segments. The first set of experiments demonstrate the solution quality and run-time performance of the planning algorithms on a single map against other methods described in literature. The second set of results are presented as integrated experimental results to highlight the considerations required for applying the presented path planning methodology as a component of the autonomous sample return solution. We first consider the interaction between mapping and coverage planning, and second consider the effect of the object detection sensor footprint on planning efficiency.



(a) Decomposition, rectangular region (b) Tour Path, rectangular region

Figure 2.9: Path planning on sample drivability maps generated at Waterloo Park. On the decomposition, grey spots are obstacles, blue lines are boundaries, red lines are single cuts, and green lines are matching cuts. On the right, generated paths are shown overlaid on a satellite image.

2.9.1 Coverage Planning Simulations

The results demonstrating basic path planning are conducted in a centrally sparse region, with an approximate area of 11,000 m^2 that is bounded by shrubbery along a nearly rectangular perimeter. The proposed planning algorithm is validated by making comparisons to the Eulerian path approximation method by Arkin [2]. The main metric of comparison is total path length, and our coverage solutions are displayed in Figure 2.9. Table 2.1 presents detailed decomposition results and collective processing times for the subroutines used in the planning algorithm, excluding initialization routines such as message handling and collision checks. For the experiments, a sweep pitch of 6m is used to reflect the current sample detection camera range of the robot, and the on-board robot computers were used to measure execution times. Table 2.2 shows detailed path length results for the environment, including scenarios for various sweep pitches. In these comparisons, an exaggerated lower bound on the path length is found by dividing the area by the sensor footprint diameter (sweep pitch). Path lengths for the lower bound, Arkin's Eulerian path approximation and the decomposition approach are presented.

It can be noted that the greedy algorithm consistently outperforms the Eulerian path approximation by an average of 25%, with the performance gap widening as pitch is en-

Table 2.1: Greedycut decomposition results, 6m pitch.

Total Area (m^2)	# Obstacles	# Sectors	# Vertices	Run-Time(s)
10,943	4	32	100	0.55

Table 2.2: Path generation results, rectangular region.

Pitch (m)	Lower Bound (m)	Arkin (m)	Coverage Planning		
			Length (m)	Impr. (%)	Run-Time (s)
3	3,648	4,682	4,024	14.05	0.94
6	1,823	2,858	2,124	25.72	0.55
12	9,16	1,946	1,317	32.32	0.31

larged. Given a larger pitch, the distance required to sweep an area is reduced, and the savings from setting sweep directions to avoid circumnavigating obstacles make up a larger percentage of the total path length. In comparing absolute path lengths, experimental results show a distance approximately 1.1 to 1.6 times the lower bound for both test cases. These results demonstrate that the algorithm can successfully decompose a search region into convex search sectors, given a boundary and obstacle data derived from an online 2D drivability map. Execution times have been demonstrated to be under 3 seconds for the given environments, proving its viability for real-time applications. For further experimental analysis, Section 2.9.2 shows how by dynamically reacting to new obstacles, the algorithm is capable of generating tour paths which guarantee full sensor coverage to all areas of a changing map.

2.9.2 Integrated Experimental Results

This section highlights the considerations required for applying the planning algorithms as a component in an integrated solution to solving the autonomous coverage and sample return problem.

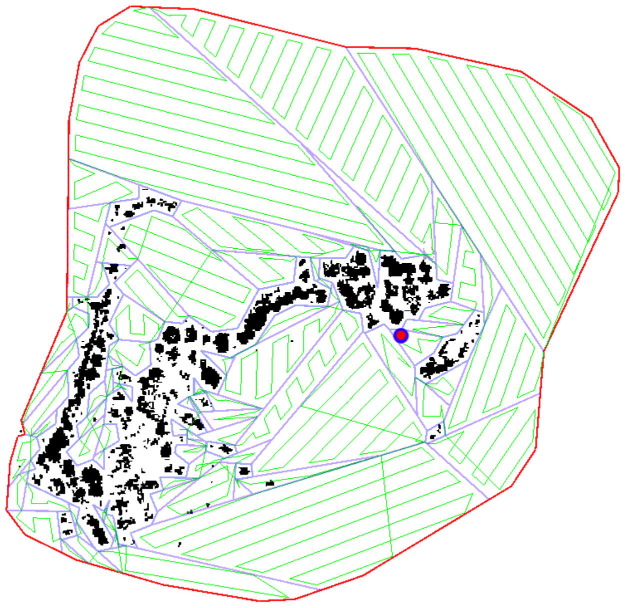
The results showing the dynamic re-planning technique are presented on two drivability maps generated by the mapping module at different time steps in the coverage process, as shown in Figure 2.10. In each map, we assume a known governing boundary for coverage planning. The first map in Figure 2.10(a), is the view of the environment in the initial position of the robot. It shows the tentative path generated to cover the entire environment

Table 2.3: Path generation results for varying sensor footprints.

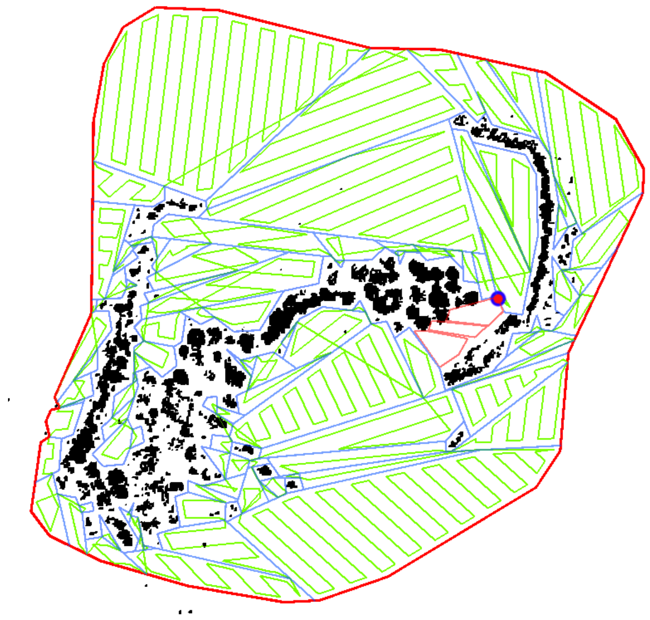
Pitch (m)	Lower Bound (m)	Arkin (m)	Coverage Planning	
			Path length (m)	Improvement (%)
1	16,992	19,286	17,614	8.67
2	8,496	10,790	9,175	14.96
3	5,664	7,957	6,332	20.42
4	4,248	6,541	4,860	25.71
5	3,398	5,692	4,218	25.89

given the current information. In Figure 2.10(b), the path is re-planned from the current robot pose using new map information that is available once the robot is within the line of sight of a previously occluded hedge of shrubs. At each re-planning step, the robot retains a memory of visited sectors as denoted by the red polygons.

While the mapping and planning modules are heavily coupled in their application to navigating the robot, the object detection module is coupled to the planning module through the sensor footprint which sets the width of the sweep pattern. In Table 2.2, it is observed that the radius of the sensor footprint has a significant impact on the quality of the coverage path generated. As the sensor footprint grows, the path not only gets shorter by virtue of a larger footprint, but it also progressively performs better than the alternatives because of the cost savings with inter-sector transitions. A larger sensor radius also implies that the robot can cover the environment at a safer distance from obstacles and reduce the probability of unexpected collisions. Table 2.3 illustrates the effect of varying the width of the sweep pattern on the quality of the computed coverage path for the drivability map in Figure 2.10.



(a) Path plan at initial time step



(b) Re-plan at second time step

Figure 2.10: Demonstration of re-planning at two time steps based on an updated map. The robot pose is denoted by a red dot. The maps show the polygonal decomposition and path plan at each instance.

Chapter 3

Persistent Surveillance

This chapter presents a recharging or refuelling strategy for a team of *working robots* (UAVs) performing a surveillance task, using one or more dedicated *charging robots* (UGVs) to keep them operational over extended periods of time. The objective is to plan a set of paths for the charging robots to rendezvous with the working robots along their trajectories and recharge them as needed during a surveillance operation. It is advantageous to use mobile charging robots because of the ease of deployment in unknown environments and a greater flexibility that comes with dynamic charging locations. The charging process could be performed, for example, by automated docking and battery swap systems, as demonstrated in [55] and [53]. The charging robots carry a payload of batteries than can be swapped into docked working robots to replenish their charge.

3.1 Related Work

The problem of persistent coverage and surveillance with mobile robots has been previously investigated in a variety of contexts. Cortes et al. [11] employ Lloyd's algorithm to develop a centroidal Voronoi tessellation-based controller that optimally covers a convex area with a team of mobile robots. Smith et al. [51] design optimal velocity controllers along precomputed paths to persistently cover a set of discrete points with varying desired frequencies of observation, taking advantage of the ability of mobile robots to sense while in motion. While both these works present persistent surveillance scenarios, neither tackle the problem of limited energy resources in the robotic agents.

Persistent surveillance tasks by definition will exceed the range capabilities of any inspection robot, and therefore naturally require inclusion of recharging in their formulations.

Derenick et al. [13] propose a modification to [11] which introduces a combined coverage and energy dependent control law to drive each robot toward a fixed docking station as their energy levels become critical. Their work considers only the static coverage case with the assumption that the combined sensor footprint of the agents is sufficient to cover the entire environment. There is also no notion of charge scheduling as each agent is assigned a dedicated static charging station. In the worst case scenario all agents will move towards their charging locations simultaneously, leaving the environment unattended.

Contrary to [13], we introduce the notion of mobile charging stations to minimize hindrances to the surveillance objective and plan optimal paths for charging robots to rendezvous with each working robot. Litus et al. [33, 32] consider the problem of finding a set of meeting points for working robots and a single charging robot, given a static set of locations for all robots and a fixed order of working robots to charge. Since our work addresses a dynamic surveillance scenario, we discretize UAV trajectories into rendezvous locations using a sampling-based roadmap method employed by Obermeyer et al. [45] to plan paths for a UAV conducting visual reconnaissance. Obermeyer et al. abstract the path planning problem onto a roadmap graph, formulate it as a variant of the Travelling Salesman Problem and develop approximation algorithms to solve it. In this work, we will extend these graph-based techniques to design rendezvous paths for charging robots.

The work presented in [33], [32] and [45] only considers optimal paths that visit desired targets once to fulfill mission objectives. However, in persistent surveillance missions, working robots may require multiple periodic recharges to ensure functionality over the lifetime of the mission. A common approach, as investigated by Bellingham et al. [4] is to use a receding horizon strategy to dynamically extend shorter trajectories over a larger planning horizon. A challenge with this approach is ensuring that each subsequent planning iteration can provide a feasible solution path. Schouwenaars et al. [50] present an iterative MILP path planning approach with implicit safety guarantees that ensure feasibility of successive planning iterations.

An alternative approach is to formulate an optimal path planning problem over the entire planning horizon. Michael et al. [37] investigate a persistent surveillance problem for a team of UAVs to periodically visit a set of interest points with varying frequencies. They formulate the problem as a Vehicle Routing Problem with Time Windows (VRPTW), which is a variant of the classical Vehicle Routing Problem (VRP) that seeks to design routes for multiple vehicles to visit all vertices in a graph. In this work we address an additional challenge of planning optimal paths for a team of charging robots to rendezvous with moving targets (UAVs).

3.2 Motion Planning For Charging Robots

Given a team of robots performing a persistent task, the problem in question is to compute paths for the team of charging robots such that they optimally rendezvous with every working robot exactly once. We extend our results to multiple charging rendezvous over finite planning horizons in Section 3.5.1. The working robots are not required to divert from their trajectories. This minimizes hindrances to the persistent mission caused by the recharging process. The assumption is made that charging robots possess sufficient energy resources and need not be refuelled or restocked within the planning horizon. The problem can now be formally stated.

3.2.1 Continuous Problem Formulation

Consider an environment, $\mathcal{E} \subset \mathbb{R}^2$, which contains R working robots, denoted by the set $\mathcal{R} = \{1, \ldots, R\}$, performing a persistent task. Each working robot, indexed by $r \in \mathcal{R}$, is described by its motion along a known trajectory, $p_r(t) \in \mathcal{E}$ within a planning horizon $t \in [0, T_r]$ determined by its lifetime on a single charge, and a charging time window $[\underline{T}_r, \overline{T}_r] \subseteq [0, T_r]$.

The environment also contains M charging robots, denoted by the set $\mathcal{M} = \{1, \ldots, M\}$, that are free to move arbitrarily within $\mathcal{E}$. Each charging robot, indexed by $m \in \mathcal{M}$, is described by its initial position $p_m(0)$ and its maximum speed, υ. We assume that all charging robots have the same maximum speed. The problem is to find optimal paths for the charging robots, $p_m(t) \in \mathcal{E}$ (where $|\dot{p}_m(t)| \leq \upsilon$) such that for each $r \in \mathcal{R}$, there exists a charging robot $m \in \mathcal{M}$ and a time $t_r \in [\underline{T}_r, \overline{T}_r]$ for which $p_m(t_r) = p_r(t_r)$.

This constraint states that the team of charging robots must rendezvous at least once with each working robot at a point along its respective path before it runs out of charge. Figure 3.1 illustrates the problem statement with a team of four working robots following a single path, along with two charging robots.

The continuous-time problem, as stated, requires an optimization over the space of all charging robot trajectories [49]. Hence, discretizing the formulation converts the problem into a more tractable form and allows the application of graph-based linear program techniques to obtain a solution.

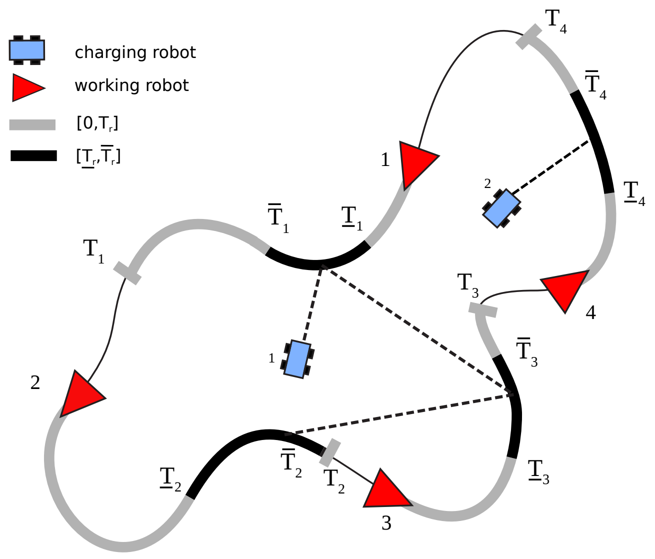


Figure 3.1: Four working robots (red triangles) travelling along one path. Each working robot's lifetime horizon is denoted by a bold grey line and the charging window, by a bold black line. The two blue charging robots must meet all working robots on their paths within their charging windows to guarantee persistent operation.

3.2.2 Problem Discretization

For each working robot r, given that the trajectory $p_r(\cdot)$ is known over the planning horizon, we can discretize its charging time window to generate a set of K_r charging times $\tau_r = \{t_{r,1}, \ldots, t_{r,K_r}\} \subseteq [\underline{T}_r, \overline{T}_r]$ at which it can be reached along its trajectory. The set of charging points that result are defined as,

$$C_r = \{(p_r(t), t) \,|\, t \in \tau_r\}.$$

Each charging point $(p_r(t_{r,i}), t_{r,i})$ is described by its time of occurrence, $t_{r,i}$, and its position along the robot path $p_r(t_{r,i})$.

A charging robot, subject to its speed constraints, will attempt to charge a working robot by arriving at one of its charging points $(p_r(t_{r,i}), t_{r,i}) \in C_r$ at a time $t \leq t_{r,i}$ and staying there until time $t_{r,i}$ such that $p_m(t_{r,i}) = p_r(t_{r,i})$. This definition satisfies the previously stated condition for a rendezvous in continuous time. Note that for the sake of simplicity, the formulation assumes instantaneous charge, but it can be extended directly to the case of nonzero charging durations, as discussed in Remark 3.2.1.

The discrete problem is one of finding paths for the charging robots that visit one charging point in each set C_r. We can encode every possible charging path in a partitioned directed graph G, defined as follows.

Vertices The vertices, are defined by $R+1$ disjoint vertex sets, $V_0, V_1, \ldots, V_R$. The set V_0 is the set of initial locations of the charging robots. Each vertex in set V_r, for $r \in \mathcal{R}$ corresponds to a charging point in C_r, the set of all charging points for robot r. The complete vertex set is then $V = V_0 \cup V_1 \cup \cdots \cup V_R$.

Edges An edge (v_i, v_j) is added to E, where $v_i \in V_{r_1}$ and $v_j \in V_{r_2}$ for some $r_1, r_2 \in \mathcal{R}$ with $r_1 \neq r_2$, to E if there exists a feasible traversable path from charging point $(p_{r_1}(t_{r_1,i}), t_{r_1,i})$ to $(p_{r_2}(t_{r_2,j}), t_{r_2,j})$. That is, if

$$\frac{\|p_{r_2}(t_{r_2,j}) - p_{r_1}(t_{r_1,i})\|}{\upsilon} \leq t_{r_2,j} - t_{r_1,i}. \tag{3.1}$$

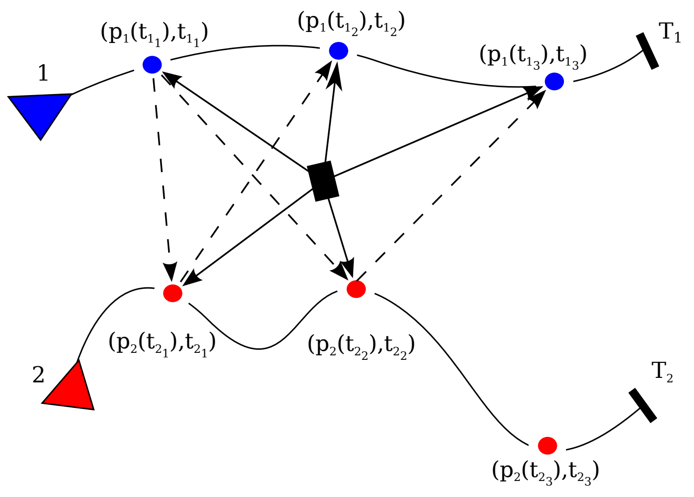
Edge Costs Each edge $e = (v_i, v_j) \in E$ is associated with a non-negative cost $c(e)$ that can be chosen based on the objective of the optimization such as minimizing total distance travelled by charging robots or total makespan of the recharging process. Further, in order to avoid recharging UAVs too early, a penalty proportional to the voltage level of robot r at its charging point $(p_r(t_{r,i}), t_{r,i})$ can be added to all incoming edges at each vertex $v_i \in V_r$.

Remark 3.2.1 (Nonzero Charging Durations). For simplicity of presentation we have assumed that charging occurs instantaneously. Thus, if a charging robot performs a rendezvous with a working robot at charging point $(p_r(t_{r,i}), t_{r,i})$, it can leave that charging point at time $t_{r,i}$. We can extend this formulation to charging points described as triples $(p_r(t_{r,i}), t_{r,i}, \Delta t_{r,i})$, where $\Delta t_{r,i}$ is the time required to charge robot r at the i^{th} charging point. In this case the charging robot can leave the charging point at time $t_{r,i} + \Delta t_{r,i}$. The condition to add an edge in equation 3.1 then changes slightly to

$$\frac{\|p_{r_2}(t_{r_2,j}) - p_{r_1}(t_{r_1,i})\|}{\upsilon} \leq t_{r_2,j} - (t_{r_1,i} + \Delta t_{r_1,i}). \tag{3.2}$$

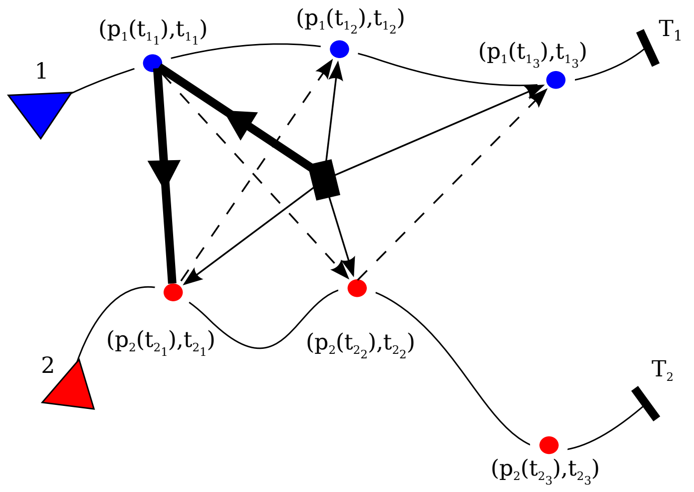
●

As a simple illustrative example, Figure 3.2(a) shows two working robots r_1 (blue) and r_2 (red) following arbitrary trajectories and one charging robot m_1 in an environment $\mathcal{E} \subset \mathbb{R}^2$. Each robot path is discretized into three charging points and graph G is constructed on them based on the feasibility conditions.



$(p_1(t_{1_1}), t_{1_1})$
$(p_1(t_{1_2}), t_{1_2})$
$(p_1(t_{1_3}), t_{1_3})$
T_1
1
2
$(p_2(t_{2_1}), t_{2_1})$
$(p_2(t_{2_2}), t_{2_2})$
T_2
$(p_2(t_{2_3}), t_{2_3})$

(a) Sampled UAV trajectories and roadmap graph



$(p_1(t_{1_1}), t_{1_1})$
$(p_1(t_{1_2}), t_{1_2})$
$(p_1(t_{1_3}), t_{1_3})$
T_1
1
2
$(p_2(t_{2_1}), t_{2_1})$
$(p_2(t_{2_2}), t_{2_2})$
T_2
$(p_2(t_{2_3}), t_{2_3})$

(b) Optimal recharge path solution

Figure 3.2: Building a traversal graph for two working robots and one charging robot. The resulting graph is a directed acyclic graph with vertex partitions.

In addition to the vertex partition, an interesting property of the constructed graph G is that there are no edges between vertices of the same vertex set. This property makes the graph *multipartite* in nature. Further, since the edges represent rendezvous conditions between pairs of time-stamped locations and all edges are directed towards vertices increasing in time, it is impossible for G to contain any directed cycles. Hence, by definition G is a *partitioned directed acyclic graph (DAG)*.

3.2.3 Optimization on a Partitioned Directed Acyclic Graph

Given a partitioned DAG G, the goal is to find an optimal path or set of paths that collectively visit each set in the partition once, as shown in Figure 3.2(b). To characterize the complexity of our problem, it will be helpful to state it as a graph optimization.

We begin by defining the stated problem of computing a set of charging robot rendezvous paths as a decision problem known as the *One-in-a-set DAG Path Problem*.

Problem 3.2.2 (The One-in-a-set DAG Path Problem)**.** Consider a partitioned DAG G and a partition $(V_0, V_1, \dots, V_R)$ of V where $V_0 = \{v_m | m \in \mathcal{M}\}$. Does there exist a set of paths $P = \{P_1, \dots, P_M\}$ in G, where $P_m \in P$ starts at $v_m \in V_0$, such that $|V_i \cap V_P| = 1$ for all $i \in \mathcal{R}$?

We will say that the partitioned DAG G contains One-in-a-set path(s) if and only if the answer to the corresponding decision problem is yes.

The One-in-a-set Path problem has been proved to be NP-hard for the case of undirected, complete, or general directed graphs, because they contain, as special cases, the undirected and directed TSP problems, respectively, which are both NP-hard. Unlike these TSP problems, the One-in-a-set DAG Path problem consists of a path through a directed acyclic graph, which is not trivially provable as NP-hard given that the *longest path problem* for directed acyclic graphs is solvable in polynomial time using dynamic programming [16]. However, in the following section we prove that the One-in-a-set DAG Path problem is in fact an NP-hard problem.

3.2.4 Hardness of Discrete Problem

We will prove NP-hardness of the One-in-a-set DAG Path problem by using a reduction from the NP-Complete Hamiltonian path problem [28].

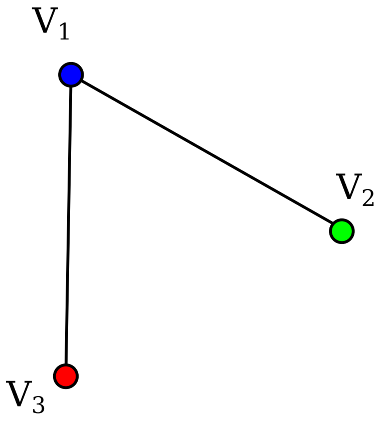
Theorem 3.2.3 (NP-Completeness of Problem 3.2.2)**.** *The One-in-a-set DAG Path problem is NP-Complete.*

Proof. Suppose we have an instance of the Hamiltonian path problem defined on graph G. We will give a polynomial transformation of G into an input $\overline{G}$ for the One-in-a-set DAG Path decision problem.

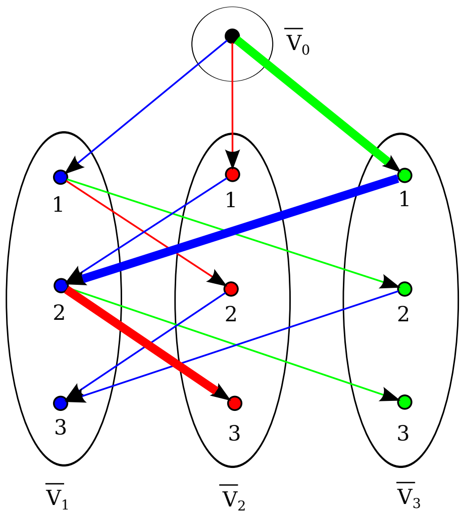
Given the undirected graph G, we need to create a DAG $\overline{G} = (\overline{V}, E)$ along with the vertex partition $(\overline{V}_0, \overline{V}_1, \ldots, \overline{V}_R)$. Our approach will be to encode every possible Hamiltonian path order in $\overline{G}$. The One-in-a-set DAG decision problem will then have a yes answer if and only if the graph G contains a Hamiltonian path.

Let $V = (v_1, \ldots, v_R)$ and for each $r \in \{1, \ldots, R\}$, let $\overline{V}_R$ be given by R copies of v_r, which we will denote by $\overline{V}_r := (v_{r,1}, \ldots, v_{r,R})$. The jth copy of v_r will correspond to all paths in G that have v_r as their jth vertex. Finally, we create a (dummy) vertex $\overline{V}_0$ and define $\overline{V} = \overline{V}_0 \cup \overline{V}_1 \cup \cdots \cup \overline{V}_R$.

Now, we define the edges $\overline{E}$ as follows. We begin by adding an the edge $(\overline{V}_0, v_{r,1})$ to $\overline{E}$ for each $r \in \{1, \ldots, R\}$. Then for any two sets $\overline{V}_i$ and $\overline{V}_j$ and for $k \in \{1, \ldots, R-1\}$ we add the edge $(v_{i,k}, v_{j,k+1})$ if and only if $(v_i, v_j) \in E$. Figure 3.3 illustrates this reduction and shows that a feasible path is found in the DAG. It is clear that a feasible solution to the described One-in-a-set DAG Path problem yields a feasible solution to the Hamiltonian path problem.



(a) $G = (V, E)$



(b) $\overline{G} = (\overline{V}, \overline{E})$

Figure 3.3: A reduction of the Hamiltonian Path Problem to the One-in-a-set DAG Problem. Each color in graph G represents an individual vertex. Each vertex color in graph G corresponds to a unique vertex set in graph $\overline{G}$

This defines the input $\overline{G}$ to the One-in-a-set DAG decision problem. It is easy to see that $\overline{G}$ is acyclic since it has a topological sort: Define the partial ordering as $v_{i,k} \leq v_{j,\ell}$ if and only if $k \leq \ell$ and note that there is an edge from $v_{i,k}$ to $v_{j,\ell}$ only if $\ell = k + 1$. Also, note that $\overline{G}$ has $R^2 + 1$ vertices.

Finally, we just need to show that G contains a Hamiltonian path if and only if $\overline{G}$ contains a One-in-a-set path. Suppose G contains a Hamiltonian path $P = v_{r_1} v_{r_2} \cdots v_{r_R}$, where $(v_{i_j}, v_{i_{j+1}}) \in E$ for each $j \in \{1, \ldots, R-1\}$. Then, the path $\overline{P} = \overline{V}_0, v_{r_1,1} v_{r_2,2} \cdots v_{r_R,R}$ is a One-in-a-set path in $\overline{G}$ since each edge $(v_{r_j,j},\, v_{r_{j+1},j+1})$ is in $\overline{E}$.

Conversely, suppose that $\overline{G}$ contains a One-in-a-set path $\overline{P}$. By the definition of the edges $\overline{E}$, the path must be of the form $\overline{V}_0, v_{r_1,1} v_{r_2,2} \cdots v_{r_R,R}$. This implies that $(v_{r_j}, v_{r_{j+1}}) \in E$ for each $j \in \{1, \ldots, R-1\}$ and thus $P = v_{r_1} \cdots v_{r_R}$ is Hamiltonian path in G. $\square$

NP-Completeness of the One-in-a-set DAG decision problem implies that Problem 3.2.2 is NP-Complete, and thus our recharging optimization problem is NP-hard. In what follows we present our approach to the problem from the bottom up. We first formulate the MILP for the single charging robot case and use it to characterize the structure of the optimization and inform our solution methods. We then extend the problem to include multiple charging robots and investigate algorithmic alternatives to generate near optimal solutions.

3.3 Mixed Integer Linear Program Formulation

The One-in-a-set DAG Path problem can be stated as a MILP and optimally solved for smaller instances of the problem. For ease of presentation we first formulate the MILP for a single charging robot path in a partitioned DAG.

Given a partitioned graph G defined for the One-in-a-set DAG Path problem, we make a small modification to apply degree constraints. A dummy finish vertex, v_f, is added to V_0 and equal cost edges are assigned from every vertex back to v_f.

Given the partitioned graph G with vertex sets $(V_0, V_1, \ldots, V_R)$, we define a decision variable, $x_{ij} \in \{0, 1\}$ with $x_{ij} = 1$ if, in the resulting path, a visit to vertex v_i is followed by a visit to vertex v_j, where $i \in V_{r_1}$, $j \in V_{r_2}$ and $r_1 \neq r_2$, $r_1, r_2 \in \mathcal{R}$. The cost of the edge traversal x_{ij} is denoted by c_{ij}, and is defined as follows. For the edge $e = (v_i, v_j)$ (with associated decision variable x_{ij}) we define.

$$c_{ij} = \begin{cases} c(e), & \text{if } e \in E, \\ \infty, & \text{if } e \notin E. \end{cases} \tag{3.3}$$

The start vertex is denoted by index d. The solution path must end at the dummy vertex denoted with index f. The single charging robot MILP is now defined as follows.

$$\min \quad \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{3.4}$$

subject to

$$\sum_{j \in V \setminus V_0} x_{dj} = 1 \tag{3.5}$$

$$\sum_{i \in V \setminus V_0} x_{if} = 1 \tag{3.6}$$

$$\sum_{j \in V_r} \sum_{i \in V} x_{ij} = 1 \qquad \forall r \in \mathcal{R} \tag{3.7}$$

$$\sum_{i \in V_r} \sum_{j \in V} x_{ij} = 1 \qquad \forall r \in \mathcal{R} \tag{3.8}$$

$$\sum_{i,j \in V} (x_{ik} - x_{kj}) = 0 \qquad \forall k \in V \setminus V_0 \tag{3.9}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{3.10}$$

The objective function (3.4) seeks to minimize the total path cost defined as the travel distance of the charging robot. Constraint (3.5) and (3.6) guarantee that the tour starts at the start vertex and ends at the finish vertex. Constraint (3.7) and (3.8) ensure that each vertex set is visited only once. Constraint (3.9) is a flow constraint to guarantee that the entering and exiting edge for each vertex set are both incident on the same vertex in the group. Finally Constraint (3.10) specifies binary constraints on the decision variables x_{ij}. Given $M = 1$, the total number of constraints in this formulation is $(2M + 2R + N)$. The maximum number of binary decision variables on the edges of a complete graph is $N(N-1)$. However, owing to the multipartite nature of the graph G, a decision variable x_{ij} is only defined if v_i and v_j belong to different vertex sets.

The complexity of the MILP problem is influenced by the number of binary variables and constraints, which grows with the number of vertices in graph G. For a given environment and configuration of working and charging robots, the size of G is determined by the length of the charging window $[\underline{T}_r, \overline{T}_r] \subseteq [0, T_r]$, and the density of charging locations along each robot path.

3.3.1 Special Problem Characteristics

The optimal solution to the MILP provides a minimum cost path that passes through each vertex set of a DAG exactly once. We observe from the formulation that the problem can be modelled as the Generalized Travelling Salesman Problem (GTSP) [43] as stated in

Problem 1.4.5.

Despite structural similarities to the GTSP, it is interesting to note that the MILP formulated for the One-in-a-set DAG Path problem introduces a significantly smaller constraint set than TSP and GTSP routing problems. In addition to the degree and flow constraints stated, routing problems require subtour elimination constraints to ensure a continuous path and avoid disjoint subtours in the solution. A sub-tour elimination constraint in both classes of problems is formulated as

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \quad \forall S \subset V,\ 2 \leq |S| \leq N - 2,$$

where, for a graph G, N is the total number of vertices in V, S is any subset of vertices in V that can form a sub-tour and x_{ij} is the decision variable on edge $(v_i, v_j) \in E$. A TSP with N vertices requires $2^N - 2N - 2$ subtour elimination constraints. Likewise for a GTSP with N vertices and R vertex sets, the linear program contains as many as $2^{R-1} - R - 1$ subtour elimination constraints [42].

In comparison, the lack of directed cycles in a DAG eliminates any need for sub-tour elimination constraints in our formulation. Further, the multipartite nature of the graph removes the need for binary decision variables on intraset edges. This significant reduction in the number of constraints means that we can solve larger problems with relatively lesser computational effort. In practice, we observed that problems with an order of magnitude increase in the number of vertices that could be solved in comparable time. Nevertheless, given the NP-hardness of the problem, optimally solving the MILP will not always be computationally tractable and Section 3.4 describes the algorithmic approach we use to compute near-optimal solutions.

3.3.2 Extending the MILP for Multiple Charging Robots

The linear program in Section 3.3 can be easily extended to the multiple charging robot problem, using a three-index flow formulation. We highlight the differences here and refer a reader to [35] for more details.

In the extended formulation each charging robot m is represented by an independent route p_m. Thus the binary decision variables on edges are defined as $x_{ijm} \in \{0, 1\}$ with $x_{ijm} = 1$ if, in route p_m, the vertex v_j is visited after vertex v_i, where $i \in V_{r_1}, j \in V_{r_2}, r_1 \neq$

r_2 and $r_1, r_2 \in \mathcal{R}$. The new objective function is

$$\min \quad \sum_{m=1}^{M} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ijm}.$$

This expression represents a *min-sum* objective that seeks to minimize the total path cost of all charging vehicles. The MILP can be redefined to include a *min-max* objective that minimizes the maximum path cost of any single charging robot, or, to add constraints to bound each path cost, similar to the well-known Capacitated Vehicle Routing Problem (CVRP) [29]. These extensions are applicable when we wish to set limits on the maximum load capacities of the charging robots, balance their work loads or include depots to restock their charging payload.

Similar to the single charging robot problem, the multiple charging robot problem can be modelled as an MGTSP (see Definition 1.4.6), where each of the M charging robots is assigned a start-depot and, at most, M paths that optimally visit each vertex set once must be computed.

3.4 Algorithmic Approach: Graph Transformations

As covered in Section 1.4, there are a number of algorithmic approaches for solving the GTSP. A common approach is the Noon-Bean Transformation [45], which transforms any GTSP instance into an equivalent TSP instance. The method guarantees that the optimal TSP solution to the transformed problem will always correspond to the optimal solution to the original GTSP.

Section 3.4.1 presents an implementation to transform the One-in-a-set DAG path problem, with a single charging robot, into a TSP using the Noon-Bean Transformation. The Noon-Bean method applies only to a problem modelled as a GTSP and in order to solve the multiple charging robot route problem, a specialized solution approach for MGTSP instances is required. Hence, in Section 3.4.2, we propose a novel modification to the Noon-Bean method to allow a transformation of any MGTSP instance into a TSP. The optimal solution to the TSP can then be used to construct the optimal MGTSP solution.

3.4.1 Path Computation for a Single Charging Robot

In what follows we define a sequence of problems, beginning with the One-in-a-set DAG Path problem, and ending with a TSP problem. To begin, we define Problem (P0) to be an instance of the One-in-a-set DAG Path problem for a single charging robot. Problem (P0) is a GTSP instance defined on a Partitioned DAG G^0 with a partition of vertices into $R+1$ mutually exclusive sets $V^0 = (V_0^0, V_1^0, \dots, V_R^0)$. Set V_0^0 contains the start-depot of the charging robot, v_d. We seek the shortest path starting at v_d, and visiting each vertex set exactly once. Figure 3.4(a) shows a sample instance of the problem.

As shown in [43], the Noon-Bean Transformation can now be used to transform the graph instance in Problem (P0) to new problem (P1), which is a TSP defined on a graph G^1. The vertices V^1, edges E^1 and cost function c^1 are defined as follows.

(i) Define the set of vertices of G^1, as $V^1 = V^0$. In set V_0^1, add a depot v_f, as the charging robot route finish-depot. Add edges (v_j, v_f), where $v_j \in V^1 \setminus V_0^1$ and assign a cost based on the desired optimization objective.

(ii) For each vertex set V_r^1, create an arbitrary ordering of its vertices $(v_i, v_{i+1}, \dots, v_{|V_r^1|})$. Add zero-cost directed edges that create a directed cycle through the vertices in the chosen order. The dotted black edges in figure 3.4(b) show these intraset edges in Problem (P1).

(iii) Shift the tail end of each interset edge $(v_i, v_k) \in E^0$, to (v_{i-1}, v_k), the vertex immediately preceding it in the corresponding intraset cycle. Add these edges to E^1.

(iv) A large penalty $\beta > \sum_{e \in E^1} c^1(e)$ is added to all the interset edges to ensure that the lowest cost TSP tour will never exit a vertex set without traversing the entire intraset vertex cycle.

Problem (P1) is a TSP instance, which can be solved using a variety of freely and commercially available TSP solvers. The goal of the Noon-Bean method is to transform the GTSP into a TSP instance, in which an optimal tour visits all vertices in a vertex set in a clustered manner before moving on to other sets. The penalty β added to interset edges ensures that the shortest tour always contains a clustered solution.

The TSP solution to Problem (P1), denoted by Υ^1 can be used to construct the GTSP solution, Υ^0, to Problem (P0), given that it satisfies $\sum_{e \in E_{\Upsilon^1}} c^1(e) \leq (R+2)\beta$. This condition ensures the clustered nature of the TSP tour and stems from the fact that a feasible GTSP solution through $R+1$ vertex sets contains only $R+1$ interset edges. Since

$\beta > \sum_{e \in E^1} c^1(e)$, has been added to every interset edge, we know that the cost of a TSP tour that corresponds to a feasible GTSP tour cannot be greater than $(R+1)\beta + \beta = (R+2)\beta$.

A feasible GTSP solution Υ^0, to Problem (P0), can now be constructed by sequentially extracting the entry vertex at every cluster in the TSP tour Υ^1. The total cost of the reconstructed GTSP solution $\sum_{e \in E_{\Upsilon^0}} c^0(e) = \sum_{e \in E_{\Upsilon^1}} c^1(e)$.

3.4.2 Path Computation For Multiple Charging Robots

In this section we propose a novel extension to the Noon-Bean method to transform the MGTSP into a TSP. The transformation ensures that the optimal solution to the TSP can be used to construct the optimal solution to the MGTSP. The algorithm is implemented to solve the One-in-a-set DAG path problem for multiple charging robots.

We begin by stating the One-in-a-set DAG Path problem as an MGTSP and call this Problem (P2). See Figure 3.5 as an example. Problem (P2) is an instance of an MGTSP, defined over a partitioned DAG, G^2, with a partition of its vertices V^2 into $R+1$ sets, $(V_0^2, V_1^2, \ldots, V_R^2)$. The vertex set V_0^2 contains M start-depots for charging robots. We seek a set of paths starting at the depots $v_d^i, i \in \mathcal{M}$ that visit all the vertex sets $V_1^2, \ldots, V_R^2$ exactly once.

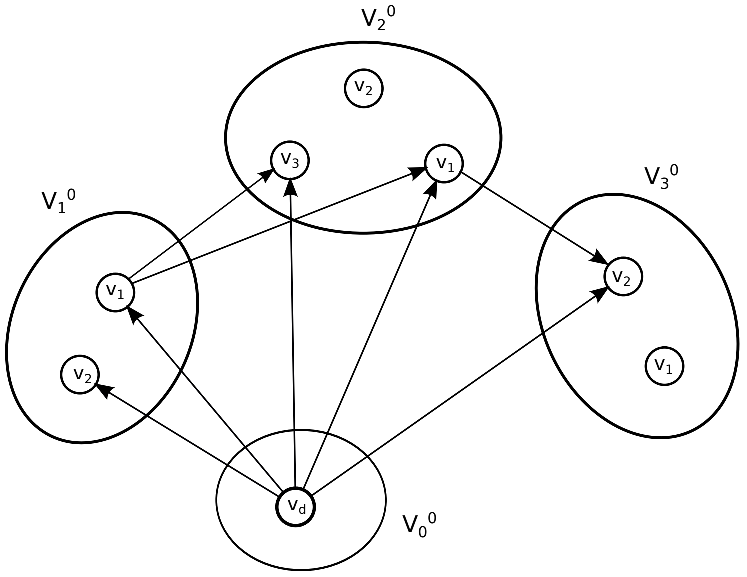
Transformation Algorithm

The new transformation algorithm converts the MGTSP problem instance (P2), into a new problem instance (P3) on which a TSP solution may be computed. Problem (P3) is a TSP defined over a graph G^3. The vertices, V^3, edges E^3 and cost function c^3 are defined as follows.

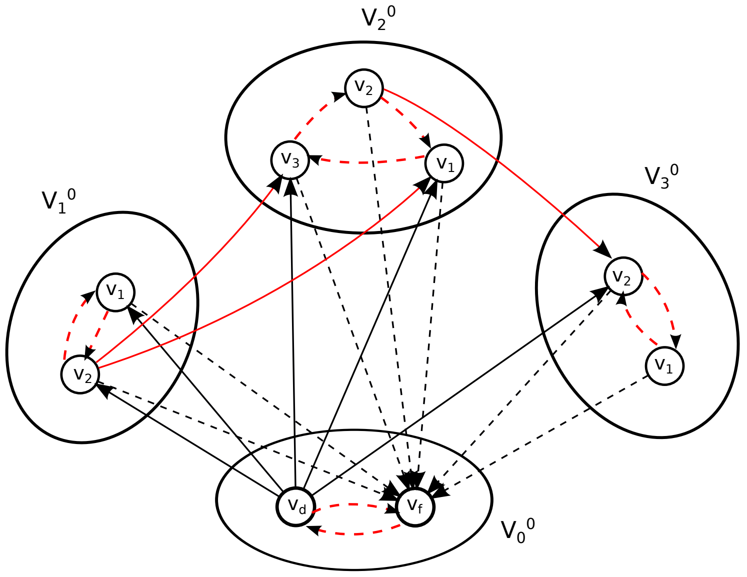
(i) Define the set of vertices of G^3, as $V^3 = V^2$. In set V_0^3, add M vertices, $v_f^i, i \in \mathcal{M}$, as the charging robot route finish-depots. At each vertex v_f^i, add edges (v_j, v_f^i), where $v_j \in V^2 \setminus V_0^2$ and assign costs based on the optimization objective.

(ii) In vertex set V_0^3, arrange all start-finish depot pairs (v_d^i, v_f^i) in an arbitrary sequential ordering to obtain $V_0^3 = \{v_d^1, v_f^1, v_d^2, v_f^2, \ldots, v_d^M, v_f^M\}$. Create zero-cost intraset edges forming a single directed cycle through all vertices in V_0^3, in the chosen order. Hence, create edges $(v_d^1, v_f^1), (v_f^1, v_d^2), \ldots, (v_d^M, v_f^M), (v_f^M, v_d^1)$.

(iii) For the definition of all edges (v_i, v_j) where $v_i, v_j \in V^3 \setminus V_0^3$ and $i \neq j$, use the original Noon-Bean method presented in Section 3.4.1.



(a) A sample instance of (P0) with vertex sets (V_1^0, V_2^0, V_3^0) and set V_0^0, which contains the charging robot depots.



(b) The problem instance (P1) generated using the Noon-Bean Transformation. Transformed interset and instraset edges are shown in red.

Figure 3.4: The Noon-Bean transformation for GTSPs

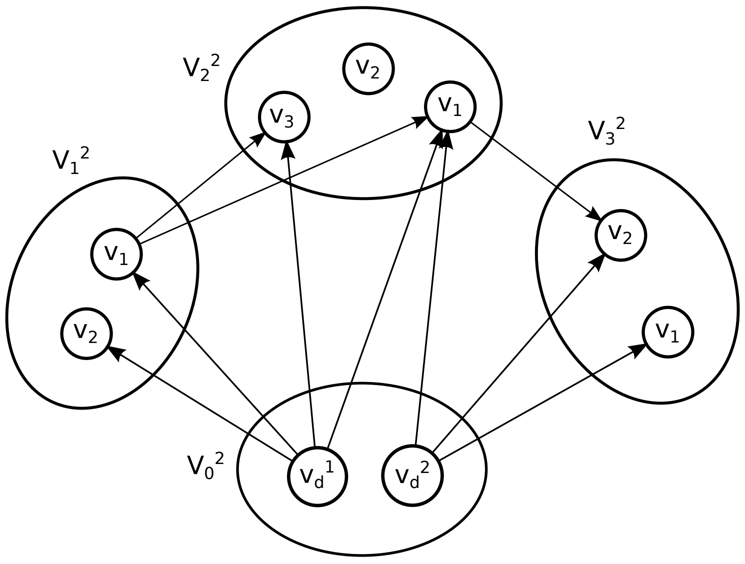


Figure 3.5: An sample instance of Problem (P2), with $R = 3$, and $M = 2$

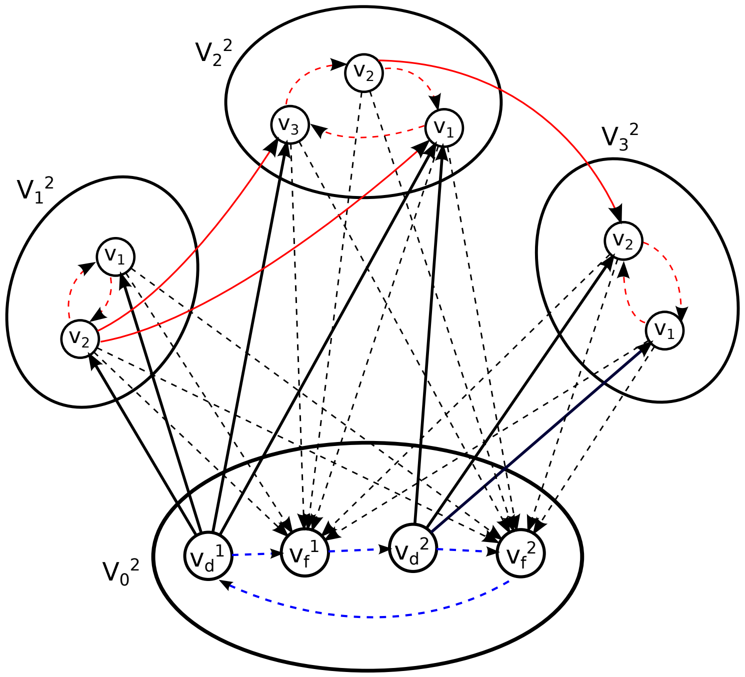


Figure 3.6: Problem instance (P3), generated using the modified Noon-Bean algorithm. Red edges represent the Noon-Bean transformation and blue edges represent new additions in the modified algorithm.

(iv) Add the penalty $\beta > \sum_{e \in E^3} c^3(e)$ to all edges (v_i, v_j) where $v_i, v_j \in V^3 \setminus V_0^3$ and $i \neq j$. Further, add penalty β to all outgoing interset edges from start-depots v_d^i in set V_0^3. Penalty β is not added to any edges incident on finish-depot vertices in V_0^3.

Figure 3.6 illustrates the transformed graph G^3, for Problem (P3). Before proving correctness of the modified Noon-Bean theorem, we require some intermediate results.

Lemma 3.4.1. *In any TSP solution to Problem (P3), each start-depot vertex $v_d^i \in V_0^3$ will be immediately preceded by the finish-depot vertex, $v_f^{i-1} \in V_0^3$ in the chosen cyclic ordering of vertices in set V_0^3.*

Proof. Every start-depot $v_d^i \in V_0^3$ has an in-degree of one. Hence a path visiting a start-depot can do so only through the preceding finish-depot vertex in the given cyclic ordering of V_0^3. □

This simple result implies that the indices of the finish-depot vertices will allow us to "cut" a single TSP tour into paths for each working robot. Lemmas 3.4.2 and 3.4.3 define the method and conditions under which the TSP solution to Problem (P3) provides the MGTSP solution to Problem (P2).

Lemma 3.4.2. *The optimal TSP solution to Problem (P3) can be used to construct the optimal MGTSP solution to Problem (P2).*

Proof. According to the modified Noon-Bean transformation, if an optimal MGTSP solution to Problem (P2), Υ^2, is defined by the set of M paths as,

$$\{\{v_d^1, v_j, \ldots, v_k, v_f^1\}, \ldots, \{v_d^M, v_a \ldots, v_b, v_f^M\}\},$$

then the corresponding optimal TSP solution Υ^3 to the transformed problem (P3) will be,

$$\begin{aligned} &v_d^1, v_j, v_{j+1}, \ldots, v_{j-1}, \ldots, v_k, v_{k+1}, \ldots, v_{k-1}, v_f^1, \\ &v_d^M, v_a, v_{a+1}, \ldots, v_{a-1} \ldots, v_b, v_{b+1}, \ldots, v_{b-1}, v_f^M, v_d^1 \end{aligned}$$

The optimal TSP path visits all vertices in vertex sets $\{V_1^3, \ldots, V_R^3\}$ in a clustered manner as shown in the Noon-Bean transformation. The vertices of set V_0^3 are visited intermittently between interset transitions in finish-depot, start-depot pairs as specified in Lemma 3.4.1. As stated in Lemma 3.4.1, the TSP tour can be cut into optimal paths for each of the charging robots. Further, given that each interset edge of Υ^3 has a cost equal to the corresponding interset edge in Υ^2, we can determine that $\sum_{e \in E_{\Upsilon^3}} c^3(e) = \sum_{e \in E_{\Upsilon^2}} c^2(e)$.

□

We know that an optimal solution (P3) always corresponds to the optimal solution to (P2). Lemma 3.4.3 extends this result to define the condition under which a feasible TSP solution to (P3) can provide a feasible solution to (P2)

Lemma 3.4.3. *A feasible TSP solution, Υ^3, to Problem (P3) provides a feasible MGTSP solution, Υ^2, to Problem (P2) given that $\sum_{e \in E_{\Upsilon^3}} c^3(e) < (R+2)\beta$.*

Proof. From Subsection 3.4.1, we know that a feasible GTSP solution through $R+1$ vertex sets contains $R+1$ interset edges and the cost of a corresponding TSP solution cannot exceed $(R+2)\beta$.

In the case of multiple charging robots, the number of interset edges in the solution depends on the number of charging robot routes. However, since the edges incident on finish-depots in V_0^3 do not have the penalty, β, added to their cost, the number of large-cost interset edges in the solution is $R+1$, independent of the number of charging robot routes used. Hence, a feasible solution to Problem (P2) can be constructed from a solution to Problem (P3), if $\sum_{e \in E_{\Upsilon^3}} c^3(e) < (R+2)\beta$.

□

In the case of both Lemma 3.4.2 and 3.4.3, the cost of the constructed MGTSP solution is equal to the cost of the TSP solution. Hence, $\sum_{e \in E_{\Upsilon^3}} c^3(e) = \sum_{e \in E_{\Upsilon^2}} c^2(e)$.

3.4.3 Reconstructing the MGTSP Solution

The transformed graph G^3 defined in Problem (P3) can now be used to compute the TSP solution using a variety of freely and commercially available TSP solvers. The experimental simulations in this work use the LKH solver based on the Lin-Kernighan Helsgaun heuristic to solve TSP instances. .

Given the optimal solution Υ^3 to Problem (P3), we can construct, Υ^2, the optimal solution to Problem (P2) as follows. Find the indices of all the finish-depot vertices used in Υ^3. If the indices are $\{l_1, l_2, \ldots, l_M\}$, pick the vertices immediately following them in the tour as $\{l_1+1, l_2+1, \ldots, l_M+1\}$. These are the start-depots of each individual path. Between every pair of start-depot and finish-depot indices (l_i+1, l_{i+1}), use the Noon-Bean method to select vertices for each set in $\{V_1^2, \ldots, V_R^2\}$, as described in Section 3.4.1. The

MGTSP solution to (P2) can be constructed from the TSP solution to (P3) only if Lemma 3.4.3 is satisfied.

Combining the methods described by Noon and Bean and the lemmas stated in this section, we can transform an MGTSP to a TSP, solve it, and use the TSP solution to re-construct the original MGTSP solution. Our final result can be formally stated as the *Modified Noon-Bean Theorem.*

Theorem 3.4.4 (Modified Noon-Bean Theorem)**.** *Given a MGTSP in the form of Problem (P2) with R vertex sets and M depots, we can transform the problem into a TSP in the form of Problem (P3). Given a solution Υ^3 to Problem (P3), we can construct a corresponding solution Υ^2 to Problem (P2) if $\sum_{e \in E_{\Upsilon^3}} c^3(e) < (R+2)\beta$.*

3.5 Extending the Planning Horizon

To this point, we have restricted our problem formulation to a single recharge per working robot. For persistent surveillance tasks, however, it is necessary to consider multiple recharging events per working robot to maintain functionality over longer planning horizons. One approach is to formulate the recharging problem as a path optimization over the entire planning horizon, known as a *fixed horizon* plan. Computing an optimal fixed horizon path may be intractable for larger problem sizes and an alternative approach to reduce computational effort is to consider an iterative computation of the single recharge cycle plan over a receding planning horizon. We first formulate the fixed horizon plan as a MILP to generate an optimal recharge schedule over a finite planning horizon and then present the *receding horizon* planning approach as an extension of the single recharge cycle problem.

3.5.1 Optimal Periodic Recharging

The fixed horizon approach to path planning involves computing an optimal path over the entire planning horizon. This approach, although significantly increasing the size of the problem, guarantees optimality of rendezvous paths over the lifetime of the mission. In this section, we formally state the *optimal periodic recharging* problem and present a MILP solution, which extends the approach in Section 3.3 for a single recharge cycle.

As in the single recharge cycle computation, in the periodic recharging problem, working robot trajectories are known for the entire planning range $[0, T]$. However, the objective is now to compute charging robot paths that rendezvous with working robots at a sequence of charging points such that no working robot runs out of charge over the planning range. Our approach to the problem is as follows.

Approach

Three main factors distinguish the periodic charging problem from the single charge cycle problem:

(i) Arrival times of working robots at charging points cannot be determined a-priori, since they depend on previous rendezvous in their paths.

(ii) The time elapsed between consecutive recharges of each robot must be constrained to ensure successful continued operation.

(iii) The variability of arrival times at charging points implies that the feasibility condition applied on a path between them, as defined in Equation 3.2, cannot be predetermined.

Given these considerations, we formulate the periodic charging problem as an optimization on a partitioned graph, G, for a set of working robots $\mathcal{R}$ and a set of charging robots $\mathcal{M}$ in an environment $\mathcal{E} \subset \mathbb{R}^2$. The graph G is defined as follows.

Vertices: Define a set of vertices V that is partitioned into $R+1$ disjoint vertex sets, $V_0, V_1, \ldots, V_R$. The set V_0 is the set of start-depots of the charging robots. The vertices in each set V_r, $r \in \mathcal{R}$, correspond to charging locations in C_r.

The charging point set for periodic charging, $C_r = \{(p_r(t)|t \in T\}$ for robot $r \in \mathcal{R}$ is defined as the set of locations $p_r(t)$ that a robot would visit along its trajectory, given infinite charge and no recharge stops. The estimated arrival times at the charging points will be updated as part of the optimization.

Edges: Edge-feasibility is subject to change based on UAV arrival times at charging points. Hence define all edges (v_i, v_j) where $v_i \in V_a$, and $v_j \in V_b$ for $a, b \in \mathcal{R}$ and $a \neq b$ as valid edges in the periodic charging graph. The edge-feasibility condition will be applied as a constraint in the optimization.

Costs: The cost on an edge can be defined based on the optimization objective. In this formulation we consider the distance between two charging locations.

In addition to the graph G, we introduce two sets of variables, $y_{r,i}$ and $t_{r,i}$. The variable, $y_{r,i} \in \mathbb{R}_+$, stores the value of the time elapsed since the last recharge of robot r, at each

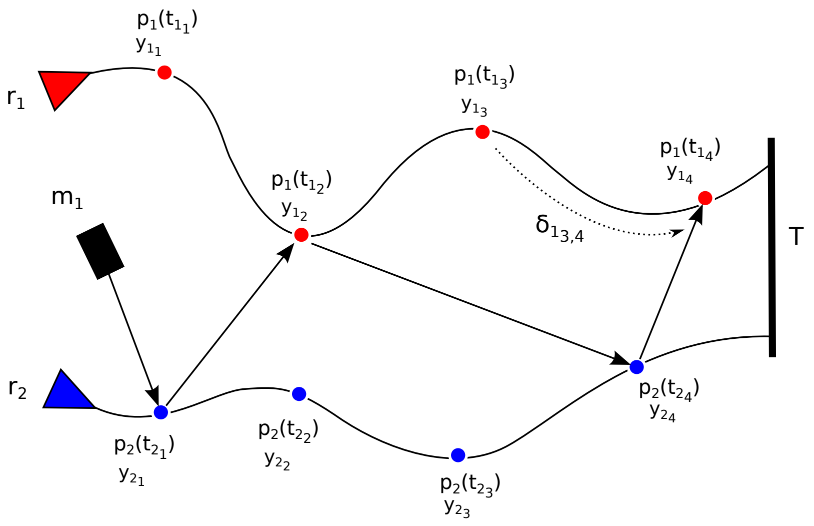


Figure 3.7: The periodic MILP representation: An sample problem instance illustrating charging point discretization and key variables. A path for charging robot m_1 is computed to visit charging point sets V_{r_1} and V_{r_2}, for robots r_1 and r_2, periodically to ensure $y_{r,i} < \tau_r$.

charging point i in C_r. By placing a bound, τ_r, on the maximum value of $y_{r,i}$, we can ensure that robot r will always rendezvous with a charging robot before it is completely discharged. The variable, $t_{r,i} \in \mathbb{R}_+$, computes the time of arrival of a UAV r at its charging point i in set C_r. The value of $t_{r,i}$ is computed at each point taking into account the service times at charging points chosen for rendezvous.

A sample instance of the discretized problem for optimal periodic charging is shown in Figure 3.7.

We can now formally state the optimal periodic recharging problem.

Problem 3.5.1 (Optimal Periodic Charging Problem)**.** Consider a partitioned DAG G with the partition $(V_0, V_1, \ldots, V_R)$ of V where $V_0 = \{v_m | m \in \mathcal{M}\}$. Find a set of paths $P = \{P_1, \ldots, P_M\}$ in G that minimize $\sum_{i=1}^{M} \sum_{e \in E_{P_i}} c(e)$ and satisfy the constraints (i) $P_m \in P$ starts at $v_m \in V_0$, such that $|V_r \cap V_P| \geq 1$ for all $r \in \mathcal{R}$ and (ii) $y_{r,i} < \tau_r$ for all $r \in \mathcal{R}$ and all $v_i \in V_r$.

Periodic MILP Formulation

Given the problem statement, the periodic charging MILP can be defined as an extension to the single charge cycle MILP defined in Section 3.3. For ease of presentation, the

MILP is formulated to compute a single charging robot path through a team of UAVs performing a persistent task. The extension to multiple robots is straightforward as shown in Section 3.3.2.

The periodic charging MILP refers to a vertex v_i with an index i in the context of the complete vertex list V, as well as an index within each working robot vertex set V_r. Hence, to avoid ambiguities in the indices, we define the set of vertex indices of V_r as $I_{V_r} = \{1, \ldots, |V_r|\}$, and the set of vertex indices of V as $I_V = \{1, \ldots, N\}$. Finally, we define the index function $\sigma : \mathcal{R} \times I_{V_r} \to I_V$ as a function that takes a working robot index $r \in \mathcal{R}$ and the local index of the charging vertex $i \in I_{V_r}$ and returns the global index of the vertex in the complete vertex list I_V.

The objective of the periodic charging problem, as inherited from the single recharge cycle MILP, is to minimize the total sum of path costs of the charging robots.

$$\min \quad \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{3.11}$$

The constraints of the periodic optimization inherit degree and flow constraints of the One-in-a-set DAG path problem and extend the problem definition to fulfill periodic charging. Constraints (3.5), (3.6) and (3.9) are inherited directly. Set degree constraints (3.8) and (3.9) are modified to form Constraints (3.12) and (3.13) to allow multiple recharge rendezvous within each set V_r of a robot r:

$$\sum_{j \in V_r} \sum_{i \in V} x_{ij} \geq 1, \quad \forall r \in \mathcal{R} \tag{3.12}$$

$$\sum_{i \in V_r} \sum_{j \in V} x_{ij} \geq 1, \quad \forall r \in \mathcal{R} \tag{3.13}$$

Constraint (3.14) computes the value of $t_{r,i}$, the arrival time at each charging point, as the sum of the arrival time at the previous charging point, $t_{r,i-1}$, the service time s_r at the point if a recharge has taken place, and the travel time, $\delta_{r_{i-1,i}}$, between two consecutive charging points.

$$\begin{aligned} t_{r,i} = t_{r,i-1} + s_r \sum_{j \in V} x_{j\sigma(r,i-1)} + \delta_{r_{i-1,i}}, \\ \forall r \in \mathcal{R} \quad \forall i \in I_{V_r} \end{aligned} \tag{3.14}$$

Given the value for $t_{r,i}$ at each charging point, an edge feasibility constraint for every edge in the graph can now be defined. Constraint (3.15) is defined as a logical or implication constraint which ensures that if the value of $x_{\sigma(r_1,i)\sigma(r_2,j)} = 1$, signifying an active edge in the solution, then the feasibility constraint as shown in constraint (3.15) must be satisfied. Logic constraints can be reformulated into MILP constraints using linear relaxations and *big-M* formulations as shown in [21].

$$\begin{gathered} x_{\sigma(r_1,i)\sigma(r_2,j)} = 1 \implies t_{r_2,j} - t_{r_1,i} > \frac{d_{\sigma(r_1,i)\sigma(r_2,j)}}{\upsilon} \\ \forall r_1, r_2 \in \mathcal{R};\ r_1 \neq r_2 \quad \forall i \in I_{V_{r_1}} \quad \forall j \in I_{V_{r_2}} \end{gathered} \tag{3.15}$$

Note that $d_{\sigma(r_1,i)\sigma(r_2,j)}$ is the distance between the two charging points. The final three constraints (3.16), (3.17) and (3.18) compute the value of $y_{r,i}$ and ensure that it is bounded by τ_r. Constraint (3.16) computes the value of $y_{r,i}$ at every charging point, where a rendezvous does not occur, as the sum of $y_{r,i-1}$ and $\delta_{r_{i-1,i}}$.

$$\begin{gathered} \sum_{j \in V} x_{j\sigma(r,i)} = 0 \implies y_{r,i} - y_{r,i-1} = \delta_{r_{i-1,i}} \\ \forall r \in \mathcal{R} \quad \forall i \in I_{V_r} \end{gathered} \tag{3.16}$$

Constraint (3.17) resets the value of $y_{r,i}$ to 0 at a charging point chosen for rendezvous. Thus the value of $y_{r,i}$ increments throughout the charging point set, occasionally resetting to 0 at points where recharge rendezvous occur.

$$\begin{gathered} \sum_{j \in V} x_{j\sigma(r,i)} = 1 \implies y_{r,i} = 0 \\ \forall r \in \mathcal{R} \quad \forall i \in I_{V_r} \end{gathered} \tag{3.17}$$

Finally Constraint (3.18) limits the growth of $y_{r,i}$ to guarantee that robot r is consistently charged through the mission.

$$0 \leq y_{r,i} \leq \tau_r, \quad \forall r \in \mathcal{R} \quad \forall i \in I_{V_r} \tag{3.18}$$

The full MILP formulation block can be found in Appendix A.

The total number of constraints in this formulation is $2M + 2R + N^2 + 5N$, of which $N^2 + N$ are implication constraints. Similar to the single recharge cycle MILP defined in Section 3.3, the complexity of the fixed horizon problem is influenced by the number of vertices in the graph G. For a given scenario of working and charging robots, the size of the G grows with the length of the planning horizon $[0, T]$ and the density of charging locations on each robot path.

This formulation produces a significantly larger constraint set than the single recharge cycle MILP and as a result, the fixed horizon MILP quickly becomes intractable for larger problem instances. An alternative approach to minimize computational effort is a receding horizon strategy.

3.5.2 Receding Horizon Planning

Receding horizon methods have been extensively applied to MILP based motion planning [4] to minimize computational effort and enhance robustness of the computed path. In a general receding horizon formulation, a path plan is computed and implemented over a shorter time window and then iteratively updated from the state reached at each planning event, for the duration of the planning horizon.

In the persistent recharging scenario, we maintain a set of R working robots waiting to be charged. At each planning iteration, a time horizon $[0, T_r]$ is defined using the current state of the robots as their starting state. A One-in-a-set DAG problem is defined over the R working robots and a set of rendezvous paths for charging robots are computed using the single recharge cycle method of either Section 3.3 or Section 3.4. Each time a working robot is charged (i.e., a rendezvous occurs), we have an option to re-plan. Thus, given a team of R working robots, the receding horizon window can be varied from 1 to R rendezvous between re-plans.

The size of this window influences the quality of solutions generated. A larger planning window results in fewer planning iterations, a lower cumulative computation time and a lower total path cost per recharge cycle. However, it suffers from a greater possibility that a subsequent planning iteration will produce an infeasible path problem. A smaller planning window, while producing a more myopic path and a larger number of planning iterations, is more robust to uncertainties and is less likely to reach an infeasible solution.

In Section 3.6.2, we examine the effects of the planning window on the performance, and compare this method to the multiple charge MILP.

3.6 Simulation Results

The optimization framework for this work was implemented and tested in simulated experiments generated in MATLAB®. The mixed integer linear programs were solved optimally using the IBM CPLEX® solver and the TSP heuristic used in the computation was the freely available LKH Solver [20]. The solutions were computed on a laptop computer running a 32 bit Ubuntu 12.04 operating system with a 2.53 GHz *Intel Core2 Duo* processor and 4GB of RAM.

The simulation environment consists of a test set of planar trajectories that are assigned to a team of R working robots. Each working robot r is defined by its assigned trajectory, current pose, voltage level and battery lifetime T_r. The environment also contains a set of M randomly located charging robots, each defined by an initial position and a maximum velocity, v. The goal is to enable the working robots to persistently traverse their assigned trajectories for the duration of mission.

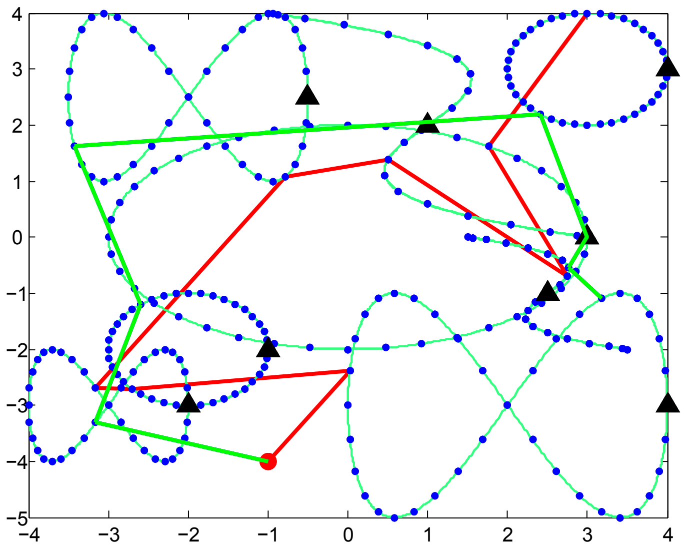
Using these simulations, we benchmark rendezvous path solutions and examine the performance of the receding horizon and fixed horizon strategies for persistent recharging.

3.6.1 Single Recharge Cycle Path Computation

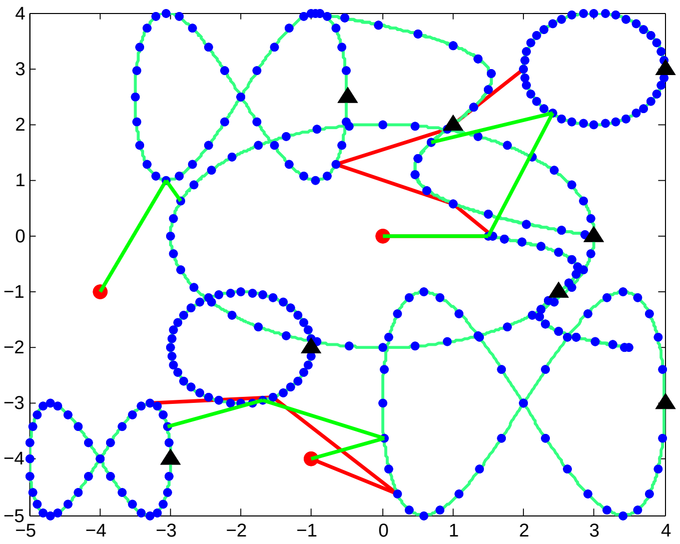
The recharge path is algorithmically computed using the Noon-Bean Transformation in the case of a single charging robot and the Modified Noon-Bean Transformation for multiple charging robots.

Figure 3.8(a) illustrates a sample problem instance with 8 working robots distributed along 8 paths and 1 charging robot. The result compares the heuristic solution obtained with the Noon-Bean transformation and LKH solver against the optimal MILP solution computed with CPLEX, for a single charging robot. The generated DAG contains 500 vertices. The optimal solution for a single charging robot was computed by CPLEX in 102 seconds. The heuristic solution was computed by LKH in 2 seconds and resulted in a path cost 12.8% higher than the optimal cost.

Figure 3.8(b) presents a sample problem instance with eight working robots distributed among eight paths and 3 charging robots. The result compares the heuristic solution obtained with the Modified Noon-Bean transformation followed by the LKH solver against the optimal MILP solution for multiple charging robots. The DAG consists of 500 vertices. The optimal solution was computed by CPLEX in 97 seconds. The heuristic solution was computed by LKH in 1.2 seconds and resulted in a path cost 7.8% higher than the optimal cost.



(a) Comparison of the optimal CPLEX solution (light grey/green path) against the Noon bean Transform and LKH Heuristic solution (dark grey/red path). The problem consists of 8 working robots (triangles) on 8 paths.



(b) Comparison of the optimal CPLEX solution (light grey/green path) against the Modified Noon bean Transform and LKH Heuristic solution (dark grey/red path). The problem consists of 8 working robots (triangles) on 8 paths and 3 charging robots.

Figure 3.8: Rendezvous path computation for a single recharge cycle.

The results demonstrate that the performance of the Modified Noon-Bean transformation closely matches the original Noon-Bean method as an algorithmic strategy in comparison with the optimal MILP solutions.

To further benchmark the performance of the LKH heuristic against the optimal CPLEX solution, we conducted an experiment to examine the effect of growth in problem complexity on the runtime and solution quality for both solvers.

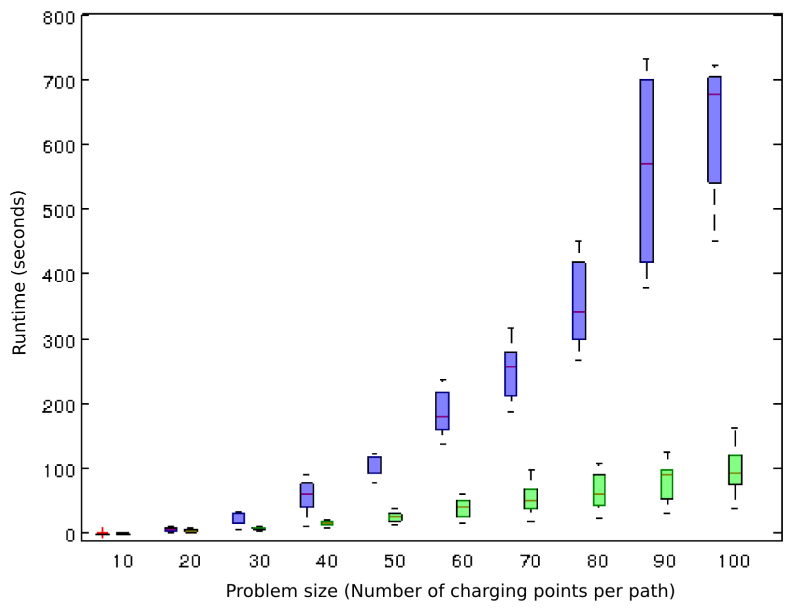
A test set of simulation environments with different path and robot configurations was created. Given each environment configuration, the complexity of the path optimization was varied by incrementing the density of charging points along each working robot trajectory from $\{10, 20, 30, \ldots, 100\}$ charging points per path. The recharge path was computed several times for each charging point density level. The resulting runtimes and path lengths for each environment are normalized to show trends in performance with growth in problem complexity. The results are summarized Figure 3.8 using box plots that show the spread of results over each charging point density level, using quartiles (box edges), extreme data points (whiskers) and outliers (crosses). Boxes for the optimal and heuristic solutions are plotted adjacently for each x-axis data point.

Figure 3.9(a) demonstrates the the growth in runtime for the optimal CPLEX solution and the LKH heuristic solution with a growth in problem complexity. Similarly, Figure 3.9(b) demonstrates the trend in path costs for the optimal and heuristic solutions. The optimal path cost for each simulation environment is generally consistent for all problem sizes since the complexity is varied by only increasing the number of charging points per path. The results show that, on average, as problem complexity grows, the optimal solver grows exponentially in runtime and the heuristic solver consistently provides solutions within 10% of the optimal with significant savings in computational effort.

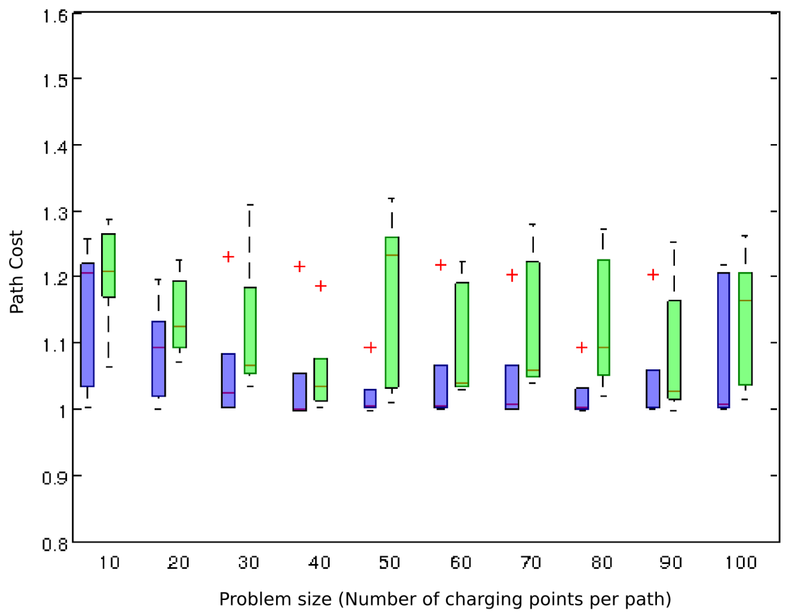
Finally, since an optimal MILP solution is not computationally tractable for large problem sizes an extremely large problem was solved as a performance benchmark. The environment consists of sixteen working robots, evenly distributed among eight paths and one charging robot. The resulting graph for the TSP solution contains 5761 vertices. The LKH solver found a TSP solution in 368 seconds. The optimal MILP solver was not able to compute an solution within a reasonable time frame.

3.6.2 Recharging in Extended Planning Horizons

The following simulation experiments examine the receding horizon and fixed horizon methods of computing recharge paths over an extended planning horizon. For appropriate benchmarking, the receding horizon strategy is implemented by computing the optimal



(a) Runtime comparison: Optimal CPLEX (blue) vs. LKH heuristic (green).



(b) Cost comparison: Optimal CPLEX (blue) vs. LKH heuristic (green).

Figure 3.9: Performance comparison of Optimal CPLEX and LKH TSP heuristic solutions

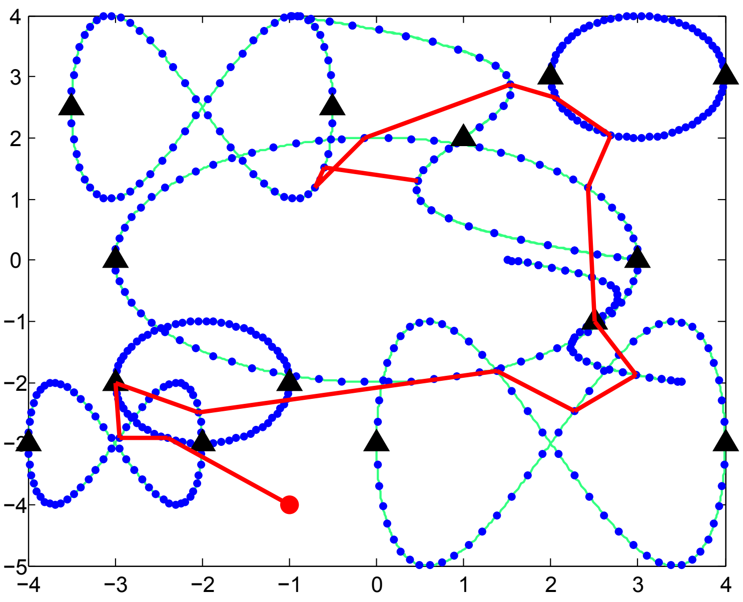


Figure 3.10: The heuristic solution(computed using LKH) to the single charging robot problem with sixteen working robots distributed among eight paths.

MILP solution at each planning iteration and compared with the optimal fixed horizon path over the planning horizon.

The computational effort required by the receding horizon method is significantly less than the fixed horizon strategy due to a shorter planning window and a much smaller MILP formulation. For the same reason, however, global optimality is not guaranteed over the entire planning horizon. To investigate this trade-off, we conducted an experiment, similar to the single recharge cycle tests, to examine the effect of growth in problem complexity on the runtime and solution quality for both methods.

A test set of simulation environments with different path and robot configurations was created. For each simulated environment, the recharge path was computed using both the receding and fixed horizon methods for a set of different planning horizons from $\{10, 15, 20, \ldots, 40\}$ minutes assuming the estimated lifetime of each working robot to be 6 minutes. For all receding horizon simulations, the replanning window was chosen to be $R/2$ rendezvous per iteration. The optimization of each planning strategy was aborted if a solution was not found in 1000 seconds. The aggregate results for cumulative runtime and total path cost are summarized in Tables 3.1 and 3.2 due to the large differences in results between the two strategies.

Table 3.1: Fixed horizon runtimes

Horizon (minutes)	Runtime Quartiles (seconds)		
	25%	Median	75%
10	0.12	0.33	0.42
15	5.34	10.23	14.04
20	12.65	21.14	45.87
25	91.71	200.14	500.32
30	254.03	401.55	801.39
35	712.28	900.61	+1000
40	968.75	+1000	+1000

Table 3.2: Receding horizon runtimes

Horizon (minutes)	Runtime Quartiles (seconds)		
	25%	Median	75%
10	0.11	0.14	0.18
15	0.12	0.15	0.20
20	0.16	0.20	0.30
25	0.21	0.28	0.38
30	0.25	0.32	0.49
35	0.31	0.43	0.58
40	0.37	0.54	0.68

Tables 3.1 and 3.2 demonstrate the spread in the growth of runtime for the fixed horizon strategy and the receding horizon strategy respectively with a growth in planning horizon, using quartiles, similar to the box plots. For each planning horizon, the 25th percentile, 75th percentile and median of the runtime results are shown. Figure 3.11 compares the normalized path costs for both methods with a box plot.

The results show that, on average, as problem complexity grows, the growth in runtime for the fixed horizon solver is exponential with a wide spread of growth rates based on the problem configuration. On the contrary, the receding horizon strategy consistently results in a significantly smaller cumulative runtime even with an optimal MILP solution at each iteration. On average, the receding horizon method is seen to produce solutions with a total path cost within 20% of the optimal fixed horizon solution.

Next, we investigate the effect of varying the planing window size on the receding

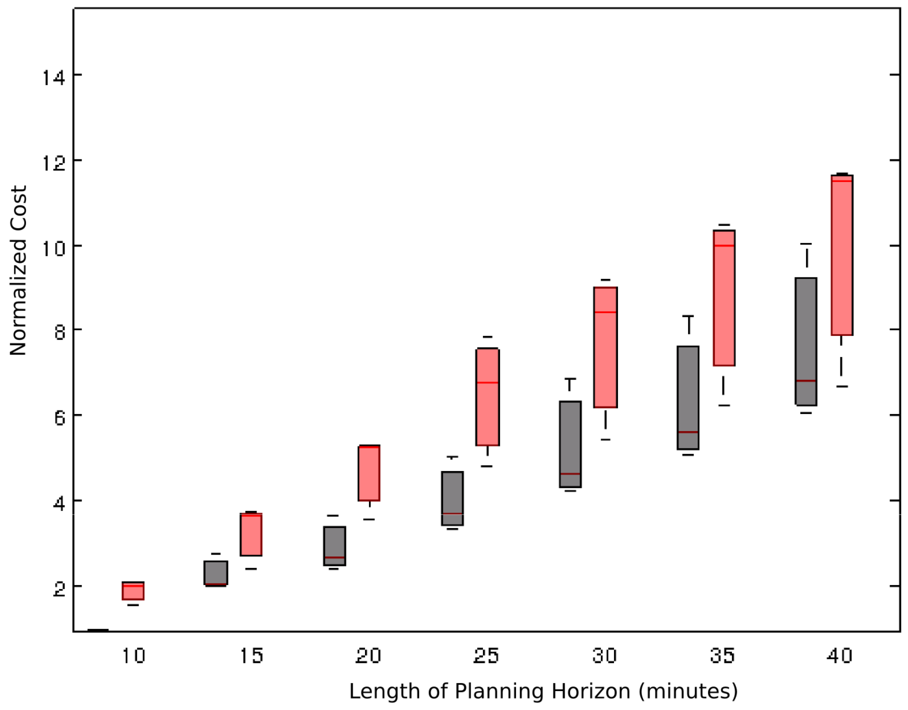


Figure 3.11: Total path cost: Fixed horizon (dark/black) and Receding horizon (light/red)

Table 3.3: Receding horizon cumulative runtime

Planning window	1	2	4	6	8
Runtime (seconds)	6.14	3.49	2.03	1.41	infeasible

Table 3.4: Receding horizon total path cost

Planning window	1	2	4	6	8
Total Path Cost	1.22	1.01	1.23	1.02	infeasible

horizon strategy. For a set of 8 working robots and 3 charging robots, a test set of simulation environments with different path configurations and charging point densities was generated. For each environment, the planning window was varied from 1 rendezvous to 8 rendezvous and the receding horizon solution was computed for each window size. Tables 3.3 and 3.4 show the normalized results of cumulative runtime and path cost, respectively, averaged over all the experiments.

It is important to note that in the presented receding horizon strategy, at each iteration, regardless of planning window, the optimal recharge path is computed to visit all working robots. Hence, Table 3.3 shows that the cumulative runtime generally drops as the planning window grows, due to fewer replanning iterations. However, a larger planning window also increases the possibility of the path reaching an infeasible solution as seen with the planning window of 8 rendezvous per iteration. Table 3.4 shows that the cumulative path cost over the planning horizon is not significantly affected by the size of the planning window.

Chapter 4

Conclusions and Future Directions

This work addresses the motion planning problem as a component of autonomy in mobile robots, given a fully or partially known map of the environment and an estimate of the robot pose. The complexity of motion planning tends to grow exponentially with problem size in decision based planning problems such as coverage and exploration due to the nature of the optimization. To mitigate this complexity, the discrete planning approach involves simplifying the solution space by discretizing the environment and using integer programs on a graph to compute shortest paths. The graph-based optimization structure lends itself well to the development of novel heuristics and approximation algorithms to compute near-optimal solutions with minimal computational effort. This thesis investigates two motion planning problems in mobile robotics, namely coverage and persistent surveillance, and presents contributions in each area under the overarching framework of discrete planning.

The coverage path planning approach in this work presents two major optimization problems to generate a minimum length path that guarantees search coverage over a bounded environment. The map decomposition algorithm presents a novel application of the greedy cut approach on a 2D drivability map of the environment, which is demonstrated to consistently produce solutions which outperform other well known decomposition methods in terms of minimizing the number of convex regions produced. The path generation algorithm computes search paths that are, on average, 25% shorter than algorithms presented in the literature that do not use decomposition methods. Further, the presented planning approach provides a complete global path which can also be used to calculate the total required mission time. Knowledge of the mission provides a significant advantage over random search approaches such as the Frontier Exploration methods, for which no reliable predictions on path length can be made. Experiments illustrated that the algorithm was capable of coverage planning on test areas between 10,000 and 20,000 m^2 in under 3 seconds. Given the speed of the vehicle and the desire to only re-plan after searching each convex sector, this update rate is sufficient for online coverage planning applications. The integrated approach is shown to be successful in a sample return application, and is able to achieve real-time performance running on-board our test vehicle.

Most coverage approaches in existing literature consider theoretical environments with idealistic assumptions like fully known maps, perfect localization or simple indoor environments such as floor cleaning applications. This thesis attempts to solve the complete coverage problem in real outdoor environments in which the map is unknown and gradually updated online as planning is executed. The presented approach is robust and dynamic but guaranteeing complete coverage in such applications requires a more rigorous treatment of map and pose uncertainty. For example, if the map changes over time when loop closure is used to correct global consistency, the path planner may incorrectly mark areas as obstacles or leave out small sections of the environment uncovered. Further, noise in the robot pose estimate, can render the planner unable compute a path due to an infeasible initial state. Given that the map and pose are the only two inputs to the planner it is difficult to correct errors unless additional information from a second local mapping sensor or an improved vehicle motion model are incorporated into the motion planner to enhance performance.

This thesis also presents the problem of persistent recharging with coordinated teams of autonomous robots. The One-in-a-set DAG problem is proved to be NP-hard and the MILP formulation for a single recharge cycle is presented. A solution approach is developed, using the Noon-Bean transformation to obtain a TSP problem instance which can be solved with a TSP heuristic solver. Subsequently, a novel modification to the Noon-Bean transformation is proposed to address the MGTSP case and find multiple rendezvous paths for a team of charging robots. Simulation results show that the heuristic solution using the Modified Noon-Bean transformation and LKH solver is a viable alternative that produces solutions of comparable cost and significant runtime savings. Finally, the problem is extended to longer planning horizons using a receding horizon and fixed horizon approach. Simulations demonstrate the trade-off between optimality and computational complexity presented by the two alternatives.

The main challenge faced by the receding horizon approach is ensuring that each subsequent planning iteration admits a feasible solution. One way to mitigate this issue is to incorporate terminal constraints for each planning iteration to ensure continued feasibility of path solutions [15]. Implementing safety constraints in the MILP formulation is a future direction for this work. In the fixed horizon strategy, in addition to high computational complexity, another drawback is poor robustness to uncertainties or modelling errors. Since the computation is performed offline, this strategy does not adapt the charging schedule to incorporate disturbances and mistiming errors in the execution of the optimal plan. However, robustness strategies such as *reactive rescheduling* [36] may be used to make it an effective planning strategy. Robustness of optimal path plans is another potential direction for this work.

In the future, the foundation laid in this work can be built upon in a number of interesting research directions. Especially in problems involving energy awareness and recharging it is justified to assume that owing to uncertainties in the environment and vehicle dynamics, some robots may fail during the course of a long-term mission. In such cases the motion planners must ideally be fault tolerant and make up for the loss of an agent by allowing other robots to compensate for surveillance blind spots, or possibly summon a spare agent to substitute it. Fault tolerant motion planners could be a useful direction to take this work.

While the problems explored in this thesis have largely been treated as two separate application domains of discrete planning, they have the potential to compliment each other in the case of autonomous exploration, surveillance or coverage with added energy constraints. For instance, the coverage problem could be augmented to formulate multi-robot UAV or UGV coverage with a separate team of service robots managing the overall health of the working robot team. In general the recharging problem could be combined with the path planning problem to generate optimal, yet energy aware motion plans for exploration problems like mapping or maximizing information gain about an environment. Further given a set of deterministic or stochastically arriving locations to survey, a team of UAVs could potentially be enabled to generate distributed path plans to consistently visit regions of interest and periodically schedule optimal rendezvous for recharging.

While a significant body of robotics research is dedicated to navigation, coverage and exploration, implementing robust long-term autonomous missions still remains a challenging problem for the robotics community. A complete solution for long-term autonomy necessitates the simultaneous treatment of mission specific planning as well as energy awareness and recharging strategies to enable continuous operation. The methods presented in this work are a step in this direction and coupled with advancements in robotic perception and control, will significantly influence numerous application domains with a pervasive technological impact.

APPENDICES

Appendix A

Optimal Periodic Charging MILP

A.1 Notation

Functions

$\sigma : \mathcal{R} \times I_{V_r} \to I_V$:
The function takes a working robot index $r \in \mathcal{R}$ and the local index of the charging vertex $i \in I_{V_r}$ and returns the global index of the vertex in the complete vertex list I_V.

Decision Variables

x_{ij} **(Binary):**
Define $x_{ij} = 1$ if, in the solution path, the vertex v_j is visited after vertex v_i, where $i \in V_{r_1}, j \in V_{r_2}, r_1 \neq r_2$ and $r_1, r_2 \in \{1, \dots, R\}$.

$y_{r,i}$ **(+ve Real):**
For robot r, at a charging vertex indexed by $i \in I_{V_r}$, $y_{r,i}$ is the time elapsed since its last recharge.

$t_{r,i}$ **(+ve Real):**
$t_{r,i}$ is the time at which robot r arrives at the charging vertex indexed by $i \in I_{V_r}$.

A.2 MILP Formulation

For each charging robot m, index d represents the initial position of the charging robot, $p_m(0)$. The solution paths must end at the dummy vertex denoted by index f.

$$\min \quad \sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij} \tag{A.1}$$

subject to

$$\sum_{j\in V\setminus V_0} x_{dj} = 1 \tag{A.2}$$

$$\sum_{i\in V\setminus V_0} x_{if} = 1 \tag{A.3}$$

$$\sum_{j\in V_r}\sum_{i\in V} x_{ij} \geq 1 \qquad \forall r \in \mathcal{R} \tag{A.4}$$

$$\sum_{i\in V_r}\sum_{j\in V} x_{ij} \geq 1 \qquad \forall r \in \mathcal{R} \tag{A.5}$$

$$t_{r,i} = t_{r,i-1} + s_r \sum_{j\in V} x_{j\sigma(r,i-1)} + \delta_{r_{i-1,i}} \qquad \forall r \in \mathcal{R} \quad \forall i \in I_{V_r} \tag{A.6}$$

$$x_{\sigma(r_1,i)\sigma(r_2,j)} = 1 \implies t_{r_2,j} - t_{r_1,i} > \frac{d_{\sigma(r_1,i)\sigma(r_2,j)}}{\upsilon} \qquad \forall r_1, r_2 \in \mathcal{R} \quad r_1 \neq r_2 \quad \forall i \in I_{V_{r_1}} \quad \forall j \in I_{V_{r_2}} \tag{A.7}$$

$$\sum_{j\in V} x_{j\sigma(r,i)} = 0 \implies y_{r,i} - y_{r,i-1} = \delta_{r_{i-1,i}} \qquad \forall r \in \mathcal{R} \quad \forall i \in I_{V_r} \tag{A.8}$$

$$\sum_{j\in V} x_{j\sigma(r,i)} = 1 \implies y_{r,i} = 0 \qquad \forall r \in \mathcal{R} \quad \forall i \in I_{V_r} \tag{A.9}$$

$$0 \leq y_{r,i} \leq \tau_r \qquad \forall r \in \mathcal{R} \quad \forall i \in I_{V_r} \tag{A.10}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V \tag{A.11}$$

References

[1] D. Applegate, R. Bixby, V. Chvtal, and W. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, New Jersey, USA, 2011.

[2] E.M. Arkin, M. Held, and C.L. Smith. Optimization problems related to zigzag pocket machining. *Algorithmica*, 26(2):197–236, 2000.

[3] J. Barraquand, B. Langlois, and J.C. Latombe. Numerical potential field techniques for robot path planning. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(2):224–241, 1992.

[4] J. Bellingham, A. Richards, and How J.P. Receding horizon control of autonomous aerial vehicles. In *in Proceedings of the American Control Conference*, pages 3741–3746, 2002.

[5] D.W. Casbeer, R.W. Beard, T.W. McLain, Sai-Ming Li, and R.K. Mehra. Forest fire monitoring with multiple small UAVs. In *American Control Conference, 2005. Proceedings of the 2005*, pages 3530–3535 vol. 5, 2005.

[6] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C. K. Yap, editors, *Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[7] B. Chazelle and D. Dobkin. Decomposing a polygon into its convex parts. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, STOC '79, pages 38–48, New York, NY, USA, 1979. ACM.

[8] L.P. Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1):97 – 108, 1989.

[9] H. M Choset. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT press, 2005.

[10] J. Corrales, Y. Madrigal, D. Pieri, G. Bland, T. Miles, and M. Fladeland. Volcano monitoring with small unmanned aerial systems. In *American Institute of Aeronautics and Astronautics Infotech Aerospace Conference*, page 2522, 2012.

[11] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243 – 255, 2004.

[12] M. de Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[13] J. Derenick, N. Michael, and V. Kumar. Energy-aware coverage control with docking for robot teams. In *IEEE International Conference on Intelligent Robots and Systems*, pages 3667–3672, 2011.

[14] M. Dunbabin and L. Marques. Robots for environmental monitoring: Significant advancements and applications. *Robotics & Automation Magazine, IEEE*, 19(1):24–39, 2012.

[15] M.G. Earl and R. D'Andrea. Iterative MILP methods for vehicle-control problems. *Robotics, IEEE Transactions on*, 21(6):1158–1167, 2005.

[16] D. Eppstein. Finding the k shortest paths. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 154 –165, 1994.

[17] A. Goel and V. Gruhn. A general vehicle routing problem. *European Journal of Operational Research*, 191(3):650–660, 2008.

[18] D. H. Greene. The decomposition of polygons into convex parts. *Computational Geometry, Advances in Computing Research*, 1:235–259, 1983.

[19] E. Guizzo. Japan earthquake: Robots help search for survivors. http://spectrum.ieee.org/automaton/robotics/industrial-robots/japan-earthquake-robots-help-search-for-survivors, 2011.

[20] K. Helsgaun. General k-opt submoves for the Linkernighan TSP heuristic. *Mathematical Programming Computation*, 1:119–163, 2009.

[21] J.N. Hooker and M.A. Osorio. Mixed logical-linear programming. *Discrete Applied Mathematics*, 9697(0):395 – 442, 1999.

[22] D. Karapetyan and G. Gutin. Linkernighan heuristic adaptations for the generalized traveling salesman problem. *European Journal of Operational Research*, 208(3):221 – 232, 2011.

[23] L.E. Kavraki and J. Latombe. Probabilistic roadmaps for robot path planning, 1998.

[24] J. M. Keil. Decomposing a polygon into simpler components. *SIAM J. Comput.*, 14(4):799–817, 1985.

[25] J. M. Keil. *Handbook of Computational Geometry*, volume 2, chapter Polygon decomposition. North Holland, 2000.

[26] D. Kingston, R.W. Beard, and R.S. Holt. Decentralized perimeter surveillance using a team of UAVs. *Robotics, IEEE Transactions on*, 24(6):1394–1404, 2008.

[27] K. Konolige, J. Augenbraun, N. Donaldson, C. Fiebig, and P. Shah. A low-cost laser distance sensor. In *IEEE International Conference on Robotics and Automation*, pages 3002–3008, 2008.

[28] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithmics and Combinatorics*. Springer, 4 edition, 2007.

[29] G. Laporte, H. Mercure, and Y. Nobert. An exact algorithm for the asymmetrical capacitated vehicle routing problem. *Networks*, 16(1):33–46, 1986.

[30] S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[31] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):pp. 498–516, 1973.

[32] Y. Litus, R.T. Vaughan, and P. Zebrowski. The frugal feeding problem: Energy efficient, multi-robot, multi-place rendezvous. In *IEEE International Conference on Robotics and Automation*, pages 27–32, 2007.

[33] Y. Litus, P. Zebrowski, and R.T. Vaughan. A distributed heuristic for energy-efficient multirobot multiplace rendezvous. *IEEE Transactions on Robotics*, 25(1):130 –135, 2009.

[34] Y. Liu and G. Nejat. Robotic urban search and rescue: A survey from the control perspective. *Journal of Intelligent and Robotic Systems*, 2013.

[35] N. Mathew, S.L. Smith, and S. Waslander. A graph based approach to multi-robot rendezvous for recharging in persistent tasks. In *IEEE International Conference on Robotics and Automation*, 2013.

[36] C. Mendez and J. Cerd. An MILP framework for batch reactive scheduling with limited discrete resources. *Computers & Chemical Engineering*, 28(67):1059 – 1068, 2004. FOCAPO 2003 Special issue.

[37] N. Michael, E. Stump, and K. Mohta. Persistent surveillance with a team of MAVs. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2708–2714, 2011.

[38] B. M. E. Moret, M. J. Collins, J. Saia, and Y. Ling. Ice rinks and cruise missiles: sweeping a simple polygon. In *Workshop on Algorithm Engineering (WAE)*, Venice, Italy, 1997.

[39] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen. Search and rescue robotics. In B. Siciliano and K. Oussama, editors, *Springer Handbook of Robotics*, pages 1151–1173. Springer Verlag, 2008.

[40] R.R. Murphy. Trial by fire [rescue robots]. *Robotics Automation Magazine, IEEE*, 11(3):50–61, 2004.

[41] V. Nguyen, S. Gächter, A. Martinelli, N. Tomatis, and R. Siegwart. A comparison of line extraction algorithms using 2D range data for indoor mobile robotics. *Autonomous Robots*, 23(2):97–111, August 2007.

[42] C. E. Noon and J. C. Bean. A Lagrangian based approach for the asymmetric generalized traveling salesman problem. *Operations Research*, 39(4):pp. 623–632, 1991.

[43] C. E. Noon and J. C. Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR*, 31(1):39 – 44, 1993.

[44] S. Ntafos. Watchman routes under limited visibility. *Computational Geometry*, 1(3):149–170, 1992.

[45] K. J. Obermeyer, P. Oberlin, and S. Darbha. Sampling-based path planning for a visual reconnaissance unmanned air vehicle. *Journal of Guidance, Control, and Dynamics*, 35(2):619–631, 2012.

[46] The Office of the Chief Technologist. NASA sample return robot challenge 2013. http://www.nasa.gov/robot, 2013. Accessed: 3/12/2012.

[47] F. Pasqualetti, J. W. Durham, and F. Bullo. Cooperative patrolling via weighted tours: Performance analysis and distributed algorithms. *IEEE Transactions on Robotics*, 28(5):1181 –1188, 2012.

[48] C. Pintea, P.C. Pop, and C. Chira. The generalized traveling salesman problem solved with ant algorithms. *Journal of Universal Computer Science*, 13(7):1065–1075, 2007.

[49] A. Scheuer and T. Fraichard. Continuous-curvature path planning for car-like vehicles. In *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 2, pages 997–1003 vol.2, 1997.

[50] T. Schouwenaars, J. How, and E. Feron. Receding horizon path planning with implicit safety guarantees. In *American Control Conference, 2004. Proceedings of the 2004*, volume 6, pages 5576–5581 vol.6, 2004.

[51] S. L. Smith, M. Schwager, and D. Rus. Persistent robotic tasks: Monitoring and sweeping in changing environments. *IEEE Transactions on Robotics*, 28(2):410–426, 2012.

[52] L.V. Snyder and M.S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, 174(1):38 – 53, 2006.

[53] K. Suzuki, P. Kemper Filho, and J. Morrison. Automatic battery replacement system for UAVs: Analysis and design. *Journal of Intelligent and Robotic Systems*, 65:563–586, 2012.

[54] S. Suzuki and K. Be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, April 1985.

[55] K.A. Swieringa, C.B. Hanson, J.R. Richardson, J.D. White, Z. Hasan, E. Qian, and A. Girard. Autonomous battery swapping system for small-scale helicopters. In *IEEE International Conference on Robotics and Automation*, pages 3335 –3340, May 2010.

[56] M.P. Vitus, S.L. Waslander, and C.J. Tomlin. Locally optimal decomposition for autonomous obstacle avoidance with the tunnel-MILP algorithm. In *IEEE Conference on Decision and Control (CDC)*, pages 540–545, Cancun, Mexico, 2008 2008.

[57] D. Wettergreen, M. Wagner, D. Jonak, V. Baskaran, M. Deans, S. Heys, D. Pane, T. Smith, J. Teza, D. Thompson, P. Tompkins, and C. Williams. Long-distance autonomous survey and mapping in the robotic investigation of life in the Atacama desert. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*, 2008.

[58] B.A. White, A. Tsourdos, I. Ashokaraj, S. Subchan, and R. Zbikowski. Contaminant cloud boundary monitoring using network of UAV sensors. *Sensors Journal, IEEE*, 8(10):1681–1692, 2008.

[59] B. Yamauchi. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 146–, Washington, USA, 1997.

[60] J. Yang, X. Shi, M. Marchese, and Y. Liang. An ant colony optimization method for generalized TSP problem. *Progress in Natural Science*, 18(11):1417 – 1422, 2008.