UNIVERSITY OF CALIFORNIA, SAN DIEGO

Sampling-Based Motion Planning Algorithms for Replanning and Spatial Load Balancing

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Engineering Sciences (Aerospace Engineering)

by

Beth Leigh Boardman

Committee in charge:

Sonia Martínez, Chair
Robert Bitmead
Jorge Cortes
Troy Harden
Ken Kreutz-Delgado
Melvin Leok

2017

The dissertation of Beth Leigh Boardman is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2017

DEDICATION

To Patricia, Aiden, and Savina. Always follow your dreams and never give up.

EPIGRAPH

Someday I must read this
scholar Everyone. He seems
to have written so much
– all of it wrong.
—Tamora Pierce, Emperor Mage

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank everyone who has supported me throughout the years. In particular, I would like to acknowledge:

The University of California, San Diego professors for sharing their expert knowledge. Especially those in the Mechanical and Aerospace Engineering Department.

Los Alamos National Laboratory (LANL) and the Engineering Institute for providing the funding for my PhD research. Without programs like this I, and many other graduate students, would be unable to pursue a PhD.

This work was supported by Los Alamos National Laboratory and is approved for public release under LA-UR-17-29319. Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Prof. Sonia Martinez, thank you for being my research advisor these last five year. I truly appreciate all the hard work you have done to help me complete my PhD research. Our discussions during research meeting have been invaluable to my work.

My Los Alamos National Laboratory mentor and advisor, Troy Harden. I cannot thank you enough for all the advice you have given me over the last five years. The opportunities to do robotics research that you helped provide have been instrumental in

me completing my PhD.

My committee members, Professor Robert Bitmead, Professor Jorge Cortes, Professor Ken Kreutz-Delgado, and Professor Melvin Leok.

The Applied Engineering and Technology division of LANL, especially group 5, who hired me on as an intern during the summers. The experience I gain by working for AET-5 has beed invaluable. I have thoroughly enjoyed working with the members of AET-5; they are an excellent group of hardworking people.

My fellow lab mates, in particular Evan Gravelle and Aaron Ma. Thank you have all the conversations both research and non-research related. All of the lunch time breaks we shared we a lot of fun and great way to recharge before going back to work. Good luck in the rest of your adventures, what ever they may be.

My fellow LANL student interns, especially Katherine Jensen. Thanks for the friendship and making the summers at LANL that much more enjoyable. I appreciate all of our conversations and research discussions. I would also like to thank you for all of your hard work on getting the SIA5 robot up and running.

My first graduate advisor, Prof. Kristi Morgansen. Thank you for hiring me into your lab as a masters student. The research I did while working in your lab was an excellent foundation for pursuing my PhD. I also appreciate all of the control theory I learned from you, both in the classroom and in our research meeting.

My parents, Mark and LeAnn Boardman, for encouraging my interest in science and math. I appreciate all the life lessons that you instilled upon me as I was growing up. Those lessons have made me a better person and given me the tools I need to succeed in my chosen field.

My grandparents, Larry and Shirley Stokes, Paul and Marion Brittain, and Richard and Sue Boardman, for always believing in and encouraging me.

Nao Murakami, for being my friend. Without you I don't know how I would

have survived graduate school. I appreciate all the times you have let me bounce ideas off of you, answered MATLAB, math, and physics questions, read and edited papers for me, for listening to all my rants when I was frustrated, and most important, sharing in the joy of all the little accomplishments that led to a PhD.

Chapter 3, in part, contains material as it appears in ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference 2015. "Focused Refinement in the RRT*: Trading Optimality for Improved Performance", Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, contains material submitted to ASME Journal of Dynamic Systems, Measurement and Control 2017. "Improved Performance of Asymptotically Optimal Rapidly-Exploring Random Trees," Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, contains material as it appears in the proceeding of the 52nd Annual Allerton Conference on Communication, Control, and Computing 2014. "Optimal kinodynamic motion planning in environments with unexpected obstacles", Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, contains material submitted to ASME Journal of Dynamic Systems, Measurement and Control 2017. "Improved Performance of Asymptotically Optimal Rapidly-Exploring Random Trees" Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, contains material that will be submitted, "Reactive Multi-Agent Path Planning: Combining the RRT* with Collision Cones" 2017. Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, contains material as it appears in American Control Conference 2016. "Spatial load balancing in non-convex environments using sampling-based motion planners" Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, contains material as it appears in American Control Conference 2017. "Limited range spatial load balancing for multiple robots" Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, contains material that has been submitted to Autonomous Robots 2017. "Limited Range Spatial Load Balancing in Non-Convex Environments using Sampling-Based Motion Planners", Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

VITA

2010	B. S. in Aeronautics and Astronautics, University of Washington, Seattle
2012	M. S. in Aeronautics and Astronautics, University of Washington, Seattle
2017	Ph. D. in Engineering Sciences (Aerospace Engineering), University of California, San Diego

PUBLICATIONS

B. Boardman and T. L. Hedrick and D. H. Theriault and N. W. Fuller and M. Betke and K. A. Morgansen, "Collision avoidance in biological systems using collision cones", *American Control Conference (ACC)*, Washington D.C., USA, June 2013.

B. Boardman and T. Harden and S. Martínez, "Optimal kinodynamic motion planning in environments with unexpected obstacles", *Communication, Control, and Computing (Allerton), 52nd Annual Allerton Conference on*, Monticello, IL, USA, October 2014.

B. Boardman and T. Harden and S. Martínez, "Focused Refinement in the RRT*: Trading Optimality for Improved Performance", *ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Boston, MA, USA, August 2015.

B. Boardman and T. Harden and S. Martínez, "Spatial load balancing in non-convex environments using sampling-based motion planners", *American Control Conference (ACC)*, Boston, MA, July 2016.

B. Boardman and T. Harden and S. Martínez, "Limited range spatial load balancing for multiple robots", *American Control Conference (ACC)*, Seattle, WA, USA, May 2017.

B. Boardman and T. Harden and S. Martínez, "Limited Range Spatial Load Balancing in Non-Convex Environments using Sampling-Based Motion Planners", Submitted. *Autonomous Robots*, 2017.

B. Boardman and T. Harden and S. Martínez, "Improved Performance of Asymptotically Optimal Rapidly-Exploring Random Trees", Submitted to *Journal of Dynamic Systems, Measurement and Control*, 2017.

B. Boardman and T. Harden and S. Martínez, "Reactive Multi-Agent Path Planning: Combining the RRT* with Collision Cones", To be Submitted. 2017.

ABSTRACT OF THE DISSERTATION

Sampling-Based Motion Planning Algorithms for Replanning and Spatial Load Balancing

by

Beth Leigh Boardman

Doctor of Philosophy in Engineering Sciences (Aerospace Engineering)

University of California, San Diego, 2017

Professor Sonia Martínez, Chair

The common theme of this dissertation is sampling-based motion planning with the two key contributions being in the area of replanning and spatial load balancing for robotic systems. Here, we begin by recalling two sampling-based motion planners: the asymptotically optimal rapidly-exploring random tree (RRT*), and the asymptotically optimal probabilistic roadmap (PRM*). We also provide a brief background on collision cones and the Distributed Reactive Collision Avoidance (DRCA) algorithm. The next four chapters detail novel contributions for motion replanning in environments with unexpected static obstacles, for multi-agent collision avoidance, and spatial load

balancing. First, we show improved performance of the RRT* when using the proposed Grandparent-Connection (GP) or Focused-Refinement (FR) algorithms. Next, the Goal Tree algorithm for replanning with unexpected static obstacles is detailed and proven to be asymptotically optimal. A multi-agent collision avoidance problem in obstacle environments is approached via the RRT*, leading to the novel Sampling-Based Collision Avoidance (SBCA) algorithm. The SBCA algorithm is proven to guarantee collision free trajectories for all of the agents, even when subject to uncertainties in the knowledge of the other agents' positions and velocities. Given that a solution exists, we prove that livelocks and deadlock will lead to the cost to the goal being decreased. We introduce a new deconfliction maneuver that decreases the cost-to-come at each step. This new maneuver removes the possibility of livelocks and allows a result to be formed that proves convergence to the goal configurations. Finally, we present a limited range Graph-based Spatial Load Balancing (GSLB) algorithm which fairly divides a non-convex space among multiple agents that are subject to differential constraints and have a limited travel distance. The GSLB is proven to converge to a solution when maximizing the area covered by the agents. The analysis for each of the above mentioned algorithms is confirmed in simulations.

Chapter 1

Introduction

Robotics and automation are being integrated into our everyday lives more and more. We mostly see automation that is very structured, resulting in repetitive motion. Theses robots are designed to do one, and only one, task efficiently. More recently, robotics has ventured into accomplishing versatile tasks. This includes navigating various environments, known, unknown, or uncertain.

We have also seen an increase in human-robot interaction. Robots are being used to improve safety and efficiency for humans. The key to humans and robots working cooperatively is trust. The human needs to trust that the robot will avoid collisions and move in a predicable, natural manner. The research presented here works toward developing motion planning algorithms that produce paths, in an uncertain environment, that are efficient and smooth.

Motion planning is about determining how a robot should move to complete a given task. The possible tasks include navigation, coverage, localization, and mapping. Motion planning can be model based (offline planning), sensing based (reactive planning), a hybrid that switches between sensing or modeling based, or probabilistic which fuses together modeling and sensing based.

Some of the challenges facing robotic motion planning are non-convex spaces, agents subject to differential constraints, high dimensional robots, uncertainties, and dynamic and uncertain environments, all of which have a great impact of the speed of the algorithms. Unlike more classical algorithms, sampling-based motion planners find a solution quickly, handle non-convex environments easily, and are able to find a solution for high dimensional robots. Even though current sampling-based motion planners are quick, they do not return an optimal or near optimal solution in real-time.

Sampling-based motion planning algorithms sample the environment incrementally and quickly connect samples from the free configuration space to find a collision-free path from an initial configuration to a goal configuration. Improving the performance of sampling-based algorithms, in convergence speed and its adaptation to moving obstacles, is key in the development of real-time motion planning algorithms. This motivated for the research in this dissertation. The research contained within this dissertation develops novel improved performance sampling-based motion planning algorithms with applications to environments with unknown static and dynamic obstacles. The last part of this dissertation applies sampling-based motion planners to a novel multi-agent coverage problem, by which dynamic agents with a limited travel range aim to balance the coverage area of a region in a fair way.

1.1 Literature Review

Rapidly-exploring Dense Tree algorithms (RDTs, also known as RRTs) [37] and Sampling-Based Roadmaps (SBRs, including Probabilistic Roadmaps (PRMs) [31]) are sampling-based motion planners which are resolution or probabilistically complete, and are able to find a feasible path to the goal without the explicit modeling of the configuration space. As opposed to SBRs, RDTs do not require pre-processing and can find

a path relatively quickly. However, the path produced by these planners can be very jagged and result in unnecessary motion that can increase the execution time. Consequently, this motivated research into how to obtain better paths from these planners.

One way to obtain improved paths is to apply a post-processing algorithm. In [61], one of such algorithms is presented and applied on any given path. The algorithm limits the allowable deviation from the original path and results in a new path with fewer nodes. A divide and conquer method is used in [11] in order to shorten a given path by connecting the first and last nodes in the path directly. If not successful, then the set of nodes in the path list is bisected until the connections are successful. Similarly, the post-processing algorithm in [54] randomly selects two points from the path list and attempts to replace the segment between them with a straight line. This process is repeated a predetermined number of times.

A subsequent effort focuses on obtaining paths that guarantee asymptotic optimality with probability one. At the core of this line of work are the PRM* and RRT* [29]. The RRT* handles any-time applications [30] and manipulators [50]. The RRT* paths can be composed of many more nodes than is strictly necessary. The Ball-tree algorithm, [57], is a sampling-based motion planner that improves the performance of the RRT and RRT* by using volumes of free-space instead of points as the vertices of the tree. More recently, the RRT$^{\#}$ [2] is another sampling-based planner that returns an optimal path by maintaining a graph and a spanning subtree. The RRT$^{\#}$ separates the exploration and exploitation tasks so the algorithm can be run in parallel to improve performance. Another algorithm, the Fast Marching Tree (FMT*) [24], performs a "lazy" dynamic programming recursion on samples from the configuration space to produce a tree of paths. A key result from [24] is the algorithm convergence rate.

The following papers also study the effects of exploitation versus exploration on the RRT*. Akgun et al. [1] uses local biasing to choose the sampling point based upon

the current best path to the goal. The RRT*-Smart in [22] finds an initial path to the goal, then it optimizes it using first a smoothing technique, and then it further shapes it by biasing sampling to balls around the nodes in the optimized path. While these two papers share the same idea of exploitation of a given path to the goal, the method focuses on a single path only, which results, at most, in a locally-optimal path.

The idea of re-adapting motion plans when finding new unexpected obstacles has been exploited significantly in the literature. The discrete-time D*, and D* lite algorithms [32], [58] re-adapt A* algorithms to find the optimal path in a discretized space. However, these algorithms become intractable as the dimensionality increases, while they have a limited ability to handle differential constraints.

The sampling-based algorithms in [19], [64] [8], [39], and [21] all extend the RRT algorithm to deal with dynamic environments. The Dynamic Rapidly-exploring Random Tree (DRRT) [19] roots the tree at the goal and trims branches in the tree that are obstructed by the new obstacle. The trimming is done by removing nodes that are within a region that contains the obstacle and whose edge is in conflict. The descendants of the affected nodes are also removed so that only one tree is maintained. The remaining paths in the tree still lead to the goal but are not optimal.

In [39] the Reconfigurable Random Forest (RRF) algorithm maintains a forest of trees. The trees are from previous plannings and have been broken apart according to the new obstacle information and initializing new trees at the new initial and goal configuration. The RRF attempts to connect the trees as in the RRT-connect [23] making this framework good for multi-query problems. The trees are trimmed by removing all nodes from within a bounding box containing the obstacle that are determined to be in conflict with the new obstacle. The RRF also prunes its trees to maintain a manageable number of nodes to reduce searching time. The lazy reconfiguration forest (LRF) is presented in [21], and uses the idea of maintaining multiple RRT trees from the RRF but

only checks for invalid edges along the task path instead of checking the entire tree.

To rebuild a tree from the initial position, way points from the previous tree are reused to increase the likelihood of a successful connection in the execution extended RRT (ERRT) [8]. The ERRT also uses an adaptive cost function that improves the generated paths. The multipartite RRT (MP-RRT) [64] combines several of the above mentioned planners and an opportunistic strategy for reusing information during replanning in a dynamic environment. However, none of these algorithms produce optimal paths. An asymptotically optimal replanning algorithm, RRT^X, was developed by the authors of [47]. The RRT^X maintains a graph and a shortest path sub-tree rooted at the goal. When an obstacle is added or removed only the effected edges are updated.

The following selection of works deal with multiple agents in a workspace. A first sampling of papers solve the multi-agent path planning problem without a graph or uncertainty. The algorithm in [6] is based on vectorized particle swarm optimization. A real-time algorithm is presented in [12] that uses sequential convex programming. A dynamic programming scheme is used in [41] to develop an online coordinated motion planner. While the algorithms in [56] and [44] use graphs, they do not use sampling-based motion planners. Sampling-based motion planners such as the RRT are used in [62] and [52].

Most similar to the reactive path planning research presented here are [60, 45, 33] which all use RRT based planners and uncertainties. The motion planning problem with uncertain obstacles is solved using a game theoretic formulation in [60]. Sequential path finding is used in [45] to develop a real-time algorithm that handles uncertainties and disturbances. The real-time algorithm that handles uncertain dynamic obstacles in [33] uses chance constraints.

1.1.1 Multi-Agent Coverage and Spatial Load Balancing

The gradient-based descent Lloyd algorithm, [40], is the basis of many multi-agent coverage strategies. A limited sample of work building on this approach includes limited sensor footprints [38], heterogeneous agents (different sensing radii) [59, 51], non-holonomic agents [35, 18, 55], and power constraints [34]. The area-constrained problem is studied in [14, 48, 49], where a partition of the environment, dependent on weights, is employed. This partition, combined with an appropriately modified objective, is sufficient to enforce agents' regions to have the desired area. In [25], a sink node is used to aggregate information collected by all of the agents. While the communication cost is minimized, the agents must all communicate with the sink node. These papers assume the agents have unlimited ranges and are deployed in a convex environment. In [42], a convex area covered by sensors is maximized by minimizing the area uncovered in the agents' cell. Even though the algorithms have certain distributed properties, they are not, in general, implementable over a limited-range communication graph.

The research on multi-agent coverage in non-convex environments is closely related to the research found in Chapter 5. A first paper is [10], which employs a diffeomorphism to transform the non-convex environment into an almost convex one, where only a finite number of points have been subtracted. The coverage problem is then solved in the transformed environment and a solution is obtained via the inverse transformation. The diffeomorphism limits this algorithm to two-dimensional environments. Environments with polygonal obstacles are considered in [63, 7]. A solution to the multi-agent coverage problem for non-point robots in non-convex environments can be found in [51]. The approach to solve the non-convex multi-agent coverage problem taken by [26] is to use visibility-based Voronoi diagrams while maximizing the coverage area. The authors of [26] extend their work in [27] to include heterogeneous sensing. Most of the above papers do not present solutions for agents subject to differential constraints. And those

papers that do account for differential constraints do not consider obstacles.

The following papers are for coverage in an unknown non-convex environment with agents not subject to differential constraints. A Voronoi partition and potential field is used in [53] to find a solution to the coverage problem in non-convex environments with unknown obstacles. The authors of [5] let the agents learn and build a gridded environment map and then determine which grid points belong to each agents' generalized Voronoi cells. This grid-based approach is limited to low dimensional spaces and differential constraints are difficult to handle in this manner. The paper [4] is an extension of [5] to Reimannian manifolds with non-convex boundaries.

1.2 Contributions

The following section details the contributions of the motion planning research contained within this manuscript. The contributions fall into four categories: improved performance, replanning, multi-agent deconfliction and spatial load balancing.

We detail two algorithms that improve the performance of the RRT*, then present extended simulations and analysis results. The two algorithms are the Focused-Refinement (FR) and the Grandparent-Connection (GP).

The Focused-Refinement (FR) algorithm is a modification of the RRT* that reduces the computation time needed to obtain a low-cost path to the goal. This is done by exploring the environment quickly until a set of paths to the goal is found. Then, the algorithm focuses on lowering the cost of the paths in this set while periodically exploring the environment. In this way, the algorithm quickly returns an asymptotically optimal path within the regions that are more intensively exploited. We present a novel way of uniformly sampling randomly within these regions that, with the right parameters, can recover the entire configuration space.

The Grandparent-Connection (GP) is a modification to the RRT* algorithm that attempts to connect the added vertex to its grandparent instead of its parent vertex. This essentially straightens the computed paths and lowers the cost. We prove that the asymptotic optimality and probabilistic completeness results for the RRT* are maintained when using the GP modification. The GP is also proven to recover the optimal path to the goal for all configuration in the visible set.

In the area of replanning, we developed the novel Goal Tree (GT) algorithm that is used to reduce the RRT* replanning time in the presence of a new obstacle. The Goal Tree (GT) algorithm reuses information from the RRT* rooted at the goal configuration, $\mathcal{T}_G$. More precisely, when a previously unknown obstacle obstructs the best path, $\mathcal{T}_G$ is trimmed to reflect this information. The tree is then incrementally extended in the affected region of the configuration space. In this setting, we identify a new sampling region, strictly contained in the configuration space, such that, when used with the GT algorithm, guarantees the recovery of an asymptotically optimal path. First, a region is proven to exist, then a characterization is provided for a general robot in a d dimensional environment. By exploiting the known path types of vehicles with no differential constraints in a d dimensional configuration space and a Dubins' vehicle, alternative characterizations of the new planning region are given.

Next, we approach multiple robot coordination problems using sampling-based motion planners. The SAMPLING-BASED COLLISION AVOIDANCE (SBCA) algorithm combines motion planning with collision avoidance for multiple agents. The agents travel to their individual goal configurations while avoiding collision. Each of the N agents builds an RRT* that is used to determine the best path to its goal from all other configurations. While executing the current best path, the agent checks for conflict with the other agents in the environment. When a conflict is determined, the agent performs a deconfliction maneuver. The deconfliction maneuver is based on the Distributed

Reactive Collision Avoidance (DRCA) from [36] which uses collision cones. The deconfliction maneuver updates the agents' velocity so that it is not in conflict with any of the other agents. A new node is added to the agents' RRT* that corresponds to the new velocity. The agent then updates its current best path and continues to travel along repeating the above process. The algorithm is first given for perfectly known position and velocity of the other agents. The algorithm is then extended to handle uncertainty in the position and velocity knowledge of the other agents.

We analyze the SBCA algorithm to prove that the agents will never collide with one another or a static obstacle. A proof is given for both the certain and uncertain algorithm. We prove that, under certainty assumptions, the agents will, with probability one, reach their goal configurations. The simulations show that the agents reach their goal configuraions on a collision free trajectory. The simulations also examine the amount of uncertainty seen by the agents.

1.2.1 Spatial Load Balancing

In Chapter 5, we consider a limited range, non-convex spatial load balancing problem for dynamically constrained agents. We approach this problem by employing locational optimization techniques that are based on generalized Voronoi partitions of the environment. Due to the obstacles and differential constraints, obtaining an exact Voronoi region description can be difficult. Thus, we expand our objective by employing an approximation of the agents' configuration space by means of a probabilistic roadmap star (PRM*).

More precisely, we start from a continuous-space version of the load balancing problem subject to a variable area constraint. With the objective of obtaining distributed algorithms, we introduce limited ranges and associated area and mixed-type performance metric functions. To solve the problem, we introduce "sub-partitions" of

the space, $\mathcal{V}^{\mathrm{LR}}(P, \omega)$, dependent on a set of weights, ω, and agent positions, P. Then, we prove the existence of a set of weights that make the sub-partition satisfy the variable area constraint and thus solve the problem. Based on this, we provide an update law for agents that allows them to converge to a set of such weights. The agent positions are updated using a gradient law aimed at decreasing the cost function with respect to position. We then define a class of deployment algorithms for solving the problem, and show convergence to a solution for the area-type performance metric case.

Building on the continuous-space version, the graph-based algorithm builds a PRM* type graph to recover the cost of an agent subject to differential constraints moving between any two configurations in a known non-convex environment. In this way, the coverage regions are approximated by assigning each agent a subset of nodes from the PRM*. This subset of nodes is further employed to estimate the coverage regions' corresponding areas. We provide a characterization of how the limited range algorithm is distributed over a communication graph for radially-unbounded cost functions. We provide an analysis of algorithm convergence as the number of nodes in the optimal probabilistic roadmap tends to infinity. Finally, we present in simulation the convergence of agents whose cost to move is given by the Euclidean norm squared.

Chapter 2

Background

This chapter introduces some basic background, terminology, and notation related to motion planning. For a more in-depth review of the topics discussed in this chapter refer to [13, 43]. This dissertation focuses on navigation and coverage. Motion planning algorithms can be offline, online, or have elements of both. Offline algorithms completely determine the robots trajectory before execution is begun. Online algorithms determine the robot's path during execution.

A robot's path is determined in its configuration space. A robot configuration is a complete description of the robot. A configuration space is a collection of all robot configurations. The obstacle space is a description of the obstacles and robot self collisions in the configuration space. The free space is the configuration space minus the obstacle space. The robot can occupy any configuration in the free space and be collision free. Here, the d-dimensional configuration space is denoted as $X \subseteq \mathbb{R}^d$. The obstacle space is X_{obs} and the free space, $Q = X \setminus X_{\text{obs}}$.

Sampling-based motion planners construct graphs to help in the determination of the robot's path. A graph is a collection of nodes and edges, $\mathcal{G}.(V,E)$. The nodes, $v \in \mathcal{G}.V$, are at specific configurations in a configuration space. Each edge, $e \in \mathcal{G}.E$, is

an ordered pair of nodes $e_{1,2} = (v_1, v_2)$, where v_1 is the parent and v_2 is the child. Each edge in $\mathcal{G}$ has a cost associated with it, denoted $c_{\text{edge}}(e)$. If two nodes are connected by an edge, $(v_1, v_2) \in \mathcal{G}.E$, then they are considered neighbors.

Graphs can have special attributes, such as directed or undirected and sinks or sources. A directed graph, contains edges that must be followed in a specific direction, i.e. from node v_1 to node v_2. Let $\mathcal{G}$ be a directed graph and $v \in \mathcal{G}.V$, then $(v, v_1) \in \mathcal{G}.E$ is said to be an outgoing edge (from v to v_1) and $(v_2, v) \in \mathcal{G}.E$ an incoming edge (entering v from v_2). A undirected graph means the edge can be followed in either direction, i.e. from node v_1 to node v_2 or from node v_2 to node v_1. A sink is a graph node that only has incoming edges. A source node is the opposite, where all the edges associated with the source are outgoing edges. A Cycle is a closed loop set of edges that leads back to itself, $\{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$.

A specific type of graph used heavily in this thesis is a tree, $\mathcal{T}$. A n node tree has $n-1$ edges and no cycles. The trees used here have a sink (or source) that is referred to as the root. This root is the starting or ending configuration for the robot's path.

Before we introduce the sampling-based algorithms this research is related to, the general algorithm properties of optimality and completeness are defined. Optimality refers to the minimization of a cost function that is, possibly, subject to a set of constraints. When an algorithm accurately returns a solution, or that a solution does not exist, in a finite amount of time, the algorithm is said to be complete. Two weaker notions of completeness are resolution complete and probabilistically complete. A resolution complete algorithm will return the correct solution in a finite amount of time, but if a solution does not exist the algorithm will run forever. LaValle, [37], defines an algorithm as probabilistically complete if "with enough points, the probability that it finds an existing solution converges to one.

The research presented in this dissertation builds on previously developed

sampling-based motion planners. In particular, we use the Asymptotically Optimal Rapidly-exploring Random Tree Algorithm (RRT*) and Asymptotically Optimal Probabilistic Roadmap (PRM*). A brief description on the build process of both motion planners is given below in Sections 2.1 and 2.2, respectively. The final section in this chapter, Section 2.3, gives an introduction to collision cones in relation to collision avoidance.

2.1 The Asymptotically Optimal Rapidly-exploring Random Tree Algorithm

The RRT* algorithm by Karaman and Frazzoli is theoretically analyzed in [29]. The kinodynamic RRT* is presented in [28].

The RRT*, outlined in Algorithm 1, builds a tree, $\mathcal{T}$ which is dense with probability one in the entire configuration space, X, as the number of samples, n, goes to infinity. We use Cost as the notation for the cost function being minimized. In the original work by [29], the edge cost considered is the *cost-to-go*; that is the cost of $e_{1,2} = (v_1, v_2)$ is the cost of moving from the parent v_1 to the child v_2. Most of our work uses a *cost-to-come*, or the cost of moving from child to parent. This allows use to build a tree rooted at the goal configuration instead of the initial configuration. Then, the cost of a vertex, $\mathsf{Cost}(v)$, is the sum of the costs of the edges connecting the root to v. The paths in $\mathcal{T}$ are asymptotically optimal, meaning that as $n \to \infty$ the optimal path from the initial configuration, $x_I \in X_{\text{free}}$, to any other configuration in X_{free} is recovered. More precisely, the functions involved in the RRT* process are described as follows. With some abuse of notation, we will define a robot configuration as x_v instead of v.

After initializing $\mathcal{T}$ at x_I, the RRT* begins by using the Sample function to output x_{rand}, a uniformly sampled random configuration from X_{free}. The Nearest function finds

the nearest vertex, $x_{\text{nearest}} \in \mathcal{T}$, and extends $\mathcal{T}$ a distance ε from x_{nearest} to get x_{new}.

Next, the function Near determines the set X_{near}, which is the of vertices from $\mathcal{T}$ that are near x_{new}. Vertices that are farther than $\delta_T = \min(\varepsilon, \gamma_T(\log(n_v)/n_v)^{(1/d)})$, where n_v is the number of vertices in $\mathcal{T}$, d is the dimension of the configuration space, and γ_T is an independent parameter, are omitted from X_{near}. The best parent for x_{new}, determined in FindBestParent, is the vertex in X_{near} that has a collision-free path with the lowest $\mathsf{Cost}(x_{\text{new}})$, as outlined in Algorithm 2. The paths that connect the vertices to each other (determined using Steer), do so according to the system dynamics. Only collision-free edges are added to $\mathcal{T}$. The collision checker, CollisionCheck, returns true if the edge is collision-free. If x_{new} is added to $\mathcal{T}$, then Rewire attempts to add the other vertices in X_{near} as children of x_{new} based upon a lower cost and collision-free edge. The Rewire function is outlined in Algorithm 3.

Algorithm 1 $\mathcal{T} = (V, E) \leftarrow \mathsf{RRT}^*(x_I, \varepsilon)$

$\mathcal{T} \leftarrow \mathsf{InitializeTree}()$;
$\mathcal{T} \leftarrow \mathsf{InsertNode}(\emptyset, x_I, \mathcal{T})$;
for $i = 1$ to $i = N$ **do**
  $x_{\text{rand}} \leftarrow \mathsf{Sample}(i)$;
  $x_{\text{new}} \leftarrow \mathsf{Nearest}(\mathcal{T}, x_{\text{rand}}, \varepsilon)$;
  $X_{\text{near}} \leftarrow \mathsf{Near}(\mathcal{T}, x_{\text{new}})$;
  $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$;
  if $x_{\text{parent}} \neq \mathsf{NULL}$ **then**
    $\mathcal{T} \leftarrow \mathsf{InsertNode}((x_{\text{parent}}, x_{\text{new}}), x_{\text{new}}, \mathcal{T})$;
    $\mathcal{T} \leftarrow \mathsf{Rewire}(\mathcal{T}, X_{\text{near}}, x_{\text{new}})$;
  end if
end for
return $\mathcal{T}$

Algorithm 2 $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$

$x_{\text{parent}} \leftarrow \emptyset$;
$c_{\min} \leftarrow \infty$;
for $x_{\text{near}} \in X_{\text{near}}$ **do**
  $e_{\text{near,new}} \leftarrow \mathsf{Steer}(x_{\text{near}}, x_{\text{new}})$;
  $c_{\text{near}} \leftarrow \mathsf{Cost}(x_{\text{near}}) + c_{\text{edge}}(e_{\text{near,new}})$;
  if $c_{\text{near}} < c_{\min}$ and $\mathsf{CollisionFree}(e_{\text{near,new}})$ **then**
    $x_{\text{parent}} \leftarrow x_{\text{near}}$;
    $c_{\min} \leftarrow c_{\text{near}}$;
  end if
end for
return x_{parent};

Algorithm 3 $\mathcal{T} \leftarrow \mathsf{Rewire}(\mathcal{T}, X_{\text{near}}, x_{\text{new}})$

for $(x_{\text{near}}) \in X_{\text{near}}$ **do**
  $e_{\text{near,new}} = \mathsf{Steer}(x_{\text{new}}, x_{\text{near}})$;
  if $\mathsf{Cost}(x_{\text{new}}) + c_{\text{edge}}(e_{\text{near,new}}) < \mathsf{Cost}(x_{\text{near}})$ **then**
    if $\mathsf{CollisionFree}(e_{\text{near,new}})$ **then**
      $x_{\text{oldparent}} \leftarrow \mathsf{Parent}(\mathcal{T}, x_{\text{near}})$;
      $\mathcal{T}.\mathsf{remove}((x_{\text{oldparent}}, x_{\text{near}}))$;
      $\mathcal{T}.\mathsf{add}((x_{\text{new}}, x_{\text{near}}))$;
    end if
  end if
end for
return $\mathcal{T}$;

2.2 The Asymptotically Optimal Probabilistic Roadmap Algorithm

The second sampling-based motion planner the work in this dissertation uses is the asymptotically optimal probabilistic roadmap (PRM*) [29]. This section briefly describes how to construct a PRM*, denoted as G, which assumes the environment is known. While there are similarities between the RRT* and PRM*, the greatest difference is the graph structure. The PRM* can have multiple incoming and multiple outgoing edges. Note that the PRM* is limited to dynamics that can be solved with a two point boundary value problem.

The graph G allows the agents to recover the approximately optimal path cost between two configurations in the graph as the sum of the costs of the edges defining the path. The edge cost is the cost of an agent to travel between the nodes that define the edge, e.g. it can be given by length of the edge. The graph G is composed of a set of nodes $\mathcal{N}_G$ and a set of edges $\mathcal{E}_G$ constructed as follows. A node $q \in \mathcal{N}_G$ is a sampled configuration from Q, and each edge, $e \in \mathcal{E}_G$, is an ordered pair, $e = (q_1, q_2)$, which corresponds to an optimal path in Q, satisfies all constraints, and has a cost $J_e(q_1, q_2)$. The cost $J(q_1, q_2)$ is assumed to be additive; given an optimal path from q_1 to q_2, and a node q' in that path, it holds that, $J(q_1, q_2) = J(q_1, q') + J(q', q_2)$. We denote the out neighbors of q in G as $\mathcal{N}_G^{\text{out}}(q) = \{q_j \in \mathcal{N}_G \mid (q, q_j) \in \mathcal{E}_G\}$.

Each iteration of the construction of G begins by taking a uniformly sampled random configuration from the free-space, $q_{\text{rand}} \in Q$. Next, all graph vertices that are within a ball centered at q_{rand} with radius, $\delta_G = \gamma_G(\log(m)/m)^{1/d}$, are determined. Here, γ_G is a fixed parameter, m is the number of vertices currently in $\mathcal{N}_G$, and d is the dimension of Q. Let the near vertices of q_{rand} be denoted as Q_{near}. If Q_{near} is empty, then the graph vertex that is closest to q_{rand} is added to Q_{near}. The least-cost paths, whose cost

is $J_e(q_{\text{rand}}, q_{\text{near}})$, from q_{rand} to $q_{\text{near}} \in Q_{\text{near}}$ are determined; these are outgoing edges of q_{rand}. If the direction matters, as is the case with differential constraints, then the least-cost path from q_{near} to q_{rand} is also determined, these are the incoming edges of q_{rand}. Each collision-free path, e, is added to $\mathcal{E}_G$ as an edge. The application of the PRM* to this work requires that G be strongly connected; a necessary condition is that all $q \in \mathcal{N}_G$ must have both an outgoing and incoming edge, so that if an agent reaches that node it may also leave that node. A sufficient condition is to construct a graph which only allows edge pairs, that is, $(q_1, q_2) \in \mathcal{E}_G$ if and only if $(q_2, q_1) \in \mathcal{E}_G$.

The free-space Q is discretized by G while maintaining an asymptotically optimal roadmap of the environment. Each node $q \in \mathcal{N}_G$ has an associated area, $\beta(q)$, which is calculated as follows. Let $\mathcal{N}_X = \mathcal{N}_G \cup \mathcal{N}_{\text{obs}}$, where $\mathcal{N}_{\text{obs}}$ is a set of configurations inside X_{obs}. Then, determine the Voronoi partition of X using $\mathcal{N}_X$. The $\beta(q)$ for each $q \in \mathcal{N}_G$ is the area of its associated cell in this partition. A description of the external boundary of X is needed, and we assume this is available. In this dissertation, nodes refer to the vertices in the graph, $q \in \mathcal{N}_G$, and not to the n agents that move in the environment.

2.3 Deconfliction using Collision Cones

This section is based on the collision cones and collision avoidance algorithm in [36]. Let N be the number of agents in the $\mathbb{R}^2$ environment. Collision cones are a way of determining whether or not two agents are in conflict. Two agents $i, j \in \{1, \ldots, N\}$ are in *conflict* if, by staying on their current trajectory (heading and speed), they will eventually collide. *Deconfliction* is the act of an agent i changing its velocity to avoid future collision.

Let each agent i be approximated as a circle with radius $\rho_i \in \mathbb{R}_{>0}$ and let $d_{\text{sep},0}$

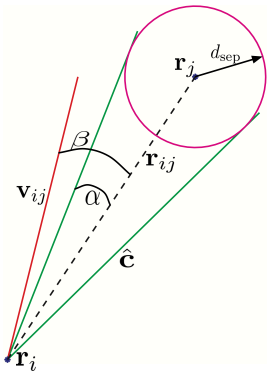


Figure 2.1: Depiction of a collision cone

be the minimum allowable separation distance between any two agents. Then, agents i and j can be approximated as points that are to be separated by a minimum distance, $d_{\text{sep},ij} = \rho_i + \rho_j + d_{\text{sep},0}$. Agents i and j have position vectors, $\mathbf{r}_i$ and $\mathbf{r}_j$ and velocity vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, respectively. The relative position vector between agent i and j is defined as $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ and the relative velocity vector is defined as $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$. The speed of agent i is denoted as $v_i = \|\mathbf{v}_i\|$.

The collision cone half angle is defined as $\alpha_{ij} = \arcsin\left(\frac{d_{\text{sep},ij}}{\|\mathbf{r}_{ij}\|}\right)$. The angle between the relative position and relative velocity vectors is $\beta_{ij} = \arccos\left(\frac{\mathbf{r}_{ij}\cdot\mathbf{v}_{ij}}{\|\mathbf{r}_{ij}\|\|\mathbf{v}_{ij}\|}\right)$. The two edges of the collision cone between agents i and j are defined using the unit vector $\hat{c}_{ij} = \mathcal{R}(\pm\alpha_{ij})\frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}$. Note that the half angle $\alpha_{ij} = \alpha_{ji}$ and that each cone contains the circle centered at agent j's position with radius $d_{\text{sep},ij}$. A depiction of a collision cone is in Fig. 2.1. The following proposition characterizes conflicts between pair of agents in terms of their collision cones.

Proposition 1 ([36])**.** Let $\beta_{ij} = \angle\mathbf{v}_{ij} - \angle\mathbf{r}_{ij}$, $\alpha_{ij} = \arcsin\left(\frac{d_{\text{sep},ij}}{\|\mathbf{r}_{ij}\|}\right)$ and $\mathbf{r}_{ij}$ be the relative position vector at the time conflict is being checked. A necessary and sufficient condition for there to exists no conflict is $|\beta_{ij}| \geq \alpha_{ij}$.

Agent i determines the collision cones between itself and all other agents $j \neq i$ in the environment. When in conflict with at least one other agent, agent i applies a

deconfliction maneuver. This maneuver moves agent i out of conflict with all other agents while minimizing its change in velocity,

$$\mathbf{v}_i' = \underset{\mathbf{v}\in V_i}{\operatorname{argmin}} \|\mathbf{v} - \mathbf{v}_i\|,$$

where $\mathbf{v}_i'$ is the new velocity vector picked from a set V_i of feasible velocities V_i determined by the maneuvers described below.

If the agents cannot speed up or slow down then a *constant speed maneuver* is executed. The new velocity vector is chosen from the set V_i,

$$V_i = \{\mathbf{v}_j - a_{ij}\hat{\mathbf{c}}\}, \quad \forall\, j \neq i,$$

where

$$a_{ij} = \hat{\mathbf{c}}^\top \mathbf{v}_j \pm \sqrt{(\hat{\mathbf{c}}^\top \mathbf{v}_j)^2 - \mathbf{v}_j^\top \mathbf{v}_j + \mathbf{v}_i^\top \mathbf{v}_i}.$$

Note that a_{ij} has two values and only its real positive values are valid. If V_i is non-empty, then,

$$\mathbf{v}_i' = \underset{\mathbf{v}\in V_i}{\operatorname{argmin}} \|\mathbf{v} - \mathbf{v}_i\|,$$

such that $\mathbf{v}_i'$ is not put agent i in conflict with any other agents. If V_i is an empty set or there are no velocities in V_i that are conflict free, then $\mathbf{v}_i' = \mathbf{0}$. Theorem 2 from [36] gives conditions on the existence of an admissible constant speed velocity, i.e. V_i will be non-empty.

When agents have the ability to change their speed, a *variable speed maneuver* is executed. Let $v_{i,\max}$ be the maximum allowable speed of all agents. The new velocity

for agent i will result from one of three maneuvers:

1. nearest point on each edge of the cones,
   $\mathbf{v}_i' = \hat{\mathbf{c}}\hat{\mathbf{c}}^\top \mathbf{v}_{ij} + \mathbf{v}_j$,

2. intersection between $v_{i,\max}$,
   $\mathbf{v}_i' = \mathbf{v}_j - \hat{\mathbf{c}}\hat{\mathbf{c}}^\top \mathbf{v}_j \pm \hat{\mathbf{c}}\sqrt{(\hat{\mathbf{c}}^\top \mathbf{v}_j)^2 - \mathbf{v}_j^\top \mathbf{v}_j + v_{i,\max}^2}$,

3. intersection between two cones.

Each maneuver described above produces a set of possible velocities to be checked for feasibility and put into the set V_i. Agent i checks the set of potential velocities from each maneuver type until it finds the $\mathbf{v}_i'$ that minimizes its change in velocity, has a speed less than the maximum allowable $\|v_i'\| < v_{i,\max}$, and is conflict free. Again if V_i is empty or does not contain a feasible velocity, $\mathbf{v}_i' = \mathbf{0}$. Theorem 3 from [36] gives conditions on the existence of an admissible variable speed velocity, i.e. V_i will be non-empty.

2.4 Notation

Finally, some notation that is used throughout this dissertation is define. The set of natural numbers, $\{1,2,3,...\}$, is denoted as $\mathbb{N}$. A two dimensional ball, centered at x with radius r is $\mathbb{B}(x,r)$ Let $\mathbf{1}_n = (1,\ldots,1)^\top \in \mathbb{R}^n$ and $\mathbf{0}_n = (0,\ldots,0)^\top \in \mathbb{R}^n$. The two dimensional rotation matrix is defined as $\mathcal{R} \subset \mathbb{R}^{2\times 2}$. Finally, the 2-norm is $\| \begin{bmatrix} x & y \end{bmatrix}^\top \| = \sqrt{x^2+y^2}$.

Chapter 3

Improvements to Sampling-Based Motion Planning Algorithms

In this chapter, two sampling-based algorithms are developed that modify the RRT* to improve the performance by lowering the path cost more quickly. The details of the Focused-Refinement (FR) algorithm is in Section 3.1 and the Grandparent-Connection (GP) is in Section 3.2. Section 3.3 analyzes the Grandparent-Connection algorithm. The GP algorithm is probabilistically complete and in convex configuration space produces the optimal path. The simulations confirming the results for the FR and GP algorithms are presented below. The simulations also compare our two algorithms to similar algorithms.

3.1 The Focused-Refinement Algorithm

As shown in [29], the RRT* initially constructs a tree that is the same as the RRT and then, as more nodes are added, the RRT* begins to look at many neighboring vertices to recover an asymptotically optimal path. The RRT* finds and refines all paths

in the configuration space. Refer to Section 2.1 for the detail of the RRT* algorithm. The refinement extension, Focused-Refinement (FR), focuses on refining only those paths that have already reached the goal region in hopes of reducing the amount of time needed to find a sufficiently optimal path.

The FR begins the construction of a tree using the RRT* algorithm until there exists at least one path that reaches the goal region. This set of paths is denoted as Π, with p vertices defining a set V_Π. The FR has two options: exploring the configuration space or exploiting Π to lower its cost. If exploring, the algorithm proceeds as the RRT*, but if exploiting, the set of vertices in Π, V_Π, is determined. The sample x_{new} is determined by perturbing a vertex randomly drawn from the set V_Π. The FR then proceeds as the RRT*.

The pseudo code for the FR is presented in Algorithm 4, and uses three parameters. The first is $C_{\text{exploit}} \in \mathbb{N}$, the number of consecutive iterations the FR will exploit Π. The number of consecutive iterations to explore X_{free} is the second parameter needed, $C_{\text{explore}} \in \mathbb{N}$. The third parameter, $C_{\text{reset}} \in \mathbb{N}$, tells the algorithm when to update V_Π. The sampling region defined by V_Π does not change dramatically every iteration, therefore, to save computation time, the set V_Π is only updated every $C_{\text{reset}} + C_{\text{explore}}$ iterations. If $C_{\text{exploit}} = 0$ and $C_{\text{explore}} = \infty$, the FR becomes the RRT*. In order to take advantage of the exploitation property of the FR, C_{exploit} should be greater than C_{explore}. In environments with multiple routes to the goal, C_{explore} can be increased in hopes of finding a better route than what was initially found.

Exploitation only occurs if GoalReach returns true (there exists at least one path to X_{G}) and exploitation has occurred less than C_{exploit} consecutive times. Once Π has been exploited C_{exploit} iterations, the RRT* is allowed to explore the space for C_{explore} iterations. The following are the details on choosing x_{new} during the exploitation stage of the FR.

The new sample, x_{new}, is determined as illustrated in Fig. 3.1 and in Algorithm 5. Given a d-dimensional configuration space, $X \subset \mathbb{R}^d$, consider $k \in \{1, 2, \cdots, d\}$. First, the minimum and maximum k-component from $V_\Pi \in \mathbb{R}^{d \times p}$, $w_{\min} = \min V_\Pi^k$ and $w_{\max} = \max V_\Pi^k$, are found. Here, V_Π^k is the set of all k-components of the vertices in V_Π. Next, the k-component of x_{new} (x_{new}^k) is taken as a uniformly random sample between $w_{\min} - \varepsilon$ and $w_{\max} + \varepsilon$, $\varepsilon > 0$. For every $j \neq k$, the j-component of the vertex whose k-component is nearest to x_{new}^k is determined, x_{nearest}^j, $x_{\text{nearest}} = \text{argmin}_{x \in V_\Pi} \ \|x_{\text{new}}^k - x^k\|$. The j-component of x_{new} is uniformly sampled between $x_{\text{nearest}}^j - \varepsilon$ and $x_{\text{nearest}}^j + \varepsilon$. The FR alternates which k-component is used to determine x_{new}, this provides a uniform distribution of samples around Π. As ε is increased, the entire configuration space is uniformly sampled randomly, thus recovering the original RRT*.

Note that V_Π can consist of multiple distinct paths to the goal. Determining distinct paths is non-trivial and potentially time-consuming. In general, and in the simulation section, V_Π is taken to be only the current best path. Efficiently determining distinct paths is a subject of future work. Because the FR restricts exploration of the entire free configuration space, it is not ideal for use in conjunction with the GT.

3.2 The Grandparent-Connection Algorithm

The Grandparent-Connection (GP) algorithm was inspired by reducing the number of nodes in, and cost of, a given path. Before adding a node to the tree, the modified algorithm attempts to connect directly to its grandparent node, as outlined in Algorithm 6. A successful connection to the grandparent occurs when a lower cost, collision-free path is found. It is also predicted that the GP algorithm will produce smoother paths with fewer nodes. Because the GP algorithm is applied during construction of the tree as every node is added to the tree, the grandparent connection smooths out every

Algorithm 4 $\mathcal{T} = (V, E) \leftarrow$ $\mathsf{FR}(x_i, \varepsilon, d, C_{\text{exploit}}, C_{\text{explore}}, C_{\text{reset}})$

$\mathcal{T} \leftarrow \mathsf{InitializeTree}()$;
$\mathcal{T} \leftarrow \mathsf{InsertNode}(\emptyset, x_i, \mathcal{T})$;
$c_{\text{reset}} = 1$; $c_{\text{exploit}} = 1$; $c_{\text{explore}} = 1$; $k = 1$;
for $i = 1$ to $i = N$ **do**
  if GoalReach and $c_{\text{exploit}} < C_{\text{exploit}}$ **then**
    if $c_{\text{reset}} = 1$; **then**
      $V_\Pi = \mathsf{PathSet}(\mathcal{T})$;
    end if
    $(c_{\text{reset}}, c_{\text{exploit}}) \leftarrow \mathsf{UpdateParameters}(c_{\text{reset}}, c_{\text{exploit}}, C_{\text{reset}})$;
    $x_{\text{new}} = \mathsf{NewPointPathSet}(V_\Pi, \varepsilon, k, d)$;
    $k \leftarrow \mathsf{UpdateDimension}(d)$;
  else
    $(c_{\text{explore}}, c_{\text{exploit}}) \leftarrow \mathsf{UpdateParameters}(c_{\text{explore}}, c_{\text{exploit}}, C_{\text{explore}})$;
    $x_{\text{rand}} \leftarrow \mathsf{Sample}(i)$;
    $x_{\text{new}} \leftarrow \mathsf{Nearest}(\mathcal{T}, x_{\text{rand}}, \varepsilon)$;
  end if
  $X_{\text{near}} \leftarrow \mathsf{Near}(\mathcal{T}, x_{\text{new}})$;
  $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$;
  if $x_{\text{parent}} \neq \mathsf{NULL}$ **then**
    $\mathcal{T} \leftarrow \mathsf{InsertNode}((x_{\text{parent}}, x_{\text{new}}), x_{\text{new}}, \mathcal{T})$;
    $\mathcal{T} \leftarrow \mathsf{Rewire}(\mathcal{T}, X_{\text{near}}, x_{\text{new}})$;
  end if
end for
return $\mathcal{T}$

Algorithm 5 $x_{\text{new}} \leftarrow \mathsf{NewPointPathSet}(V, \varepsilon, k, d)$

$w_{\min} = \min(V^k)$;
$w_{\max} = \max(V^k)$;
$x^k_{\text{new}} = \mathsf{Rand}(w_{\min} - \varepsilon, w_{\max} + \varepsilon)$;
$x_{\text{nearest}} = \mathsf{NearestComponent}(V, x^k_{\text{new}})$;
for $j = 1$ to $j = d$; **do**
  if $j \neq k$ **then**
    $v_{\min} = x^j_{\text{nearest}} - \varepsilon$;
    $v_{\max} = x^j_{\text{nearest}} + \varepsilon$;
    $x^j_{\text{new}} = \mathsf{Rand}(v_{\min}, v_{\max})$;
  end if
end for

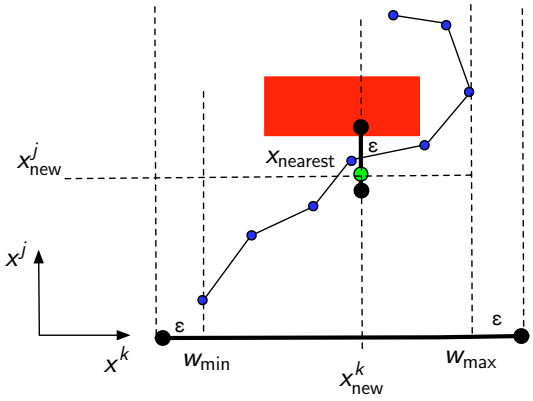


Figure 3.1: An illustrative example on choosing x_{new} when refining a single path. The red rectangle is an obstacle in the environment. The blue dots are the the set of vertices, V_{Π}, used to determine the region from which x_{new} is sampled. The k-component of x_{new} is a uniform random sample between the maximum and minimum (plus and minus ε, respectively) k-component values from V_{Π}. Next, with respect to x^k_{new}, determine the nearest k-component from V_{Π} and label its corresponding j-component as x^j_{nearest}. Finally, the x^j_{new} is a random value from between $x^j_{\text{nearest}} - \varepsilon$ and $x^j_{\text{nearest}} + \varepsilon$, $\varepsilon > 0$. Sample x_{new} is represented as the green dot.

path in the tree. The GP gains the advantage over smoothing a single path when paired with the Goal Tree (GT) algorithm, our novel replanning algorithm introduced in Chapter 4, or similar replanning algorithms. The grandparent connection can also be used in combination with the FR algorithm of Section 3.1.

3.3 Analysis of Grandparent-Connection Algorithm

The same probabilistic completeness and asymptotic optimality results for the RRT* algorithm in [29], also hold true for the Grandparent-Connection algorithm. Theorem 1 is a restatement of Theorems 23 and 38 from [29] but for the GP algorithm.

Proposition 2. If the RRT* and GP algorithms solve the same path planning problem, using the same parameters, then the vertex sets of both graphs are equal, $V_n^{\text{RRT}^*} = V_n^{\text{GP}}$, $\forall\, n \in \mathbb{N}$.

Proof. Proof by induction. When $n = 1$, no Grandparent-Connection is possible, there-

Algorithm 6 $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$

$x_{\text{parent}} \leftarrow \emptyset$;
$c_{\min} \leftarrow \infty$;
for $x_{\text{near}} \in X_{\text{near}}$ **do**
  $e_{\text{near,new}} \leftarrow \mathsf{Steer}(x_{\text{near}}, x_{\text{new}})$;
  $c_{\text{near}} \leftarrow \mathsf{Cost}(x_{\text{near}}) + c_{\text{edge}}(e_{\text{near,new}})$;
  if $c_{\text{near}} < c_{\min}$ and $\mathsf{CollisionFree}(e_{\text{near,new}})$ **then**
    $x_{\text{parent}} \leftarrow x_{\text{near}}$;
    $c_{\min} \leftarrow c_{\text{near}}$;
  end if
  if $x_{\text{parent}} \neq \emptyset$ **then**
    $x_{\text{grandparent}} \leftarrow \mathcal{T}.\mathsf{parent}(x_{\text{parent}})$;
    $e_{\text{grandparent,new}} \leftarrow \mathsf{Steer}(x_{\text{grandparent}}, x_{\text{new}})$;
    $c_{\text{grandparent}} \leftarrow \mathsf{Cost}(x_{\text{grandparent}}) + c_{\text{edge}}(e_{\text{grandparent,new}})$;
    if $c_{\text{grandparent}} < c_{\min}$ and $\mathsf{CollisionFree}(e_{\text{grandparent,new}})$ **then**
      $x_{\text{parent}} \leftarrow x_{\text{grandparent}}$;
      $c_{\min} \leftarrow c_{\text{grandparent}}$;
    end if
  end if
end for
return x_{parent};

fore, both the vertex and edge sets are identical. When $n = 2$, even though in the last step of the iteration the added vertex may connect to its grandparent the vertex is still added to the tree, making both vertex sets identical but the edges set different. When $n = 3$, because the vertex sets are the same, the vertex to be added, x_{new}, is the same for both algorithms, so is the set X_{near}. Then, if an edge exists between x_{new} and an $x \in X_{\text{near}}$ in one algorithm it also exists in the other. Therefore, even if the parents are different x_{new} will be added to both trees, maintaining the identical vertex sets. □

Theorem 1. *The GP algorithm is probabilistically complete. Furthermore, for any robustly feasible path planning problem* $(X_{\text{free}}, x_i, X_{\text{goal}})$*, there exist constants* $a > 0$ *and* $n_0 \in \mathbb{N}$*, both dependent only on* X_{free} *and* X_{goal}*, such that,*

$$\mathbb{P}(\{V_n^{\text{GP}} \cap X_{\text{goal}} \neq \emptyset\}) > 1 - e^{-an} \quad \forall\, n > n_0.$$

Also, if $\gamma > (2(1+\frac{1}{d})^{\frac{1}{d}}(\frac{\mu(X_{\text{free}})}{\zeta_d})^{\frac{1}{d}}$*, then the GP algorithm is asymptotically optimal.*

Proof. First, from Prop. 2, we have $V_n^{\text{RRT}^*} = V_n^{\text{GP}}$. Then, by construction, the GP builds a connected graph. Therefore, the probabilistic completeness of the GP follows from the probabilistic completeness of the RRT*. Finally, the asymptotic optimality result comes from $\text{cost}_n^{\text{RRT}^*}(x_i, x) \geq \text{cost}_n^{\text{GP}}(x_i, x)$ for all $x \in (V_n^{\text{RRT}^*} = V_n^{\text{GP}})$. □

Lemma 1. *When* X_{free} *is convex, the GP recovers the optimal path from* x_i *to* X_{goal}*. Furthermore, define the visibility set of* x_i *as the subset of* X_{free} *such that* $\forall\ x \in \mathit{Vis}(x_i)$ *there exists a collision-free geodesic from* x_i *to x. Then, the GP algorithm will recover the optimal paths in* $\mathit{Vis}(x_i)$*.*

Proof. By construction, with a convex X_{free}, every vertex in the graph has x_i as its parent. Then, extending to other metric spaces, any node in the visibility set has x_i as its parent. □

3.4 Simulations

There are two different robot types simulated in this section: Euclidean metric and Dubins' vehicle.

3.4.1 Euclidean Metric

This set of simulation results is for a point robot with no dynamics and Euclidean Metric edge cost. There are three environments, 25, 50, or 75 obstacles, which were chosen to compare how the different algorithms do with various obstacle densities. The results are an average of 25 simulations.

First, the Grandparent-Connection and Focused-Refinement are compared to the RRT*, RRT*-Smart, and RRT with path smoothing, Table 3.1. The algorithms were run for 100 seconds in the 25 obstacle environment, 120 seconds in the 50 obstacle environment, and 450 seconds in the 75 obstacle environment. The mean initial cost of each algorithm is compared to the RRT*'s mean initial cost to obtain the percent difference. A negative percent difference indicates that the cost is less than the RRT*'s cost. As expected the GP algorithms find the lowest cost initial path, without incurring much of a time increase. While the GP algorithms' initial path cost is about the same as the path found by an RRT with a post-processing path smoothing technique, the cost difference in the final path is much larger. The GP and FR algorithms have lower final path costs than the RRT*.

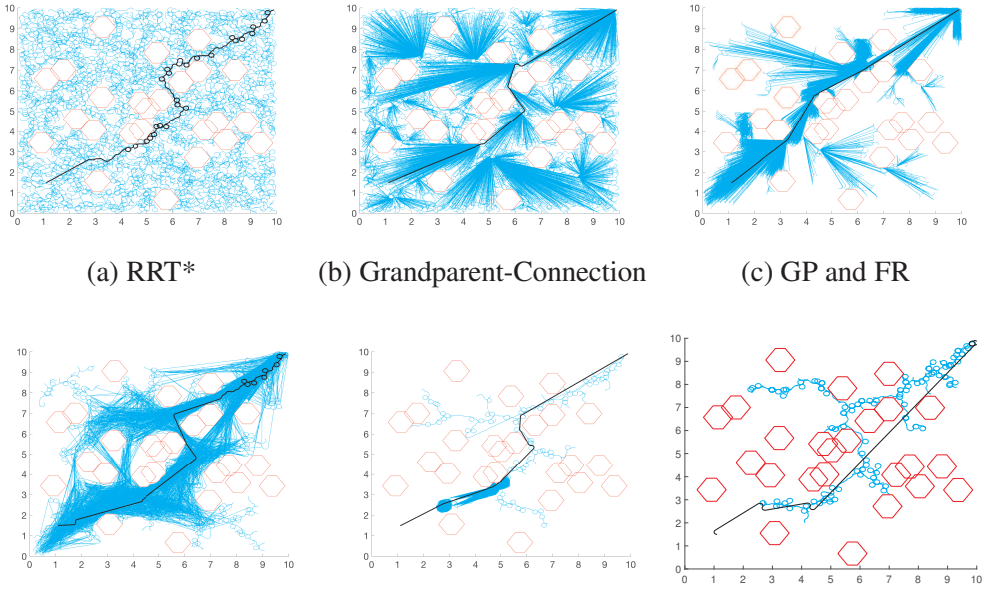
Table 3.1: The mean and standard deviation results for the Euclidean metric. This is a summarization that compares the Grandparent-Connection and Focused-Refinement algorithms to the RRT*, RRT*-Smart, and RRT with path smoothing.

		RRT*	GP	FR	FR-GP	RRT*-Smart	Smoothing
Initial Cost	25	17.65 (0.83)	14.78 (0.55)	17.65 (0.83)	14.78 (0.55)	17.65 (0.83)	15.13 (0.65)
	50	18.08 (1.01)	15.99 (1.02)	18.08 (1.01)	15.99 (1.02)	18.08 (1.01)	16.39 (0.98)
	75	18.84 (1.23)	17.90 (1.20)	18.84 (1.23)	17.90 (1.20)	18.84 (1.23)	18.74 (1.23)
% Difference	25	0	-16.3	0	-16.30	0	-14.29
	50	0	-11.53	0	-11.53	0	-9.32
	75	0	-4.98	0	-4.98	0	-0.53
Initial Time	25	13.77 (7.39)	12.98 (6.62)	13.77 (7.39)	16.20 (9.31)	9.69 (6.07)	11.92 (5.23)
	50	37.72 (16.82)	32.92 (15.38)	35.61 (16.53)	32.87 (15.30)	22.10 (11.32)	34.00 (13.64)
	75	139.3 (47.4)	115.7 (55.4)	140.2 (47.4)	119.4 (57.1)	137.2 (46.7)	81.19 (30.72)
Final Cost	25	15.33 (0.32)	14.60 (0.34)	14.59 (0.37)	14.54 (0.27)	15.01 (0.59)	15.13 (0.65)
	50	15.93 (0.50)	15.45 (0.78)	15.15 (0.64)	14.98 (0.64)	15.80 (0.72)	16.39 (0.98)
	75	16.85 (0.90)	16.60 (1.05)	16.67 (0.98)	16.46 (0.94)	16.46 (0.94)	18.74 (1.23)
% Difference	25	0	-4.79	-4.89	-5.21	-2.15	-1.33
	50	0	-2.98	-4.91	-5.95	-0.78	2.92
	75	0	-1.52	-1.12	-2.31	-2.31	11.2

3.4.2 Dubins' Vehicle

The simulations results in this section are for a Dubins' vehicle in the 25 and 50 obstacle environment. Each algorithm is averaged over 10 runs in each environments. Typical trees found by the RRT* and GT algorithm's are in Figs. 3.2a- 3.2f. The tree produced by the RRT* is the only one with many excessive loops in its final path.

Table 3.2 compares the Grandparent-Connection and Focused-Refinement algorithms to the RRT*, RRT*-smart, and RRT with path smoothing. The loops and curves of the path found by the RRT* increase the path cost significantly. This translates to the GP algorithms finding initial paths that have costs 50% less than the initial cost of the RRT*. The RRT with path smoothing and GP algorithms have initial path costs that are about the same. The excess loops and curves are not present in the GP algorithms final paths, and are reduced in the FR and RRT*-smart algorithms' final path.



(a) RRT* (b) Grandparent-Connection (c) GP and FR

(d) Focuses Refinement (e) RRT*-Smart (f) RRT + Path Smoothing

Figure 3.2: Typical Dubins' vehicle trees in the 25 obstacle environment found by the RRT*, Grandparent-Connection, Grandparent Connection with Focused-Refinement, Focused-Refinement, RRT*-Smart, and RRT with path smoothing algorithms.

Table 3.2: Mean with standard deviation results summarizing the comparison of the Grandparent-Connection and Focused-Refinement algorithms to the RRT*, RRT*-Smart, and RRT with path smoothing for Dubins' Vehicle.

		RRT*	GP	FR	FR-GP	RRT*-Smart	Smoothing
Initial Cost	25	342.1 (39.1)	132.6 (7.8)	342.1 (39.1)	132.6 (7.8)	342.1 (39.1)	126.9 (3.0)
	50	365.6 (45.4)	140.9 (51.5)	365.6 (45.4)	158.3 (19.2)	365.6 (45.4)	131.6 (7.4)
% Difference	25	0	-61.23	0	-61.27	0	-62.9
	50	0	-61.45	0	-56.71	0	-64.00
Initial Time	25	9.46 (9.02)	47.64 (44.17)	9.05 (8.71)	34.27 (17.51)	14.63 (15.93)	18.73 (11.73)
	50	34.70 (19.79)	130.5 (102.4)	44.63 (26.37)	116.2 (44.1)	38.82 (21.21)	187.6 (226.8)
Final Cost	25	300.1 (61.1)	131.2 (7.2)	260.0 (52.0)	128.7 (4.0)	134.4 (6.0)	126.9 (3.0)
	50	349.4 (32.2)	152.3 (12.0)	280.9 (28.3)	149.9 (16.2)	151.9 (11.8)	131.6 (7.4)
% Difference	25	0	-56.29	-13.36	-57.12	-55.72	-57.70
	50	0	-56.40	-19.59	-57.09	-56.53	-62.33

3.5 Summary

The Focused-Refinement, and Grandparent-Connection are the two algorithms presented in this chapter that improve the performance of the asymptotically optimal Rapidly-exploring Random Tree (RRT*). The GP algorithm is proven to maintain the asymptotic optimally of the RRT*. The simulations for the Euclidean metric show that our algorithms improve performance most in the initial cost. The Dubins' vehicle simulations show the GP and FR algorithms can lower the initial and final path cost significantly.

Publications associated with this chapter

Chapter 3, in part, contains material as it appears in ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference 2015. "Focused Refinement in the RRT*: Trading Optimality for Improved Performance", Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, contains material submitted to ASME Journal of Dynamic Systems, Measurement and Control 2017. "Improved Performance of Asymptotically Optimal Rapidly-Exploring Random Trees," Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 4

A Sampling-Based Algorithm for Replanning in Environments with Unknown Static Obstacles

This chapter extends the RRT* to allow for path planning when encountering unexpected static obstacles. Section 4.1 details the Goal Tree algorithm and then the algorithm is proven to produce an asymptotically optimal tree in Section 4.2. Fianlly, the Goal Tree and the Goal Tree with Grandparent-Connection algorithms are compared to the RRT* in Section 4.3.

4.1 The Goal Tree Algorithm

In this section, the Goal Tree (GT) algorithm is described in detail. The GT is a method for replanning when unexpected or moving obstacles obstruct the execution of the path determined by the RRT*. The RRT* produces paths that are asymptotically optimal from the initial configuration to any other point in the configuration space. By

a slight modification to the RRT* algorithm, one can produce a tree, $\mathcal{T}_G$, rooted at the goal configuration, x_G, such that the asymptotically optimal path from any point in X_{free} to x_G can be recovered. To do this, the cost associated with each edge $e_{1,2} = (v_1, v_2)$ in the RRT* tree becomes the *cost-to-come*; i.e. the cost of traveling from the child v_2 to the parent v_1.

Once the new obstacle, O, has been discovered, the GT trims $\mathcal{T}_G$ and then it is extended, use the RRT*, in some subset $R \subseteq X \setminus O$. To trim $\mathcal{T}_G$, the edges are checked for conflict with O and removed using PropagateCost. Instead of checking every edge in $\mathcal{T}_G$ we can define a subregion that contains all possible vertices whose trajectories are in conflict with O. We want to define the region as all points within some Euclidean distance from a point in O. Denote the center point of O as x_c, and the maximum Euclidean distance from x_c to the boundary of O as $r_{\max}$. A graph search is done over $\mathcal{T}_G$ to determine the maximum edge cost, $r_{\text{cost}} = \max_{e \in \mathcal{T}.E}\{c_{\text{edge}}(e)\}$. Since $c_{\text{edge}}(e_{1,2}) \geq \|x_1 - x_2\|$, the set of vertices whose trajectories are in conflict with O is contained in

$$V_{\text{conflict}} = \{v \in \mathcal{T}_G.V \mid \|x_c, x_v\| \leq r_{\max} + r_{\text{cost}}\}.$$

All trajectories of the vertices in V_{conflict} are checked for conflict with O. All vertices found in conflict with O, and their descendants, are trimmed from $\mathcal{T}_G$.

4.2 Optimality of Goal Tree algorithm

Reducing the sampling region for rebuilding $\mathcal{T}_G$ can lead to faster convergence but can also prevent global optimality. We first prove that there exist a generic restricted region of the space which can be used to extend $\mathcal{T}_G$ so that convergence to a globally optimal path is guaranteed. Then we aim to characterize these regions for common cases.

Theorem 2. *Let $X = [0,1]^d$ be a d-dimensional C-space, $d \in \mathbb{N}$ and $d \geq 2$. Let X_{obs} be the C-obstacles space. Assume O is newly found obstacle information, i.e. $O \not\subset X_{\text{obs}}$, and there exists a ball, $B(x_G, r) \subset (X_{\text{obs}} \cup O)^c$, $r > 0$. Suppose the feasible dynamic paths of vehicles in a free environment are at least $\mathcal{C}^3$. Then, there exists a generic $R \subsetneq X$ such that if $\mathcal{T}_G$ is originally built in X using the RRT* with information X_{obs}, then trimmed using O, and finally extended in R using the RRT* with information $X_{\text{obs}} \cup O$, then an optimal path, $\pi : x_I \to x_G$, can be asymptotically recovered by the GT algorithm as $n \to \infty$.*

Proof. The ball $B(x_G, r)$ is an obstacle free environment where the restricted optimal solution is a sufficiently smooth curve. A generic property of smooth curves is that they have a finite number of inflection points and vertices; see [9] focusing on planar curves, but from which results are valid for curves in any dimensions. Thus, there exists a final piece of the optimal path, say σ, to x_G which is convex or concave and does not contain any vertex or inflection point in it. Using σ, there exists a smaller radius $r' < r$ such that $B(x_G, r') \cap \sigma$ reduces to a single intersection point. The optimal path π from $x_{I'}$ to x_G must go through the $\partial B(x_G, r')$ at this point. Then, taking $R = X \setminus B(x_G, r')$ will yield π asymptotically. □

In d dimensional environments, a rebuilding region guaranteed to recovery an asymptotically optimal trajectory can be found as follows. Consider the goal and new initial configurations, $x_G$and $x_{I'}$, and a new obstacle O such that $x_{I'}, x_G \notin O$. For simplicity, assume that $O \cap X_{\text{obs}} = \varnothing$.

First, a region in the environment is defined and then, using this region for sampling, the GT is proven to recover a geodesic from $x_{I'}$ to x_G. Due to the obstacles in the environment, any configuration in X_{free} could have more than one geodesic to x_G. Note that in the following, the distinction is made between position and configuration. Position is the $(p_1, p_2, ...)$ position in the environment, while configuration can also include

orientations or velocities.

Definition 1. *The* shadow *of x_G on O, $\mathcal{S}_O$, is the envelope or hull, as defined by position rather than configuration, formed by the geodesics from all configurations in X_{free} going to x_G that are in conflict with O.*

Note that $x_{I'} \in \mathcal{S}_O$ must be true, otherwise, there is no need for replanning. Also note that $\mathcal{S}_O$ is a set of positions and not configurations. In this way each position could have an infinite number of possible configurations associated with it.

Definition 2. *Let $S \subset X$ be a set such that $x_{I'} \in S$ and whose boundary is denoted as ∂S. Then, an* outgoing configuration *on ∂S is defined as a configuration whose position is in ∂S and whose orientation or velocity will force the vehicle to leave S.*

Lemma 2. *All outgoing configurations on $\partial\mathcal{S}_O$ have geodesics to x_G that are not in conflict with O.*

Proof. Let x be an outgoing configuration on $\partial\mathcal{S}_O$. Consider a geodesic from x to x_G which is in conflict with O, then by the definition of an outgoing configuration on $\partial\mathcal{S}_O$, any motion from x forces the vehicle position strictly outside $\partial\mathcal{S}_O$. However, this is in contradiction with the definition of $\mathcal{S}_O$, which contains all positions obtained from geodesics to x_G that are in conflict with O. Therefore, there must only exist geodesics from x to x_G that are not in conflict with O. □

The main result, Theorem 3, states that using the shadow of x_G on O as the new sampling region will allow the Goal Tree algorithm to asymptotically recover an optimal path from $x_{I'}$ to x_G. Due to the tree structure used by the GT, only one of the geodesics from $x_{I'}$ to x_G will be recovered.

Theorem 3. *Let $\mathcal{S}_O$ be as in Definition 1. If the Goal Tree algorithm uses $\mathcal{S}_O$ as the new sampling region to rebuild $\mathcal{T}_G$, then it will converge to a globally optimal path as $n \to \infty$.*

Proof. Let π be an optimal path from $x_{I'}$ to x_G. If π lies entirely in $\mathcal{S}_O$, then, it will be recovered by sampling in $\mathcal{S}_O$. Otherwise, π must cross $\partial\mathcal{S}_O$ at an outgoing configuration. Let x_1 be the outgoing configuration in π that first crosses $\partial\mathcal{S}_O$. Then the subpath of π, from $x_{I'}$ to x_1, lies entirely in $\mathcal{S}_O$ and can be recovered by sampling in $\mathcal{S}_O$. By Lemma 2, a geodesic from x_1 to x_G is in $\mathcal{T}_G$. Thus, the GT algorithm can recover a geodesic from $x_{I'}$ to x_G by sampling in $\mathcal{S}_O$. □

By exploiting what is known about geodesics in the Euclidean metric, we can provide an alternative characterization of a feasible sampling region for use in the GT by a robot with no differential constraints.

Theorem 4. *Let X be a d-dimensional C-space such that $d \in \mathbb{N}$ and $d \geq 2$. Let the initial obstacle space be X_{obs} and let $O \not\subset X_{\text{obs}}$ be new obstacle information. For simplicity, assume that $O \cap X_{\text{obs}} = \varnothing$. If*

1. *X is the Euclidean metric space,*
2. *$O \subset R \subset X$,*
3. *R is convex, and*
4. *$x_{I'} \in R$*

then the GT algorithm will converge to a globally optimal path, π, as $n \to \infty$ by employing $\mathcal{T}_G$ with the previous X_{obs} and trimming $\mathcal{T}_G$ using the O information and then extending $\mathcal{T}_G$ in R.

Proof. In Euclidean space, an optimal path, π, is composed of straight lines and segments that follow the boundary of the obstacles. In particular, an optimal path from $x_{I'}$ to any point on ∂R, with respect to the new obstacle information, is a concatenation of path segments included among the following:

1. collision-free straight line paths from $x_{I'}$ to a point on the boundary of O; i.e., a *visible* point on ∂O from $x_{I'}$.

2. any path along the boundary of O and the boundary of the convex hull of O, and

3. collision-free straight line paths from ∂O to the visible boundary of R.

The convexity of R implies that all straight lines that begin and end in R are entirely contained in R. Any path that follows ∂O is entirely in R because $O \subset R$. A globally optimal path from $x_{I'}$ to x_G will have to cross ∂R if $x_G \notin R$. Let the boundary point at this crossing be x_B. By the above discussion, the subpath from $x_{I'}$ to x_B can be recovered asymptotically by means of sampling in R with the new obstacle information. Consider the optimal subpath from x_B to x_G with respect to the old obstacle information X_{obs}. By the same considerations as above, this optimal subpath is made of a concatenation of segments from the list above but with respect to X_{obs}. Thus, it can be asymptotically recovered by means of $\mathcal{T}_G$ with information in X_{obs}. □

Note that, if $O \cap X_{\text{obs}} \neq \varnothing$, then R would have to be a convex region containing the connected component of $O \cup X_{\text{obs}}$ that contains O. This connected set would then be used in the above proof in place of O.

The region characterization from Definition 1 can be used to approximately determine where to sample from the geodesics obtained from the initial tree for planning problems. However, and as for the Euclidean case, alternative regions can be used if the particular dynamics are amenable to direct analysis. The following leads to a characterization of a new sampling region, R, for use in rebuilding $\mathcal{T}_G$ during replanning with the Dubins' vehicle. The Dubins' vehicle has three states, x- and y-position and orientation

θ. The dynamics for the Dubins' vehicle are

$$\dot{x}(t) = v\cos(\theta)$$
$$\dot{y}(t) = v\sin(\theta)$$
$$\dot{\theta}(t) = u, \quad |u| \leq \frac{v}{\rho},$$

where v is the speed of the vehicle and ρ is the minimum turning radius. It is assume that both v and ρ are constant. The optimal trajectory between two configurations for these dynamics are discussed in [17]. The locally optimal trajectory defined by the above dynamics is one of six paths, RSL, LSR, RSR, LSL, RLR, and LRL, where L means left, R means right, and S means straight. Geodesics with respect to Euclidean length are characterized as concatenations of circular arcs and straight lines. The minimum turning radius for the Dubins' vehicle is denoted as ρ.

The following lemmas are useful in obtaining the main Dubins' vehicle result of this subsection.

Lemma 3. *Given a circular arc that begins at x_1 and ends at x_2, that has an angle strictly less than π radians, let x_c be the point where the tangent lines of the arc at x_1 and x_2 cross. Then, the* outer approximation *is defined as the union of the line from x_1 to x_c with the line from x_c to x_2. Then, the length of the outer approximation of a given circular arc is greater than or equal to the arc length.*

Lemma 4. *Given a circular arc that begins at x_1 and ends at x_2, define the* inner approximation *as the straight line connecting x_1 to x_2. Then, the length of the inner approximation of a given circular arc is less than or equal to the arc length.*

The proof of Lemma 3 and 4 follows directly from basic geometric considerations employing the triangular inequality and the convexity of circular arcs. It can be

seen that the result can be extended to any convex curve and any inner approximation defined using points on the curve and joining them through lines in a similar way.

Now, using O, a region that contains at least one valid path around O is defined.

Definition 3. *Define the region R_O as the smallest convex set that contains the union of O with circles of radius 2ρ centered at each corner of O.*

Now, R_O is extended to contain feasible paths from $x_{I'}$ to the previous region R_O.

Definition 4. *Define the region R, as the smallest convex region that contains R_O and $B(x_{I'}, 2\rho)$.*

The first Dubins' vehicle result of this subsection states the existence of valid trajectories in R.

Lemma 5. *The region R, as in Definition 4, contains at least one feasible Dubins' vehicle trajectory from $x_{I'}$ to any outgoing configuration on ∂R.*

Proof. First, it is shown that the Dubins' vehicle dynamics can be satisfied while traveling from $x_{I'}$ to any point on $\partial B(x_{I'}, 2\rho)$. To do this, note that the Dubins' vehicle can leave $x_{I'}$ and travel on a curve of radius ρ for π radians that places the vehicle on the boundary of R. This will put the Dubins' vehicle 2ρ way from $x_{I'}$ and the vehicle can now travel along $\partial B(x_{I'}, 2\rho)$. From $\partial B(x_{I'}, 2\rho)$ the Dubins' vehicle can travel to R_O along the tangent line forming part of the boundary of the convex hull between R_O and $B(x_{I'}, 2\rho)$. The Dubins' vehicle is now on ∂R_O, the circles centered at the corners with radius 2ρ allow it to stay on ∂R_O traveling completely around the obstacle. Given that ∂R is 2ρ from ∂O, implies that the vehicle can do a circular maneuver from some point on ∂R to get to a specific outgoing configuration on ∂R. Thus, R will contain at least one trajectory, not necessarily optimal, from $x_{I'}$ to any outgoing configuration on ∂R. □

The second result is that, the optimal path, from a configuration inside R to an arbitrary outgoing configuration on ∂R, will be entirely inside R.

Lemma 6. *Let σ be a feasible path for a Dubins' vehicle that starts at $x_{I'}$ and ends at an outgoing configuration $x_{\text{end}} \in \partial R$. If σ leaves and returns to R, then there exists another path, π, from $x_{I'}$ to x_{end}, that is entirely in R and that has a lower path length than σ.*

Proof. Define B_ρ to be a ball

1. of radius ρ
2. that is tangent to both ∂R and σ
3. that is contained in R, $B_\rho \subseteq R$.

Every time σ crosses ∂R, two such balls can be created. Take $B_{\rho 1}$ to be the B_ρ that is tangent to σ before leaving R and that is closest to where σ crosses back into R. Let x_1 be the configuration in σ that is tangent to $B_{\rho 1}$. Now, take $B_{\rho 2}$ to be the B_ρ that is tangent to σ after returning to R and that is closest to where σ crossed outside of R. Let x_2 be the configuration in σ that is tangent to $B_{\rho 2}$.

Break σ into three subpaths: $\sigma_{I'1}$ from $x_{I'}$ to x_1, σ_{12} from x_1 to x_2, and $\sigma_{2\text{end}}$ from x_2 to x_{end}. Now, construct π as follows. Let $\pi_{I'1} = \sigma_{I'1}$ and $\pi_{2\text{end}} = \sigma_{2\text{end}}$. The subpath π_{12} is taken to be the path from x_1 that travels along $\partial B_{\rho 1}$ until it reaches ∂R, concatenated with the path that travels along ∂R until it reaches $\partial B_{\rho 2}$, and concatenated with the path that travels along $\partial B_{\rho 2}$ until it reaches x_2.

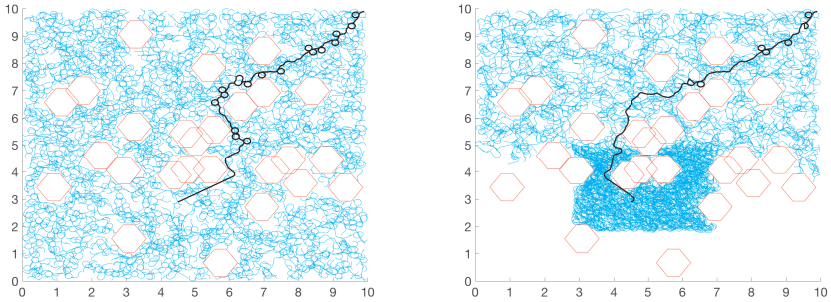
Now, π has been constructed to be a path, from $x_{I'}$ to x_{end}, that is entirely inside R. To prove that π has a shorter path length than σ, outer approximate every arc in π_{12}, π^o_{12}, and inner approximate every arc in σ_{12}, σ^i_{12}. This leaves us with two paths that start at x_1 and end at x_2 and that are only composed of straight lines. By construction, π^o_{12} is convex (because R and the outer approximation are convex). From the triangle

inequality and the convexity of π^o_{12}, the length of σ^i_{12} cannot be longer than the length of π^o_{12}. Which means the length of π_{12} must be less than the length of σ_{12}. Combining this with the fact that the length of $\pi_{I'1}$ is equal to the length of $\sigma_{I'1}$ and the length of $\pi_{2\text{end}}$ is equal to the length of $\sigma_{2\text{end}}$, the length of π is less than the length of σ. Therefore, any feasible Dubins' vehicle path that leaves and returns to R can be shortened to a path that is entirely inside R. □

The main Dubins' vehicle result of this subsection says that R is a sampling region that allows the GT to recover the optimal path. This is stated more precisely in Theorem 5.

Theorem 5. *Consider a Dubins' vehicle at $x_{I'}$ for which minimum-length paths are to be found to x_G. Assume that the new obstacle, O, is a convex polygon and does not intersect any other obstacles. Let R be as in Definition 4 and assume $x_G \notin R$. If the Goal Tree algorithm uses R as the new sampling region to rebuild $\mathcal{T}_G$, then it will converge to a globally optimal path as $n \to \infty$.*

Proof. By Lemma 5, there exists a path in R from $x_{I'}$ to any x_{end}, an outgoing configuration on ∂R. Then, by Lemma 6, if the optimal path π^* to any x_{end} leaves and returns to R, a shorter path can be found, which is a contradiction to the optimality of π^*. Therefore π^* must be entirely in R and can be found by sampling in R. Let x_B be the outgoing configuration on ∂R where the optimal path from $x_{I'}$ to x_G crosses outside of R. From the above, the path from $x_{I'}$ to x_B can be recovered asymptotically by sampling in R with the information from O. Now consider the path from x_B to x_G, this path is outside of R and can be asymptotically constructed by sampling outside of R with respect to X_{obs}. Thus, the optimal path from x_B to x_G can be recovered asymptotically from $\mathcal{T}_G$. □



(a) RRT*

(b) Goal Tree

Figure 4.1: Typical Dubins' vehicle trees found when replanning in the 26 obstacle environment using the RRT* and GT algorithms.

4.3 Simulations of the Goal Tree Algorithm

There are three different robot types simulated in this section: Euclidean metric, Dubins' vehicle, and a seven degree-of-freedom manipulator. These simulations find an approximation of the shadow for use as their sampling set when replanning. The shadow approximation, $\tilde{\mathcal{S}}$, is a collection of all the vertices removed from the original tree as a consequence of O. The $\tilde{\mathcal{S}}$ can be made denser by adding samples via the primitive NewPointPathSet.

Figs. 4.1a and 4.1b show two typical trees produced by the RRT* and GT algorithms when replanning. The original environment has 25 obstacles, one unknown obstacle is found, therefore the replanning is done in a 26 obstacle environment. The empty space in Fig. 4.1b is from trimming.

4.3.1 Euclidean Metric

Table 4.1 compares replanning in the 50 obstacle environment using the GT with the RRT* algorithms. The Goal Tree algorithm's initial path cost and its path cost at the mean time the RRT* first finds a path are compared to the initial path cost of the RRT*. The GT algorithm out performs the RRT* in initial path cost by 13%. The cost of

Table 4.1: Mean with standard deviation Euclidean metric results summarizing the comparison the Goal Tree and RRT* Algorithms.

		Time (s)	Cost	%
Initial	Goal Tree	0.69 (0.76)	7.23 (0.90)	-13.5
		2.02 (0.01)	7.19 (0.90)	-13.9
	RRT*	2.02 (2.16)	8.35 (1.33)	0
Final	Goal Tree	80.01 (0.01)	6.37 (0.14)	-2.8
	RRT*	80.01 (0.01)	6.55 (0.22)	0

Table 4.2: Mean with standard deviation results summarizing the comparison the Goal Tree and RRT* Algorithms for Dubins' Vehicles in the 25 obstacle environment.

		Time (s)	Cost	%
Initial	Goal Tree	0.16 (0.03)	260.8 (68.1)	4.5
		6.13 (0.03)	228.4 (44.5)	-8.4
	RRT*	6.10 (5.15)	249.6 (47.8)	0
Final	Goal Tree	80.03 (0.03)	209.8 (16.1)	-3.2
	RRT*	80.01 (0.01)	216.8 (21.8)	0

the final path found, after 80 seconds, by each algorithm is much closer, only a 2.7% difference.

4.3.2 Dubins' Vehicle

The Dubins' vehicle Goal Tree algorithm simulation comparison results are in Table 4.2. The GT algorithm's initial path cost and path cost at the time the RRT* finds an initial path are compared to the initial path cost of the RRT*. The GT has a mean initial cost that is higher than the RRT*, but that path is found much quicker. By the time the GT algorithm reaches the mean time it take the RRT* to find an initial path (6 seconds) the GT has reduced the path cost significantly. The GT, after 6 seconds, has a path cost 8% lower than the RRT* initial path cost. The comparison of the final path cost (at 80 seconds) drops to 3%.

The Grandparent-Connection can be incorporated into the Goal Tree Algorithm. Table 4.3 compares the Goal Tree with Grandparent-Connection (GT-GP) to the Grand-

Table 4.3: Mean with standard deviation results summarizing the comparison of the Goal Tree and Grandparent-Connection Algorithms for Dubins' Vehicles in the 25 obstacle environment.

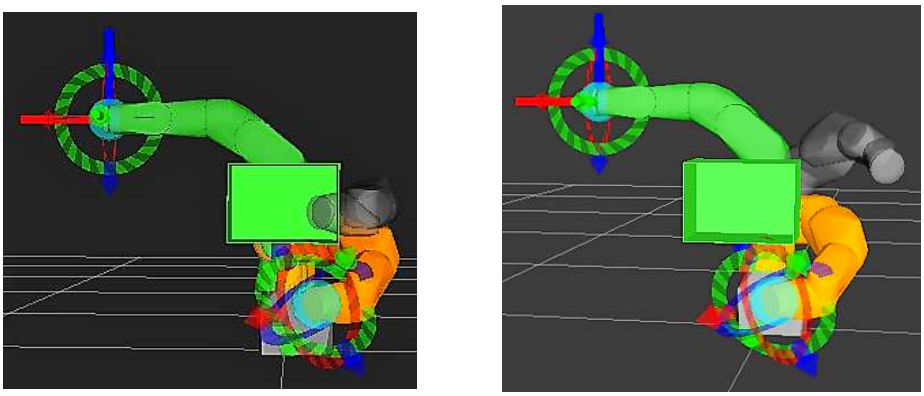
		Time (s)	Cost	%
Initial	GT-GP	4.97 (1.45)	106.9 (22.1)	3.2
		241.5 (0.3)	98.1 (15.1)	-5.4
	GP	240.6 (201.8)	103.6 (17.1)	0
Final	GT-GP	2500.6 (0.5)	92.92 (5.36)	-0.48
	GP	2500.7 (0.6)	93.36 (5.74)	0

parent - Connection algorithm. The GT with GP was able to find an initial path quicker in comparison to the GP. At the mean time the GP finds an initial path to the goal, the GT with GP has a path to the goal whose cost is 5.4% better. The final cost for both algorithms, after running for 2500 seconds, is very close, with only a 0.48% difference.

4.3.3 Seven Degree-of-Freedom Manipulator

A Motoman seven degree-of-freedom manipulator is simulated using MoveIt! in Robot Operating System (ROS). The Goal Tree and RRT* algorithms are run in the Open Motion Planning Library (OMPL). The RRT* finds the best path for the manipulator in an obstacle free environment. Next, a box that is in conflict with the manipulator's path is introduced to the environment, Fig. 4.2a. The Goal Tree and RRT* algorithms are then each used to replan and find a collsion-free path. The RRT* algorithm in the obstacle free environment is run 10 times. For each of the 10 obstacle free trees, the Goal Tree algorithm is run 10 times. Each 10 runs of the Goal Tree algorithm is referred to here as a set, therefore there are 10 sets of 10 runs each. Fig. 4.2b is a typical collision-free path found by the Goal Tree algorithm. For comparison, the RRT* algorithm is run in the same environment 25 times. The results of this setup are summarized in Table 4.4.

Table 4.4 compares the mean costs and initial iterations. The table shows that



(a) (b)

Figure 4.2: Left: The box is added to the environment so that it is in conflict with the manipulator's path. Right: The Goal Tree algorithm successfully replans to find a collision-free path.

the Goal Tree algorithm out performs the RRT* in the quality of the initial path. While the mean cost of the final path found by the Goal Tree algorithm is better than the mean cost of the RRT* algorithm it is not significantly less. The percent difference is calculated as the difference between the mean costs divided by the RRT* mean cost. These percentages reiterate that the Goal Tree algorithm finds a much better mean initial path compared to the RRT*. The mean number of iterations to find an initial path is significantly lower for the Goal Tree algorithm compared to the RRT*. This is because the Goal Tree algorithm reuses part of the original tree, therefore it already starts with nodes in the tree. At the bottom of the table is the failure rate that was encountered during the simulations. The percentage is calculated as the number of failures divided by the total number of attempted runs for each algorithm. The Goal Tree algorithm is probably less likely to fail because it reuses part of the original tree, therefore has a shorter path to determine.

Table 4.4: Mean results summarizing the comparison between the Goal Tree algorithm and RRT* algorithm for the seven degree-of-freedom manipulator.

Initial Cost	Goal Tree	8.83 (0.84)
	RRT*	9.66 (1.95)
Final Cost	Goal Tree	7.694 (0.65)
	RRT*	7.78 (0.93)
% Cost Difference	Initial Cost	-8.7%
	Final Cost	-1.1%
Initial Iterations	Goal Tree	19.54 (17.66)
	RRT*	143.0 (151.2)
% Failure	Goal Tree	-23.7%
	RRT*	-39.0%

4.4 Summary

The Goal Tree (GT) algorithm for replanning due to unexpected static obstacles is introduced and detailed. The GT algorithm is then proven to asymptotically optimal. Simulations show that the GT algorithm has improved performance compared to rerunning the RRT* algorithm. When run on the seven degree-of-freedom manipular, showed the most improvement in the initial path cost and showed that the initial path was found more quickly.

Publications associated with this chapter

Chapter 4, in part, contains material as it appears in the proceeding of the 52nd Annual Allerton Conference on Communication, Control, and Computing 2014. “Optimal kinodynamic motion planning in environments with unexpected obstacles”, Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, contains material submitted to ASME Journal of Dynamic Systems, Measurement and Control 2017. “Improved Performance of Asymptotically

Optimal Rapidly-Exploring Random Trees" Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 5

A Sampling-Based Motion Planning Algorithm for Replanning in Environments with Multiple Dynamic Agents

We extend from replanning due to static obstacles to replanning due to dynamic agents in this chapter. Specifically, we develop a novel approach, the SAMPLING-BASED COLLISION AVOIDANCE algorithm, to multi-agent motion planning where the other agents are not known perfectly.

There are N agents in an environment tasked with getting from their initial configurations, $x_{I,i}$, to their goal configurations, $x_{G,i}$. The agents must also never collide with any of the other agents or static obstacles. We say that, two agents $i, j \in \{1,\ldots,N\}$ are in conflict if, by staying on their current trajectory (heading and speed), they will eventually collide before reaching their destinations. Deconfliction is the act of an agent i changing its velocity to avoid future collision. Each agent, $i \in \{1,\ldots,N\}$ is approxi-

mated as a two dimensional ball, centered at x with radius ρ_i, $\mathbb{B}(x, \rho_i)$.

To reach its goal configuration while avoiding collision, the agent follows a predicted path from an RRT*. At each node in the path, the agent checks for conflict with the other agents. If there is no conflict, then the agent updates its velocity to follow the next edge in the predicted path. The path minimizes the distance traveled by the agent. When there is a conflict, the agent updates its velocity using a maneuver described in Section 2.3. This maneuver minimizes the change in velocity and not the distance traveled.

Agent i describes the environment with a configuration space, $X_i \in \mathbb{R}^2$. The agents are all in $\mathbb{R}^2$ therefore, $X_i = X_j = X$. The static obstacles, but not the other agents, are accounted for in the agent's obstacle space, $X_{\text{obs},i}$. Then, the free space for each agent is, $Q_i = X \setminus X_{\text{obs},i}$. Agent i has position, $\mathbf{r}_i \in Q_i$, and velocity, $\mathbf{v}_i \in \mathbb{R}^2$, both in the world frame. The final executed path for agent i, Π_i, is entirely in $X_{\text{free},i}$ and collision free with respect to the other agents.

Our solution to this problem requires communication between the agents. Initially we solve the problem with all agents communicating their position and velocity with all agents at every time step. Then, the solution is extended using an event-triggered approach to limit the communication by only requesting the position and velocity from a fellow agent when certain conditions are met. In the initialization of the problem, all agents are given the maximum velocity and the approximation radius, ρ_i, of all other agents.

Section 5.1 details the Sampling-Based Collision Avoidance (SBCA) algorithm that merges the RRT* (Section 2.1) with collision cones (Section 2.3). Section 5.2 presents results and Section 5.3 simulations for the SBCA algorithm.

5.1 The Sampling-Based Collision Avoidance Algorithm

This section presents two algorithms for multi-agent motion planning with collision avoidance in the presence of static obstacles. First, we combine the RRT* algorithm with the deconfliction strategy based on the collision cones of Section 2.3 for an algorithm under perfect information, see Section 2.1. This requires perfect communication among all agents. In order to reduce communications among agents, we then present an alternative algorithm in Subsection 5.1.2 that employs RRT*s with dynamically growing collision cones, in order to account for the uncertainty on other agents under sporadic communication.

Both algorithms are run on each agent independently of other agents, therefore they are described from the point of view of a single agent i. Each agent uses its motion planning tree $\mathcal{T}_i$, defined in Section 2.1, to determine, and update, its path to the goal and avoid the static obstacles. The collision cones are then used to keep the agent from colliding with other agents. Below, information is used in reference to position and velocity knowledge of the other agents.

5.1.1 Perfect Information Case

Pseudo code for the SAMPLING-BASED COLLISION AVOIDANCE with perfect information is presented in Algorithm 7. A quick overview of the algorithm that each agent runs is as follows. First, the agent extracts a path to the goal. While the agent is not at the goal and no conflicts are detected, it moves along its path. At each node in the path, the agent receives the current position and velocity from the other agents. If a conflict is detected, $\beta_{ij} < \alpha_{ij}$, for some other j, then the agent updates its own velocity and gets a new path to the goal. The agent also keeps track of the path it traveled. The details of each step are shown below.

Algorithm 7 $\Pi_i \leftarrow \mathsf{Sampling-BasedCollisionAvoidance}(x_{I,i}, x_{G,i}, \varepsilon)$

$[\mathcal{T}, \pi_i] \leftarrow \mathsf{RRT}*(x_{I,i}, x_{G,i}, \varepsilon)$;
$\mathbf{r}_i = x_{I,i}$;
$\Pi_i.\mathsf{add}(\mathbf{r}_i)$;
while $\mathbf{r}_i \neq \mathbf{x}_{G,i}$ **do**
  $\mathbf{r}_i, \mathbf{v}_i \leftarrow$ Move along path π_i to next node
  $\mathbf{r}_j, \mathbf{v}_j \leftarrow$ Communicate with ALL agents
  if Conflict ($\beta_{ij} < \alpha_{ij}$) with agent j **then**
    $\mathbf{v}_i \leftarrow$ Update velocity via maneuvers, Section 2.3;
    if $\mathbf{r}_i' \in X_{\text{obs},i}$ **then**
      Add obstacle to agent list
      Restart while iteration
    else
      $\pi_i \leftarrow GetPath(\mathcal{T}, \mathbf{r}_i, x_{G,i})$;
    end if
  end if
  $\Pi_i.\mathsf{add}(\mathbf{r}_i)$;
  Remove obstacle from agent list
  if $\mathbf{v}_j = \mathbf{0} \; \forall \; j$ **then**
    Replan
  end if
end while
$\mathbf{v}_i = \mathbf{0}$;
return Π_i

To initialize the algorithm, each agent recovers its best path to the goal from $\mathcal{T}_i$. The path for agent i is denoted as $\pi_i = \{x_i(t_1) = x_{I,i}, x_i(t_{i_2}), \ldots, x_i(t_{i_m}), \ldots, x_i(t_{i_f}) = x_{G,i}\}$, where $m \in \mathbb{N}$. Note that each agent may reach its specific goal at different times, t_{i_f}. Once at the goal the agents resets its velocity to zero, $\mathbf{v}_i = \mathbf{0}$. The algorithm also requires both the constant and variable speed agents to be able to stop.

As agent i travels along its path, it checks for conflict with the other agents $j \neq i$. For each node in the path π_i, agent i knows its current time t_{i_m}, position, $\mathbf{r}_i = \mathbf{x}_i(t_{i_m}) \in \pi_i$, and velocity $\mathbf{v}_i = v_i \frac{\mathbf{x}_i(t_{i_{m+1}}) - \mathbf{x}_i(t_{im})}{\|\mathbf{x}_i(t_{i_{m+1}}) - \mathbf{x}_i(t_{im})\|}$. The time to the next node in the path is $t_{i_{m+1}}$ and $dt = t_{i_{m+1}} - t_{i_m}$. Agent i receives the current position, $\mathbf{x}_j(t_{i_m})$, and current velocity, $\mathbf{v}_j(t_{i_m})$, from all agents $j \neq i$.

Once agent i has all the information for t_{i_m}, it calculates the collision cones. If there is a conflict, the new velocity vector, $\mathbf{v}'_i$, is calculated using one of the maneuvers from Section 2.3. When performing a variable speed maneuver that increases the speed, the next position of agent i, $\mathbf{r}'_i = \mathbf{r}_i + \mathbf{v}'_i dt$, will be farther away from $\mathbf{r}_i = \mathbf{x}_i(t_{i_m})$ than $\mathbf{x}_i(t_{i_{m+1}})$. When this occurs, dt is decreased so that $\mathbf{r}'_i = \mathbf{r}_i + \mathbf{v}'_i dt = \mathbf{x}_i(t_{i_{m+1}})$, $dt = \frac{\|\mathbf{x}_i(t_{i_{m+1}}) - \mathbf{r}_i\|}{\|\mathbf{v}'_i\|}$.

In some cases the new node added to the tree is really close to another node already in the tree, creating tiny edges. These tiny edges can cause the agent to spend more time doing algorithm calculations than moving. To avoid this, the edges can be removed as follows. Recall, the current path for agent i is $\pi_i = \{x_i(t_{i_1}), x_i(t_{i_2}), x_i(t_{i_3}), \ldots, x_i(t_{i_f})\}$. If an edge in π is less than 75% of the smallest edge length in the original tree, say for the first edge, $\|x_i(t_{i_1}) - x_i(t_{i_2})\| < 0.75 \min(\mathcal{T}^{\text{original}}.E)$, then $x_i(t_{i_2})$ is removed from π_i so that $\pi_i = \{x_i(t_{i_1}), x_i(t_{i_3}), \ldots, x_i(t_{i_f})\}$.

The collision cones do not account for the static obstacles, this is done to reduce computation time, and therefore, must make sure their next position, $\mathbf{r}'_i$, is not in collision with a known obstacle from X_{obs}. If the $\mathbf{r}'_i$ calculated is in collision with X_{obs}, then

the obstacle that is part of the collision needs to be accounted for when updating the agent's velocity. The obstacle is treated as an agent with zero velocity, $\mathbf{v}_{\text{obs}} = \mathbf{0}$. Agent i then recalculates an updated velocity vector. If there is not a feasible new velocity vector, then, agent i stops. Agent i can move again after at least one other agent has changed its velocity vector and if it finds a conflict free velocity vector.

If agent i was in conflict, the new velocity vector, $\mathbf{v}_i'$ is related back to $\mathcal{T}_i$. The new position, $\mathbf{r}_i'$, is added to $\mathcal{T}_i$ as x_{new} in the usual way, see Section 2.1. Once agent i adds $\mathbf{r}_i'$ to $\mathcal{T}_i$, the path to the goal, π_i is updated. In this way, the deconfliction maneuver will make sure the agents never collide and the underlying RRT* will make sure that the agents' reach their respective goal configurations.

5.1.2 Collision-Triggered Information Case

Here, we introduce uncertainty into the position and velocity knowledge of the other agents $j \neq i$. It is at the expense of this increased uncertainty that agents can reduce their communications with other agents. Thus, agent i only receives $\mathbf{r}_j$ and $\mathbf{v}_j$ when the uncertain information makes agent i think it is in collision or a solution to $\mathbf{v}_i'$ does not exist. In the first case, agent i only needs to update the $\mathbf{r}_j$ and $\mathbf{v}_j$ for the agent that is causing the "collision." In the second case, information from all other agents needs to be updated.

We introduce a few key differences with respect to Algorithm 7 when uncertainty is added. Algorithm 8 is the pseudo code for the new algorithm, with those differences highlighted in blue and italics. First, note that the variables $\mathbf{r}_j$ and $\mathbf{v}_j$ are not received at each path node but rather only when a "collision" is determined. Once that information is updated, the while loop iteration restarts. If a conflict is determined but there is no feasible $\mathbf{v}_i'$ then all $\mathbf{r}_j$ and $\mathbf{v}_j$ are updated and the loop is restarted.

Let t_{ij} be the time agent i last received $\mathbf{r}_j$ and $\mathbf{v}_j$ from agent j. Every time

Algorithm 8 $\Pi_i \leftarrow$ Sampling – BasedCollisionAvoidance

$[\mathcal{T}, \pi_i] \leftarrow \mathrm{RRT}*(x_{I,i}, x_{G,i}, \varepsilon)$;
$\mathbf{r}_i = x_{I,i}$;
Π_i.add($\mathbf{r}_i$);
while $\mathbf{r}_i \neq \mathbf{x}_{G,i}$ **do**
  $\mathbf{r}_i, \mathbf{v}_i \leftarrow$ Move along path to next node
  if $\|\mathbf{r}_i - \mathbf{r}_j(t_{ij})\| < \tilde{d}_{\text{sep},ij}$ **then**
    $\mathbf{r}_j, \mathbf{v}_j \leftarrow$ Communicate with agent j ONLY
    Restart While Iteration
  else if Conflict ($\beta_{ij} < \alpha_{ij}$) with agent j **then**
    $\mathbf{v}_i \leftarrow$ Update velocity
    if $\mathbf{v}_i$ Does Not Exist **then**
      for $j \neq i$ **do**
        $\mathbf{r}_j, \mathbf{v}_j \leftarrow$ Communicate with ALL agents
      end for
      Restart While Iteration
    end if
    if $\mathbf{r}'_i \in X_{\text{obs},i}$ **then**
      Add obstacle to agent list
      Restart while iteration
    else
      $\pi_i \leftarrow GetPath(\mathcal{T}, \mathbf{r}_i, x_{G,i})$;
    end if
  end if
  Π_i.add($\mathbf{r}_i$);
  Remove obstacle from agent list
  if $\mathbf{v}_j = \mathbf{0} \; \forall \; j$ **then**
    Replan;
  else if Livelock **then**
    $\mathbf{v}_i = \mathbf{0}$;
  end if
end while
$\mathbf{v}_i = \mathbf{0}$;
return Π_i

information is requested, $t_{ij} = t_{i_m}$, agent i calculates $\mathbf{v}_i'$ using perfect information from agent j. In time $\delta t = t_{i_m} - t_{ij}$ from the last information update, agent j can be anywhere within a ball centered at $\mathbf{r}_j$ with radius $v_{j,\max}\delta t$, $\mathbb{B}(\mathbf{r}_j, v_{j,\max}\delta t)$. Agent i estimates the velocity of agent j by calculating of $\tilde{\mathbf{v}}_j'$. The uncertain values are used in place of the certain values when calculating the collision cones and possible velocity update maneuvers. The separation distance is now, $\tilde{d}_{\text{sep},ij} = d_{\text{sep},ij} + v_{j,\max}\delta t$. The uncertain relative position is $\tilde{\mathbf{r}}_{ij} = \mathbf{r}_j(t_{ij}) - \mathbf{r}_i$ and the uncertain relative velocity is $\tilde{\mathbf{v}}_{ij} = \mathbf{v}_i - \tilde{\mathbf{v}}_j$. This leads to an uncertain half angle, $\tilde{\alpha}_{ij} = \arcsin\left(\frac{\tilde{d}_{\text{sep},ij}}{\|\tilde{\mathbf{r}}_{ij}\|}\right)$, and a uncertain angle between the relative velocity and position, $\tilde{\beta} = \arccos\left(\frac{\tilde{\mathbf{r}}_{ij} \cdot \tilde{\mathbf{v}}_{ij}}{\|\tilde{\mathbf{r}}_{ij}\|\|\tilde{\mathbf{v}}_{ij}\|}\right)$.

5.2 Analysis

This section details the analytical results for the Sampling-Based Collision Avoidance Algorithm. First, we prove that the algorithm using perfect information will never cause an agent to collide with another agent or a static obstacle. Next, we prove that the algorithm with uncertainty will also never cause a collision.

This first result, Lemma 7, guarantees that the agents will remain collision free while running the SBCA algorithm.

Lemma 7. *Let there be N agent in an environment with static obstacles all running the SBCA algorithm. If the agents begin collision free then, with probability one, they will stay collision free.*

Proof. There are two things an agent could collide with, other agents and static obstacles. First, we look at guaranteeing that agents will not collide with one another. Recall that the agents check and update their velocity at each node in the RRT* graph. Because the RRT* is built using random samples, the probability that more than one agent will update their velocity at the same time is zero. Every time an agent is in conflict, see

Proposition 1, it updates its velocity to be out of conflict with all other agents. This keeps the agent from colliding with another agent. Recall, that if the agent's updated velocity will put it in collision with a static obstacle it then treats the static obstacle as a static agent. Thus, we can use the above discussion on avoiding agents to again show the agent is collision free with static obstacles as well. Finally, we address the case when there are no feasible velocities. Here, the agent will stop and therefore not collide with any other agent or static obstacle. These three pieces combine to cover all possible scenarios an agent may encounter, thus keeping the agent collision free. □

There are two types of situations that may cause the agent to never reach their goal: deadlock and livelock. A *deadlock* occurs when none of the agents are able to find a feasible velocity and therefore must set their velocities to zero to avoid collision. A *livelock* is when the agents can find a feasible velocity but will never reach their goals.

A cyclic livelock is a specific type of livelock where the agents end up at the same set of nodes over and over. Lemma 8.

Lemma 8. *Agents using the SBCA algorithm will never enter a cyclic livelock.*

Proof. For an agent to arrive at a node in the graph the agent must be conflict free. If the agent arrived at the same set of nodes over and over, i.e. a cyclic path, all the nodes would be in the graph. Here, the graph is a tree, therefore it contains no cyclic paths. If the agent is in conflict, then it is creating new nodes; this new node has probability zero that it is already in $\mathcal{T}$ since the nodes are randomly sampled. Since this node has not been visited, the path created could not be cyclic and a cyclic livelock is avoided. □

The following is a procedure that will determine if the agents are in a livelock. To do this, we compare the M previously visited node in Π, Π_{last}. This sequence of M nodes is then compared to sequences of M nodes in Π. In particular, the agent takes one

node $x_m \in \Pi_{\text{last}}$ and find all nodes in $\Pi \setminus \Pi_{\text{last}}$, that are within $0.75 \min(\mathcal{T}^{\text{original}}.E)$ from x_m,

$$X_{\min} = \{x \in \Pi \setminus \Pi_{\text{last}} \mid \|x - x_m\| < 0.75 \min(\mathcal{T}^{\text{original}}.E)\}.$$

Now, the agent needs to check Π_{last} against the sequence of M nodes in Π that corresponds to $x \in X_{\min}$. If each corresponding node from the sequences is within $0.75 \min(\mathcal{T}^{\text{original}}.E)$ of each other, then the agent may be in a livelock. The agent then stops, $\mathbf{v}_i = \mathbf{0}$. If all of the agent stop then they can proceed with replanning as discussed in the deadlock case. This procedure on determining livelock is conservative in the sense that, if a livelock occurs the agents will stop, but they may also stop even when they are not in a livelock.

First, we introduce Assumption 1 on the deadlock situation. Assumption 1 says there will always exist a solution for at least one agent.

Assumption 1. At all times, there is at least one agent able to treat the other agents as enlarged static obstacles and replan using the RRT* to find a collision free path to the goal. Let agent i be at node $\mathbf{r}_i$ in path π and let the path after replanning be π_{new}. The other agents are enlarged to encompass how far that agent could travel in the time it takes for agent i to reach a node, $x \in \pi_{\text{new}}$, that has a lower cost-to-come compared to the cost-to-come before replanning, $\mathsf{Cost}(\mathbf{r}_i \in \pi) > \mathsf{Cost}(x \in \pi_{\text{new}})$.

To prove that the agents converge to their goal configurations, we need to show that the agent's cost-to-come satisfies the discrete time Lyapunov stability theorem. We do this by showing the cost-to-come is always decreasing. This is difficult to do with only local information and a decentralized algorithm.

The first part of the discrete time Lyapunov theorem is showing that the goal configuration is the only configuration that has a cost-to-come of zero. This is true by

construction of the RRT* tree. The second piece of the discrete time Lyapunov theorem is to prove that at each node the cost-to-come is decreased.

Under Assumption 1, we prove that, with probability one, the agents will always decrease their cost-to-come, when exiting a deadlock or livelock, Lemma 9

Lemma 9. *Let there be N agent in an environment with static obstacles all running the SBCA Algorithm 7. Let Assumption 1 hold and when an agent has reached its goal configuration, it will not keep any other agent from reaching its goal configuration. Then, with probability one, all agents will decrease their cost-to-come when exiting a deadlock or livelock.*

Proof. First, from Lemma 7, we know, with probability one, that the agents will never collide with one another or with the static obstacles in the environment. If all the agents stop, Assumption 1 says at least one will be able to replan, either using a replanning algorithm such as the Goal Tree or re-run the RRT* after adding the agents to $X_{\text{obs},i}$, and then find a collision free path to the goal. Also, from Assumption 1, we know that the one agent can travel along their new path without collision until the cost-to-come has decreased to below the cost-to-come before replanning.

Lemma 8 says the agents will never enter a cyclic livelock. The livelock case that is still unreasoned occurs when the agents are on a path whose limit is a cyclic path. In this case, the agent may never reach the cyclic path, but stays near the cyclic path, and therefore, never reach its goal. Once a livelock is discovered, the agents stop. Now, using Assumption 1 again, we follow the procedure outlined above for relieving deadlock. □

5.2.1 A New Deconfliction Maneuver

This section describes a new deconfliction maneuver that that minimizes the agent's cost-to-come instead of its change in velocity. This new maneuver allows the agents to all agents eventually reaching their goal configurations via a collision free path, with probability one.

The minimization problem for the new deconfliction maneuver is defined as,

Problem 1.

$$\begin{aligned} &\min \quad \mathsf{Cost}(\mathbf{r}_i) \\ \text{s.t.} \quad &\mathsf{Cost}(\mathbf{r}_i) \;=\; \mathsf{Cost}(\mathbf{r}'_i) + \|\mathbf{r}_i - \mathbf{r}'_i\| \\ &\mathsf{Cost}(\mathbf{r}_i \in \pi_i) \;\geq\; \mathsf{Cost}(\mathbf{r}_i \in \pi_{i,\text{new}}) \\ &\mathbf{r}'_i \in (X_{\text{near}} = \{x \in \mathcal{T} \mid \|x - \mathbf{r}_i\| \leq \delta_T\}) \\ &\alpha_{ij} < \beta_{ij}(\mathbf{v}'_i = \|\mathbf{r}'_i - \mathbf{r}_i\|). \end{aligned}$$

In other words, the cost-to-come is minimized such that the new cost-to-come is less than or equal to the current cost-to-come at $\mathbf{r}_i$. The next agent position $\mathbf{r}'_i$ will be within δ_T (see Section 2.1) of $\mathbf{r}_i$. Finally, the new velocity, $\mathbf{v}'_i = \|\mathbf{r}'_i - \mathbf{r}_i\|$, must make agent i conflict free.

In Theorem 6, agents using the new deconfliction maneuver are shown to decrease the cost-to-come at each step when subject to Assumption 1.

Theorem 6. *Let there be N agent in an environment with static obstacles all running the SBCA Algorithm 7 with the deconfliction maneuver that solves Problem 1 . Let Assumption 1 hold and when an agent has reached its goal configuration, it will not keep any other agent from reaching its goal configuration. Then, with probability one, all agents will decrease their cost-to-come at each step and converge to their goal configurations.*

Proof. The agent will either move to the next node in π_i, perform the new deconfliction maneuver, or stop.

First, we will examine the situation where the agent travels to the next node in π_i. Recall that the RRT* tree for each agent is asymptotically optimal with respect to the goal configuration for that agent. If the agent is traveling along a path extracted from its RRT* then the cost-to-come decreases,

$$\mathsf{Cost}(\mathbf{r}_i \in \pi_i) > \mathsf{Cost}(\mathbf{r}_i' \in \pi_i).$$

Secondly, we look at the deconfliction maneuver case. By changing the maneuvers to minimize the cost-to-come, Problem 1 (instead to the change in velocity), we can guarantee that at each step the cost-to-come decreases,

$$\mathsf{Cost}(\mathbf{r}_i \in \pi_i) > \mathsf{Cost}(\mathbf{r}_i' \in \pi_i).$$

Lastly, we address the situation where the agent stops. The agent could be part of a deadlock or the agent could start moving again because another agent moved.

When the agent stops (not in a deadlock) and is able to start moving again, the cost evolves as,

$$\mathsf{Cost}(\mathbf{r}_i \in \pi_i) = \mathsf{Cost}(\mathbf{r}_i \in \pi_i) > \mathsf{Cost}(\mathbf{r}_i' \in \pi_i).$$

Lastly, we address the deadlock situation. Let Assumption 1 hold. The deadlocked agents replan treating the other agents as static obstacles, $\mathbf{v}_j = \mathbf{0}$. Recall that the agents are overestimated to account for the movement before the SBCA algorithm can take over. When agent i stops at node $\mathbf{r}_i$, the cost-to-come stays constant, $\mathsf{Cost}(\mathbf{r}_i \in \pi_i) = \mathsf{Cost}(\mathbf{r}_i \in \pi_i)$, after replanning it may increase, $\mathsf{Cost}(\mathbf{r}_i \in \pi_i) \leq \mathsf{Cost}(\mathbf{r}_i' \in \pi_{i,\text{new}})$. Recall that from Assumption 1,agent i will move to node $\mathbf{r}_i'$ which will decrease the

cost-to-come,

$$\mathsf{Cost}(\mathbf{r}_i \in \pi_i) > \mathsf{Cost}(\mathbf{r}'_i \in \pi_{\mathrm{i,new}}).$$

Also, note that the other agents are moving along their paths in a similar fashion. The other agents are therefore decreasing their cost-to-come, $\mathsf{Cost}(\mathbf{r}_j \in \pi_j)$. From the above we can see that the cost-to-come (and sum of the cost-to-come) decreases at every node. Therefore, we have shown that the agents will asymptotically converge to their goal regions.

Finally, we prove that the agents will in fact reach their goal configurations in finite time. The sum of the cost-to-come of the agents cannot converge to a strictly positive value. This is due to the fact that, by construction of the RRT* trees, the edges have a cost that is uniformly lower bounded away from zero. Recall, the SBCA removes any edge that has a cost less than $0.75 \min(\mathcal{T}^{\mathrm{original}}.E)$. Therefore, the RRT* tree will never add an infinite number of edges between any two nodes already in the tree.

□

5.2.2 Collision-Triggered Algorithm Analysis

Now we turn to the algorithm with uncertain information. First, we look at a result that proves the existence of conflict free velocity due to the variable speed maneuvers, Proposition 3. The existence result mirrors Theorem 3 from [36] but for agents with no uncertainty. A similar result hold for the existence of a velocity due to constant speed maneuvers.

Proposition 3. For vehicle i of an N-vehicle system in the plane, let vehicle i's speed be constrained by $\|\mathbf{v}_i\| \leq v_{i,\max}$, while each other vehicles' speed is constrained by the uniform bound $\|\mathbf{v}_j\| \leq v_{\max}$. Let X_{obs} be the set of static obstacle. There exists an admissible velocity vector, $\mathbf{v}'_i$, which is conflict-free with the other $N-1$ vehicles and

X_{obs}, given that vehicle i is separated from the other vehicles such that

$$\sum_{j\in D} \tilde{\alpha}_{ij} \leq \begin{cases} \arcsin(\frac{v_{i,\max}}{v_{\max}}), & v_{i,\max} < v_{\max} \\ \frac{\pi}{2}, & v_{i,\max} \geq v_{\max}, \end{cases}$$

Proof. The uncertainty in velocity does not change the proof from Theorem 3 [36] because the worst case scenario is when the j agents are at max velocity. The uncertainty in the position is placed into the $d_{\text{sep},ij}$, therefore we use $\tilde{\alpha}_{ij}$ instead of α_{ij}. □

Next, the collision cone defined by the uncertain information is compared to the collision cone defined by the true information. Lemma 10 states that the true collision cone is contained within the uncertain collision cone.

Lemma 10. *If $\tilde{d}_{\text{sep},ij} \leq \|\mathbf{r}_i - \tilde{\mathbf{r}}_j\|$ for all time then $d_{\text{sep},ij} \leq \|\mathbf{r}_i - \mathbf{r}_j\|$ for all time.*

Proof. By definition of $\tilde{d}_{\text{sep},ij}$, we know $\tilde{d}_{\text{sep},ij} \geq d_{\text{sep},ij}$ and $\|\tilde{\mathbf{r}}_j - \mathbf{r}_j\| \leq v_{j,\max} dt = \tilde{d}_{\text{sep},ij} - d_{\text{sep},ij}$. The triangle inequality gives $\|\mathbf{r}_i - \mathbf{r}_j\| + \|\tilde{\mathbf{r}}_j - \mathbf{r}_j\| \geq \|\mathbf{r}_i - \tilde{\mathbf{r}}_j\|$. Putting everything together gives,

$$\begin{aligned} \tilde{d}_{\text{sep},ij} &\leq \|\mathbf{r}_i - \tilde{\mathbf{r}}_j\| \leq \|\mathbf{r}_i - \mathbf{r}_j\| + \|\tilde{\mathbf{r}}_j - \mathbf{r}_j\|, \\ \tilde{d}_{\text{sep},ij} &\leq \|\mathbf{r}_i - \mathbf{r}_j\| + \tilde{d}_{\text{sep},ij} - d_{\text{sep},ij}, \\ \tilde{d}_{\text{sep},ij} - \tilde{d}_{\text{sep},ij} + d_{\text{sep},ij} &\leq \|\mathbf{r}_i - \mathbf{r}_j\|, \\ d_{\text{sep},ij} &\leq \|\mathbf{r}_i - \mathbf{r}_j\|. \end{aligned}$$

□

The following Theorem 7 is the parallel result to Lemma 9 or Theorem 6, but for the collision-triggered algorithm.

Theorem 7. *Let there be N agent in an environment with static obstacles all running the SBCA algorithm 8. Let Assumption 1 hold and when an agent has reached its goal configuration, it will not keep any other agent from reaching its goal configuration. Then, with probability one, all agents will eventually reach their goal configurations via a collision free path.*

The proof for Theorem 7 follows that of Lemma 9 or Theorem 6, but using the uncertain values. If Lemma 9 or Theorem 6 holds, then, agent i is never within $\tilde{d}_{\text{sep},ij}$ of any agent j and will reach its goal configuration. Then, by Lemma 10, agent i is never within $d_{\text{sep},ij}$ of any agent j and will remain collision free.

5.3 Simulations

The following simulations involve eight agents initially spaced around the edge of the space. There are ten randomly placed static obstacles that must also be avoided. The underlying RRT* trees are constructed with respect to these obstacles. The agents are all approximated using the same radius, $\rho_i = \rho_j$ for all $i, j \in \{1, \ldots, N\}$. The maximum velocities for each agent are all different.

The black lines are the initial best path found by the tress. The colored paths are the final paths determined by the algorithm that avoids the agents and static obstacles.

The the separation distance between an agent i and all other agents j is shown in Fig. 5.2. The black line is the minimum separation distance, d_{sep}. There are three agents that get close to agent i but never violate the d_{sep} condition.

Figures 5.3a and 5.3b compare the true and estimated values of position and velocity. Both figures look at the percent difference, defined as $\% = \dfrac{\|\mathbf{r}_j - \tilde{\mathbf{r}}_j\|}{\|\mathbf{r}_j\|} * 100$. The dots indicate when agent i did a conflict check and the stars indicate the maximum uncertainty before the information was updated. In Fig. 5.3a, the uncertainty increases

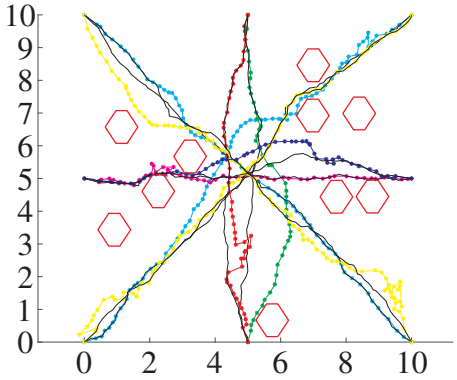


Figure 5.1: The initial paths (black) and final paths (colored). The red hexagons are static obstacles that must be avoided.

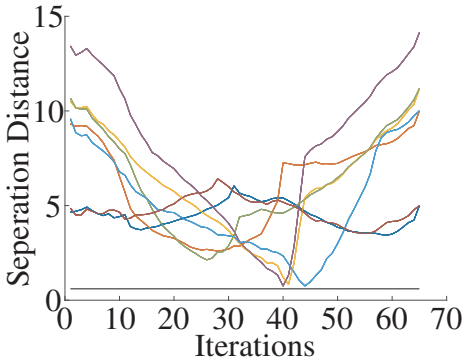


Figure 5.2: The distance between agent i and the other agents over time

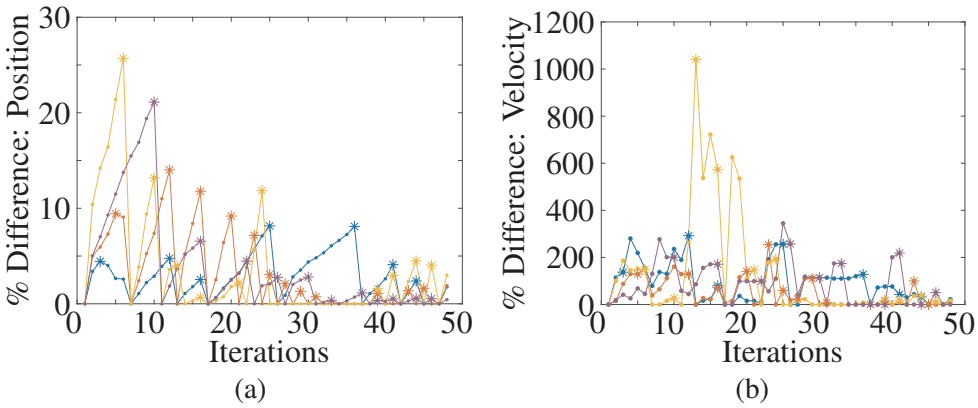


Figure 5.3: The difference between the true and uncertain position (left) and velocity (right) of the agents

monotonically until the information is updated. This is not the case with the velocity in Fig. 5.3b. The velocity estimation calculation allows the uncertainty to increase and decrease.

There are three different simulations compared in Table 5.1, each simulation has eight agents. The first two simulations are in the same environment depicted in Fig. 5.1 but with different underlying RRT* graphs. The third simulation was done in an environment without any static obstacles, but the same initial and final agent configurations as in Fig. 5.1. The results for each simulation are broke down by agent, number of algorithm iterations done by each agent, mean number of times an agent does not communicates with a particular agent, percentage of non-communication is calculated as the mean divided by number of iterations, peak difference columns compares the true value to the uncertain value, maximum error in the positions and velocities of the other agents, and finally, the mean peak error.

The mean peak looks at how much the error grows before a communication occurs. The final column is the maximum ρ value each agent saw during the algorithm. The most interesting thing to note is that in the second simulation there is an agent that communicates with all agents at every iteration. That same simulation had four of the

eight agents not communicating with other agents over half of the time. The mean percentage of non-communications for each simulation are not that far off from each other, indicating (at least in this case) that the obstacles do not affect the communication between the agents. The error also seems unaffected by the obstacles. This is because the obstacles are already accounted for in the underlying graph. The only time an obstacle would affect the algorithm would be if an agent was about to collide with one. An environment dense with obstacles could see an increase in communication compared to the same setup without any, or fewer, obstacles.

Table 5.1: Comparison of the results for three different simulations with eight agents each. The first two simulations have the same setup as Fig. 5.1. The third simulation has the same initial agent configuration as the other two but there are no static obstacles in the space.

		Non-Communications		Peak Difference: Position		Peak Difference: Velocity		
Agents	Iterations	Mean	Percentage	Maximum	Mean	Maximum	Mean	Max ρ
1	77	18.14	23.56	7.86	0.56	3.49	1.03	4.53
2	53	30.00	56.60	10.55	0.28	3.63	0.70	3.48
3	65	3.14	4.84	8.09	0.30	1.81	0.60	4.33
4	54	26.00	48.15	10.52	0.20	2.05	0.55	3.90
5	70	20.57	29.39	8.73	0.36	3.34	1.00	4.19
6	41	0.86	2.09	7.49	0.26	1.40	0.36	3.00
7	61	10.71	17.56	1.35	0.40	3.21	1.14	3.54
8	47	27.43	58.36	10.65	0.34	3.56	0.82	3.79
Mean	58.5	17.11	30.07	8.16	0.34	2.81	0.77	3.84
1	76	22.71	29.89	12.90	0.33	3.17	1.18	5.97
2	44	24.14	54.87	13.71	0.31	3.09	0.67	4.75
3	56	0 .00	0.00	0.00	0.00	0.00	0.00	0.50
4	52	29.71	57.14	13.61	0.37	3.07	0.70	4.71
5	68	18.29	26.89	12.51	0.40	2.86	0.94	6.00
6	48	28.43	59.23	13.63	0.31	3.12	0.63	5.42
7	81	15.00	18.52	7.76	0.41	2.98	1.17	4.54
8	50	27.43	54.86	13.70	0.26	3.02	0.73	4.30
Mean	59.3	20.71	37.67	10.98	0.30	2.66	0.75	4.52
1	67	12.86	19.19	1.34	0.40	2.49	0.78	3.10
2	42	22.86	54.42	10.13	0.42	2.67	0.76	3.32
3	65	29.71	45.71	9.32	0.40	3.18	0.87	3.50
4	48	10.00	20.83	7.64	0.79	2.27	1.23	4.27
5	70	11.86	16.94	7.53	0.56	2.84	1.09	4.62
6	47	25.14	53.50	10.09	0.41	3.07	0.71	3.25
7	67	13.43	20.04	2.82	0.61	2.86	1.52	4.24
8	49	31.00	63.27	10.34	0.40	2.99	0.81	3.61
Mean	56.9	19.61	36.74	7.40	0.50	2.79	0.97	3.74

5.4 Summary

This chapter detailed our algorithm for multi-agent path planning under uncertainty. The algorithm merges the sampling-based path planner RRT* with collision cones for collision avoidance. The algorithm is shown to handle uncertainty in an agent's knowledge of the position and velocity of the other agents. After the RRT* and collision cones are introduced the algorithm details are given. The algorithm is analyzed to prove that the algorithm with perfect and uncertain information will never cause the agents to collide. Simulations show that a collision free solution is found under uncertainty

knowledge.

Publications associated with this chapter

Chapter 5, in part, contains material that will be submitted, “Reactive Multi-Agent Path Planning: Combining the RRT* with Collision Cones” 2017. Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Sampling-Based Spatial Load Balancing for Multiple Robots

In this final chapter, we study a problem for multiple robots. Sampling-based motion planning is an integral part of our proposed algorithm, as it allows us to handle differential and obstacle constraints. Our research extends the notion of spatial load balancing when robots have limited travel ranges after reaching their equilibrium configuration.

6.1 Continuous Space Spatial Load Balancing

The limited range spatial load balancing problem aims to find optimal locations for n agents, with positions $P = \{p_1, \ldots, p_n\}$, $p_i \in Q$, $i \in \{1, \ldots, n\}$, and region assignments $W_i \subseteq Q$, $i \in \{1, \ldots, n\}$, as described below. Let the optimal cost of robot i to move from configuration $q_1 \in Q$ to $q_2 \in Q$ when subject to the differential constraint, $\dot{p}_i = f(p_i, u_i)$ with control input u_i, be $J(q_1, q_2) \geq 0$. A probability density function, $\phi(q)$, defined over Q, $\phi : Q \to \mathbb{R}_{\geq 0}$, describes the likelihood of an event occurring at a

configuration in Q.

Let $a_1, \ldots, a_n \in \mathbb{R}_{>0}$, be targeted cell areas that distribute the load of covering Q among the group of agents. The a_i, $i \in \{1, \ldots, n\}$, are such that $\sum_{i=1}^{n} a_i = \int_Q \phi(q)dq$, which can be understood as a full load-balancing condition. Ideally, we would like to achieve $\int_{W_i} \phi(q)dq = a_i$, for $i \in \{1, \ldots, n\}$, with the region assignment $\{W_i\}_{i=1}^{n}$. However, if the $\{W_i\}_{i=1}^{n}$ strictly satisfy $\cup_{i=1}^{n} W_i = Q' \subsetneq Q$, full load balancing will not be achievable. Because of this, a new variable area constraint is defined,

$$a_i' = \frac{a_i \int_{Q'} \phi(q)dq}{\int_Q \phi(q)dq}, \quad i \in \{1, \ldots, n\},$$

which corresponds to load-balancing over the restricted space Q'. Of particular interest is the equal area case, $a_i = a_j$ for all i, j, which results in $a_i' = a_j'$, for all $i, j \in \{1, \ldots, n\}$. Note that, when $Q' = Q$, we recover the original, full load-balancing condition. Then, the objective is to find positions, p_i, and regions, $W_i \subseteq Q$, for $i \in \{1, \ldots, n\}$, that solve the following minimization problem subject to the area and dynamic constraints,

Problem 2 (Multicenter Optimization Problem with Dynamic and Area Constraints)**.**

$$\begin{aligned} \min \quad & \mathcal{H}(P, \mathcal{W}) \\ \text{s.t.} \quad & p_i \in Q, \quad \dot{p}_i = f(p_i, u_i), \\ & a_i' = \int_{W_i} \phi(q)dq, \quad i \in \{1, \ldots, n\}. \end{aligned}$$

In other words, the n agents want to minimize a cost functional while making sure each agent's cell, W_i in the partition $\mathcal{W}$, has area a_i'. The agents cannot leave Q and must obey the differential constraints that define their movement. In what follows, we describe specific $\mathcal{H}$ and types of subpartitions motivated by coverage control objectives.

6.1.1 Unlimited Range Agents in Convex Spaces

The solution to Problem 2 with limited ranges builds on the previous work in [14, 48, 49] where Q is convex and the agents have unlimited range such that $\cup_{i=1}^{n} W_i = Q$. Then $a_i = a_i'$ and the cost function that is minimized takes the form,

$$\mathcal{H}^{\text{centroid}}(P, \mathcal{W}) = \sum_{i=1}^{n} \int_{W_i} J(p_i, q)\phi(q)dq.$$

The cost function $\mathcal{H}^{\text{centroid}}(P, \mathcal{W})$ quantifies the network performance and is called centroid because the agent positions that minimize it are the centroids of the cells. This is the problem solved in [14].

For trivial first order dynamics, the results in [14] state that, given a set of positions, P, there exists a weight assignment, ω, that makes a generalized weighted Voronoi partition, $\mathcal{V}^{\text{weighted}}(P, \omega; J)$, feasible and that this partition is the best among all the partitions $\mathcal{W}$ that satisfy the area constraint. Here, $\mathcal{V}^{\text{weighted}}(P, \omega; J) = \{V_i^{\text{weighted}}(\omega)\}_{i=1}^{n}$ is such that, for all $i \in \{1, \ldots, n\}$,

$$V_i^{\text{weighted}}(\omega) = \{q \in Q \mid J(p_i, q) - \omega_i \leq J(p_j, q) - \omega_j, \forall\, i \neq j\}.$$

The feasible set of weights is $U = \{\omega \in \mathbb{R}^n \mid |\omega_i - \omega_j| \leq J(p_i, p_j) \;\; i, j \in \{1, \ldots, n\}\}$. If $\omega \notin U$, then at least one cell is empty. In convex environments, given a partition, the best agent positions are the centroids of their cells. For certain metrics, such as Euclidean metrics, these centroids are given by a closed-form formula. However, in non-convex environments multiple agent positions may minimize the cost function, referred to as a generalized centroid.

6.1.2 Limited Ranges

When the agents have a limited travel range, referred to as limited range, a sub-partition, $\cup_{i=1}^{n} W_i \subset Q$, is found and $a_i \geq a_i'$. Here, limited travel range refers to the maximum distance an agent can travel from its final coverage configuration position. Since the area of the sub-partition changes as a function of agent position, the cost function being minimized is modified to account for the current area covered by the agents. This leads to a cost function that either maximizes the area covered by the regions,

$$\mathcal{H}^{\text{area}}(P, \mathcal{W}) = -\sum_{i=1}^{n} \int_{W_i} \phi(q) dq,$$

or combines $\mathcal{H}^{\text{centroid}}(P, \mathcal{W})$ and $\mathcal{H}^{\text{area}}(P, \mathcal{W})$ in the convenient form of,

$$\mathcal{H}^{\text{mixed}}(P, \mathcal{W}) = \sum_{i=1}^{n} \left(\int_{W_i} J(p_i, q)\phi(q) dq - k_i \int_{W_i} \phi(q) dq \right),$$

where $k_i \in \mathbb{R}_{>0}$, are constants, see Section 6.1.3 for a particular choice.

Limited Range Sub-Partition

Define the limited ranges of the agents as the reachable sets, $\mathcal{D} = \{D_1, \ldots, D_n\}$, such that, for all $i \in \{1, \ldots, n\}$,

$$D_i = \{q \in Q \mid J(p_i, q) - \omega_i + \frac{1}{n} \sum_{k=1}^{n} \omega_k \leq c\},$$

where $c \in \mathbb{R}$ is the travel range of the agents and is a constant. When $\omega_i = \frac{1}{n} \sum_{k=1}^{n} \omega_k$, then $J(p_i, q) \leq c$ making c the upper bound, or limited range, on the cost $J(p_i, q)$. When $\omega_i \neq \frac{1}{n} \sum_{k=1}^{n} \omega_k$, those terms act as a perturbation on the limited range c. This is why the affective limited range for each agent can be different. There are certain properties

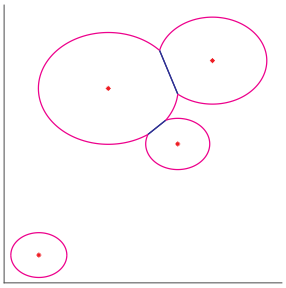


Figure 6.1: An example of a limited range sub-partition for four agents, each with a different ω_i.

of $\mathcal{V}^{\text{weighted}}$ from [14], that the limited range sub-partition should maintain in order to obtain analogous results. One such property is that the cells, V_i^{weighted}, are invariant under translation of ω, $V_i^{\text{weighted}}(P,\omega;J) = V_i^{\text{weighted}}(P,\omega + t\cdot\mathbf{1}_n;J)$. This property leads to the area of V_i^{weighted} also being invariant under translations of ω. Including the mean of ω term allows the set D_i to be invariant under translation in ω. The limited range sub-partition is defined as $\mathcal{V}^{\text{LR}}(P,\omega,c) = \{V_i^{\text{LR}}\}_{i=1}^n$, such that, for all $i \in \{1,\ldots,n\}$, $V_i^{\text{LR}} = V_i^{\text{weighted}} \cap D_i$. An example of a limited range sub-partition can be found in Fig. 6.1. The shared boundaries are from V_i^{weighted} while the unshared arcs are from D_i. Note that one of the agents has no shared boundaries, so its cell $V_i^{\text{LR}} = D_i$. In this example, the D_i boundaries are circular arcs because the cost is the Euclidean norm squared, $J(p_i,q) = \|p_i - q\|^2$.

The limited range sub-partition defines a graph which is used to describe the algorithm and its properties. This graph, $\mathcal{G}_{\text{LR}}(P,\omega) = (\mathcal{N},\mathcal{E})$, has vertices, $v_i \in \mathcal{N}$, that correspond to the n agents. In this graph, $e = (v_i,v_j) \in \mathcal{E}$, between agents i and j, if and only if $V_i^{\text{LR}} \cap V_j^{\text{LR}} \neq \emptyset$. If agents i and j share an edge in $\mathcal{G}_{\text{LR}}(P,\omega)$, $(v_i,v_j) \in \mathcal{E}$, then agents i and j are neighbors. Let $\mathcal{N}_i$ denote the set of neighbors for agent i. When $Q' \neq Q$, $\mathcal{G}_{\text{LR}}(P,\omega)$ may not be connected. The edges of $\mathcal{G}_{\text{LR}}(P,\omega)$ change as the agent positions and weights are updated. Because D_i, and hence V_i^{LR}, is dependent upon all agent weights, $\mathcal{G}_{\text{LR}}(P,\omega)$ is not necessarily representative of the agents' communication

graph. The remainder of the chapter will abbreviate $\mathcal{G}_{\mathrm{LR}}(P,\omega)$ as $\mathcal{G}_{\mathrm{LR}}$. This graph is analyzed further in Section 6.3.2.

Let η be the number of connected components in $\mathcal{G}_{\mathrm{LR}}$ and $\mathcal{G}_l$ be a single connected component of $\mathcal{G}_{\mathrm{LR}}$ such that $\mathcal{G}_{\mathrm{LR}} = \cup_{l=1}^{\eta} \mathcal{G}_l$ and $\mathcal{G}_l \cap \mathcal{G}_{l'} = \emptyset$ for all $l, l' \in \{1,\ldots,\eta\}$ and $l \neq l'$. Denote the number of vertices in $\mathcal{G}_l$ as n_l. Define the vector $\mathbf{v}_l \in \mathbb{R}^n$, such that, if agent i is in $\mathcal{G}_l$ then the i^{th} entry of $\mathbf{v}_l$ is a one and a zero otherwise. Note that $\{\mathbf{v}_l\}$ for $l \in \{1,\ldots,\eta\}$ form an orthogonal basis.

Existence and Choice of Weights

This section proves the existence of weights that allow $\mathcal{V}^{\mathrm{LR}}$ to satisfy the variable area constraint. Recall, $\mathcal{V}^{\mathrm{LR}}$ is invariant under translations in the weights and define the weights-to-area map as,

$$\mathcal{M}(P,\omega) = \left(\int_{V_1^{\mathrm{LR}}(\omega)} \phi(q)dq, \ldots, \int_{V_n^{\mathrm{LR}}(\omega)} \phi(q)dq \right).$$

For conciseness, in the following $V_i^{\mathrm{LR}} = V_i^{\mathrm{LR}}(\omega)$.

Lemma 11. *The weights-to-area map, $\mathcal{M}$, is gradient, $\nabla F = -\mathcal{M}$, where $F : \mathbb{R}^n \to \mathbb{R}$,*

$$F(\omega) = \frac{-n}{1-n} \sum_{j=1}^{n} \int_{V_j^{\mathrm{LR}}} (J(p_j,q) - \omega_j + \frac{1}{n} \sum_{k=1}^{n} \omega_k - c)\phi(q)dq.$$

Proof. Take the derivative of $F(\omega)$ with respect to ω_i, using the Leibniz rule [20], which

applies over general domains,

$$\begin{aligned}\frac{\partial F(\omega)}{\partial \omega_i} =& \frac{-n}{1-n}\Bigg(\sum_{j=1}^{n} \int_{V_j^{\mathrm{LR}}} -[\frac{\partial q}{\partial \omega_i} \quad \times (\frac{\partial}{\partial q}(J(p_j,q)-\omega_j+\frac{1}{n}\sum_{k=1}^{n}\omega_k - c)\phi(q))]\cdot dq \\ &+ \sum_{j=1}^{n}\int_{\partial V_j^{\mathrm{LR}}} \frac{\partial q}{\partial \omega_i}\cdot((J(p_j,q)-\omega_j+\frac{1}{n}\sum_{k=1}^{n}\omega_k - c)\phi(q))dq \\ &+ \sum_{j=1}^{n}\int_{V_j^{\mathrm{LR}}} \frac{\partial}{\partial \omega_i}\Bigg((J(p_j,q)-\omega_j+\frac{1}{n}\sum_{k=1}^{n}\omega_k - c)\phi(q)\Bigg)dq\Bigg).\end{aligned}$$

The first term,

$$\sum_{j=1}^{n}\int_{V_j^{\mathrm{LR}}} -[\frac{\partial q}{\partial \omega_i}\times(\frac{\partial}{\partial q}(J(p_j,q)-\omega_j+\frac{1}{n}\sum_{k=1}^{n}\omega_k - c)\phi(q))]\cdot dq = 0.$$

There are two vectors in the (q_1,q_2) plane being crossed, resulting in a vector perpendicular to the (q_1,q_2) plane, in dot product with a vector in the (q_1,q_2) plane, thus resulting in a zero value. The second term becomes

$$\begin{aligned}&\sum_{j=1}^{n}\int_{\partial V_j^{\mathrm{LR}}} \frac{\partial q}{\partial \omega_i}\cdot((J(p_j,q)-\omega_j+\frac{1}{n}\sum_{k=1}^{n}\omega_k - c)\phi(q))dq \\ =&\sum_{j=1}^{n}\int_{\triangle_{ij}} \hat{n}^{\top}\frac{\partial q}{\partial \omega_i}((J(p_j,q)-\omega_j+\frac{1}{n}\sum_{k=1}^{n}\omega_k - c)\phi(q))dq \\ +&\sum_{j=1}^{n}\int_{\Lambda_j} \hat{n}^{\top}\frac{\partial q}{\partial \omega_i}((J(p_j,q)-\omega_j+\frac{1}{n}\sum_{k=1}^{n}\omega_k - c)\phi(q))dq.\end{aligned}$$

This term then vanishes along $\triangle_{ij}$ because the opposing normal vectors are multiplying the same terms which then cancel each other out. For all $q \in \Lambda_j$, $J(p_j,q)-\omega_j+$

$\frac{1}{n}\sum_{k=1}^{n}\omega_k - c = 0$. The third term,

$$
\begin{aligned}
&\sum_{j=1}^{n}\int_{V_j^{\mathrm{LR}}}\frac{\partial}{\partial\omega_i}\Big((J(p_j,q)-\omega_j+\frac{1}{n}\sum_{k=1}^{n}\omega_k - c)\phi(q)\Big)dq \\
&=\sum_{j=1}^{n}\int_{V_j^{\mathrm{LR}}}\Big(\cancelto{0\,\forall\, j}{\frac{\partial J(p_j,q)}{\partial\omega_i}}\phi(q) - \cancelto{1,\, i=j}{\frac{\partial\omega_j}{\partial\omega_i}}\phi(q) + \frac{1}{n}\sum_{k=1}^{n}\cancelto{1,\, i=k}{\frac{\partial\omega_k}{\partial\omega_i}}\phi(q) - \cancelto{0\,\forall\, j}{\frac{\partial c}{\partial\omega_i}}\Big)\phi(q) \\
&\quad + (J(p_j,q)-\omega_j+\frac{1}{n}\sum_{k=1}^{n}\omega_k - c)\cancelto{0\,\forall\, j}{\frac{\partial\phi(q)}{\partial\omega_i}}dq \\
&=\frac{1-n}{n}\int_{V_i^{\mathrm{LR}}}\phi(q)dq.
\end{aligned}
$$

Putting everything together results in

$$\frac{\partial F}{\partial\omega_i} = -\int_{V_i^{\mathrm{LR}}}\phi(q)dq = -\mathcal{M}_i(\omega),\ \ i\in\{1,\ldots,n\}.$$

□

The following shows how to update an initial weight assignment, ω^0, to new weights that satisfy the area constraints.

Define $A \triangleq \sum_{i=1}^{n}\mathcal{M}_i(\omega^0)$ and

$$\bar{a}_i \triangleq A\frac{a_i}{\int_Q\phi(q)dq}. \tag{6.1}$$

In the equal area case, $a_i = a_j = \frac{1}{n}\int_Q\phi(q)dq$ and $\bar{a}_i = \frac{A}{n}$. Note that $\mathcal{M}_i(\omega^0) = \bar{a}_i$ does not necessarily hold. Define, $U^0 = \{\omega\in U \mid \sum_{i=1}^{n}\mathcal{M}_i(\omega) = \sum_{i=1}^{n}\mathcal{M}_i(\omega^0)\}$. Restrict $\mathcal{M}$ to $\omega\in U^0$, and denote this restriction as $\mathcal{M}^0$.

Lemma 12, which gives properties for the Jacobian of $\mathcal{M}^0$, is analogous to Prop. IV.2 from [14] but has a more complicated proof due to the sub-partition, $\mathcal{V}^{\mathrm{LR}}$. Note that $\mathcal{M}^0$ is a continuous function of ω.

Lemma 12. *Let $J(\mathcal{M}^0)$ denote the Jacobian matrix of $\mathcal{M}^0 : U^0 \subset \mathbb{R}^n \to \mathbb{R}^n$. Then*

1. *$J(\mathcal{M}^0)$ is symmetric;*

2. *Choose $\omega \in U^0$, and consider the graph $\mathcal{G}_{\mathrm{LR}}(P,\omega)$ with η connected components and associated $\mathbf{1}_n$ and $\mathbf{v}_l$, for all $l \in \{1,\ldots,\eta\}$. Then, these are eigenvectors of $J(\mathcal{M}^0)(\omega)$ with eigenvalue 0;*

3. *The rank of $J(\mathcal{M}^0)(\omega)$ is $n-\eta$.*

Proof. Fact 1 follows from $\mathcal{M}$, and thus $\mathcal{M}^0$, being gradient. For Fact 2, by definition, the sum of $\mathcal{M}^0$ stays constant when ω is updated, therefore, the range of $\mathcal{M}^0$ is in $\{m \in \mathbb{R}^n_{\geq 0} \,|\, \mathbf{1}_n^\top m = A\}$, and thus, $\mathbf{1}_n^\top J(\mathcal{M}^0) = \mathbf{0}_n$. Because $J(\mathcal{M}^0)$ is symmetric, $\mathbf{1}_n$ is a right eigenvector with eigenvalue 0. Recall that there are η connected components to $\mathcal{G}_{\mathrm{LR}}$ and that each connected component, $\mathcal{G}_l$, has a corresponding vector $\mathbf{v}_l$ associated with it. Then, because each $\mathcal{G}_l$ maintains a constant area during the weights update, additional eigenvectors with eigenvalue zero can be defined, $\mathbf{v}_l^\top J(\mathcal{M}^0) = \mathbf{0}_n$. For fact (3), first show that for $i \in \{1,\ldots,n\}$ and $(\omega_1,\ldots,\omega_i,\ldots,\omega_n)$, $(\omega_1,\ldots,\omega_i',\ldots,\omega_n)$ such that $\omega_i' \geq \omega_i$,

$$\begin{aligned}
\mathcal{M}_i^0(\omega_1,\ldots,\omega_i',\ldots,\omega_n) \geq& \mathcal{M}_i^0(\omega_1,\ldots,\omega_i,\ldots,\omega_n),\\
\mathcal{M}_j^0(\omega_1,\ldots,\omega_i',\ldots,\omega_n) \leq& \mathcal{M}_j^0(\omega_1,\ldots,\omega_i,\ldots,\omega_n),\ j \neq i.
\end{aligned}$$

Recall, $V_i^{\mathrm{LR}} = V_i^{\mathrm{weighted}} \cap D_i$. First, how $\partial V_i^{\mathrm{weighted}}$ changes with respect to ω is examined. From Eq. 6.1.1, when $q \in \triangle_{ij}$ then $J(p_i,q) - \omega_i = J(p_j,q) - \omega_j$. If ω_i increases, then the boundary $\triangle_{ij}$ moves further away from p_i which increases the area of V_i^{weighted} and decreases the area of V_j^{weighted}. Second, examine the change in the boundary of D_i. Let $J_{\max}(p_i,q)$ be the cost at the boundary of D_i, then $J_{\max}(p_i,q) = c + \omega_i - \frac{1}{n}\sum_{k=1}^n \omega_k$. If ω_i increases then so does $J_{\max}(p_i,q)$ which in turn increases the area of D_i. For D_j, if

ω_i increases then $J_{\max}(p_j, q)$ decreases and hence the area of D_j decreasesWhitehaven. Then, the partial derivatives of $\mathcal{M}^0$ for $\mathcal{V}^{\mathrm{LR}}$ satisfy

$$\frac{\partial \mathcal{M}_i^0}{\partial \omega_i} \geq 0, \quad \frac{\partial \mathcal{M}_j^0}{\partial \omega_i} \leq 0, \quad j \neq i.$$

The above, when combined with Facts 1 and 2, leads to $J(\mathcal{M}^0)$ being the Laplacian of $\mathcal{G}_{\mathrm{LR}}$. Because $\mathcal{G}_{\mathrm{LR}}$ has exactly η connected components, the $\mathrm{rank}(J(\mathcal{M}^0)) = n - \eta$. □

From here, the existence of a weight assignment that satisfies the $\bar{a}_i$ constraint can be proven. The proof of Theorem 8 follows that of Prop. IV.4 from [14] except that here $\mathcal{G}_{\mathrm{LR}}$ is not necessarily connected.

Theorem 8. *Let $a_1, \ldots, a_n > 0$ such that $\sum_{i=1}^n a_i = \int_Q \phi(q)dq$ and let $p_1, \ldots, p_n \in Q$. Let there exist some initial weights, ω^0, such that $\mathcal{M}_i(\omega^0) > 0$ for each $i \in \{1, \ldots, n\}$. Let $\{\bar{a}_1, \ldots, \bar{a}_n\}$ be as defined in* (6.1)*. Then there exists a set of weights $\omega = \{\omega_1, \ldots, \omega_n\}$ such that*

$$\int_{V_i^{\mathrm{LR}}(P,\omega,c)} \phi(q)dq = \bar{a}_i, \quad i \in \{1, \ldots, n\}.$$

Proof. Consider the function $G : U^0 \to \mathbb{R}$ defined as

$$G(\omega) = \frac{1}{2}\|\mathcal{M}^0(\omega) - (\bar{a}_1, \ldots, \bar{a}_n)\|^2. \tag{6.2}$$

Let ω^* be the minimizer of Eq. 6.2, then show that the value of the minimum is zero. Evaluate the derivative of Eq. 6.2 with respect to ω at ω^*,

$$\begin{aligned} 0 &= \frac{\partial}{\partial \omega_i}\bigg|_{\omega=\omega^*} \left(\frac{1}{2}\|\mathcal{M}^0(\omega) - (\bar{a}_1, \ldots, \bar{a}_n)\|^2\right) \\ &= \sum_{k=1}^{n} (\mathcal{M}_k^0(\omega^*) - \bar{a}_k) \frac{\partial \mathcal{M}_k^0}{\partial \omega_i}\bigg|_{\omega=\omega^*}, \ \forall\, i \in \{1, \ldots, n\}, \end{aligned}$$

which can then be expressed as $(\mathcal{M}^0(\omega^*) - (\bar{a}_1, \ldots, \bar{a}_n))J(\mathcal{M}^0)(\omega^*) = \mathbf{0}_n$. The weights-to-area map, $\mathcal{M}^0(\omega)$, is differentiable in the same way that $F(\omega)$ from Lemma 11 is differentiable. Recall there are eigenvectors such that $\mathbf{v}_l^\top J(\mathcal{M}^0)(\omega^*) = \mathbf{0}_n$ for all $l \in \{1, \ldots, \eta\}$ and that the $\text{rank}(J(\mathcal{M}^0)(\omega^*)) = n - \eta$. From here, deduce that $\mathcal{M}^0(\omega^*) - (\bar{a}_1, \ldots, \bar{a}_n) = \sum_{l=1}^{n} \beta_l \mathbf{v}_l$ for some $\beta_l \in \mathbb{R}$. Next, notice that

$$0 = \mathbf{v}_l^\top (\mathcal{M}^0(\omega^*) - (\bar{a}_1, \ldots, \bar{a}_n)) = \beta_l n_l,$$

and therefore $\beta_l = 0$ for all $l \in \{1, \ldots, \eta\}$, or $\mathcal{M}^0(\omega^*) = (\bar{a}_1, \ldots, \bar{a}_n)$. □

6.1.3 Continuous Space Algorithm

This section describes the algorithm used to solve Problem 2. The algorithm alternately updates the agents' weights and Voronoi cells until the area of the cells has converged to the desired areas. Then, the agents update their positions. Concisely, the algorithm dynamics can be expressed as,

$$\begin{pmatrix} \omega^+ \\ P^+ \end{pmatrix} = \psi \begin{pmatrix} \omega \\ P \end{pmatrix},$$

where ψ is a function combining (6.3) and (6.4). The details of the dynamic weight and position update are presented next.

Weights Update

From Theorem 8 and Lemma 12, there exists weights for which $\mathcal{V}^{\text{LR}}(\omega)$ satisfies the area constraints while maintaining constant area in each connected component of $\mathcal{G}_{\text{LR}}$. Instead of maintaining constant areas by numerically solving for the n^{th} weight, the next procedure is followed in our algorithm. All the weights are it-

eratively updated so they eventually converge to ω^*, such that, $V_i^{\mathrm{LR}}(P,\omega^*) = \bar{a}_i$, thus preserving, $\sum_{i=1}^{n} \mathcal{M}_i(P,\omega^0) = \sum_{i=1}^{n} \mathcal{M}_i(P,\omega^*)$. First, define $\mathcal{F}(\omega) = -F(\omega) - \sum_{i=1}^{n} \omega_i \bar{a}_i$, and set $\nabla\mathcal{F}(\omega) = g(\omega) = \mathbf{0}_n$, where $g(\omega) = \mathcal{M}(\omega_1,\ldots,\omega_n) - (\bar{a}_1,\ldots,\bar{a}_n) = \mathbf{0}_n$. The Jacobi algorithm, [3],

$$\omega^+ = \omega - \gamma\,\mathrm{diag}\Big(\frac{\partial g_1}{\partial \omega_1},\ldots,\frac{\partial g_n}{\partial \omega_n}\Big)^{-1} g(\omega), \tag{6.3}$$

is then used to find the ω values that optimize $\mathcal{F}$. The step size can be characterized as in [14] Prop. IV.5 to guarantee convergence in the weights. More precisely, let $\mathcal{L}$ be a level set of $\mathcal{F}(\omega)$, and then, $\gamma \in (0,\, Y/B)$, where,

$$Y = \min_{i\in\{1,\ldots,n\}} \min_{\omega\in\mathcal{L}} \frac{\partial g_i(\omega)}{\partial \omega_i} > 0, \quad B = \max_{i\in\{1,\ldots,n\}} \max_{\omega\in\mathcal{L}} \frac{\partial g_i(\omega)}{\partial \omega_i} > 0.$$

To implement this algorithm, the agents each need to compute their $g_i(\omega) = \mathcal{M}_i(\omega) - \bar{a}_i$ and $\frac{\partial g_i}{\partial \omega_i}$, where,

$$\frac{\partial g_i(\omega)}{\partial \omega_i} = \frac{\partial \mathcal{M}_i(\omega)}{\partial \omega_i} - \cancelto{0}{\frac{\partial \bar{a}_i}{\partial \omega_i}} \int_{\Lambda_i} \hat{n}^\top \frac{\partial q}{\partial \omega_i} \phi(q)dq + \int_{\triangle_{ij}} \hat{n}^\top \frac{\partial q}{\partial \omega_i} \phi(q)dq.$$

Gradient Function Computation

The agents update their positions according to the derivative of $\mathcal{H}(P, \mathcal{V}^{\mathrm{LR}}(P,\omega,c))$ with respect to position to solve Problem 2. The gradient computation details can be found in [15]. For a general $\mathcal{H}$, the dynamics for agent i are

$$p_i^+ = p_i - h\frac{\partial \mathcal{H}(P, \mathcal{V}^{\mathrm{LR}}(P,\omega,c))}{\partial p_i}, \tag{6.4}$$

where h is an appropriate step size, found via a line search. Eq. 6.4 only works for convex Q and when the agents are not subject to differential constraints, $\dot{p}_i = f(p_i, u_i)$. The graph-based algorithm is introduced specifically to handle these issues, see Section 6.2.2.

For the area only cost function, the gradient is

$$\frac{\partial \mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}}(P, \omega, c))}{\partial p_i} = -\int_{\Lambda_i} \phi(q) \hat{n}^\top \frac{\partial q}{\partial p_i} dq. \tag{6.5}$$

Here, q are at the unshared boundary configurations, $q \in \Lambda_i = \partial V_i^{\text{LR}} \cap D_i$, and $\hat{n}$ is the vector normal to the boundary at q. In other words, the agents move toward the (weighted) center of the unshared boundary of V_i^{LR}, and stay put when $\Lambda_i = \emptyset$.

When using $\mathcal{H}^{\text{mixed}}(P, \mathcal{V}^{\text{LR}})$, the right selection of k_i reduces the gradient computation to moving to a generalized centroid of V_i^{LR},

$$\frac{\partial \mathcal{H}^{\text{mixed}}(P, \mathcal{V}^{\text{LR}}(P, \omega, c))}{\partial p_i} = \int_{V_i^{\text{LR}}} \frac{\partial J(p_i, q)}{\partial p_i} \phi(q) dq$$
$$- \int_{\Lambda_i} J(p_i, q) \phi(q) \hat{n}^\top \frac{\partial q}{\partial p_i} dq + k_i \int_{\Lambda_i} \phi(q) \hat{n}^\top \frac{\partial q}{\partial p_i} dq.$$

For $J(p_i, q) = \|p_i - q\|^2$ or $J(p_i, q) = \|p_i - q\|$, choose $k_i = \mathcal{R}_i \triangleq c + \omega_i - \frac{1}{n} \sum_{k=1}^{n} \omega_k$.

6.2 Graph-based Limited Range Spatial Load Balancing

This section details the graph-based version of Problem 2. As a preprocessing step, a PRM*, G, is constructed in the non-convex environment Q, see Chapter 2.2. The cost to travel between two configurations (q_1, q_2), $J(q_1, q_2)$, is approximated by the sum of edge costs of the best path in G from q_1 to q_2. Define a sub-partition of a subset of

$\mathcal{N}_G$ as $\widetilde{\mathcal{W}} = \{\widetilde{W}_i\}_{i=1}^n$, such that $\cup_{i=1}^n \widetilde{W}_i \subseteq \mathcal{N}_G$ and $\widetilde{W}_i \cap \widetilde{W}_j = \emptyset$.

Let $\tilde{a}_1, \ldots, \tilde{a}_n \in \mathbb{R}_{>0}$, such that $\sum_{i=1}^n \tilde{a}_i = \sum_{q \in \mathcal{N}_G} \phi(q)\beta(q)$, then define the approximate variable area constraint as

$$\tilde{a}_i' = \frac{\tilde{a}_i \sum_{q \in \widetilde{\mathcal{W}}} \phi(q)\beta(q)}{\sum_{q \in \mathcal{N}_G} \phi(q)\beta(q)}, \qquad i \in \{1, \ldots, n\}.$$

The approximated area covered by $q \in \mathcal{N}_G$, $\beta(q)$, is pre-computed as described in Chapter 2.2. The $\sum_{q \in \widetilde{\mathcal{W}}} \phi(q)\beta(q)$ requires knowledge from all agents and varies with each algorithm iteration.

The n agents solve graph-based Problem 3, which approximates the integral as a summation over a set of nodes.

Problem 3 (Graph-Based Multicenter Optimization Problem with Area Constraints)**.**

$$\begin{aligned} \min \quad & \widetilde{\mathcal{H}}(P, \widetilde{\mathcal{W}}) \\ \text{s.t.} \quad & p_i \in \mathcal{N}_G, \\ & \tilde{a}_i' = \sum_{q \in \widetilde{W}_i} \phi(q)\beta(q), \quad i \in \{1, \ldots, n\}. \end{aligned}$$

Note that the differential constraint on p_i is removed because it is incorporated into G.

The cost function that the agents minimize is an approximation to either $\mathcal{H}^{\text{centroid}}$, $\mathcal{H}^{\text{area}}$, or $\mathcal{H}^{\text{mixed}}$, given as

$$\widetilde{\mathcal{H}}^{\text{centroid}}(P, \widetilde{\mathcal{W}}) = \sum_{i=1}^n \sum_{q \in \widetilde{W}_i} J(p_i, q)\phi(q)\beta(q),$$

$$\widetilde{\mathcal{H}}^{\text{area}}(P, \widetilde{\mathcal{W}}) = -\sum_{i=1}^n \sum_{q \in \widetilde{W}_i} \phi(q)\beta(q),$$

and

$$\widetilde{\mathcal{H}}^{\text{mixed}}(P,\widetilde{\mathcal{W}}) = \sum_{i=1}^{n}\Big(\sum_{q\in\widetilde{W}_i} J(p_i,q)\phi(q)\beta(q) - k_i \sum_{q\in\widetilde{W}_i}\phi(q)\beta(q)\Big).$$

To solve Problem 3 algorithmically, each agent has a copy of G and it is assumed that the agents know the positions P and the weights ω of the other agents by communicating with each other to interchange this information. The assumption on P and ω can be relaxed in some cases so that it is only necessary to know the positions and weights of a subset of the other agents; this will be discussed further in Section 6.3.

6.2.1 Approximate General Voronoi Tessellations

Different options are considered for an *approximate generalized Voronoi partition*. For conciseness, $\widehat{\mathcal{V}}$ is used to represent all approximated Voronoi partitions. In other words, if a results pertains to all the approximate Voronoi partitions, we use $\widehat{\mathcal{V}}$ (e.g. $\widehat{\mathcal{V}} = \widetilde{\mathcal{V}}^{\text{weighted}}$ or $\widehat{\mathcal{V}} = \widetilde{\mathcal{V}}^{\text{LR}}$ in the following). When the agents have unlimited range, such that $\cup_{i=1}^{n} W_i = \mathcal{N}_G$, they find the weighted approximate Voronoi partition, $\widetilde{\mathcal{V}}^{\text{weighted}} = \{\widetilde{V}_i^{\text{weighted}}\}_{i=1}^{n}$,

$$\widetilde{V}_i^{\text{weighted}} = \{q\in\mathcal{N}_G \mid J(p_i,q) - \omega_i \le J(p_j,q) - \omega_j,\ \forall\, j\neq i\}.$$

Here, $J(p_i,q)$ is the minimum sum of the edge costs of the optimal path in G defined from p_i to q. Due to the random selection of q when building G, the probability that, in $\widehat{\mathcal{V}}$, one node belongs to two different cells is zero.

The limited range agents find a sub-partition, $\cup_{i=1}^{n} W_i \subset \mathcal{N}_G$, defined as a limited range Voronoi sub-partition $\widetilde{\mathcal{V}}^{\text{LR}} = \{\widetilde{V}_i^{\text{LR}}\}_{i=1}^{n}$, $\widetilde{V}_i^{\text{LR}} = \widetilde{V}_i^{\text{weighted}} \cap \widetilde{D}_i$, where

$$\widetilde{D}_i = \{q\in\mathcal{N}_G \mid J(p_i,q) - \omega_i + \frac{1}{n}\sum_{k=1}^{n}\omega_k \le c\}.$$

In order to calculate the approximation of its own cell, $\widehat{V}_i$, agent i does a Dijkstra graph search, [16], starting from its current configuration, p_i, and keeps a queue of the vertices it needs to check. To start with, p_i is added to $\widehat{V}_i$, and all the outgoing neighboring nodes of p_i are added to the queue. The agent then takes one of the nodes, q_{check}, from the queue and checks to see if it is part of $\widehat{V}_i$. If q_{check} is a part of $\widehat{V}_i$ then its outgoing neighboring nodes are added to the queue. If q_{check} is not added to $\widehat{V}_i$, then its neighbors are not added to the queue, see Proposition 4 for the result on why this is correct. Agent i constructs $\widehat{V}_i$ until the queue is empty.

Properties of $\widehat{\mathcal{V}}$

For $\widetilde{\mathcal{V}}^{\text{weighted}}$ the weights need to belong to

$$U = \{\omega \in \mathbb{R}^n \mid |\omega_i - \omega_j| \leq J(p_i, p_j) \;\; i, j \in \{1, \ldots, n\}\}.$$

If $\omega \notin U$ then at least one cell is empty. Since $\widetilde{\mathcal{V}}^{\text{LR}}$ is a subset of $\widetilde{\mathcal{V}}^{\text{weighted}}$ then the weights must also belong to the set U.

Lemma 13 gives a lower bound on the constant c for a general $J(p_i, q)$ so that $\partial \widetilde{V}_i^{\text{weighted}} \cap \partial \widetilde{D}_i \neq \emptyset$ for each i. If the initial agent conditions lead to $\partial \widetilde{V}_i^{\text{weighted}} \cap \partial \widetilde{D}_i = \emptyset$ for all i, then, assuming that $\widetilde{\mathcal{V}}^{\text{LR}}$ satisfies the area constraint, Problem 3 is trivially satisfied.

Lemma 13. *Assume the triangle inequality holds for $J(p_i, q)$. If $\partial \widetilde{V}_i^{\text{weighted}} \cap \partial \widetilde{D}_i \neq \emptyset$, then $c \geq \frac{J(p_i,p_j)+\frac{2}{n}\sum_{k=1}^{n}\omega_k+\omega_i-\omega_j}{2}$.*

Proof. Because $q \in \partial \widetilde{D}_i$, $J(p_i, q) = c + \omega_i - \frac{1}{n}\sum_{k=1}^{n}\omega_k$. This same q is also in $\partial \widetilde{V}_i^{\text{weighted}}$,

$$J(p_i, q) - \omega_i + \omega_j = J(p_j, q), \text{ for some } j.$$

Substituting the above into the triangle inequality, $J(p_i,q)+J(p_j,q)\geq J(p_i,p_j)$, gives

$$2c-\frac{2}{n}\sum_{k=1}^{n}\omega_k+\omega_i+\omega_j\geq J(p_i,q).$$

Therefore, the c must be larger than

$$c\geq\frac{J(p_i,p_j)+\frac{2}{n}\sum_{k=1}^{n}\omega_k-\omega_i-\omega_j}{2}.$$

□

A tighter bound can be found for particular $J(p_i,q)$. Lemmas 14 and 15 compute such bounds.

Lemma 14. *Let* $J(p_i,q)=\|p_i-q\|^2$. *If* $\partial\widetilde{V}_i^{\text{weighted}}\cap\partial\widetilde{D}_i\neq\emptyset$, *then* $c\geq\frac{(\|p_i-p_j\|^2+\omega_i-\omega_j)^2}{4\|p_i-p_j\|^2}-\omega_i+\frac{1}{n}\sum_{k=1}^{n}\omega_k$.

Proof. Because $q\in\widetilde{V}_i^{\text{weighted}}$ then

$$\|p_i-q\|^2-\omega_i=\|p_j-q\|^2-\omega_j \text{ for some } j. \tag{6.6}$$

Now, squaring the triangle inequality of the Euclidean norm,

$$\|p_j-q\|^2\geq\|p_i-p_j\|^2+\|p_i-q\|^2-2\|p_i-p_j\|\|p_i-q\|.$$

Substitute this into Eq. 6.6, and rearranging, gives

$$\|p_i-q\|\geq\frac{\|p_i-p_j\|^2+\omega_i-\omega_j}{2\|p_i-p_j\|}.$$

Using $q\in\partial\widetilde{D}_i$, leads to $\|p_i-q\|=(c+\omega_i-\frac{1}{n}\sum_{k=1}^{n}\omega_k)^{1/2}$, which is substituted into the

above giving the bound on c,

$$c \geq \frac{(\|p_i - p_j\|^2 + \omega_i - \omega_j)^2}{4\|p_i - p_j\|^2} - \omega_i + \frac{1}{n}\sum_{k=1}^{n} \omega_k \text{ for all } j.$$

□

Lemma 15. *Let* $J(p_i, q) = \|p_i - q\|$. *If* $\partial \widetilde{V}_i^{\text{weighted}} \cap \partial \widetilde{D}_i \neq \emptyset$ *then* $c \geq \frac{(\|p_i - p_j\| + \omega_i - \omega_j)}{2} - \omega_i + \frac{1}{n}\sum_{k=1}^{n} \omega_k$.

The proof of Lemma 15 is similar to that of Lemma 14, and therefore omitted for brevity.

The additive property of J allows the agents to only check a connected subset of nodes, $q \in \mathcal{N}_G$, to find the approximated regions $\widetilde{V}_i^{\text{weighted}}$.

Proposition 4. Assume that J is an additive cost and let there exist an optimal path from p_i to q passing through q'. Then, if q' is not part of $\widetilde{V}_i^{\text{weighted}}$ then q is not part of $\widetilde{V}_i^{\text{weighted}}$.

Proof. The existence of an optimal path from p_i to q passing through q' leads to $J(p_i, q') + J(q', q) = J(p_i, q)$. From $q' \notin \widetilde{V}_i^{\text{weighted}}$, we have $J(p_i, q') - \omega_i \geq J(p_j, q') - \omega_j$ for some $j \neq i$. Putting these two equations together, along with the triangle inequality $J(p_j, q) \leq J(p_j, q') + J(q', q)$ gives

$$J(p_i, q') - \omega_i \geq J(p_j, q') - \omega_j$$
$$J(p_i, q) - J(q', q) - \omega_i \geq J(p_j, q) - J(q', q) - \omega_j$$
$$J(p_i, q) - \omega_i \geq J(p_j, q) - \omega_j.$$

Which implies that $q \notin \widetilde{V}_i^{\text{weighted}}$. □

Note that $\widetilde{\mathcal{V}}^{\mathrm{LR}}$ may not be connected but still only needs to check the same q as $\widetilde{\mathcal{V}}^{\mathrm{weighted}}$.

Another useful property of $\widetilde{\mathcal{V}}^{\mathrm{weighted}}$ and $\widetilde{\mathcal{V}}^{\mathrm{LR}}$ is that they are invariant under translation in the weights, $\widehat{\mathcal{V}}(\omega + t\mathbf{1}) = \widehat{\mathcal{V}}(\omega)$, which follows from the invariance property of their continuous space counterparts.

6.2.2 Discrete Space Algorithm

Algorithm 9 briefly outlines the general algorithm procedure for Problem 3 with limited ranges that leads to an approximate solution. Note that $\widehat{\mathcal{V}}$ refers to a generic approximation of a Voronoi sub-partition in terms of graph nodes; we refer to these as approximate generalized Voronoi partitions. First, the agents each determine such partition, see Section 6.2.1 for definition and details. Then the weights are updated to reflect the error in the area-constraint, the details of which are in Section 6.2.2. These two steps are alternated until the area-constraints are satisfied to within a specified error, which can be reduced by increasing the number of nodes in G. Next, the agents move to a neighboring node that will decrease $\widetilde{\mathcal{H}}$, see Section 6.2.2. The steps are repeated until none of the agents are able to update their positions, $P = P^{+}$.

Updating the Agent Weights

Recall, for Problem 3, each agent's cell should satisfy an approximate area constraint, $\tilde{a}_i'$. When the agents have unlimited ranges $\tilde{a}_i' = \tilde{a}_i$. Agents with limited range follow the procedure outlined in Section 6.1.2 to find a weight assignment for $\widetilde{\mathcal{V}}^{\mathrm{LR}}$. Let ω_0 be the set of weights, then define $\tilde{A} = \sum_{q \in \widetilde{\mathcal{V}}^{\mathrm{LR}}(\omega_0)} \phi(q)\beta(q)$. The new variable area constraint is $\bar{a}_i = \frac{\tilde{a}_i \tilde{A}}{\sum_{q \in \mathcal{N}_G} \phi(q)\beta(q)}$. Let $\hat{a}_i$ indicate either $\tilde{a}_i$ or $\bar{a}_i$.

Algorithm 9 $(P^*, \widehat{\mathcal{V}}(P^*, \omega^*; J)) \leftarrow \text{GSLB}(P_0, \omega_0, \widehat{\mathcal{V}}, Q)$

$G \leftarrow \text{PRM}^*(Q)$;
$(P, \omega) \leftarrow \text{Initialize}(P_0, \omega_0)$;
for all $\{\text{Agent } i\}_{i=1}^n$ **do**
  while $P \neq P^+$ **do**
    $P = P^+$
    $\widehat{V}_i(P, \omega; J) \leftarrow \text{VoronoiPartition}(P, \omega, c, G)$;
    $\tilde{A} \leftarrow \text{getArea}(\widehat{\mathcal{V}}(P, \omega))$;
    while $\|\omega - \omega^+\| > \text{error}$ **do**
      $\omega = \omega^+$;
      $\omega_i^+ \leftarrow \text{UpdateWeights}(P, \omega, \widehat{V}_i, \tilde{A}, G)$;
      $\omega^+ \leftarrow \text{TransmitAndReceive}(\omega_i^+)$;
      $\widehat{V}_i(P, \omega; J) \leftarrow \text{VoronoiPartition}(P, \omega, c, G)$;
    end while
    $p_i^+ \leftarrow \text{UpdateAgentPosition}(p_i, \widehat{V}_i, G)$;
    $P^+ \leftarrow \text{TransmitAndReceive}(p_i^+)$;
  end while
end for
return $(P, \widehat{\mathcal{V}}(P, \omega; J))$;

Define the error between the current and specified area as

$$\widetilde{g}(\omega) = \left(\sum_{q \in \widehat{V}_1(\omega)} \phi(q)\beta(q) - \hat{a}_1, \ldots, \sum_{q \in \widehat{V}_n(\omega)} \phi(q)\beta(q) - \hat{a}_n \right).$$

Next, each agent updates ω to reduce the area error.

From [3], the Jacobian update used to minimize $\widetilde{g}(\omega)$ is approximated as

$$\omega_i^+ = \omega_i - \gamma \left(\frac{\partial \widetilde{g}(\omega)}{\partial \omega_i} \right)^{-1} \widetilde{g}_i(\omega),$$

which converges for a small enough $\gamma > 0$. The partial derivatives of $\widetilde{g}$ approximated as,

$$\frac{\partial \widetilde{g}}{\partial \omega_i}(\omega) \approx \frac{\sum_{q \in \widehat{V}_i^i} \phi(q)\beta(q) - \sum_{q \in \widehat{V}_i} \phi(q)\beta(q)}{\Delta \omega_i}, \tag{6.7}$$

where $\widehat{V}_i^i$ is the Voronoi cell for agent i with $\omega = \{\omega_1, \ldots, \omega_i + \Delta\omega_i, \ldots, \omega_n\}$, note that

$\Delta\omega_i > 0$ needs to be small enough to guarantee convergence but also large enough that $\widehat{V}_i^i \neq \widehat{V}_i$. The $\widehat{V}_i^i$ can be computed with a single Dijkstra graph search so agent i can easily compute (6.7).

The algorithm loops through determining $\widehat{\mathcal{V}}$ and updating ω until the area constraint is satisfied to within a specified error.

Updating the Agent Positions

After $\widehat{V}_i$ and ω have been determined, agent i decides where to move. Ideally, each agent minimizing $\widetilde{\mathcal{H}}^{\text{centroid}}$ or $\widetilde{\mathcal{H}}^{\text{mixed}}$ would move to a position in the generalized centroid set on $\widehat{V}_i$, which is computationally intensive. Instead, agent i moves in the direction of one of the generalized centroids by moving to a neighboring node of p_i such that

$$p_i^+ \in \underset{p \in \mathcal{N}_G^{\text{out}}(p_i)}{\operatorname{argmin}} \sum_{q \in \widehat{V}_i} J(p,q)\phi(q)\beta(q).$$

If the agents have limited range, and are solving Problem 3 with $\widetilde{\mathcal{H}}^{\text{area}}$, they update their position by approximating (6.5),

$$\frac{\partial \widetilde{\mathcal{H}}^{\text{area}}(P, \widetilde{\mathcal{V}}^{\text{LR}})}{\partial p_i} = \sum_{i=1}^{n} \sum_{q \in \lambda_i} \phi(q)\hat{n}^\top(q)\frac{\partial q}{\partial p_i}.$$

The agent then moves to a neighboring node in the graph that is in a direction as close as possible to $\frac{\partial \widetilde{\mathcal{H}}^{\text{area}}(P, \widetilde{\mathcal{V}}^{\text{LR}})}{\partial p_i}$.

Note that because the agent is moving to another node in the graph, the new agent position will automatically be in Q. The new agent position is shared among the Voronoi neighbors of agent i, who need it to calculate their Voronoi cell. The algorithm loops through these steps until the agents' positions become fixed; indicating that convergence has been reached. Note that doing this in the continuum-space version, Problem 2,

agents are locally minimizing the functional with respect to their positions.

6.3 Distributed Algorithm Properties

This section looks at the distributed nature of the GRAPH-BASED SPATIAL LOAD BALANCING algorithm using $\widetilde{\mathcal{V}}^{\text{weighted}}$ and $\widetilde{\mathcal{V}}^{\text{LR}}$. The following discussion focuses on the discrete space case, but analogous considerations hold for the continuous space counterpart.

The information that agent i needs for the implementation of the GRAPH-BASED SPATIAL LOAD BALANCING using $\mathcal{V}^{\text{weighted}}$ is limited to those other agents j whose approximated regions are connected to the approximated region of i via boundary nodes (or $V_i^{\text{weighted}}(P,\omega;J) \cap V_j^{\text{weighted}}(P,\omega;J) \neq \emptyset$ in the continuous-space counterpart). In the case where no obstacles are present, and depending on the metric, this property generally involves a limited number of agents. This is the case of the Euclidean metric, where, for equal weights, the generic number of neighbors is six [46]. When obstacles are present, this characterization is more difficult. The distributed properties of the GRAPH-BASED SPATIAL LOAD BALANCING using $\widetilde{\mathcal{V}}^{\text{LR}}$ are discussed below, but first an alternate definition of $\widetilde{\mathcal{D}}$ using a maximum radius is introduced.

6.3.1 Alternate Definition of $\widetilde{\mathcal{D}}$

Under certain costs, $J(p_i,q)$, D_i can be defined as a ball with radius R_i. This definition is intuitive in a way that the c definition is not. Assume $J(p_i,q) = \|p_i - q\|^2$ or $J(p_i,q) = \|p_i - q\|$, then the radius of the ball defined by $\widetilde{D}_i$ is denoted as R_i for all $i \in \{1,\ldots,n\}$. Recall $\mathcal{R}_i \triangleq c + \omega_i - \frac{1}{n}\sum_{k=1}^{n}\omega_k$, then, using the definition of D_i at the boundary, $J(p_i,q) = \|p_i - q\|^2$ implies $R_i = \mathcal{R}_i^{1/2}$ and $J(p_i,q) = \|p_i - q\|$ implies $R_i = \mathcal{R}_i$.

Depending upon the physical system, an upper bound may be imposed on R_i, denoted as $R_{\max}$. We want to remove c from the definition of D_i and replace it with the fixed $R_{\max}$. To do this, let c be a function of $R_{\max}$ and ω, instead of a constant. Then, define $c_{\max} = R_{\max} - \max_k(\omega) + \frac{1}{n}\sum_{k=1}^{n}\omega_k$. Substituting $c_{\max}$ into the equation for $\mathcal{R}_i$ gives,

$$\begin{aligned}\mathcal{R}_i &= R_{\max} - \max_k(\omega) + \frac{1}{n}\sum_{k=1}^{n}\omega_k + \omega_i - \frac{1}{n}\sum_{k=1}^{n}\omega_k,\\ &= R_{\max} - \max_k(\omega) + \omega_i.\end{aligned}$$

Notice that now $\mathcal{R}_i$, and hence R_i, are no longer dependent on c or the mean of ω but on $R_{\max}$ and the maximum value of ω. Algorithm 9 needs some minor modifications to account for the maximum radius constraint. First, in Lines 6 and 12, the primitive VoronoiPartition now takes inputs $R_{\max}$ and R_i instead of c. Then, after Line 11, R_i is determined.

Let D_i denote the continuous space counterpart of $\widetilde{D}_i$, then defining D_i using $R_{\max}$ causes problems in Lemma 11 because $\max_k \omega$ is not differentiable. However, given the max operator properties, one can conjecture that an analogous result exists using generalized gradients. As a consequence, assuming the analogous result leads to $\mathcal{M}(\omega)$ being gradient (i.e. that the weights-to-area map is in the generalized gradient of F,) then all other results follow. In particular, in Lemma 12, the new D_i still satisfies the conditions on the partial derivatives. With the assumption that Lemma 11 holds, then a weight assignment exists that satisfies the area constraint. Then, the convergence result in Lemma 9 still holds for the D_i definition for $R_{\max}$.

6.3.2 Distributed Properties using $\widetilde{\mathcal{V}}^{\mathbf{LR}}$

While the algorithm in [14] for solving the spatial load balancing problem is distributed in the sense that only information is needed for neighboring agents, these neighboring agents may be significantly far away from one another. Especially when Euclidean norms are used, the limited range constraint forces the agents to only consider neighbors within a certain distance of each other.

More precisely, the computation of $\widetilde{V}_i^{\mathrm{LR}}(P, \omega)$ requires knowledge of the positions and weights of agent i's neighbors and knowledge of the mean of the weights for $\widetilde{D}_i$. The latter can be computed using a distributed consensus algorithm performed over a *connected* communication graph, not necessarily $\mathcal{G}_{\mathrm{LR}}$. If $\widetilde{D}_i$ is defined as in Section 6.3.1, the agents need the maximum ω value instead of the mean. The maximum value is found using a max operation over a connected system, requiring fewer communications between the agents.

Before each weight update, the area covered by $\widetilde{\mathcal{V}}^{\mathrm{LR}}$ needs to be determined. Again, a distributed consensus algorithm over a connected communication graph is needed. Once inside the weights update loop, agents only need the information from their cell, $\widetilde{V}_i^{\mathrm{LR}}$, to determine ω_i^+. Likewise, the position update using the gradient of $\widetilde{\mathcal{H}}^{\mathrm{area}}(P, \widetilde{\mathcal{V}}^{\mathrm{LR}})$ or the centroid of $\widetilde{V}_i^{\mathrm{LR}}$ for $\widetilde{\mathcal{H}}^{\mathrm{mixed}}(P, \widetilde{\mathcal{V}}^{\mathrm{LR}})$, only requires knowledge from the agent's own cell. In all, the algorithm is distributed over the smallest connected graph containing $\mathcal{G}_{\mathrm{LR}}$.

The $R_{\max}$ constraint can be used to define a disk graph $\mathcal{G}_{2R_{\max}}$ that is sufficient for agents to determine neighbors in $\mathcal{G}_{\mathrm{LR}}$. When using a general J, the balls are defined as a reachable set. If J is radially unbounded, the balls are compact sets. In the Euclidean metric case, the balls are circles whose radii are related to $R_{\max}$ and correspond to the standard r-disk graph. Define $\mathcal{G}_{2R_{\max}}$ over the set of agents, where a (communication) edge exists between agents i and j if and only if the balls centered at the agents' position

with radius $R_{\max}$ intersect, $\mathbb{B}(p_i, R_{\max}) \cap \mathbb{B}(p_j, R_{\max}) \neq \emptyset$. The $\mathcal{G}_{2R_{\max}}$ can be used to determine an over approximation of the sets of neighboring agents in $\mathcal{G}_{\mathrm{LR}}$. To see this, note that $\widetilde{V}_i^{\mathrm{LR}} \subseteq \widetilde{D}_i \subseteq \mathbb{B}(p_i, R_{\max})$, for all $i \in \{1, \ldots, n\}$. Therefore, $\widetilde{V}_i^{\mathrm{LR}} \cap \widetilde{V}_j^{\mathrm{LR}} \neq \emptyset$ only if $\widetilde{D}_i \cap \widetilde{D}_j \neq \emptyset$, which happens only if $\mathbb{B}(p_i, R_{\max}) \cap \mathbb{B}(p_j, R_{\max}) \neq \emptyset$. This implies that agent i can compute its cell, $\widetilde{V}_i^{\mathrm{LR}}$, communicating only with neighbors j in $\mathcal{G}_{2R_{\max}}$. In other words, an agent only needs to communicate with other agents within a distance of $2R_{\max}$ of itself, $\|p_i - p_j\| \leq 2R_{\max}$.

6.4 Algorithm Analysis

The spatial load balancing algorithm in [14] is shown to converge to a solution $(P^*, \mathcal{V}^{\mathrm{weighted}}(P^*, \omega^*; J))$ of Problem 2 for convex environments. A similar result holds for non-convex Q since there exists ω that allows a generalized Voronoi partition $\mathcal{V}^{\mathrm{weighted}}(P, \omega; J)$ to satisfy the constraints. Lemma 16 is used to extend Prop. IV.4 from [14] to non-convex Q; Prop. IV.4 sates there exists a set of weights that make $\mathcal{V}^{\mathrm{weighted}}$ satisfy the area constraints in a convex continuous space.

Lemma 16. *Let $\mathcal{V}^{\mathrm{weighted}}$, $J(p_i, q)$, $\phi(q)$, and ω be defined as above. Define the weights-to-area map as*

$$\mathcal{M}(P, \omega) = \left(\int_{V_i^{\mathrm{weighted}}(\omega)} \phi(q) dq, \ldots, \int_{V_n^{\mathrm{weighted}}(\omega)} \phi(q) dq \right).$$

Then, $\mathcal{M}$ is gradient, $\nabla F = -\mathcal{M}$, where $F : \mathbb{R}^n \to \mathbb{R}$,

$$F(\omega) = \sum_{j=1}^{n} \int_{V_j^{\mathrm{weighted}}(\omega)} (J(p_j, q) - \omega_j) \phi(q) dq.$$

The proof of Lemma 16 is similar to that of Lemma 11 and therefore omitted.

Proof. Take the derivative of $F(\omega)$ with respect to ω_i, using the Leibniz rule [20],

$$\begin{aligned}\frac{\partial F(\omega)}{\partial \omega_i} =& \frac{\partial}{\partial \omega_i}\sum_{j=1}^{n}\int_{V_j^{\text{weighted}}(\omega)}(J(p_j,q)-\omega_j)\phi(q)dq \\ =& \sum_{j=1}^{n}\int_{V_j^{\text{weighted}}(\omega)} -[\frac{\partial q}{\partial \omega_i} \quad \times (\frac{\partial}{\partial q}(J(p_j,q)-\omega_j)\phi(q))]\cdot dq \\ &+\sum_{j=1}^{n}\int_{\partial V_j^{\text{weighted}}(\omega)}\frac{\partial q}{\partial \omega_i}\cdot((J(p_j,q)-\omega_j)\phi(q))dq \\ &+\sum_{j=1}^{n}\int_{V_j^{\text{weighted}}(\omega)}\frac{\partial}{\partial \omega_i}\Big((J(p_j,q)-\omega_j)\phi(q)\Big)dq\end{aligned}$$

The first term,

$$\sum_{j=1}^{n}\int_{V_j^{\text{weighted}}(\omega)} -[\frac{\partial q}{\partial \omega_i}\times(\frac{\partial}{\partial q}(J(p_j,q)-\omega_j)\phi(q))]\cdot dq = 0,$$

There are two vectors in the (q_1,q_2) plane being crossed, resulting in a vector perpendicular to the (q_1,q_2) plane, in dot product with a vector in the (q_1,q_2) plane, thus resulting in a zero value. The second term becomes

$$\begin{aligned}&\sum_{j=1}^{n}\int_{\partial V_j^{\text{weighted}}(\omega)}\frac{\partial q}{\partial \omega_i}\cdot((J(p_j,q)-\omega_j)\phi(q))dq \\ =&\sum_{j=1}^{n}\int_{\partial V_j^{\text{weighted}}(\omega)}\frac{\partial q}{\partial \omega_i}\hat{n}_j^{\top}((J(p_j,q)-\omega_j)\phi(q))dq.\end{aligned}$$

This term then vanishes because the shared boundaries have opposing normal vectors.

Finally, the third term reduces to,

$$\begin{aligned}
&\sum_{j=1}^{n} \int_{V_j^{\text{weighted}}(\omega)} && \frac{\partial}{\partial \omega_i}\Big((J(p_j,q)-\omega_j)\phi(q)\Big)dq \\
&= \sum_{j=1}^{n} \int_{V_j^{\text{weighted}}(\omega)} && \left(\frac{\partial (J(p_j,q)-\omega_j)}{\partial \omega_i}\phi(q) + (J(p_j,q)-\omega_j)\frac{\partial \phi(q)}{\partial \omega_i}\right)dq \\
&= \sum_{j=1}^{n} \int_{V_j^{\text{weighted}}(\omega)} && \left(\cancelto{0\forall j}{\frac{\partial J(p_j,q)}{\partial \omega_i}}\phi(q) - \cancelto{1,\, i=j}{\frac{\partial \omega_j}{\partial \omega_i}}\phi(q) + (J(p_j,q)-\omega_j)\cancelto{0\forall j}{\frac{\partial \phi(q)}{\partial \omega_i}}\right)dq \\
&= -\int_{V_i^{\text{weighted}}(\omega)} && \phi(q)dq.
\end{aligned}$$

Putting everything together results in

$$\frac{\partial F}{\partial \omega_i} = -\int_{V_i^{\text{weighted}}(\omega)} \phi(q)dq = -\mathcal{M}_i$$

□

For the discrete case note that for any $\widetilde{\mathcal{W}} = \{\widetilde{W}_i\}_{i=1}^n$ partition of $\mathcal{N}_G$, one can find (many) continuous-space partitions $\mathcal{W} = \{W_i\}_{i=1}^n$ such that $\widetilde{W}_i = W_i \cap \mathcal{N}_G$ and $\mathcal{W}$ satisfies the constraints. As the number of nodes in $\mathcal{N}_G$ goes to infinity, $\widetilde{\mathcal{W}}$ will converge to a $\mathcal{W}$. Due to integration properties, and because $\mathcal{H}^{\text{centroid}}(P,\mathcal{W})$ is continuous,

$$|\mathcal{H}^{\text{centroid}}(P,\mathcal{W}) - \widetilde{\mathcal{H}}^{\text{centroid}}(P,\widetilde{\mathcal{W}})| \leq \varepsilon, \tag{6.8}$$

is true for a sufficiently small sample dispersion.

The GRAPH-BASED SPATIAL LOAD BALANCING algorithm with $\mathcal{V}^{\text{weighted}}$ cannot converge to the exact partition and centroids of the continuous problem, but an approximate solution can be guaranteed, see Theorem 9.

Theorem 9. *The unlimited range* GRAPH-BASED SPATIAL LOAD BALANCING *algo-*

rithm is guaranteed to converge to an approximate solution $(P^*, \widetilde{\mathcal{V}}^{\text{weighted}}(P^*, \omega^*))$ *which is in a continuous space set defined by*

$$\|\mathcal{H}^{\text{centroid}}(P^*, \mathcal{V}^{\text{weighted}}(P, \omega)) - \mathcal{H}^{\text{centroid}}(P^*, \mathcal{V}^{\text{weighted}}(P^*, \omega^*))\| \leq 2\varepsilon.$$

Proof. Convergence can be proved by showing that $\widetilde{\mathcal{H}}^{\text{centroid}}$ decreases monotonically at every step. Recall that the agent positions are updated specifically so that $\widetilde{\mathcal{H}}^{\text{centroid}}$ decreases, $\widetilde{\mathcal{H}}^{\text{centroid}}(P, \widetilde{\mathcal{V}}^{\text{weighted}}(P)) \geq \widetilde{\mathcal{H}}^{\text{centroid}}(P^+, \widetilde{\mathcal{V}}^{\text{weighted}}(P))$. Next, using (6.8),

$$\begin{aligned}
&\widetilde{\mathcal{H}}^{\text{centroid}}(P^+, \widetilde{\mathcal{V}}^{\text{weighted}}(P)) \\
\geq &\mathcal{H}^{\text{centroid}}(P^+, \mathcal{V}^{\text{weighted}}(P)) - \varepsilon \\
\geq &\mathcal{H}^{\text{centroid}}(P^+, \mathcal{V}^{\text{weighted}}(P^+)) - \varepsilon \\
\geq &\widetilde{\mathcal{H}}^{\text{centroid}}(P^+, \widetilde{\mathcal{V}}^{\text{weighted}}(P^+)) - 2\varepsilon.
\end{aligned}$$

Thus, $\widetilde{\mathcal{H}}^{\text{centroid}}$ decreases as long as $\mathcal{H}^{\text{centroid}}(P^+, \mathcal{V}^{\text{weighted}}(P))$ is 2ε larger than $\mathcal{H}^{\text{centroid}}(P^+, \mathcal{V}^{\text{weighted}}(P^+))$. □

Proving convergence of the GRAPH-BASED SPATIAL LOAD BALANCING with limited ranges relies on the continuous space convergence Theorem 10 which is only shown to converge for $\mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}})$.

Theorem 10. *The continuous space version of Algorithm 9, used to solve the limited range spatial load balancing problem, converges to a solution* $(P^*, \mathcal{V}^{\text{LR}}(P^*, \omega^*))$ *when* $\mathcal{H} = \mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}})$.

Proof. Let Q be compact and invariant for dynamics T, and define $\mathcal{H}_v(P)$ $= \mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}}(P, \mathcal{A}(P))$. Because Algorithm 9 alternately updates the agents' posi-

tions and weight assignment, the evolution of $\mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}})$ is,

$$\begin{aligned}
&\mathcal{H}_{\text{v}}(P) = \mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}}(P, \mathcal{A}(P))) \\
=&\mathcal{H}^{\text{area}}(T(P), \mathcal{V}^{\text{LR}}(P, \mathcal{A}(P))) \geq \mathcal{H}^{\text{area}}(T(P), \mathcal{V}^{\text{LR}}(T(P), \mathcal{A}(P))) \\
=&\mathcal{H}^{\text{area}}(T(P), \mathcal{V}^{\text{LR}}(T(P), \mathcal{A}(T(P)))) = \mathcal{H}_{\text{v}}(T(P)),
\end{aligned}$$

where $\mathcal{V}^{\text{LR}}(P, \omega)$ and $\mathcal{V}^{\text{LR}}(T(P), \mathcal{A}(T(P)))$ both satisfy the variable a' constraint. When the partition $\mathcal{V}^{\text{LR}}$ is kept constant and the position is updated, the area covered by $\mathcal{V}^{\text{LR}}$ is constant. Because $T(P)$ is found to specifically increase the area, the new sub-partition, $\mathcal{V}^{\text{LR}}(T(P), \mathcal{A}(P))$, has a greater area than $\mathcal{V}^{\text{LR}}(P, \mathcal{A}(P))$. Finally, the $\mathcal{A}(P)$ values are updated in such a way that the area is kept constant during the update. Theorem 8 says that $\mathcal{A}(T(P))$ exists. Note that if $T(P) \neq P$ then $\mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}})$ decreases implying $\mathcal{H}_{\text{v}}(P) \geq \mathcal{H}_{\text{v}}(T(P))$. Therefore, $\mathcal{H}_{\text{v}}(P) = \mathcal{H}_{\text{v}}(T(P))$ if and only if $T(P) = P$. From here, apply the La Salle invariance principle to guarantee the trajectories of T converge to the largest invariant set in $\mathcal{Z} = \{P \in Q \,|\, \mathcal{H}_{\text{v}}(P) = \mathcal{H}_{\text{v}}(T(P))\}$. It can be concluded from the above discussion that $\mathcal{Z}$ is the set of limited range generalized Voronoi configurations where the gradient of $\mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}})$ is zero. □

Then, convergence to an approximate solution can be guaranteed, Lemma 17.

Lemma 17. *The limited range* GRAPH-BASED SPATIAL LOAD BALANCING *using* $\widetilde{\mathcal{H}}^{\text{area}}$ *is guaranteed to converge to an approximate solution* $(P^*, \widetilde{\mathcal{V}}^{\text{LR}}(P^*, \omega^*))$ *which is in a continuous space set defined by*

$$\|\mathcal{H}^{\text{area}}(P^*, \mathcal{V}^{\text{LR}}(P), \omega) - \mathcal{H}^{\text{area}}(P^*, \mathcal{V}^{\text{LR}}(P^*, \omega^*))\| \leq 2\varepsilon.$$

The proof of Lemma 17 is similar to that of Theorem 9 and therefore omitted.

6.5 Simulations

The simulations in this section are for 15 agents minimizing $\mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}})$ and $\mathcal{H}^{\text{mixed}}(P, \mathcal{V}^{\text{LR}})$ with $J(p_i, q) = \|p_i - q\|^2$, a convex Q, and a uniform $\phi(q)$. The simulations also compare defining D_i with a constant maximum range c and a maximum radius $R_{\max}$. In general the maximum range is not intuitive to determine. But, when the cost $J(p_i, q) = \|p_i - q\|^2$, c becomes the radius of a ball squared minus some error that is a function of ω, $c = \|p_i - q\|^2 - \omega_i + \frac{1}{n}\sum_{k=1}^{n} \omega_k$
$= R_i^2 + \varepsilon(\omega)$. In other words, the radius of D_i, when c is a constant, is $R_i = \sqrt{c - \varepsilon\omega}$.

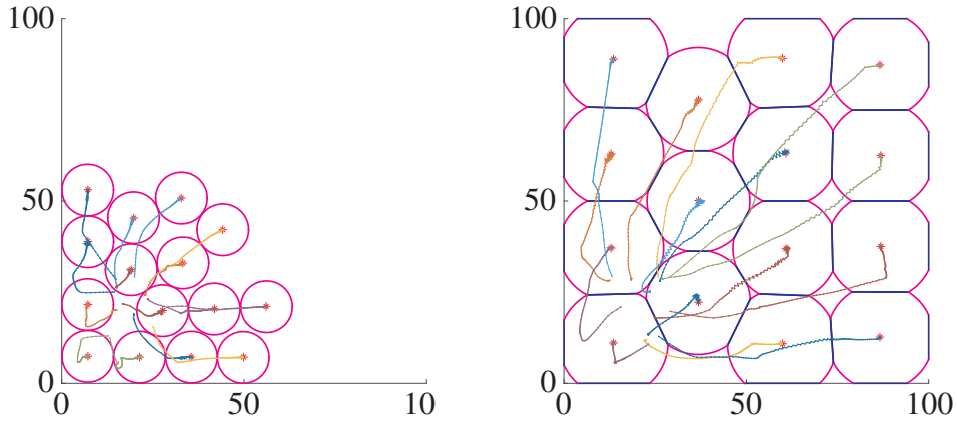
All simulations have the same initial conditions where the agents start together in the lower left corner and the weights are initially all one. Note that the initial positions of the agents do not allow for the existence of a feasible set of weights, $\omega \in U$, and therefore the area constraint, a', is not satisfied. The algorithm moves the agents away from one another and eventually the area constraint is satisfied. The convergence of the simulations are unaffected by these initial conditions.

6.5.1 Area-Only Cost Function

This section compares the limited range algorithm results with $\mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}})$. The final $\mathcal{V}^{\text{LR}}$ partition for a constant $c = 50$ is in Fig. 6.2a and for $R_{\max} = 15$ is in Fig. 6.2b.

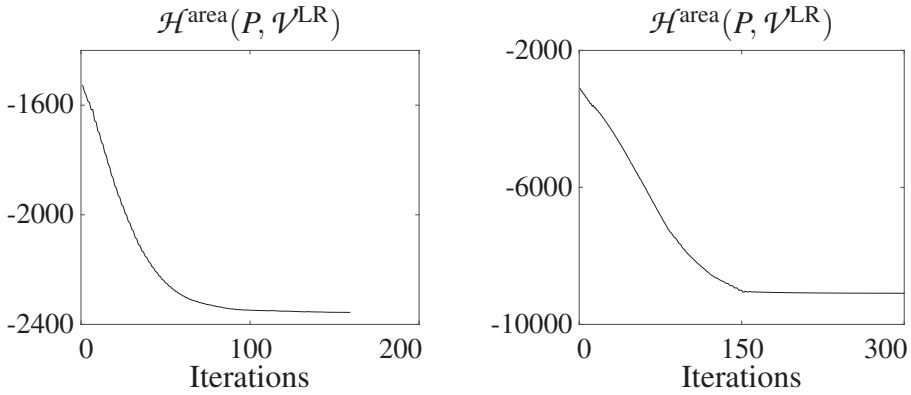
The following figure shows how $\mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}})$ evolves as the algorithm progresses. As discussed in Chapter 6.4, the cost monotonically decreases at each partition update and stays constant during the agent position update. The constant $c = 50$ partition case is in Fig. 6.3a and the $R_{\max} = 15$ case is in Fig. 6.3b.

Next, the limited range radius of D_i is compared. Fig. 6.4a shows evolution of the radii when $c = 50$ and Fig. 6.4b is the radii for $R_{\max} = 15$. Notice that the $c = 50$



(a) $c = 50$ (b) $R_{\max} = 15$

Figure 6.2: The final $\mathcal{V}^{\mathrm{LR}}$ partition that minimizes $\mathcal{H}^{\mathrm{area}}(P, \mathcal{V}^{\mathrm{LR}})$ with agent trajectories.



(a) $c = 50$ (b) $R_{\max} = 15$

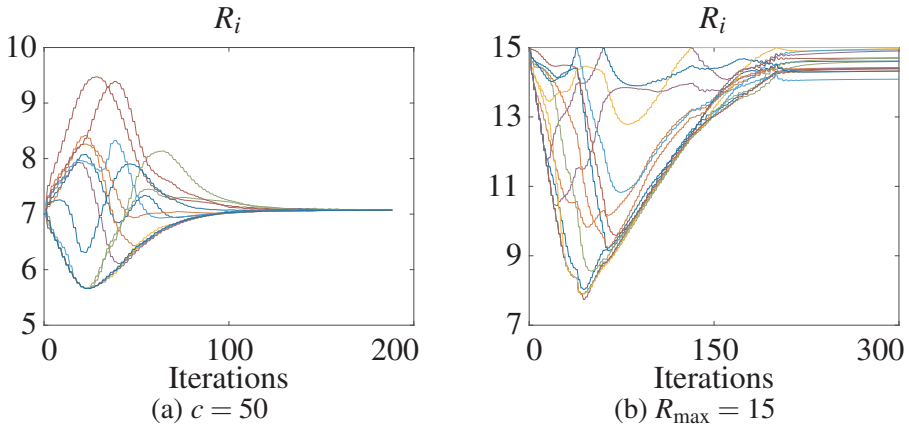
Figure 6.3: The evolution of $\mathcal{H}^{\mathrm{area}}(P, \mathcal{V}^{\mathrm{LR}})$.

radii eventually converge to a radius of $R \approx 7$, while the $R_{\max}$ radii converge to different values close to the maximum allowable radius.

6.5.2 Mixed Cost Function

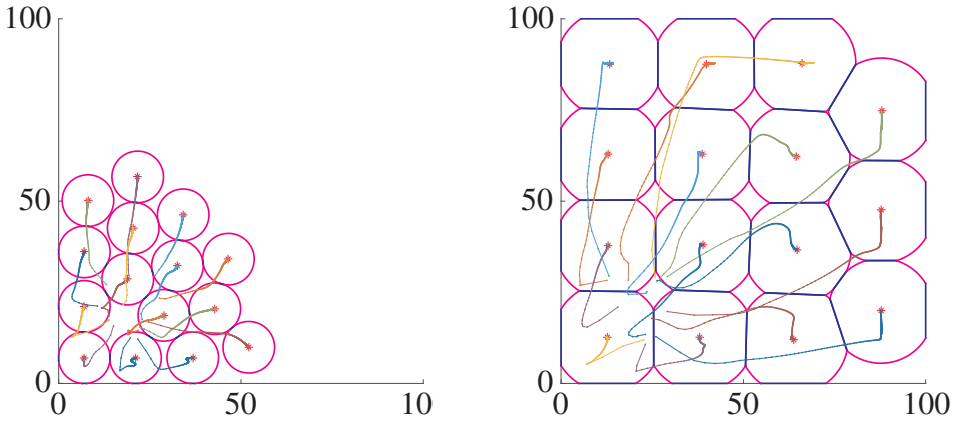
Below are the results for the limited range algorithm with $\mathcal{H}^{\mathrm{mixed}}(P, \mathcal{V}^{\mathrm{LR}})$. Fig. 6.5a and Fig. 6.5b contain the final $\mathcal{V}^{\mathrm{LR}}$ partition for a constant $c = 50$ and $R_{\max} = 15$, respectively.

The evolution of $\mathcal{H}^{\mathrm{mixed}}(P, \mathcal{V}^{\mathrm{LR}})$ as the algorithm progresses for a constant $c =$



(a) $c = 50$ (b) $R_{\max} = 15$

Figure 6.4: The evolution of the limited range radii for $\mathcal{H}^{\text{area}}(P, \mathcal{V}^{\text{LR}})$.

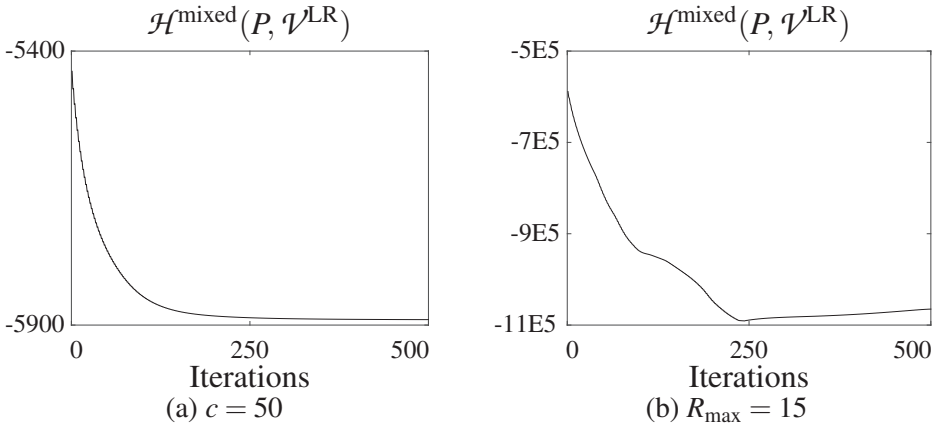


(a) $c = 50$ (b) $R_{\max} = 15$

Figure 6.5: The final $\mathcal{V}^{\text{LR}}$ partition that minimizes $\mathcal{H}^{\text{mixed}}(P, \mathcal{V}^{\text{LR}})$ with agent trajectories.

50 partition and $R_{\max} = 15$ partition are in Fig. 6.6a and Fig. 6.6b, respectively. While $D_i \subset Q$ for all $i \in \{1, \ldots, n\}$, $\mathcal{H}^{\text{mixed}}(P, \mathcal{V}^{\text{LR}})$ decreases but, as seen in Fig. 6.6b, the cost can increase when $D_i \subsetneq Q$.

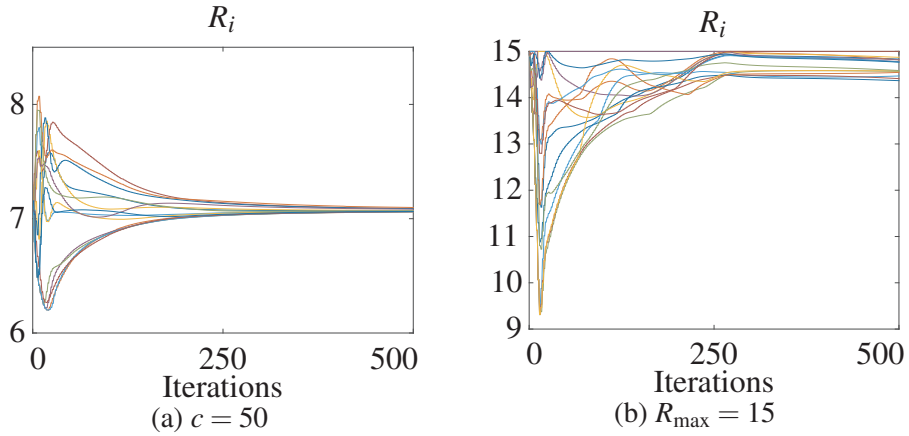
The limited range radii of the agents are compared in Fig. 6.7a when $c = 50$ and Fig. 6.7b when $R_{\max} = 15$. The radii when $c = 50$ initially are spread out and then converge to a common $R \approx 7$. Due to the definition of R_i when the maximum radius is used there is always at least one agent whose radius is the maximum value. As the algorithm progresses the agents radii converge to different values close, but not equal,



(a) $c = 50$ (b) $R_{\max} = 15$

Figure 6.6: The evolution of $\mathcal{H}^{\text{mixed}}(P, \mathcal{V}^{\text{LR}})$.

to the maximum radius.



(a) $c = 50$ (b) $R_{\max} = 15$

Figure 6.7: The evolution of the limited range radii for $\mathcal{H}^{\text{mixed}}(P, \mathcal{V}^{\text{LR}})$.

The simulations below examine agents without differential constraints whose edge cost, J_e, is Euclidean distance and for Dubins vehicle agents whose edge cost is the distance traveled. There are simulations for six agents each of them solving graph-based Problem 3 with an equal area constraint and a uniform probability density function, $\phi(q) = 1, \forall q \in \mathcal{N}_G$. Each agent is initialized with $\omega_i = 5$. The simulations compare the limited range defined by a constant $c = 3$ and $R_{\max} = 3.5$.

All simulations have the same initial agent positions; clustered together in the top right corner. The simulations were run for a graph constructed with 2,000 samples

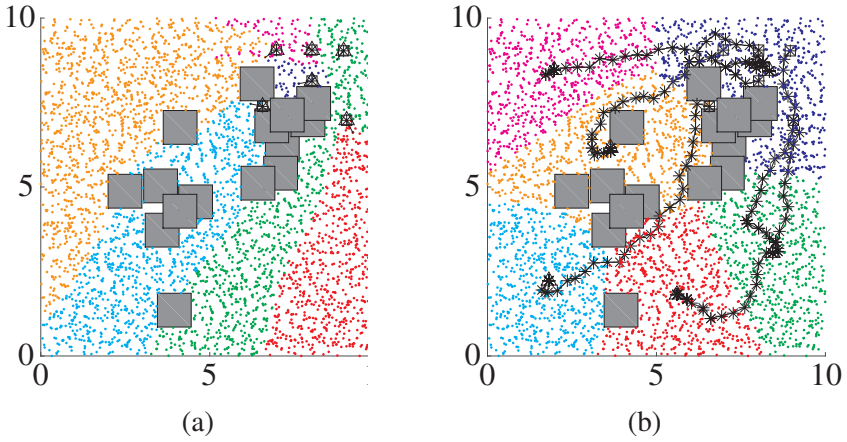


Figure 6.8: The initial (left) and final (right) 5,000 node graph $\widetilde{\mathcal{V}}^{\text{weighted}}$ for six agents obtained by solving Problem 3 with $\widetilde{\mathcal{H}}^{\text{centroid}}$

and a more dense graph constructed using 5,000 samples. The graph should be constructed such that the edge lengths are less than $\partial\widetilde{D}_i$. This will ensure that there exists neighboring nodes of p_i within $\widetilde{V}_i^{\text{LR}}$ for the agent to move to.

6.5.3 Voronoi Graph Partitions

This section compares the weighted Voronoi graph partition, $\widetilde{\mathcal{V}}^{\text{weighted}}$ for six agents solving Problem 3. Figure 6.8a and Fig. 6.8b are the initial and final $\widetilde{\mathcal{V}}^{\text{weighted}}$ partitions, respectively, in the non-convex environment using the graph with 5,000 samples.

The initial and final limited range Voronoi graph partitions for agents solving Problem 3 with $\widetilde{\mathcal{H}} = \widetilde{\mathcal{H}}^{\text{area}}$ in the non-convex environment using the 5,000 sample graph are in Fig. 6.9a -6.9c. Figure 6.9b is the final $\widetilde{\mathcal{V}}^{\text{LR}}$ partition when $c = 3$. When $R_{\max} = 3.5$, the agents' final $\widetilde{\mathcal{V}}^{\text{LR}}$ partition is in Figure 6.9c. The final $\widetilde{\mathcal{V}}^{\text{LR}}$ partitions for the 5,000 sample graph in the non-convex environment under $\widetilde{\mathcal{H}} = \widetilde{\mathcal{H}}^{\text{Mixed}}$ are Fig. 6.10a when $c = 3$, and Fig. 6.10b when $R_{\max} = 3.5$.

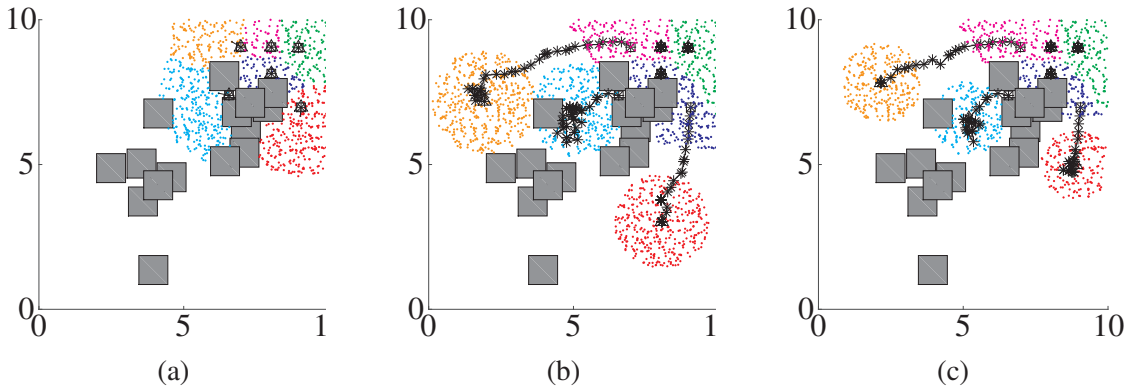


Figure 6.9: The Initial (left), final with $c = 3$ (center) and final with $R_{\max} = 3.5$ (right) 5,000 node graph $\widetilde{\mathcal{V}}^{\mathrm{LR}}$ for six agents obtained by solving Problem 3 with $\widetilde{\mathcal{H}}^{\mathrm{area}}$

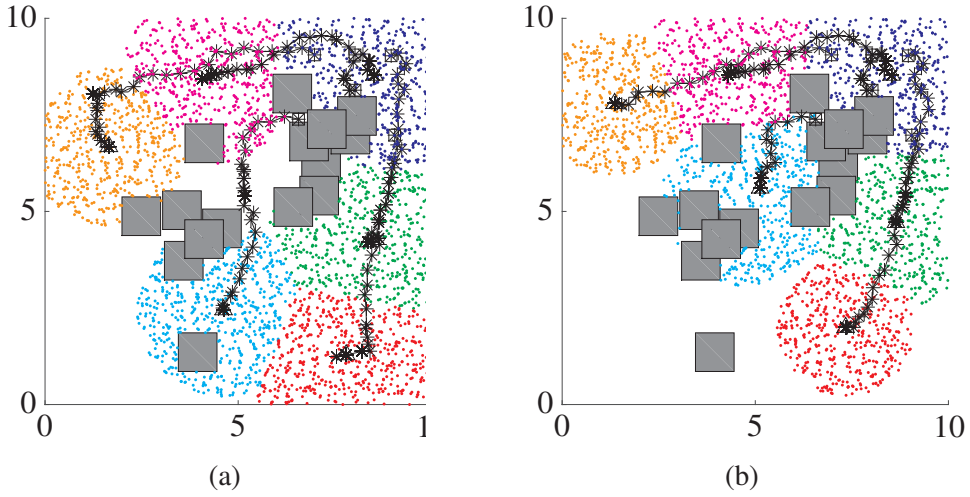


Figure 6.10: The final 5,000 node graph $\widetilde{\mathcal{V}}^{\mathrm{LR}}$ with $c = 3$ (left) and with $R_{\max} = 3.5$ (right) for six agents solving Problem 3 with $\widetilde{\mathcal{H}}^{\mathrm{mixed}}$

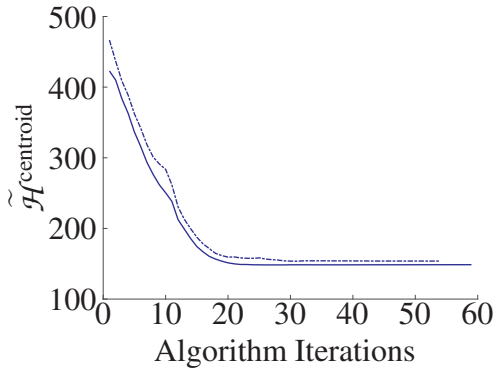


Figure 6.11: The evolution of $\widetilde{\mathcal{H}}^{\text{centroid}}$ obtained by solving Problem 3 using $\widetilde{\mathcal{V}}^{\text{weighted}}$, where the solid line is the 5,000 node graph and the dashed is the 2,000 node graph

6.5.4 Evolution of $\widetilde{\mathcal{H}}$

The following are a comparison of the evolution of the different cost functions for the various problems. The plots show the algorithm converges to a solution.

The evolution of $\widetilde{\mathcal{H}}^{\text{centroid}}$ for agents solving Problem 3 in the non-convex environment using partition $\widetilde{\mathcal{V}}^{\text{weighted}}$ is in Fig. 6.11. The cost function decreases monotonically as expected.

Figure 6.12a is the evolution of $\widetilde{\mathcal{H}}^{\text{area}}$ for the agents solving Problem 3 with limited ranges in the non-convex environment. The evolution of $\widetilde{\mathcal{H}}^{\text{mixed}}$ for the limited range agents in the convex environment solving Problem 3 is in Fig 6.12b. Both the 2,000 and 5,000 sample graphs produce costs that decrease smoothly until the algorithm gets close to convergence then chatters slightly. It is important to note that when the algorithm uses $\widetilde{\mathcal{H}}^{\text{area}}$ or $\widetilde{\mathcal{H}}^{\text{mixed}}$, the position update is approximate. The agents do not follow the gradient exactly, but rather a close approximation based on the neighboring nodes. The more nodes there are in the graph the less error there will be in following the gradient. This can be seen by comparing the 2,000 and 5,000 node graphs. The 5,000 node graph produces a cost function plot with less pronounced increases and less chattering.

The following is another simulation with seven agents in a hallway type envi-

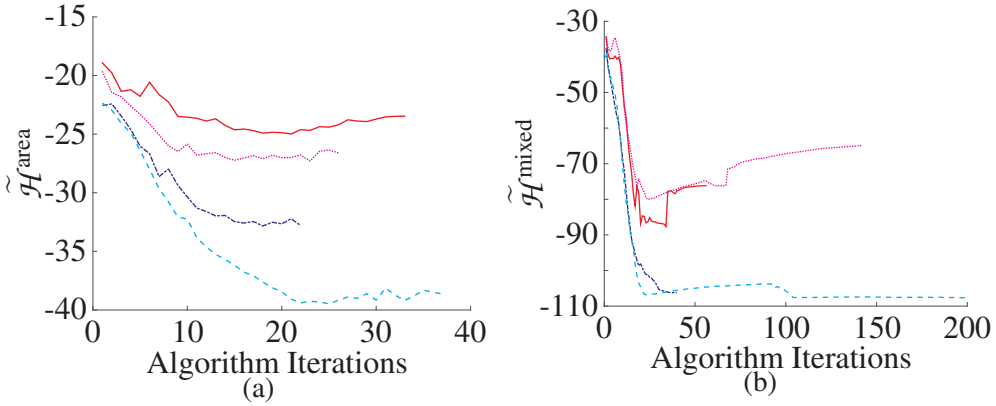


Figure 6.12: The evolution of $\widetilde{\mathcal{H}}^{\text{area}}$ (left) and $\widetilde{\mathcal{H}}^{\text{mixed}}$ (right) by solving Problem 3 using $\widetilde{\mathcal{V}}^{\text{LR}}$ from the 2,000 node graph with $c = 3$ (blue dash-dot line), $R_{\max} = 3.5$ (red solid line), from the 5,000 node graph with $c = 3$ (cyan dashed line), $R_{\max} = 3.5$ (magenta dotted line)

ronment. The initial and final agent partitions are shown in Figs. 6.13a and 6.13b. The area-only cost function is plotted in Fig. 6.14a for both the c and $R_{\max}$ definitions of the limited range sub-partition. The cost function decreases and then chatters due to the error in the gradient. Fig. 6.14b is the mixed cost function. The c definition decreases monotonically while the $R_{\max}$ definition increases first then decreases monotonically.

Dubins' vehicle Results

Simulations for Dubins' vehicle agents, subject to limited ranges, solving Problem 3 using $\widetilde{\mathcal{V}}^{\text{LR}}$ were run for a 2,000 node graph. Fig. 6.15a is the evolution of $\widetilde{\mathcal{H}}^{\text{area}}$ and Fig. 6.15b is the evolution of $\widetilde{\mathcal{H}}^{\text{mixed}}$ in the non-convex environment. The blue dashed lines are for $\widetilde{\mathcal{V}}^{\text{LR}}$ with a constant $c = 7$ and the red solid lines are for $R_{\max} = 8$. The $\widetilde{\mathcal{H}}^{\text{area}}$ and $\widetilde{\mathcal{H}}^{\text{mixed}}$ decrease initially and then chatters until convergence is reached. The $\widetilde{\mathcal{H}}^{\text{mixed}}$ decreases smoothly for $\widetilde{\mathcal{V}}^{\text{LR}}$ with a constant $c = 7$. The chattering produced by the Dubins' vehicle is due to the resolution in the graph. If the graph were to have more nodes, the position update would have less error and therefore less chattering. Because an increasing in the number of nodes in the graph causes the algorithm to increase

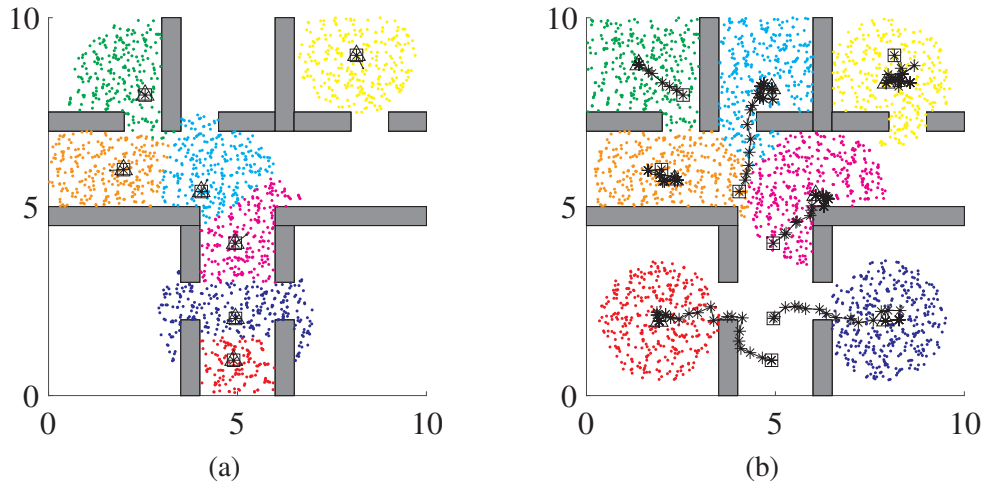


Figure 6.13: A 5,000 node graph initial (left) and final (right) $\widetilde{\mathcal{V}}^{\mathrm{LR}}$ for seven agents obtained by solving Problem 3 with $\widetilde{\mathcal{H}}^{\mathrm{area}}$

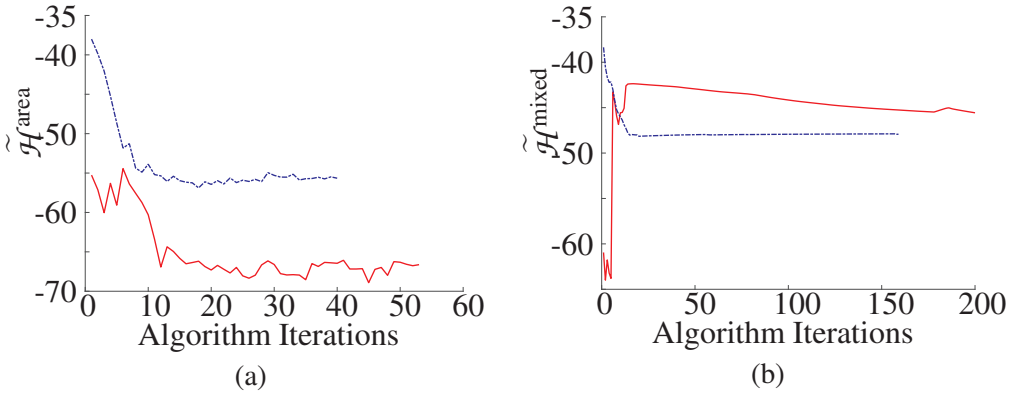


Figure 6.14: The evolution of $\widetilde{\mathcal{H}}^{\mathrm{area}}$ (left) and $\widetilde{\mathcal{H}}^{\mathrm{mixed}}$ (right) obtained by solving Problem 3 using $\widetilde{\mathcal{V}}^{\mathrm{LR}}$ from a 5,000 node graph with $c = 3$ (blue dash-dot line), $R_{\max} = 3.5$ (red solid line)

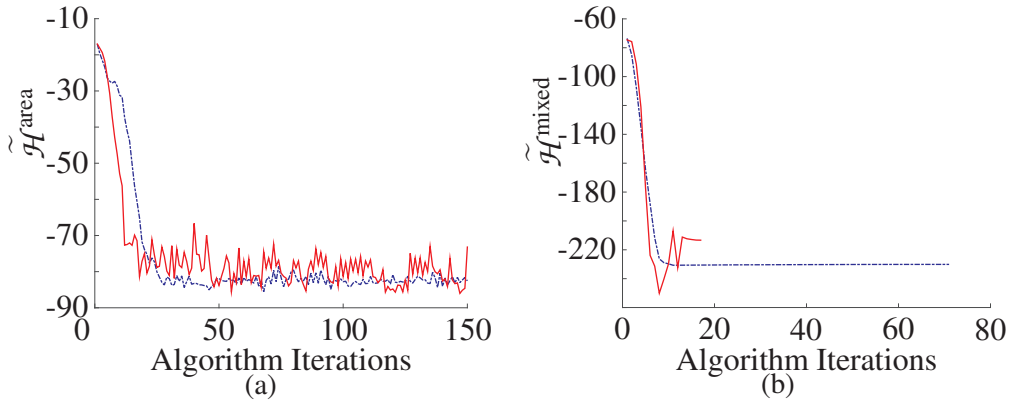


Figure 6.15: The evolution of $\widetilde{\mathcal{H}}^{\text{area}}$ (left) and $\widetilde{\mathcal{H}}^{\text{mixed}}$ (right) obtained by solving Problems 3 for Dubins' vehicle using $\widetilde{\mathcal{V}}^{\text{LR}}$, where the blue dashed line is with $c = 7$ and solid red line is with $R_{\text{max}} = 8$

in run time, a balance between the run time and the smoothness of the cost function decrease is needed.

6.6 Summary

We introduce a novel algorithm to solve the spatial load balancing problem in non-convex configuration spaces and for agents with limited ranges, subject to differential constraints. First, the problem without limited ranges is introduced and solved. Then, the limited range spatial load balancing problem in convex environments is solved. Finally, the two problems are combined to solve the non-convex, limited range, spatial load balancing problem with agents subject to differential constraints. The convergence of the algorithms is proven and confirmed in simulation.

Publications associated with this chapter

Chapter 6, in part, contains material as it appears in American Control Conference 2016. "Spatial load balancing in non-convex environments using sampling-based

motion planners" Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, contains material as it appears in American Control Conference 2017. "Limited range spatial load balancing for multiple robots" Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, contains material that has been submitted to Autonomous Robots 2017. "Limited Range Spatial Load Balancing in Non-Convex Environments using Sampling-Based Motion Planners", Boardman, Beth; Harden, Troy; Martínez, Sonia. The dissertation author was the primary investigator and author of this paper.

Chapter 7

Conclusion

The research within this dissertation pertains to robotic motion planning and falls into one of three categories. The first, Chapter 3, is improving the performance of the RRT* so that better cost paths are found quicker. The second category is replanning using sampling-based motion planners. The two replanning algorithms developed in this research are the Goal Tree algorithm in Chapter 4 and the Sampling-Based Collision Avoidance algorithm in Chapter 5. The final piece of this dissertation is the application of sampling-based motion planners to multi-agent spatial load balancing.

The Focused-Refinement (FR) and Grandparent-Connection (GP) are the two algorithms created to improve the performance of the asymptotically optimal Rapidly-exploring Random Tree (RRT*). Both algorithms maintain the asymptotic optimality and probabilistic completeness of the RRT*. Simulations compare the FR and GP to the RRT* and other similar planners.

The first replanning algorithm, the Goal Tree (GT) algorithm, is used when there are unknown static obstacles. Replanning regions are introduced that make the GT algorithm asymptoticly optimal. The GT algorithm is simulated on three different types of robots: Euclidean metric, Dubins' vehicle, and a seven degree-of-free manipulator.

From replanning due to static obstacles we moved to multiple agents in the same space. Where, the agents only account for the static obstacles when building their RRT*. The other agents are avoided by using collision cones and deconfliction maneuvers. In this way, the agents are replanning every time they detect a conflict with another agent. The Sampling-Based Collision Avoidance (SBCA) algorithm is shown to handle uncertainty in an agent's knowledge of the position and velocity of the other agents. After the RRT* and collision cones are introduced the algorithm details are given. The algorithm is analyzed to prove that the algorithm with perfect and uncertain information will never cause the agents to collide. Simulations show that a collision free solution is found under uncertainty knowledge.

Lastly, a limited range spatial load balancing problem for agents subject to differential constraints in a non-convex environment is defined and discussed. To handle differential constraints, the problem is redefined using a probabilistic roadmap star (PRM*) and the GRAPH-BASED SPATIAL LOAD BALANCING algorithm is given that finds an approximate solution to the original problem. We discuss how introducing the limited range sub-partition, and determining a subset of agents containing the Voronoi neighbors of a specific agent, limits the amount of communication between agents. A convergence proof is given for the GRAPH-BASED SPATIAL LOAD BALANCING algorithm with limited ranges. All other defined approximated problems are shown to converge in simulation.

7.0.1 Future Work

This section briefly discusses some area of future work that could be pursued. First, in the area of improved performance of the RRT*, future work includes implementing the two algorithm modifications on robots in more complex environments. It would be beneficial to see how the improvements scale for higher dimensional systems

such as a seven degree-of-freedom manipulator. In the area of static replanning, a future research direction is extending the Goal Tree algorithm to handle the removal of obstacle. One option is to run the existing path through a path smoothing algorithm every time an obstacle is removed. For dynamic replanning, it would be interesting to incorporate the GT algorithm into the SBCA algorithm for handling unknown static obstacles. It is also of interest to expand the definition of collision cone for agents with differential constraints. In example, if the agents were Dubins' vehicles or unicycle model, could an object be defined that checks the trajectories for future collision. Finally, in the area of spatial load balancing, future work includes making more extensive differential constraint simulations as well as implementation of the algorithm on a set of mobile robots.

Bibliography

[1] B. Akgun and M. Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 2640–2645, 2011.

[2] O. Arslan and P. Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *IEEE Int. Conf. on Robotics and Automation*, pages 2421–2428, 2013.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[4] S. Bhattacharya, R. Ghrist, and V. Kumar. Multi-robot coverage and exploration on riemannian manifolds wth boundaries. *International Journal of Robotics Research*, 33:113–137, 2014.

[5] S. Bhattacharya, N. Michael, and V. Kumar. Distributed coverage and exploration in unknown non-convex environments. In *International Symposium on Distributed Autonomous Robotic Systems*, pages 61–75. Springer, 2013.

[6] S. Biswas, S. G. Anavatti, and M. A. Garratt. Obstacle avoidance for multi-agent path planning based on vectorized particle swarm optimization. In *Intelligent and Evolutionary Systems: The 20th Asia Pacific Symposium, IES 2016, Canberra, Australia, November 2016, Proceedings*, pages 61–74. Springer, 2017.

[7] A. Breitenmoser, M. Schwager, J.-C. Metzger, R. Siegwart, and D. Rus. Voronoi coverage of non-convex environments with a group of networked robots. In *IEEE Int. Conf. on Robotics and Automation*, pages 4982–4989, 2010.

[8] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, volume 3, pages 2383–2388, 2002.

[9] J. W. Bruce and P. J. Giblin. An elementary approach to generic properties of plane curves. *Proceedings of the American Mathematical Society*, pages 455–458, 1984.

[10] C. H. Caicedo-Nunez and M. Zefran. Performing coverage on nonconvex domains. In *IEEE Conf. on Control Applications*, pages 1019–1024, 2008.

[11] S. Carpin and G. Pillonetto. Motion planning using adaptive random walks. *IEEE Transactions on Robotics*, 21(1):129–136, 2005.

[12] Y. Chen, M. Cutler, and J. P. How. Decoupled multiagent path planning via incremental sequential convex programming. In *icra*, pages 5954–5961. IEEE, 2015.

[13] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms and Implementations*. The MIT Press, 2005.

[14] J. Cortés. Coverage optimization and spatial load balancing by robotic sensor networks. *IEEE Transactions on Automatic Control*, 55(3):749–754, 2010.

[15] J. Cortés, S. Martínez, and F. Bullo. Spatially-distributed coverage optimization and control with limited-range interactions. *ESAIM. Control, Optimisation & Calculus of Variations*, 11(4):691–719, 2005.

[16] E. W. Dijkstra. A note on two problems on connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[17] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, pages 497–516, 1957.

[18] J. Enright, K. Salva, and E. Frazzoli. Coverage control for nonholonomic agents. In *IEEE Int. Conf. on Decision and Control*, pages 4250–4256, 2008.

[19] D. Ferguson, N. Kalra, and A. Stentz. Replanning with RRTs. In *IEEE Int. Conf. on Robotics and Automation*, pages 1243–1248, 2006.

[20] H. Flanders. Differentiation under the integral sign. *The American Mathematical Monthly*, 80(6):615–627, June 1973.

[21] R. Gayle, K. R. Klingler, and P. G. Xavier. Lazy Reconfiguration Forest (LRF) - An approach for motion planning with multiple tasks in dynamic environments. In *IEEE Int. Conf. on Robotics and Automation*, pages 1316–1323, 2007.

[22] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan. RRT-smart: Rapid convergence implementation of RRT towards optimal solution. In *IEEE Int. Conf. on Mechatronics and Automation*, pages 1651–1656, 2012.

[23] K. J. J, J and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 995–1001, 2000.

[24] L. Janson and M. Pavone. Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions. In *International Symposium on Robotic Research*, 2013.

[25] W. Jiang and M. Zefran. Coverage control with information aggregation. In *IEEE Int. Conf. on Decision and Control*, pages 5421–5426. IEEE, 2013.

[26] Y. Kantaros, M. Thanou, and A. Tzes. Visibility-oriented coverage control of mobile robotic networks on non-convex regions. In *IEEE Int. Conf. on Robotics and Automation*, pages 1126–1131. IEEE, 2014.

[27] Y. Kantaros, M. Thanou, and A. Tzes. Distributed coverage control for concave areas by a heterogeneous robot–swarm with visibility sensing constraints. *Automatica*, 53:195–207, 2015.

[28] S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *IEEE Int. Conf. on Decision and Control*, pages 7681–7687, 2010.

[29] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

[30] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the RRT*. In *IEEE Int. Conf. on Robotics and Automation*, pages 1478–1483, 2011.

[31] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[32] S. Koenig and M. Likhachev. $d^\star$ lite. In *American Association for Artificial Intelligence*, 2002.

[33] M. Kothari and I. Postlethwaite. A probabilistically robust path planning algorithm for uavs using rapidly-exploring random trees. *jirs*, pages 1–23, 2013.

[34] A. Kwok and S. Martínez. Deployment algorithms for a power-constrained mobile sensor network. *International Journal on Robust and Nonlinear Control*, 20(7):725–842, 2010.

[35] A. Kwok and S. Martínez. Unicycle coverage control via hybrid modeling. *IEEE Transactions on Automatic Control*, 55(2):528–532, 2010.

[36] E. Lalish. *Distributed reactive collision avoidance for multivehicle systems*. PhD thesis, PhD thesis, University of Washington, 2009.

[37] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[38] K. Laventall and J. Cortés. Coverage control by robotic networks with limited-range anisotropic sensory. In *American Control Conference*, pages 2666–2671, Seattle, WA, 2008.

[39] T.-Y. Li and Y.-C. Shie. An incremental learning approach to motion planning with roadmap management. In *IEEE Int. Conf. on Robotics and Automation*, volume 4, pages 3411–3416, 2002.

[40] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[41] W. Luo, N. Chakraborty, and K. Sycara. Distributed dynamic priority assignment and motion planning for multiple mobile robots with kinodynamic constraints. In *acc*, pages 148–154. IEEE, 2016.

[42] H. Mahboubi, K. Moezzi, A. G. Aghdam, K. Sayrafian-Pour, and V. Marbukh. Distributed deployment algorithms for improved coverage in a network of wireless mobile sensors. *IEEE Transactions on Industrial Informatics*, 10(1):163–174, 2014.

[43] S. Martínez. Yo, robot (I, Robot). MatePoster (a math poster for high-school students). Published through the Spanish digital magazine website at `http://www.matematicalia.net` as an invited contribution.

[44] A. Murano, G. Perelli, and S. Rubin. Multi-agent path planning in known dynamic environments. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 218–231. Springer, 2015.

[45] A. A. Neto, D. Macharet, L. Chaimowicz, and M. Campos. Path planning with multiple rapidly-exploring random trees for teams of robots. In *icar*, pages 1–6. IEEE, 2013.

[46] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley Series in Probability and Statistics. John Wiley, 2 edition, 2000.

[47] M. Otte and S. Frazzoli. rrt^x: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *International Journal of Robotics Research*, 2015.

[48] R. Patel, P. Frasca, and F. Bullo. Centroidal area-constrained partitioning for robot networks. *ASME Journal on Dynamic Systems, Measurement, and Control*, 136(3):031024–1–031024–8, 2014.

[49] M. Pavone, A. Arsie, E. Frazzoli, and F. Bullo. Distributed algorithms for environment partitioning in mobile robotic networks. *IEEE Transactions on Automatic Control*, 56(8):1834–1848, 2011.

[50] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. R. Walter. Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 4307–4313, 2011.

[51] L. Pimenta, V. Kumar, R. C. Mesquita, and G. Pereira. Sensing and coverate for a network of heterogeneous robots. In *IEEE Int. Conf. on Decision and Control*, pages 3947–3952, Cancun, Mexico, December 2008.

[52] M. Ragaglia, M. Prandini, and L. Bascetta. Multi-agent poli-rrt. In *International Workshop on Modelling and Simulation for Autonomous Systems*, pages 261–270. Springer, 2016.

[53] A. Renzaglia and A. Martinelli. Distributed coverage control for a multi-robot team in a non-convex environment. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2009.

[54] G. Sánchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *International Symposium on Robotic Research*, pages 403–417. Springer, 2003.

[55] K. Savla, F. Bullo, and E. Frazzoli. The coverage problem for loitering Dubins vehicles. In *IEEE Int. Conf. on Decision and Control*, pages 1398–1403, New Orleans, LA, Dec. 2007.

[56] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470–495, 2013.

[57] A. Shkolnik and R. Tedrake. Sample-based planning with volumes in configuration space. *Computing Research Repository*, arXiv:1109.3145, 2011.

[58] A. Stentz. The focussed D* algorithm for real-time replanning. In *International Joint Conference on Artificial Intelligence*, volume 95, pages 1652–1659, 1995.

[59] Y. Stergiopoulos and A. Tzes. Coverage-oriented coordination of mobile heterogeneous networks. In *Mediterranean Conf. on Control and Automation*, pages 175–180, 2011.

[60] N. Virani and M. Zhu. Robust adaptive motion planning in the presence of dynamic obstacles. In *acc*, pages 2104–2109. IEEE, 2016.

[61] M. Waringo and D. Henrich. Efficient smoothing of piecewise linear paths with minimal deviation. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 3867–3872, 2006.

[62] Z. Zhen, C. Gao, Q. Zhao, and R. Ding. Cooperative path planning for multiple uavs formation. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on*, pages 469–473. IEEE, 2014.

[63] M. Zhong and C. G. Cassandras. Distributed coverage control and data collection with mobile sensor networks. *IEEE Transactions on Automatic Control*, 56(10):2445–2455, 2011.

[64] M. Zucker, J. Kuffner, and M. Branicky. Multipartite RRTs for rapid replanning in dynamic environments. In *IEEE Int. Conf. on Robotics and Automation*, pages 1603–1609, 2007.